opentext™

OpenText™ Information Archive

Fundamentals Guide

Learn the concepts, functionality, and features required to preserve, maintain, and control continuing access to valuable enterprise information assets.

EARCORE250400-ACS-EN-01

OpenText™ Information Archive
Fundamentals Guide
EARCORE250400-ACS-EN-01
Rev.: 2025-Sept-08

This documentation has been created for OpenText™ Information Archive CE 25.4.
It is also valid for subsequent software releases unless OpenText has made newer documentation available with the product, on an OpenText website, or by any other means.

Open Text Corporation

275 Frank Tompa Drive, Waterloo, Ontario, Canada, N2L 0A1

Tel: +1-519-888-7111
Toll Free Canada/USA: 1-800-499-6544 International: +800-4996-5440
Fax: +1-519-888-0677
Support: https://support.opentext.com
For more information, visit https://www.opentext.com

Table of Contents

1 What is OpenText Information Archive? 7
1.1 What can be archived? 8
1.2 Key features 8
1.3 High-Level overview: How to use OpenText Information Archive 10
1.4 Architecture 12

2 Archiving overview 15
2.1 Backing up vs. archiving data 15
2.2 Standards 15
2.3 Table archiving and SIP archiving 16
2.3.1 File archiving 17
2.3.2 Data record archiving 17
2.3.3 Compound-record archiving 18
2.4 The archiving process 18

3 What type of archiving should I use? 21
3.1 Setting archiving goals 21
3.2 Understanding the source application 21
3.3 Selecting the appropriate archiving method 22
3.4 Use cases 23

4 Applications and data 25
4.1 How data is ingested 26
4.2 How data is stored 26
4.2.1 Cloud storage 27
4.2.2 Custom 27
4.3 How data is searched 28
4.3.1 Search results and saved searches 29
4.3.2 Matters 30
4.4 How OpenText Information Archive is configured for data 30
4.4.1 Configuration objects 31
4.5 How information is protected 32

5 Table archiving fundamentals 33
5.1 Metadata file 34
5.2 Table data files 35
5.3 The configuration YAML file 36
5.4 Searches 36
5.5 Installation and ingestion 37

6 SIP archiving fundamentals 39
6.1 Glossary 39

6.2 SIP ... 39
6.2.1 Best practices for SIPs ... 40
6.3 SIP descriptor ... 41
6.3.1 About data submission sessions (DSS) ... 42
6.3.2 About external ID ... 43
6.4 PDI ... 44
6.5 PDI schema ... 45
6.6 AIP ... 46
6.7 Holding ... 47
6.7.1 Ingestion mode ... 48
6.7.2 Close ... 50
6.8 Cache-in and cache-out ... 51
6.9 Adding a SIP to an archive ... 53
6.9.1 Invalidation and rejection ... 54

7 Operational concepts ... 55
7.1 Authentication mechanisms ... 55
7.1.1 Authenticating users – configuring OpenText Directory Services ... 56
7.1.1.1 Types of OTDS integrations ... 57
7.2 Authorization ... 59
7.3 User roles and actions ... 59
7.4 Configuring groups to access OpenText Information Archive ... 61
7.5 Auditing ... 62
7.5.1 Default retention on audits ... 63
7.6 Dashboards ... 63

8 Background processing – General concepts ... 65
8.1 Jobs ... 65
8.1.1 Anatomy of a job definition ... 66
8.2 Background requests ... 66
8.3 Scheduled tasks ... 67

9 Compliance – General concepts ... 69
9.1 Compliance-related roles ... 70
9.2 The retention lifecycle ... 70
9.3 Architecture ... 70
9.4 What is a retention policy? ... 71
9.4.1 What is a retention set? ... 73
9.4.2 Applying retention ... 74
9.4.2.1 Deciding where to apply retention ... 74
9.4.2.2 Applying retention to records ... 76
9.4.2.3 Storage considerations ... 77
9.4.2.4 Applying retention automatically ... 77

9.4.2.5 Using jobs to apply retention 78
9.4.2.6 Configuring retention options on the holding after ingestion 79
9.4.2.7 Applying multiple retention policies 80
9.5 What is a hold? 81
9.5.1 What is a hold set? 81
9.5.2 Legal Matters 81
9.5.3 Where to apply a hold 82
9.5.4 Rerunning and removing items of a saved search with holds 83
9.6 Granularity 84
9.7 What is disposition? 84
9.7.1 What is a purge candidate list? 86
9.7.2 Disposition processing 87
9.7.3 What is granular disposition? 88
9.7.3.1 Timing of applying holds when using granular retention 89
9.7.4 Disposition and unstructured content 90
9.7.5 Table disposition and storage footprint 91
9.8 Event-based retention 91
9.8.1 What are compliance events? 91
9.8.2 Basic flow 92
9.8.3 How does aging work? 94
9.8.4 Audits 94
9.9 Rules 95
9.9.1 Accessing external systems 97

Chapter 1

What is OpenText Information Archive?

OpenText Information Archive is a powerful, secure, and scalable archiving solution. It preserves, maintains, and controls continuing access to valuable enterprise information assets. It is also application agnostic, providing one unified archive for all your application data.

As organizations generate more data, and retain data longer, IT costs continue to rise. With OpenText Information Archive, you can decommission legacy applications to reduce costs (maintenance, special hardware, and consultants). You can also archive data from live systems which will reduce the load on the systems, reduce hardware requirements, and improve performance.

OpenText Information Archive helps make better use of your structured and unstructured data by opening availability and putting the data to work in new ways for the business. It is standards compliant, presents data in any format, and features robust compliance functionality so that you can meet all retention requirements.

There are two main use cases for OpenText Information Archive:

Decommissioning legacy applications

Often, legacy applications are no longer officially supported, run on older hardware that is nearing the end of its lifecycle, and contain data that is no longer being updated. But this data can be valuable to the business and important to preserve for compliance reasons.

You can archive the data in a standard format on lower-cost storage, create searches to make the data accessible to your users, and then safely switch off the legacy application. With decommissioning, data is typically extracted all at once from the source application and then archived in OpenText Information Archive.

Live archiving

You can decrease the load on a live application by offloading some of the application's data to OpenText Information Archive, which improves performance and reduces hardware costs,

The data that you choose to archive, and the frequency of archiving, is based on your business rules. For example, a bank can archive all bank statements that are a year old or more on a weekly basis.

After the data has been archived, OpenText Information Archive sends a confirmation to the source application to purge the data. With live archiving, data is continually added to the source application and then archived at defined points.

1.1 What can be archived?

OpenText Information Archive stores both structured and unstructured data:

- Structured data is information that is separated into independent, predictable parts, defined by metadata. It stores this content as XML. Examples of structured data include transactions from ERP systems, legal records, and economic data.
- Unstructured data is information that does not follow a specified format or predefined model. The system can store this content on a file system or cloud storage, and stores the content's metadata as XML. Examples of unstructured content include Microsoft Office documents, print streams, image files, and videos.

OpenText Information Archive can be configured to extract and index textual content from unstructured content during the archiving process, to be archived along with the structured data. You can include language hints/metadata in your configuration and your ingested data to allow for language-specific indexing and searching.

Depending on the data source, your data might be a combination of structured and unstructured data. OpenText Information Archive supports the following types of data sources:

Data source	Type of data	Example
Tables	Mostly structured, might also have some unstructured data	Tables from an RDBMS, such as an Oracle database
Files	Often unstructured, might also have some structured data	Media archives
Data records	Often structured, might also have some unstructured data	Transaction histories
Compound records	Unstructured and structured data in a single record	Laboratory reports

1.2 Key features

- Archives structured, unstructured, and all information types in a single archive.
- Flexible, controlled access to archived data
  - Allow authorized users to search, view, and export content.
  - Role-based permissions and data encryption
  - Customize searches and the look and feel of the interface.
  - Use APIs to integrate query capabilities into your business applications.
- Compliance with open industry standards

  - Reference Model for an Open Archival Information System (OAIS)
  - Extensible Markup Language (XML)
  - SQL
- No dependencies on the originating application
- Synchronous (transactional) and asynchronous (batch) ingestion
  - Archive large and steady input streams of data
  - Schedule ingestion in batches for optimal performance when information comes in intermittently.
- Powerful tools for archiving operations and management, including compliance tasks, audits, and metrics.
  - Create and apply retention policies and legal holds.
  - Apply policies at different levels, including application, package, and record.
  - For SIP applications, apply retention during or after ingestion.
  - Granular disposition of records.
- Extensible and customizable
  - Multi-vendor support for storing unstructured data, including Amazon S3 and Microsoft Azure.
  - Integrate any storage type using the storage API interface.
  - Configure the presentation of search results using standard HTML templates.
- Highly scalable, can meet increasing demands and complexity.
  - Supports large volumes of data.
  - Supports large numbers of source applications.
  - Supports increasing complexity of regulations.
- Minimizes IT costs.
  - Move data to lower-cost storage.
  - Save maintenance and licensing costs for legacy applications.
  - Reduce the load on live applications.

1.3 High-Level overview: How to use OpenText Information Archive

OpenText Information Archive is a powerful and versatile tool. Having a sense of how its functionality fits together, and when to use which features, can speed up your success.

The following table presents the major goals and tasks when using OpenText Information Archive, from learning about the product through maintaining and monitoring a fully configured and live production deployment. For information, refer to section 3 "Quickly installing a demo configuration" in *OpenText Information Archive - Installation Guide (EARCORE-IGD)* and section 10 "Upgrading OpenText Information Archive" in *OpenText Information Archive - Installation Guide (EARCORE-IGD)*.

Goal	Task	Where to find more information
Learn about OpenText Information Archive and archiving	• Understand what OpenText Information Archive does • Understand how OpenText Information Archive archives data • Understand how to satisfy your archiving goals • Determine which types of archiving to use • Understand the areas of functionality and technical components	• What is OpenText Information Archive? • Archiving overview • What type of archiving should I use? • Section 1.1 "System components" in *OpenText Information Archive - Installation Guide (EARCORE-IGD)*
Install a demo configuration	• Test and get familiar with the software.	• Section 3 "Quickly installing a demo configuration" in *OpenText Information Archive - Installation Guide (EARCORE-IGD)*
Prepare the source application for archiving	• Model the information that you want to archive. • Prepare data for ingestion (configure ETL tools or connectors, and generate SIPs).	• Applications and data • Data organization for table archiving • Data organization for SIP archiving • Operational concepts

Goal	Task	Where to find more information
Install OpenText Information Archive and configure general settings	• Install a secure production configuration. • Set up disaster recovery. • Upgrade an older version of OpenText Information Archive. • Connect to user directory services. • Configure backup and restore. • Configure search. • Configure language support and customize branding.	• Section 5 "Planning a production configuration" in *OpenText Information Archive - Installation Guide (EARCORE-IGD)* • *Best Practises OpenText Information Archive Setup: Demo vs. Production* on support.opentext.com (hthttps://support.opentext.com/) • *Best Practises OpenText Information Archive Disaster Recovery* on support.opentext.com (https://support.opentext.com/) • *OpenText Information Archive - A Quick Guide to OpenText Information Archive (EARCORE-AQS)* • *OpenText Information Archive - Configuration Guide (EARCORE-CGD)* • *OpenText Information Archive - Administration Guide (EARCORE-AGD)*
Configure archiving and administer OpenText Information Archive	• Create and configure OpenText Information Archive applications, searches, and audits. • Archive data • Create and perform searches. • Manage user accounts and permissions. • Create storage. • Manage packages and jobs. • Administer auditing and compliance. • Maintain and monitor OpenText Information Archive.	• *OpenText Information Archive - Configuration Guide (EARCORE-CGD)* • *OpenText Information Archive - Administration Guide (EARCORE-AGD)* • *OpenText Information Archive Shell Guide (EARCORE-ARE)*

1.4 Architecture

OpenText Information Archive has the following components:

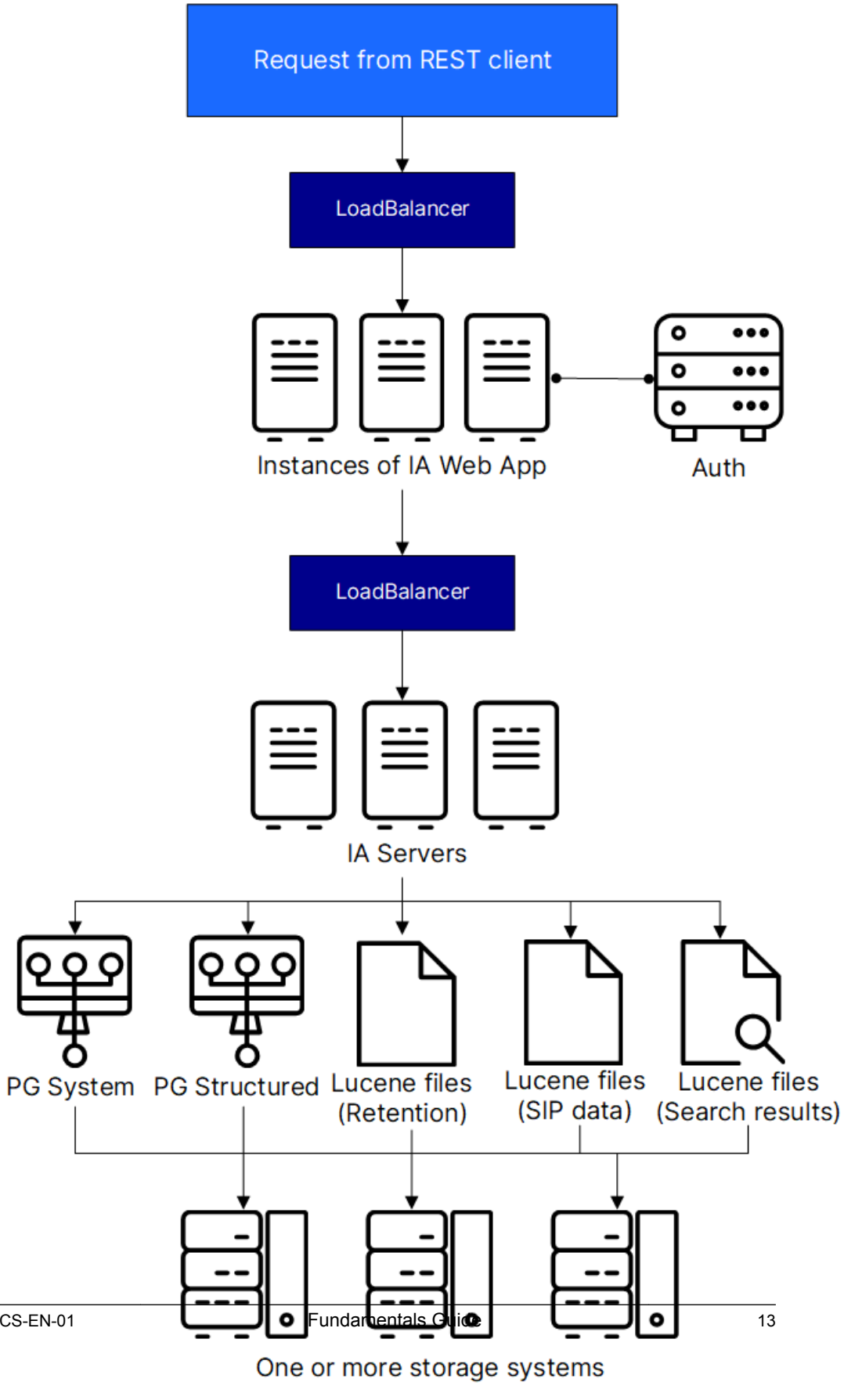
- IA Server performs most of the functions. The server leverages PostGreSQL and Lucene for your information.
- IA Web App provides the user interface for interacting with OpenText Information Archive.
- IA Shell provides a command line interface for interacting with OpenText Information Archive.

Typically, in a production environment, multiple IA Servers and IA Web App components would be deployed with a load balancer. Storage systems are used for your unstructured data.

The following diagram illustrates the OpenText Information Archive architecture, showing its layers and components. Requests, such as data ingestion, search, application configuration, *etc*. typically originate from IA Shell or a browser client (signified by the box at the top of the diagram marked "Request from REST client"). From here, a load balancer passes them to an IA Web App/Gateway server. Here, authentication is checked and authorization granted, typically in combination with another component, such as OTDS. For more information, see Authentication mechanisms. Requests are then passed on (again through a load balancer) to an IA Server, which processes the request and returns a response.

Dependent on the request, IA Server accesses (read, update, write, delete) various types of data:

- (always) system data, which is stored in PostgreSQL. Examples of system data are the configuration of every archived application, pipelines for data export, definitions of administrative processes (so-called jobs), a directory structure keeping track of the location (in other storage) of every archived piece of content, *etc*.
- (dependent on request) other data, stored in PostgreSQL, so-called Lucene indexes, and storage systems, such as file systems and object stores. This is mainly the archived data itself, but also data to keep track of applied compliance policies on (collections of) data records and temporary data, such as search results. For more information on data types and storage, see How data is stored.


Request from REST client
LoadBalancer
Instances of IA Web App
Auth
LoadBalancer
IA Servers
PG System
PG Structured
Lucene files (Retention)
Lucene files (SIP data)
Lucene files (Search results)
One or more storage systems

Refer to the following to learn more about OpenText Information Archive architecture:

- Section 1.1 “System components” in *OpenText Information Archive - Installation Guide (EARCORE-IGD)*
- Section 1.2 “System directories” in *OpenText Information Archive - Installation Guide (EARCORE-IGD)*
- Section 3 “Quickly installing a demo configuration” in *OpenText Information Archive - Installation Guide (EARCORE-IGD)*
- *OpenText Information Archive Setup: Demo vs. Production* whitepaper in the Documentum Champion Toolkit
- Section 2.4 “OpenText Information Archive applications” in *OpenText Information Archive - Installation Guide (EARCORE-IGD)*
- Section 3.3 “Installing applications for a demo configuration” in *OpenText Information Archive - Installation Guide (EARCORE-IGD)*
- Section 2.5 “Connectors for the ETL process” in *OpenText Information Archive - Installation Guide (EARCORE-IGD)*

Chapter 2

Archiving overview

2.1 Backing up vs. archiving data

Archiving data differs from backing up data. When you back up data, you create a copy of the data in a recovery mechanism, to protect against the event that the data is accidentally destroyed or corrupted. When you archive data, you store and protect information that is not needed for everyday operations, but still the information must be available. You can archive the data on less expensive storage systems and retain it to meet operational or regulatory requirements.

Backup	Archive
Offers protection: frequent snapshots of data to protect against data loss	Offers preservation: movement of data to a low-cost platform to ensure compliance and reduce costs
Copies data	Moves data off production disk
Supports operations and recovery	Supports business and compliance
Supports availability	Supports operational efficiencies
Point-in-time only	Comprehensive in nature
Poor solution for regulatory compliance	Ideal solution for regulatory compliance
Not easily searched	Easily searched
Often, old backups cannot be restored	Provides historic reference

2.2 Standards

OpenText Information Archive uses the following standards to ingest, store, query, and retrieve data during the archiving process:

Standard	Definition	Use in OpenText Information Archive
OAIS (ISO 14721)	A reference model used by a wide variety of organizations for archiving digital information for long-term preservation.	Specifies the format that data is ingested into, stored in, and retrieved from OpenText Information Archive.

Standard	Definition	Use in OpenText Information Archive
XML	A markup language for encoding information in a format that is readable by humans and machines. Designed for long-term, platform-independent retention.	Format for archiving structured data and metadata.
XQuery	A query language for XML data.	Used for value lists and other functionality.
SQL	A query language for relational data.	Used for search in archived data.

2.3 Table archiving and SIP archiving

An archive can be either table-based or SIP-based:

- A *table-based archive* uses a schema to ingest structured data and linked files from a table (for example, an RDBMS).

  You should use table archiving to migrate structured data in application tables and linked files from transaction systems to OpenText Information Archive with few transformations. Table archiving can reduce the up-front analysis involved with decommissioning an application and eliminate the data-integrity risks associated with other archiving methods.

- A *SIP-based archive* uses submission information packages (SIPs) to ingest data from files, data records, or compound records.

  For example, a customer record could include contact information (structured content), a picture of the customer (unstructured content), transactions (structured content), and a contract (unstructured content). A package bundles these different types of content together into XML data and metadata, and content files.

There are three types of SIP-based archives: file archives, data-record archives, and compound-records archives.

2.3.1 File archiving

A file archive stores unstructured data and its associated metadata in a single record. The data is preserved in its original format or transformed into a future-proof format, such as PDF-A. One record can contain several files to create sets of related information. Metadata attributes are derived from the content itself, or associated with other systems.

File archiving is useful when you want to reuse data in a context that is different from the source application. For example, you can transform large print streams (such as customer statements) from print-oriented formats (such as AFP or metacode) into a PDF available on a web-based platform. Or you can reprocess image archives (typically multi-page TIFF files with limited attributes) to add document types and full-text optical character recognition (OCR). Users find this information more easily by searching for the associated application metadata rather than directly accessing the infrastructure.

2.3.2 Data record archiving

A data-record archive stores structured data in a single record. One record can contain multiple XML packets, and information from multiple systems are drawn together according to the requirements of the project while preserving a complex multi-system chain-of-custody.

Data record archiving is useful when you want to reuse data, while reducing costs, in a context that is different from the source application. Also, data archiving is well suited for the active archiving of live systems. It requires additional and more business-oriented analysis of application data than table archiving does. Examples include SWIFT transactions, sales histories, and patient histories.

There are two advantages of data record archiving:

- The complex data model of the source application is transformed into a simple data model in the archive. This reduces costs and simplifies future access.
- Because there is no direct link between the source application and the data in the archive, changes in the source application does not force a change in the archive. When a change in the source application results in an update to the archive data model, OpenText Information Archive ensures that results for searches of data sets include all records across all changed data. For table archiving, part of the responsibility for ensuring this resides with the search designer who writes the SQL queries for searching the archived data is partially responsible for table archiving.

By extending or recording metadata, you can harmonize records and support searching and filtering across data sets.

Data archiving is used with transaction systems for active archiving of individual structured data records (for example, transaction history tables). It is also used with interaction systems (for decommissioning data and queries, optimizing searches, and advanced analytics), and with content systems. Because it presents data as

single records, it is ideal for archiving information according to government requirements and legal mandates.

2.3.3 Compound-record archiving

A compound-record archive stores structured and unstructured data in a single record. The structured elements are modelled as XML, and the unstructured elements are preserved in the original format or transformed into a more future- proof format, such as PDF-A. One record contains multiple files of related information.

Compound-record archiving is useful when you want to archive systems with structured information (such as wikis or blogs) with unstructured information (such as attachments), while meeting compliance rules.

As business processes become more complex and regulations more demanding, there is a growing need to archive complex business records that contain both structured and unstructured data, which must be brought together to create a final business record. Examples include financial trades, cases and laboratory reports. Customers can retain business events as single records that they can reuse for analytics or regulatory audits. You can search the records using pure business logic without switching from one application to another.

2.4 The archiving process

The following steps describe the archiving process at a high level, for both table archiving and SIP archiving. The SIP archiving steps and data formats for ingestion, storage, and retrieval comply with the OAIS data model, including the naming conventions for information packages (SIP, AIP, and DIP).

Note: While OpenText Information Archive accepts data for ingestion as XML, it does not transform the data from its source format into XML. This transformation must take place before OpenText Information Archive ingests the data. You can use whichever Extract, Transform, Load (ETL) tools that you prefer to perform this transformation, or you can use connectors that are available for SAP, SharePoint, OpenText File Intelligence (OTFI), and other sources, including partner tools.

1. An ETL tool or connector extracts data and metadata from the producer (the source application). For table archiving, the tool or connector converts structured data and metadata to XML, and stores unstructured data as content files. For SIP archiving, the tool or connector packages files, data records, and compound records into Submission Information Packages (SIPs). The SIPs contain both structured data and metadata as XML and unstructured data.
2. The customer ingests the table data and SIPs into an application. For SIP applications, the ingestion process verifies the SIPs, and creates an Archival Information Package (AIP) from each SIP. The frequency of ingestion is based on the organization's business rules.
3. OpenText Information Archive transfers the table data and AIPs to archived storage. It stores structured data, including any extracted text from unstructured

data, in a relational database (table archiving) or as Lucene index files (SIP archiving) and unstructured data as files in a configurable storage system (for example, a file system, EMC ECS, Amazon S3, or CAS).

4. OpenText Information Archive can create a confirmation for a SIP application, to report on the ingested data. Table applications have the ability to run chain of custody tests to insure the integrity of the ingested data.
5. For table archiving, it is vital that you run the IA Shell command `index-build` for the database before running any searches or other operations on that data.
6. Customers search for information from the archive by performing a synchronous search (for immediate results) or creating an asynchronous background request for the records.
7. Customers can apply policies to the data that indicate when the data should be purged (disposed) from the system. Applying retention can be automated for SIP applications during ingestion.
8. Customers can determine when content is eligible for disposition and then approve and dispose of the data.

Chapter 3

What type of archiving should I use?

You must determine which of the four archive types is the best method for archiving your data:

- Table archiving
- File archiving
- Data record archiving
- Compound record archiving

There are three basic steps to making this decision:

1. Set your archiving goals.
2. Understand your source application.
3. Select the appropriate archiving method.

3.1 Setting archiving goals

You should consider the following questions when setting your archiving goals:

- Do you want to decommission legacy applications or set up live archiving for active applications?
- Do you want to apply analytics to archived data, or expose the data for other uses?
- Do you need to accommodate many different applications and data formats?

Live archiving can use any of the four archive types, except for table archiving. If information aggregation and reuse is important to you, then you should consider file, data record, and compound record archiving.

3.2 Understanding the source application

When choosing the best archiving method for a source application, you should consider the following:

- What type of data do you want to archive?
- Going forward, how will users access the data?

Most organizational data is managed in one of the following system types:

System type	Description
Transaction systems	Transaction systems have databases that hold details of past business events related to accounting processes, enterprise resource planning (ERP), enterprise asset management, and supply chain management. Transaction systems are used to maintain reference data in master files, record activities in transaction files, and store old records in transaction history tables. They include cloud-based systems and allow many people to add small bits of detail over time.
Print stream systems	Referred to as COLD (computer output to laser disk) systems, these systems store print-stream data for long-term preservation. Most of this data consists of customer communications, but another example is green-bar reporting systems.
Content and image repositories	Content and image repositories store unstructured information and metadata, typically in their native formats. Examples include traditional enterprise content management (ECM) systems, as well as storage-based systems.
Interaction systems	Interaction systems connect users with an organization for quick access to complete information. Examples include systems that support customer relationship management (CRM) and collaborative tasks. These systems include data as well as transaction, grouping, and unstructured files.
Collaborative systems	Collaborative systems address the needs of groups of individuals to share information and communicate with each other around specific topics. These systems have all the characteristics of interaction systems, but generally cater to a less-structured approach. Notable examples include eRoom, Microsoft SharePoint, and Lotus Notes.

3.3 Selecting the appropriate archiving method

The four archiving methods offered by OpenText Information Archive are optimized based on the format of the data that is being archived, the ease of extraction and up-front analysis, and how the data is to be used after it is ingested. This choice is a critical success factor for large-scale information management programs that involve a wide range of applications. For more information about these archiving methods, see the following sections:

- Table and SIP archiving
- File archiving
- Data record archiving
- Compound record archiving

3.4 Use cases

The following are examples of how you might use OpenText Information Archive:

Archiving goals	Source application analysis	Consider these archiving methods
Reduce IT costs for legacy applications	Legacy and redundant applications have been superseded by an enterprise resource planning (ERP) system, replicated during an acquisition, or must be decommissioned as part of a business sale, closure, or industry mandate. Application data must remain accessible for business reporting, audits, and compliance with data-retention regulations. Once the organization shuts down the applications, it can save on maintenance and support costs.	Application decommissioning Table archiving
Make production applications and infrastructure more efficient Compliant data retention	Production costs for live business applications have been escalating while performance has been degrading. Data should be periodically archived to reduce the demand on the applications, reduce storage and backup costs, and reduce licensing and administration costs. Large volumes of inactive transaction records (such as checks and statements) must be retained to meet industry regulations. Information from some completed projects (such as such as pharmaceutical studies, cases, and construction projects) must be archived together.	Live archiving Data record archiving Compound record archiving
Put data to work in new ways Remove information silos	New and innovative uses for data are becoming business requirements, such as advanced and predictive analysis and application modernization programs. A platform is needed for data aggregation and management, offering access to business records individually through web services, or in bulk through the Hadoop Distributed File System (HDFS).	Live archiving File archiving Compound record archiving

Chapter 4

Applications and data

An OpenText Information Archive application defines and provides access to archived data. The application defines the structure of your data and also provides the ability to associate unstructured data with your records. Typically, most end users access archived information by using the IA Web App, where they are simply called applications. It is also possible to use IA Shell or the REST API to interact with the applications. Certain operations can only be done using the IA Shell.

When creating an application, the Developer needs to choose between the following types:

- SIP
- Table

The choice made is transparent to the end users. You can use IA Web App to create and configure applications, however, once you are familiar with OpenText Information Archive, you will most likely prefer to configure applications using declarative configuration (DC). This YAML-based, human-readable format gives you advanced options and control for configuring, importing, and exporting applications. See Section 8 “Declarative configuration” in *OpenText Information Archive - Configuration Guide (EARCORE-CGD)*" for more information.

Refer to Section 2 “Managing an OpenText Information Archive application” in *OpenText Information Archive - Configuration Guide (EARCORE-CGD)* to see an overview of the steps required to create SIP and table applications.

OpenText Information Archive provides many options for creating new applications. To help decide which options are best for your needs, there are many example applications that can be installed and studied. See Section 2.4 “OpenText Information Archive applications” in *OpenText Information Archive - Installation Guide (EARCORE-IGD)* and Section 13.10 “How example applications demonstrate product features” in *OpenText Information Archive - Installation Guide (EARCORE-IGD)* for more information.

4.1 How data is ingested

Before you ingest data into OpenText Information Archive from a source application, you must extract the data from the source application using an ETL tool or connector. The three ETL functions are as follows:

Function	Description
Extract	Extract data in its natural form from a source application, whether it is in a table or file format.
Transform	Apply rules or functions to convert source data into XML. Some systems can natively export in XML.
Load	Ingest the transformed XML data. Digital data storage (DDS) provides a toolset for loading data with assigned retention and metadata.

Use one of the OpenText Information Archive connectors created by OpenText, a partner or third- party tool, or a tool of your own to perform the first two functions, extract and transform. OpenText Information Archive is open to any ETL tool that can transform data into XML or a SIP. Some applications have a built-in facility to export to XML.

The Load (ingestion) function is always performed by an OpenText Information Archive application, and you must configure the application before ingestion.

4.2 How data is stored

As discussed at the beginning of this chapter, an application defines a space, which determines where to store both the data and the cached search results.

Where the data is stored on ingestion depends on whether the data is structured or unstructured and, if it is structured, whether it is table or SIP data:

- *Structured SIP data* (AIUs) is stored in Lucene index files. Lucene is an open-source indexing and querying API and implementation allowing software to store data and its indexes as files in a file system. So, instead of storing/indexing/querying, for example, structured data through a central database server, OpenText Information Archive stores/indexes/queries such data through accessing the corresponding Lucene files directly on the file system.
- *Structured table data* (records) is stored and indexed in a relational database system (PostgreSQL)
- *Unstructured data*, such as an image file, is stored in a configurable storage system (for example, a file system, ECS, S3, or CAS). Use the storage system's tool to create the storage (for example, use the ECS Management User Interface to create the bucket), and then use OpenText Information Archive to register the storage.
- *Embedded text* within unstructured content can be configured to be extracted, and then stored together with structured data (i.e., in Lucene for SIP data and PostgreSQL for table data).

When SIP or table data is ingested, it creates a container. For SIP data, a Content Information (CI) container is created. For table data, a Table Content container is created. For each ingested record, there will be a reference in the Record Index (RI) file, which acts as an index for the ingested data. If you ingest multiple times into a table, a new Table Content container and RI file are created.

Refer to Section 2.1 “Setting up storage for structured data” in *OpenText Information Archive - Administration Help (EARCORE-H-AGD)* and Section 2.2 “Setting up storage for unstructured data” in *OpenText Information Archive - Administration Help (EARCORE-H-AGD)* for more information.

Obviously, OpenText Information Archive needs to keep track of which applications there are, what AIPs, tables, search definitions, jobs, stores, *etc.* there are and in what storage systems and in which folders, buckets, *etc.* their data lives. All this is and more is stored in the System Database, which is a PostgreSQL database, to be set up before or during installation. For more information, see Section 5.3 “Planning a PostgreSQL server configuration” in *OpenText Information Archive - Installation Guide (EARCORE-IGD)* and its subsections.

4.2.1 Cloud storage

OpenText Information Archive supports cloud storage, such as Amazon S3 and Microsoft Azure. It can be used to store the CI container. The cloud storage option provides cheaper option to store information for the archive. It is important to take into consideration the available network bandwidth for such usage.

OpenText Information Archive also supports additional capabilities with Amazon Glacier. Amazon Glacier allows content to be stored offline to reduce storage costs. You can use Amazon S3 and choose whether to use a proxy.

All cloud storage options can be configured to use a proxy, if required.

4.2.2 Custom

OpenText Information Archive can also utilize any storage that is not part of the storage certification matrix. It can be achieved by implementing custom integration with OpenText Information Archive abstract storage API. The integration is only targeted to use other storage technologies for unstructured content storage.

4.3 How data is searched

Searching is the main way that a user accesses data that OpenText Information Archive has ingested. Some user roles can run searches for applications that they have permissions to see.

End Users perform searches in IA Web App, and Developers also use IA Web App to create searches.

Searches are slightly different for table and SIP archives:

- For a table archive, a search can be associated with a schema or table. Table-based searches require a SQL query and the schema or table that the search runs against.
- For a SIP archive, a search is associated with an archive information collection (AIC) and a query configuration. SIP-based searches require the name of the AIC and the query configuration, which determines the criteria and results for the search.

The Search Designer is responsible for defining a search which includes which criteria to use, which fields to show, and in the case of tables, how to query the data from the database. For example, when creating a table-based search form, the Search Designer specifies a schema and a table within the schema.

For table-based searches, the Search Designer must understand SQL and how it is related to search design. The Search Designer must ensure the following:

- The query is valid.
- If SQL is defined, all mandatory parameters without defaults must be supplied by the form.
- Bind the elements to columns.

A SIP-based search uses two queries:

- The first query returns the appropriate AIP based on the partition key. This information appears in the wizard when selecting the field for the search criteria.
- The second query filters the suitable AIU in the selected AIP list. Search performance can be improved by using and filling at least one partition key in the search form.

Search forms are customizable. A Search Designer can add or remove fields as required. Searches provide a rich graphical user interface for both criteria and results.

Result data might be simple text, a link to a nested search, downloadable content, or viewable content. The search results pages are customizable. A Search Designer can add or remove fields as required.

Searches can sometimes run in the background for a variety of reasons. OpenText Information Archive also supports cross-application searches to effectively run a set

of searches per application. For more information, see Section 6.2.4 “Creating a cross-application search” in *OpenText Information Archive - Configuration Guide (EARCORE-CGD)*.

4.3.1 Search results and saved searches

OpenText Information Archive caches search results in a separate configurable store. It will retain these search results for a while for the user to be able to page through them, export them, *etc*. To ensure the system retains a search result indefinitely, the user who triggered the search must create a named “saved search” based on the search result. OpenText Information Archive then persists the search result records (by an internal ID) and makes them available for the user for paged access, export, *etc*., until the saved search is deleted by the user.

After creating a saved search, you can remove individual records from it to fine-tune its final contents.

Using the stored record IDs, OpenText Information Archive provides an on-demand view of the underlying records by looking up the actual records by those IDs, and caching them just like a normal search result. This is called a *reload* of the saved search.

By *only* persisting record IDs instead of copying the actual records, OpenText Information Archive avoids having to manage any extra data compliance, as retaining a persisted copy of data that was deleted (“disposed” in compliance terminology) from its primary location constitutes a compliance issue. Retaining only their IDs, however, does not constitute a compliance issue. Refer to the Compliance chapter for more information.

If the actual records are deleted/disposed from a saved search, the next time a user reloads the saved search, OpenText Information Archive cannot instantiate them from their IDs, remove their IDs from the saved search, and instantiate/cache only the records that are still there.

The entire saved search can be updated by running a search with the original search criteria, typically to include any newly ingested data to it that matches the criteria. This is called a *rerun* of the saved search. After a rerun, any information on previously removed individual records is lost. The user will have to remove them again, if needed.

Reruns can also be triggered on a schedule specified with a time unit, for example, once a day or once every two weeks.

Saved searches can be shared by their creator with one or multiple groups for viewing only or for viewing and editing. Members of a group with which a saved search has been shared for viewing only can view the results and, if needed, reload them.

Members of a group with which a saved search has been shared for editing can also edit it (including removal of individual records), export its results, rerun it, and share it further with more groups.

Note that a user (even the creator) is only allowed to edit a saved search if that user has access to the saved search's application.

Deleting a saved search is only available to the creator of the saved search or any user with the Administrator role. The latter only has the option to delete the saved search without being able to view its actual search results.

4.3.2 Matters

Saved searches can be grouped together in what is called a “matter”. As such, a matter is just a named collection of saved searches. Saved searches can be added to and removed from matters throughout the matter’s life cycle.

One reason to collect saved searches into a matter is the collection of evidence in a legal investigation. Usually, in such cases, holds are applied to the saved searches’ records. This makes the matter a so-called “legal matter”. Refer to Legal Matters for more information.

Matters, like saved searches, can be shared by their creator with one or multiple groups, for viewing only, or for viewing and editing. Members of a group with which a matter has been shared for viewing only can view only its saved searches.

Members of a group with which a matter has been shared for editing can also edit it (including adding/removing saved searches) and share it further with more groups.

Note that a user (even the creator) is only allowed to edit a matter if that user has access to all applications of all saved searches of the matter.

Deleting a matter is only available to the creator of the matter or any user with the Administrator role. The latter only has the option to delete the matter without being able to navigate into its saved searches.

4.4 How OpenText Information Archive is configured for data

As part of an OpenText Information Archive installation, a PostgreSQL data node needs to be configured to hold the system data. For more information, see Section 5.3.1 “PostgreSQL data nodes, databases and setup” in *OpenText Information Archive - Installation Guide (EARCORE-IGD)*.

This section refers to different OpenText Information Archive user roles. Learn more about roles in User roles and actions.

Users with the administrator and developer roles typically perform the configuration as follows:

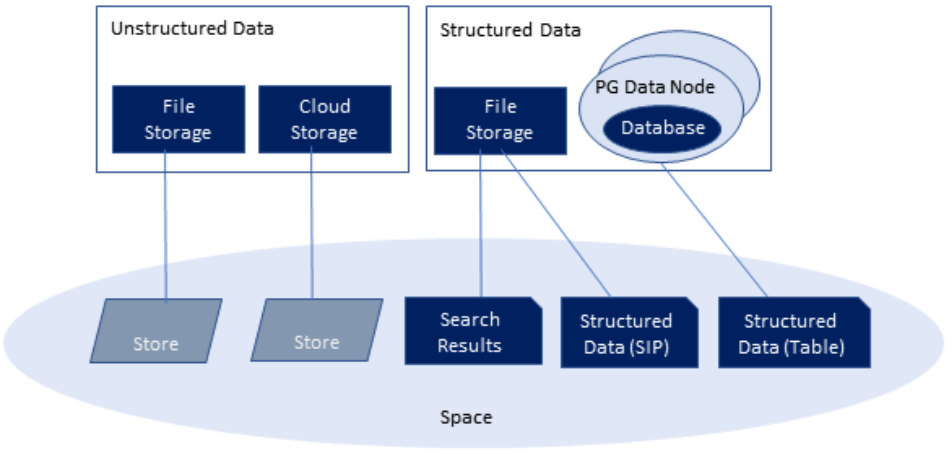
1. The administrator configures one or more PostgreSQL data nodes and databases for structured table data.
2. The administrator configures one or more (shared) file systems (NFS) for structured SIP data.

3. The administrator configures one or more storage systems for unstructured data.
4. The developer extracts data from a source application using an ETL tool or connector.
5. The developer configures an OpenText Information Archive application using a declarative configuration format (`YAML`). For SIP–based applications, the developer can instead use the holding wizard using the IA Web App.
6. For table–based applications, the developer defines the entire application, including the database and schema information using a declarative configuration.
7. The developer starts ingestion.
8. The developer creates searches that end users can use to access ingested data.

4.4.1 Configuration objects

To bind OpenText Information Archive to physical systems, administrators and developers use the following configuration objects in IA Web App:

Configuration object	Description
Data node	Binds OpenText Information Archive with a PostgreSQL data node, which acts as a container for databases.
RDB database	Binds OpenText Information Archive with a PostgreSQL database, which holds the structured content of archived table records.
Storage	Binds OpenText Information Archive with physical storage, such as file systems, PowerScale, ECS, S3, and so on.
Space	Represents the relation between a storage system and an application, which contains records and content for the application. At least one data node, database and storage system must exist to create a space for a table application.
Store	Links a space with a file system directory or bucket (a storage configuration resource used in ECS and S3 storage systems). Stores hold data (including structured SIP data residing in Lucene files) in the context of an application.



4.5 How information is protected

Sensitive information can be found:

- In configuration files,
- In system configuration objects, or
- In a specific application.

In configuration files, it is possible to encrypt information, such as credential information, to connecting to the system database. OpenText Information Archive can be configured to use keystores, Vault integration, or Gemalto to store this information.

In system configuration objects, OpenText Information Archive stores information, such as credential information, for connecting to databases containing table application information. This information can be stored in an external keystore, a keystore stored within the system database, or stored in the system database encrypted using Key Mediator. External keystores are not recommended for production use.

OpenText Information Archive also provides ability for both SIP and table applications to either encrypt sensitive fields of structured data, or unstructured content associated with records. For SIP applications, the holding wizard lets you configure what is encrypted. For table applications, this information is configured when defining the database, schema, and tables.

Chapter 5

Table archiving fundamentals

When you are preparing to archive a table application, consider the following:

- Which databases, schemas, and tables do you need to archive? You need to know the definitions of the table.
- Is there unstructured content associated with the information?
- Do you need to encrypt the structured or unstructured content?
- Do you know in advance how many records are in each table?
- Do you need to retain records across multiple tables as a unit?
- Do you understand what searches you will need so that you can build effective indices?
- What sorts of checks do you want to use to make sure that the information has been ingested into the archive correctly?

If you need to retain records across multiple tables, consider using SIP archiving instead to consolidate the information into a composite record.

Once you are clear on the above questions, you can:

- Configure an ETL tool to extract data from the source application, transform it to the `XML` format required for OpenText Information Archive to be able to ingest it, and store `XML` and any referred blobs in a folder structure suitable for IA Shell to trigger ingest. For example, see `<IA_ROOT>/examples/applications/Tickets/data/blobs`.
- Produce a metadata file describing all the application's schemas, tables, and columns (an ETL tool could also do that).
- Produce a `YAML` file (or more likely multiple related ones) for all other aspects of the application's configuration, such as storage, searches, ways to export data, etc.

5.1 Metadata file

The metadata file describes the data to be ingested. It defines, among others, the following:

- The schema name.
- The table count (the expected number of records in each table).
- The table and column metadata, including foreign key relations and other constraints.
- Whether to use indexing and encryption (the type of encryption to use is defined in the `<IA_ROOT>/examples/applications/default.properties` file).
- When to index data: while ingesting data or after all data has been ingested.
- Whether and how, during ingestion of data, text should be extracted from associated unstructured content. This will typically be done using a combination of table, column, and (custom) index entries. An example of how to configure this is in the OrderManagement application. If your associated unstructured content is in multiple languages, further configuration in the metadata file combined with language hints in the ingested data make the system store/index your content per language. It can also be searched by a language. This multi-language feature is showcased in the OrderManagementMultilanguage example application.

The file name for the metadata file is `database-<APPLICATION_NAME>SqlDb.xml`. There is one metadata file for each OpenText Information Archive application. The following is an example of the metadata file: `<INFOARCHIVE_ROOT>/examples/applications/Patent/config/application-config/data-model-config/database-PatentSqlDb.xml`.

You can see the schema name in the `<name>` element, and the number of tables in the `<tableCount>` element. For every table, the file contains one `<tableMetadata>` element. Each table has a `<recordCount>` element to specify the number of records, which OpenText Information Archive uses for validation. The `<index>` element specifies whether to index the data during ingestion.

The total time spent on ingestion and indexing is typically longer when indexing is configured to be done while ingesting, instead of after all data has been ingested. Therefore, the example applications included with OpenText Information Archive do not do indexing on ingestion. Instead, they use a command after ingestion to build an index at that time. It is vital that with this configuration, the indexing command gets issued and is completed before using the archive. Working with a non-indexed table archive will cause serious performance and locking issues, which may also affect the performance of other applications in the same deployment.

For the table metadata, for each column, the metadata file defines the type and the length of the type. You can specify for each table column, whether the column should be indexed or not.

Commonly supported SQL types are:

SQL type	Type length
INTEGER	10
BIGINT	20
VARCHAR	Depends on the size of the STRING.
CHAR	1
DATE	19

Note: For table applications, LONG is not a valid SQL type and cannot be supported by the table definition. Use BIGINT, which is the SQL type equivalent of Java Long.

For each table in the metadata file, you can also define custom indexes, for example gin indexes (possibly spanning multiple columns). If you configured extraction of text from unstructured content, you would need custom indexes to support querying that text.

5.2 Table data files

For ingestion, an application requires table data from source applications to be transformed to `XML` format in files that do not exceed a few hundred megabytes in size. Larger files can cause indexing to take a long time, or in some cases, stop unexpectedly. The tables must have at least one row. Individual tables can span multiple table data files, and the names for the additional files are appended with a dash and sequence number. The schema and table names in the table data files must match the schema and table names in the metadata file.

You can find examples of table data files in the following directory: `<IA_ROOT>/examples/applications/Tickets/data/TICKETS`.

The files `TICKETS-TICKET_ATTACHMENT.xml` and `TICKETS-TICKET_ATTACHMENT-2.xml` are examples of table data spanning multiple files.

A table row can have attached unstructured content. Each attachment must be referenced using an attribute with name `ref`. For example:

```
<TICKETS>
   <TICKET_ATTACHMENT>
     <ROW>
       <TICKET_NUMBER>10011553</TICKET_NUMBER>
       <ATTACHMENT ref="../blobs/ticket1.jpg" />
     </ROW>
```

The size of a table data file, including attached content, must not exceed 2 GB. If the total size of `XML` and content is bigger, you must split the table data file into smaller pieces. In addition, you should consider that attached content is stored with the table data file. If one content object is linked by multiple table files, it is stored multiple times.

5.3 The configuration YAML file

Each OpenText Information Archive application has its own configuration YAML file that describes the application, including the following:

- Application name, category, and description
- Searches
- Export of data
- Application-specific storage
- Encryption information

Note that `YAML` files can include other `YAML` files. All the example OpenText Information Archive applications use multiple `YAML` files.

5.4 Searches

You can create and add predefined searches using a declarative configuration, including XForm, SQL, and supporting files. Searches require XForm definitions, SQL definitions, and other files.

OpenText Information Archive supports the encryption of data during ingest. With OpenText Information Archive specific extensions to SQL, searching encrypted columns is enabled, as well as decrypting encrypted columns as part of search results. In addition, an extension is included for predicates with external variables, that allows to omit the entire predicate from the query if no value is supplied for the external variable.

OpenText Information Archive provides the ability to export data from search results. You can add exports during ingestion by creating a pipeline, configuration, and transformation.

For searching text extracted from unstructured content, you do not need special configuration other than SQL queries with full-text predicates. The OrderManagement sample application includes sample queries for this. If your unstructured content is in multiple languages, the OrderManagementMultilanguage application includes some sample queries for searching in a specific language.

5.5 Installation and ingestion

After you have set up the configuration files for an OpenText Information Archive application for table archiving, you can install the application. You run an installation script (`install.bat` in Windows and `install` in Linux), which runs the `install.iashell` script with the `iashell` command.

For an example, look at the Baseball application in the `<IA_ROOT>/examples/applications/Baseball` directory. The script for this application does the following:

1. Connects to the system.
2. Imports the configuration of the application, either to create and initialize the application or to update its existing configuration.

   When creating and initializing most applications for table archiving, the following are created:

   - A space, which includes the following:
     - A dedicated PostgreSQL database for the structured data
     - Storage for your unstructured ingested content and other unstructured content, such as export results and backups
   - In the above PostgreSQL database: schemas, tables, columns, relations, indexes etc., as specified in the metadata file.
   - A database, one or more schemas, and one or more table objects representing their physical counterparts mentioned above.
   - Cryptography objects
   - Other objects (for example, searches with associated SQL queries)

   After an application creates all the required configuration objects, it is ready to perform an initial ingestion.

   Ingests the table information. You should ingest all table data before indexing the database.
3. You should ingest all table data before indexing the database.

Caution

- You should not install the same application multiple times if it will result in ingesting the same data more than once. You cannot delete duplicate records.
- If you do not have all your data ready, then you can ingest some tables, but you should not run table indexing commands until all the data is ingested.
- You can split the application installation from the ingestion by removing both the `ingest` and `index-build` commands, and you can run these commands at any time in IA Shell.

4. Performs chain of custody tests to make sure that the data was ingested as expected, to ensure data integrity.

   System performance might be slower until these tests are completed.

5. Starts the background indexing.

   Searches will be slower until table indexing is completed. You should index only once, so that you have one index for the entire database.

Some notes about these steps:

- If you want to apply retention, you must do it after the steps above. The system cannot apply retention during ingestion. It is recommend to finish indexing if you plan to apply granular retention.
- You cannot invalidate table data, so take care not to ingest the same data twice.

  Tip: Use unique constraints to help ensure you do not ingest the same data twice.

- Chain of custody tests are optional but recommended.
- After the script finishes, you must restrict access to the application, as it might contain sensitive data.

Chapter 6

SIP archiving fundamentals

6.1 Glossary

The SIP data model is based on the reference model for an Open Archival Information System (OAIS), ISO 14721. The reference model includes terminology that the SIP data model uses, including the following terms. For more information about OAIS, see the OAIS website.

Acronym	Expansion	Notes
SIP	Submission Information Package	Package containing the structured data, unstructured content and SIP descriptor
PDI	Preservation Description Information	`XML` containing the structured data
AIU	Archival Information Unit	Record
CI	Content Information	Unstructured content (PDF, MP3) associated with the record
AIP	Archival Information Package	Package in the system repository
AIC	Archival Information Collection	View
DIP	Dissemination Information Package	Result

6.2 SIP

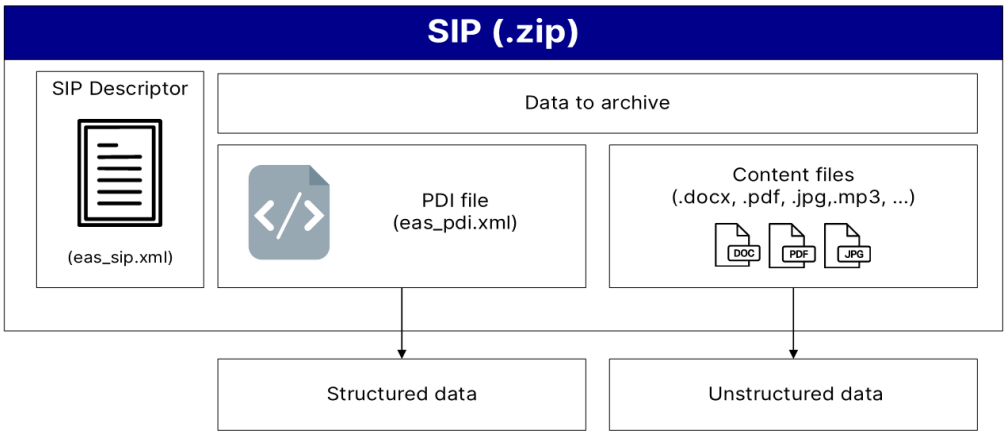
A SIP is a package of structured data, unstructured content, and a manifest file that an OpenText Information Archive application ingests.

Any type of data produced from any source application can be archived if the data is packaged into SIPs that meet the file structure and format requirements. However, OpenText Information Archive is not responsible for generating SIPs from the source application's data.

You can build and ingest a SIP using the Software Development Kit (SDK) for OpenText Information Archive available on the GitHub website. You can also use an ETL tool or connector to extract, package, and ingest the unstructured content and structured data from the producer (the source application).

A `SIP ZIP` is a compressed package (`ZIP` format) with the following files at the root level:

- SIP descriptor
- PDI file
- Content files



6.2.1 Best practices for SIPs

Use the following best practices when creating SIPs:

- Use UTF-8 encoding and version 1.0 for XML files. While XML 1.1 is supported, search performance is decreased when using version 1.1.
- The XML size of a record must be contained and should not exceed 1 MB; otherwise, performance may be impacted.
- Depending on the number of records, you should configure an optimal ingestion mode, indexes, and partitioning keys before you start ingesting data.
- If a SIP contains unstructured data, it must be stored in the root–level of the SIP. Any unstructured content that is stored in a sub-directory or in another path will not be ingested and an error will occur. For an example of a SIP with unstructured data, see <IA_ROOT>/examples/applications/PhoneCalls/data/PhoneCallsSample-2001.zip.
- Date and time values should include the time zone information to avoid time conversion problems for searches. For example, in the following text, the bold text specifies one hour offset ahead of UTC time: 2002-11-18T16:22:04.104+**01:00**.

6.3 SIP descriptor

The SIP XSD file is available in <IA_ROOT>/doc/ia_sip.xsd when you expand the infoarchive-support.zip file.

The following table lists the elements contained in the SIP descriptor. Unless indicated, all parameters are mandatory and must be in this order. If a mandatory parameter is missing or contains syntax errors, ingestion will fail:

<external_id>
Optional custom identifier assigned by the business application producing the SIP to identify the AIP after the ingestion. If the value is not provided, a random UUID is used.

If the <external_id> is not unique or contains a colon character (:), ingestion will fail. If the <external_id> already exists, the AIP has to be removed prior to ingesting the data with the same ID.

<dss>
In the <dss> block of elements, all the SIPs that belong to a particular DSS must have the same values for <holding>, <id>, and <producer>.

<holding>
Name of the holding where the SIP must be archived.

<id>
DSS identifier assigned by the business application producing the SIP.

<pdi_schema>
URN of the schema applied by the PDI file. You must put the version of the schema in the URN, as this is common XML practice.

<production_date>
The <production_date> element that is contained outside of the <dss> element is the production date of the SIP.

There are two production dates to distinguish between the production of the data versus the package. For example, when you package legacy data, the packaging date is more recent that the data. Even though these dates are mandatory, they are for information-purposes only.

<base_retention_date>
Base date to be considered for computing the retention date.

<producer>
Code of the business application producing the SIP.

<entity>
Code of the business entity that owns the data contained in the SIP.

<priority>
Ingestion sub-priority of the SIP.

`<application>`
Code of the business application producing the data contained in the `PDI` file.

`<retention_class>`
An alias that can be used to associate a retention policy that will be applied to the SIP on ingestion. The retention class overrides any default that is specified by the holding. The holding must specify the name of the retention class and map to zero or more retention policies that are to be applied.

`<production_date>`
The `<production_date>` element that is contained outside of the `<dss>` element is the production date of the SIP.

`<seqno>`
Sequence number of the SIP within the DSS that it belongs to. It is common to have only one SIP in a DSS. If a SIP is so large that there are multiple sequence numbers, none of the numbers can overlap; otherwise, ingestion will fail.

`<is_last>`
Boolean indicating whether this SIP is the last SIP of the DSS. If a SIP is so large that there are multiple sequence numbers, the `<is_last>` must be used; otherwise, ingestion will fail.

`<aiu_count>`
Number of AIUs contained in the `PDI` file. The value must match the actual AIU count or ingestion will fail.

`<page_count>`
Reserved for future use. Always set this to 0.

`<pdi_hash>`
Optional element that specifies the encoded hash value of the PDI file. If the value does not match the value in the PDI file, ingestion will fail.

For an example, see the `eas_sip.xml` file in the `<IA_ROOT>/examples/applications/PhoneCalls/data/PhoneCallsSample-2001.zip` file.

6.3.1 About data submission sessions (DSS)

Sometimes information to be archived cannot be packaged in a single SIP because of the following:

- ZIP limitations: a ZIP file has an upward size limit
- File transfer limitations: the FTP that you are using might have a built-in limitation for file size
- Time limitations: a single, large ZIP file might cause a connection to time out

A data submission session (DSS) associates multiple SIPs together. The DSS ID is specified in the SIP descriptor. OpenText Information Archive uses the DSS ID to determine that the SIPs belong to the same DSS.

OpenText Information Archive is insensitive to the order in which it receives SIPs that belong to the same DSS. It can receive and ingest the SIPs in any order and can also do so concurrently.

OpenText Information Archive provides a native commit and rollback at the DSS level. If there is an issue with one of the SIPs in the DSS, then the DSS can be rolled back in a single transaction. If there are no issues with the SIPs in the DSS, and the SIPs have all been ingested, then the DSS can be committed in a single transaction by the Commit job.

For a DSS to be valid, the following must be true:

- There must be no gaps in the `<seqno>` values. For example, if there are SIPs with the `<seqno>` values `1`, `2`, and `4`, then there must also be a SIP with the value `3` for OpenText Information Archive to commit the DSS.
- The last SIP in the DSS must have `<is_last>` set to `true`, and all other SIPs in the DSS must have `<is_last>` set to `false`.
- All of the SIPs that belong to a particular DSS must have the same values in the `<dss>` block of elements (for example, they must have the same value for `<id>`).

6.3.2 About external ID

The external ID is an optional custom identifier that is assigned by the business application producing the SIP. It is used to identify the AIP after ingestion.

The external ID can be used in the following:

- *AIU IDs*: Each record (AIU) of the AIP is assigned a unique identifier during ingestion. This identifier is formed as follows:

  ```
  <external id>:aiu:<sequence number>
  ```

- *CI IDs*: Each instance of unstructured content (CI) in the AIP is assigned a unique identifier during ingestion. This identifier is formed as follows:

  ```
  <external id>:ci:<sequence number>
  ```

- Audits related to this archive.

One typical use case for the external ID is the migration of an archive from another archival system or a version of OpenText Information Archive older than version 4.0, where the business application needs to continue identifying migrated archives with the same IDs.

The external ID must be unique for the whole application and has the following implications:

- Ingesting a SIP with an external ID that is already taken by an AIP in the same application results in a reception error.
- The reject and invalidation process must be run entirely on an AIP to be able to re-ingest a SIP with the same external ID.

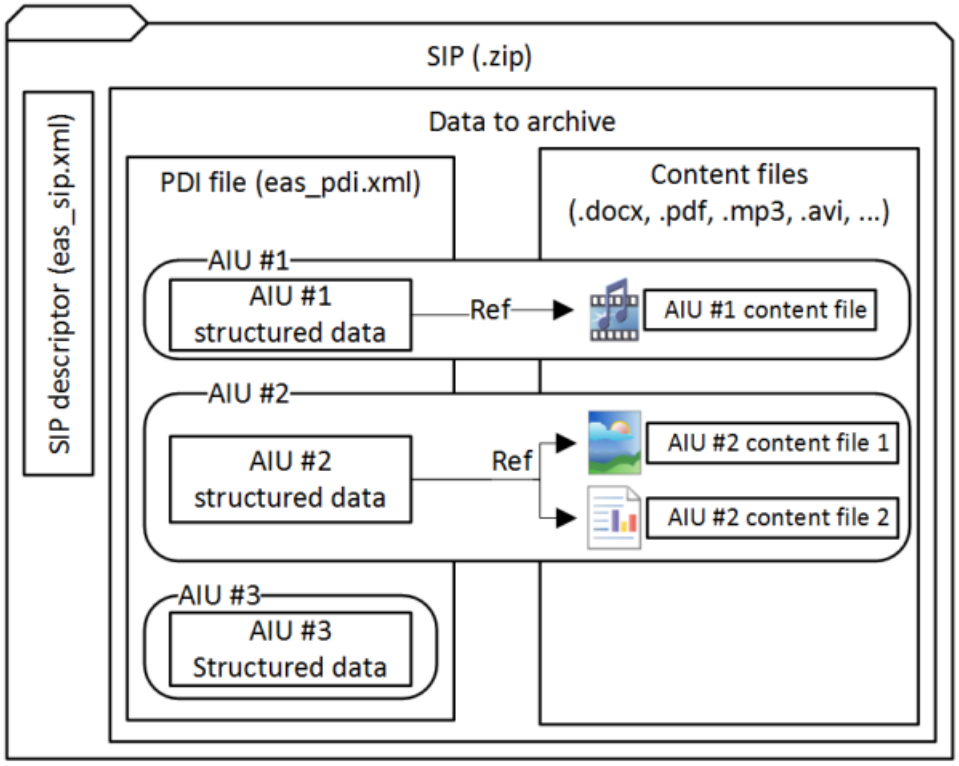
6.4 PDI

Preservation Description Information (PDI) is an XML document containing structured data. The structure of the PDI file is defined by a PDI schema and is composed by 1 to n Archival Information Units (AIU). An AIU is the smallest archival unit of an information package and an XML block containing structured data and, optionally, references to one or more associated unstructured content files.

An AIU in a PDI file consists of an XML block containing structured data and, optionally, references to one or more associated unstructured content files.

For example, in the following diagram:

- AIU #1 is described by structured data in the PDI file, with a reference to one content file.
- AIU #2 is described by structured data in the PDI file, with a reference to two content files.
- AIU #3 only contains structured data stored in the PDI file, with no content files.



Use the following formats for the different data types:

Type	Format
STRING	Printable characters and spaces are permitted.
DATE	2002-11-18
DATE_TIME	2002-11-18T16:22:04.104+01:00
INT	10
LONG	3000000000
DOUBLE	12.123456789
FLOAT	12.12345

6.5 PDI schema

You must specify the structure of the PDI file by creating a schema file.

The schema file should be driven by business requirements. OpenText Information Archive uses the schema file to validate the PDI XML file during ingestion.

You must use standard XSD features to control XML content, including the following:

- Where possible, use standard XML data types, especially for date and time information.
- Configure the minimum and maximum length of attribute and element values.
- In the PDI file, include date and time values that explicitly include the time zone information. For example, in the following text, the bold text specifies one hour offset ahead of UTC time:

  ```
  <CallStartDate>2002-11-18T16:22:04.104+01:00</CallStartDate>
  ```

- Include the version number in the schema URN, which is defined as a value of the targetNamespace:

  ```
  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="urn:eas-samples:en:xsd:office.1.0"
  xmlns:ns1="urn:eas-samples:en:xsd:office.1.0">
  ...
  </xs:schema>
  ```

  In the example above:

  - urn is the prefix
  - eas-samples can be the name of your company
  - en is the language (in this case, English)
  - xsd specifies the category or application as well as the version
- Adopt a consistent naming rule for the schema URNs. This will make it easier to remember the URNs, which are referenced in multiple places during the configuration.

- If you are defining your own custom elements, do not use reserved keywords for XSD (for example, `language`).

For examples of the schema file and a corresponding PDI file, see the `<INFOARCHIVE_ROOT>/examples/applications/PhoneCalls/config/data-model-config/pdiSchema-urnEasSamplesEnXsdPhonecalls.1.0.xsd` file and the `eas_pdi.xml` file in the `<INFOARCHIVE_ROOT>/examples/applications/PhoneCalls/data/PhoneCallsSample-2001.zip` file.

Each `<Call>` element represents an AIU.

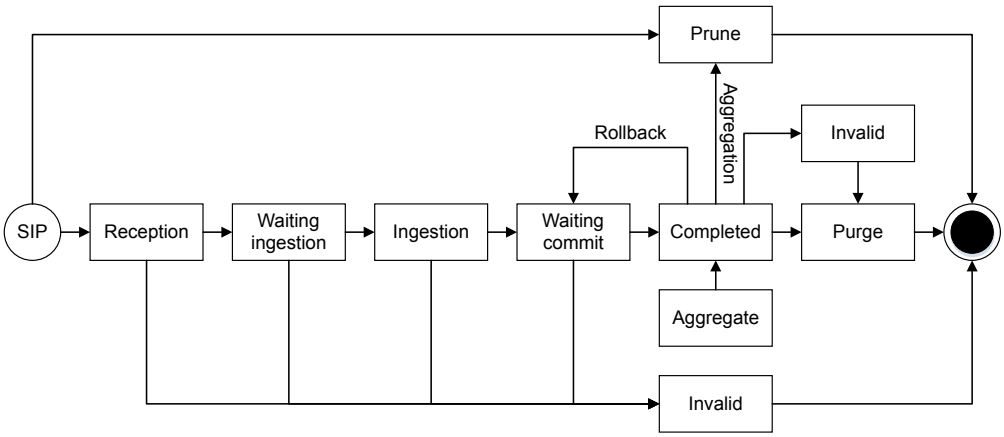
6.6 AIP

When OpenText Information Archive ingests and verifies a SIP, it creates an AIP from the SIP. AIPs are packages of information that can contain structured data, unstructured content, logs, and backup.

The structured data and unstructured content can be extracted directly from the source application, derived from other systems, or programmatically constructed. The ability to maintain separate data elements in an AIP allows OpenText Information Archive to balance the requirements to maintain exacting standards around chain-of-custody with the desire to build richer data sets than those that existed in the original applications. For example, raw transaction history data can be extracted, modeled as `XML`, and verified as 100% accurate and complete for chain-of-custody purposes. At the same time, additional information from extended systems can be made part of the AIU in another data element, making the AIU more usable without compromise.

The following diagram illustrates the process whereby a SIP is received, ingested, and transformed into an AIP. The following notes provide further clarification about the diagram:

- An AIP reaches the COMPLETED stage when all AIPs that are part of the same data submission session (DSS) have been ingested. An AIP is only searchable once it has reached this stage. For more information about DSS, see Data submission sessions.
- The PRUNE phase is only used when you have configured the ingestion mode to use AGGREGATION mode. When the aggregate is closed, all AIP children are moved to the PRUNE phase and the aggregate moves to the phase COMPLETED.
- Once the retention needs of the data have been met, the AIP can be disposed.



6.7 Holding

A holding is essentially a basic archive application, a logical destination where data that shares common characteristics is archived. Some examples of common characteristics include:

- Data from the same source application
- Data in the same format (for example, audio recordings)
- The same type of data (for example, communication through email, chat, and faxes)
- Data that belongs to the same business entity

Holdings are used exclusively with SIP–based archives. Table–based archives do not use holdings.

The SIP descriptor includes the name of the holding to be used. Typically, for the example OpenText Information Archive applications, the holding name is the same as the application name. However, an application can use multiple holdings. Multiple holdings can also exist for a single data type, which means that you are able to apply different access rights or use a different storage area.

A holding is the central configuration object in SIP archiving. The following is defined in a holding:

- Storage area
- Retention
- Ingestion mode
- Library policy

When you create a holding, you should consider the types of data that will be archived as well as the data segregation and isolation restrictions.

If a holding is properly designed, it should be able to handle a lot of data. The ingestion speed is very sensitive to a holding configuration. If performance is poor, it is most likely because of an improperly designed holding configuration. With the OpenText Information Archive Holding Configuration Wizard, you can configure a holding that is optimized by default and uses indexes efficiently during ingestion and searches.

6.7.1 Ingestion mode

OpenText Information Archive supports three ingestion modes: private, pooled, and aggregated. You can use any of these modes for small amounts of data but using the optimal ingestion mode is key for scalability and search performance. You choose an ingestion mode when you create a holding.

In general, your choice of ingestion mode depends on the SIP package characteristics and your ingestion use case, particularly the number of AIUs per AIP.

You can also dynamically select the mode during reception and ingestion. Dynamic mode allows you to optimize the aggregated or pooled mode by switching to private mode when there are too many AIUs as defined by the `libraryPolicy.aiuThreshold` property.

	Private	Pooled	Aggregated
Use Case	• Standard archiving • Close to 50 packages a day • Very large packages • Higher than 50,000 AIUs per package	• Immediate ingestion (for example, medical prescriptions for multiple patients, sent frequently). • < 1000 packages a day. • Medium-sized packages.	• Immediate ingestion (for example, signed contracts). • 1000+ small packages a day. • Less than 100 AIUs per package.
Performance	• Optimum scalability and search performance. • Source/calling application can wait until a large package is ready.	• Balances archive speed with scalability and search performance. • Source/calling application can wait until a medium-sized package is ready. • Better if you cannot easily know your partition keys.	• Archives data as fast as possible, as soon as the data is ready to archive.

	Private	Pooled	Aggregated
Library	• Each package has its own dedicated library.	• Each package has its own dedicated library before being associated with one shared combined library at the library's closing, after which all packages share the same library. • Specify the maximum number of packages and records in a library. • Specify when to close the library.	• All packages have their own dedicated library before being combined into one package and one library at the library's closing. • Specify the maximum number of packages and records in a library. • Specify when to close the library.
Search	• AIPs are searchable immediately after ingestion. • Indexes are created during ingestion. • Ideal when partition keys are used to reduce the search scope.	• AIPs are searchable immediately after ingestion. • Search might be slower than normal until the AIP is closed because many smaller libraries are queried.	• AIPs are searchable immediately after ingestion. • Search might be slower than normal until the AIP is closed because many smaller libraries are queried.
Package Retention	• Applied to each package on ingestion.	• Applied to the pooled package when it is closed.	• Applied to the aggregated package when it is closed.
Backup	• Immediate	• Immediate for individual libraries. • At library close for the merged library.	• Immediate for individual libraries. • At library close for the merged library.

6.7.2 Close

Close is the process that makes the transition between a library that is open to receive more packages in pooled and aggregated ingestion modes, and the library in its archival state. (The state that the library remains in until it is being disposed or invalidated).

For both aggregated and pooled mode, the close will do the following:

- Make the library unavailable for further ingestion.
- Merge the individual libraries into the final combined library.
- Back up the library and make it eligible to cache in and cache out.

For aggregated mode, the close will do the following:

- Combine all renditions of the aggregate's children into renditions of the aggregated AIP.
- Combine all CI containers of the aggregate's children into the CI container of the aggregated AIP.
- Calculate the partition keys of the aggregated AIP.
- Prune the aggregate's children.

The close process is handled by the Close job and should be scheduled to run on a regular basis, depending on the business use case. Jobs are discussed more in the Jobs section.

The settings to decide when a library is eligible to be closed are defined by the libraryPolicy object attached to the holding. A library can be eligible to be closed based on the following:

- A closing date calculated during the ingestion.
- Reaching several AIUs or AIPs quota.
- A manual request.

When using pooled and aggregate mode, it is better to close open libraries on a regular basis because keeping too many libraries open in an application can have a major impact on performance.

Running the Close job requires disk space. Before running this job, ensure that you have enough disk space to properly store the individual libraries being closed. For more information, see Section 3.7.10 "Close job" in *OpenText Information Archive - Administration Guide (EARCORE-AGD)*.

6.8 Cache-in and cache-out

Cache-in and cache-out provides an ability for SIP archiving applications to improve performance by reducing the number of libraries directly available in File Storage.

OpenText Information Archive stores data from SIP-based applications in several PostgreSQL databases and other storage:

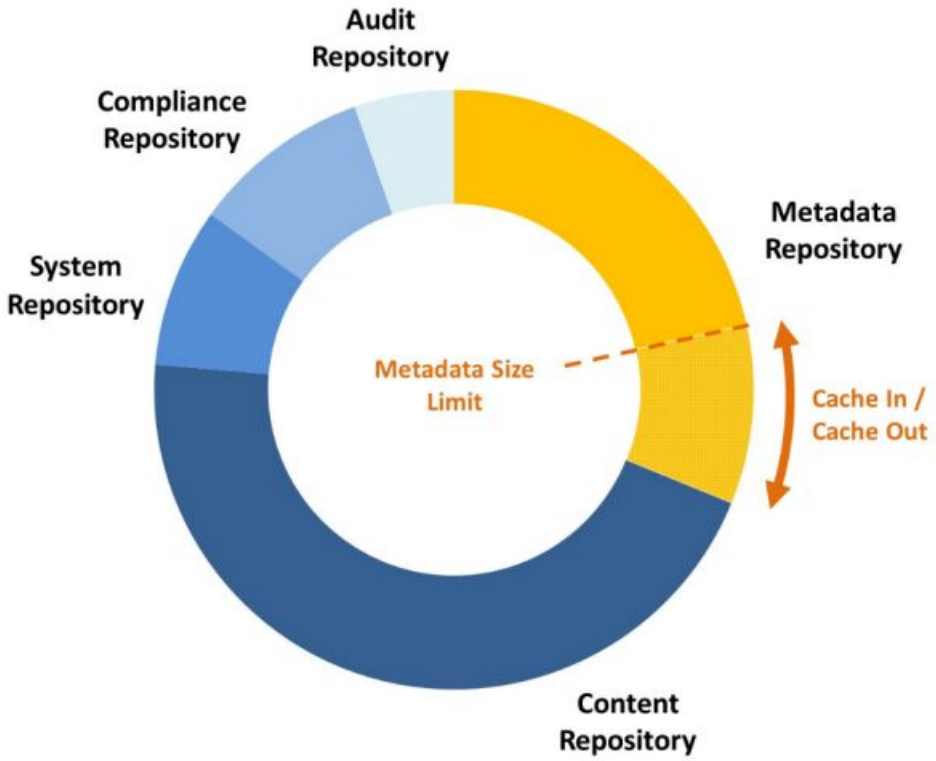
Audit repository
Contains audits generated by the system until they are archived.

Compliance repository
Contains retention and hold information.

System repository
Contains metadata for system objects (for example, configuration objects and AIPs).

Content repository
Contains the unstructured contents of archived AIPs.

Metadata repository
Contains all the metadata (in particular, PDI files) of archived AIPs.



Cache-in and cache-out functionality allows you to manage disk space for the metadata repository. You can choose to limit the data in the cache (the metadata repository) for a particular application. When this limit is exceeded, the Cache Out job removes the least frequently accessed AIPs from the metadata repository until the size of the cache satisfies the limit that you set.

For a user to perform a synchronous search for an application, the AIPs targeted by the search must be present in the metadata repository. If an AIP targeted by a synchronous search is not present in the metadata repository, OpenText Information Archive asks the user to do a background search instead. During the background search, the system finds the targeted AIP and caches it back in.

From a technical perspective, the cache-in and cache-out functionality automatically reduces the number of Lucene indexes. To perform a synchronous search, all corresponding indexes must be in the metadata repository, stored in a file hierarchy on one of the following:

- Storage Area Network (SAN), which is fast but expensive
- Network Attached Storage (NAS), which is slower but less expensive

The cache-in and cache-out mechanisms are based on the library's Lucene index individual backup. The system performs this backup during the ingestion process and saves the backup file in the content repository (for example, Dell EMC CAS (Content Addressed Storage) Elastic Cloud Storage or Amazon S3).

When the system needs to cache-in and restore the segment, it retrieves the backup file from the content store and restores it in the metadata repository (SAN/NAS) to be accessible as a Lucene index.

The CacheOut job determines which libraries should be removed from the metadata repository, based on usage statistics as follows:

- The usage statistics are stored in the system repository.
- For each library in the metadata repository, the last access date is stored. The access date is updated when the library is created or restored, as well as each time a corresponding AIP is the target of a search.
- When a library is removed from the cache, it is also removed from the usage statistics. For the compliance metrics, the records are still part of the calculations even if the library was cached-out.
- When the CacheOut job needs a candidate for removal, it takes the one with the oldest access date.

6.9 Adding a SIP to an archive

There are two different ways to add a SIP to an archive:

- An IA application can *ingest* the SIP, which means that, if there are no issues, the package is automatically committed, and its records are searchable. If the holding was configured to apply retention on ingestion, using private ingestion mode, then retention may be applied to the individual records or to the package.
- An IA application can *receive* the SIP, which means that the package has been uploaded to the server. The package is not committed, and its records are not searchable until it is ingested.

You most likely will want to ingest the SIP and use the private ingestion mode if you are not pushing retention to the hardware as well.

If you are doing a lot of ingestion and you do not want your users getting partial results, you might want to receive the SIPs so that you can make all the records available at roughly the same time. Another reason to receive the SIP is that you are using package-based retention and your store is configured to push retention to the hardware. In this case, after you apply retention, it is not possible to remove the retention until the retention date has expired.

If you receive the SIP, you will need to finish the ingestion by doing one of the following:

- Calling the `ingest` command on the AIP created during the reception. This will make the records searchable.
- Calling the `commit` command (if auto-commit is not enabled at the holding level). This will apply the retention to the package if running in private mode, but not aggregate mode.

To select the received AIP to ingest, you can use the `enumerate` command. This command lists the AIPs that are waiting for ingestion for the current application, ordered by priority.

This three-step ingestion process is for customers who used OpenText Information Archive prior to version 4.0 and want to keep the same level of control available to them.

After receiving the data, it is possible to mark the content as Error, using IA Web App. After the data has been committed, it is still possible to invalidate the package.

In most cases, newer OpenText Information Archive customers should use the `ingest` command.

6.9.1 Invalidation and rejection

You might need to invalidate or reject data that was added to a SIP. A common reason is that the data was already ingested.

When you invalidate or reject an AIP, the records that are contained in the AIP will not be returned in search results to users, but the data is still in OpenText Information Archive. To complete the invalidation or rejection process and remove the data, you must run the Invalidation job. However, the data remains on the storage system until you run the Clean job, which permanently deletes the data from the storage.

You can undo an invalidation or rejection operation for a package, before running the Invalidation job, only if the package was committed before you did the invalidation or rejection. This is because if a package is committed, OpenText Information Archive can safely put the package back to its previous state. After the rollback, the AIP is back to its completed state and available for search again.

An invalidation or rejection operation can no longer be rolled back after the Invalidation job runs. In most cases, apply the Invalidation job to a single application at a time.

A package can be invalidated from any stable state:

- Waiting for ingestion (just after reception)
- Waiting for commit
- In error
- Completed

If you received an AIP but have not yet ingested it, you can reject the AIP rather than invalidating it. Once you have ingested an AIP, you can no longer reject it, but you can invalidate it.

For more information about rejecting or invalidating an AIP and running jobs, see Section 3.3.8 “Rejecting or invalidating an AIP” in *OpenText Information Archive - Administration Guide (EARCORE-AGD)*.

Chapter 7

Operational concepts

7.1 Authentication mechanisms

OpenText Information Archive supports the following authentication mechanisms:

- Active Directory (AD)
- Lightweight Directory Access Protocol (LDAP)
- OpenText Directory Services (OTDS), which manages user and group identity information for OpenText components. OTDS contains services for identity synchronization and provides single sign on for other OpenText components. With it, you can configure multiple identity providers (IDPs), including integrating with AD and LDAP. OTDS is available free of charge.
- Example user accounts (also known as in-memory user accounts)

Authentication takes place when a user logs into IA Web App. Authentication is based on the OAuth 2.0 framework and uses JSON web tokens.

For LDAPAD–based authentication, OpenText Information Archive basically configures the Spring Security LDAP implementation. The actual authentication is implemented by the Spring Security LDAP library.

The person configuring LDAP/AD must be familiar with the specific structure of the directory in LDAP/AD server and understand how the groups and users are located and queried.

In a production configuration, you should use OTDS, AD, or LDAP to authenticate user accounts. The example user accounts are intended for use with a demo configuration, when you want to quickly set up OpenText Information Archive so that you can test its features, set up a proof of concept, give a short demonstration, or set up a basic development environment. For more information about:

- Example user accounts: Refer to Section 13.2.1 “Working with example user accounts” in *OpenText Information Archive - Installation Guide (EARCORE-IGD)*.
- Demo configurations: Refer to Section 2.2 “Demo configuration” in *OpenText Information Archive - Installation Guide (EARCORE-IGD)*.

Caution

Do not use the example user accounts in a production environment. Attackers can use one of the example user accounts to gain unauthorized access to your OpenText Information Archive system and the assets that it contains. These out-of-the-box accounts are meant for demo purposes, and the default password for each account is `password`.

There are settings for validating password strength for SQL database objects (refer to Section 4 “Security” in *OpenText Information Archive - Installation Guide (EARCORE-IGD)*).

7.1.1 Authenticating users – configuring OpenText Directory Services

OpenText Information Archive uses the OAuth2 protocol for authenticating users. It delegates access to the Lightweight Directory Access Protocol (LDAP) or Active Directory (AD) external authentication mechanisms, using JSON Web Tokens (JWT) for stateless authentication. This is implemented in the Gateway component of IA Web App.

OpenText Directory Services (OTDS) is a repository of user and group identity information and a collection of services to manage this information for OpenText components. OTDS manages the integration of many authentication systems, such as single sign-on (SSO).

You configure the following OTDS elements for use with OpenText Information Archive:

Resource
Resources represent multi-user systems, or components, that users can access. Essentially, a resource is a representation of such a component in OTDS. For example, throughout this section, `infoarchive` is used as the resource name to represent the OpenText Information Archive instance.

Access Role
Access roles are used to control which resources users can access. The default access role for a resource is automatically created. For example, the `infoarchive` resource has the `infoarchive` default access role. Any partitions that are members of the `infoarchive` access role are able to access the `infoarchive` resource. This enables all of the groups defined in those partitions to be mapped to OpenText Information Archive roles.

Partition
Partitions are self-contained copies of user information that allow you to organize users into a structured hierarchy of users, groups and organizational units. A user partition in OTDS is represented by a unique name.

User information can be imported and synchronized with AD and/or LDAP, and can be managed fully within OTDS. OTDS supports multiple, concurrent user partitions.

- *Synchronized User Partition*: Partitions are synchronized with an identity provider, such as AD or LDAP. A synchronized user partition contains users, groups and organizational units that are imported from the identity provider when the user partition is created. A synchronized user partition can be automatically kept up to date with its source directory. Users who are imported from an identity provider into a synchronized user partition are authenticated by the identity provider.

- *Non-synchronized User Partition*: These are created and maintained manually. Unlike a synchronized user partition, a non-synchronized user partition does not have an identity provider that its users and groups are imported from. Users and groups in a non-synchronized user partition are maintained entirely through the OTDS Web Client. Users who are created and maintained manually in a non-synchronized user partition are authenticated by OTDS. Configurable password policies are available for non-synchronized user partitions.

You use two key tools when configuring OTDS for use with OpenText Information Archive:

Setup process
To learn about how the setup process helps configure OTDS, see Section 7.2.4 "Using OTDS bootstrapping to create the required configuration" in *OpenText Information Archive - Installation Guide (EARCORE-IGD)*.

OTDS Web Administration Client
You use this tool to manage resources, access roles, and partitions.

7.1.1.1 Types of OTDS integrations

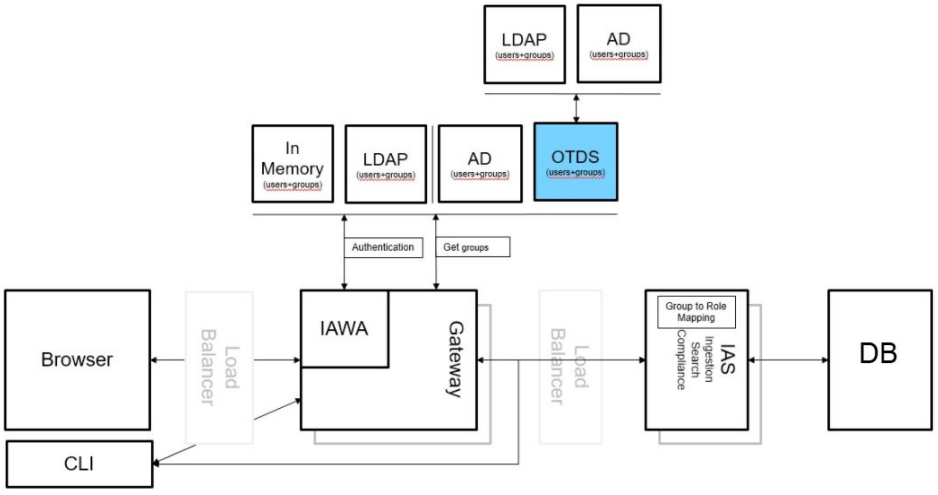
When configuring OTDS for use with OpenText Information Archive, you have the option of using two different types of integrations:

- *Authentication provider mode*: Also known a simple integration.
- *Single sign-on (SSO) mode*: Also known as a full integration.

The primary difference between the two types of integration is that the full integration allows the system to include SSO functionality.

Authentication provider mode
This mode allows for authentication using usernames and passwords. OTDS acts the same as other authentication providers, such as LDAP, AD, and example user accounts (also known as in-memory user accounts). However, because OTDS allows the configuration of many Identity Providers (IDPs), it is possible to use multiple IDPs with OTDS. In authentication provider mode, Gateway is still in charge of issuing the JWT tokens that are used for stateless access.

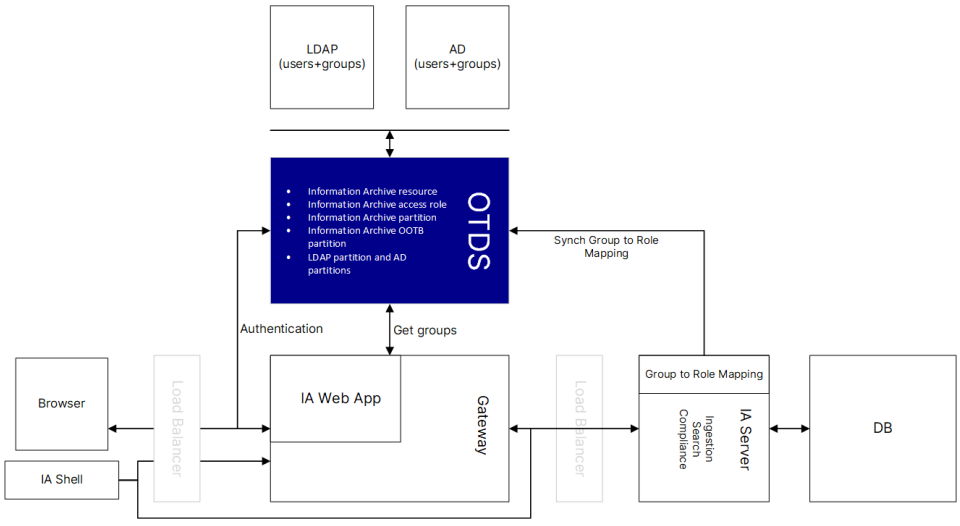
The following diagram illustrates the architecture of OTDS integration in authentication provider mode. You can use example (in-memory) user accounts for bootstrapping only, LDAP, and AD, and OTDS as authentication providers, and OTDS can also connect to LDAP and AD.



OpenText Information Archive uses OTDS as an external authentication system in the same manner as it uses LDAP and AD. You activate the use of OTDS using a Spring profile, and you use IA Web App to map the user groups to their respective roles. When this is done, the users in OTDS who can access the resource associated with the OpenText Information Archive instance can log in. For more information about managing groups, see Section 3.10.2.3 “Managing application permissions” in *OpenText Information Archive - Administration Guide (EARCORE-AGD)*.

SSO mode

In this mode, OTDS takes over the authentication and issuing of the JWT tokens completely. This enables OTDS to also provide SSO functionality. LDAP and AD connect to OTDS, which connects to OpenText Information Archive.



Therefore, it is necessary to store the group-role mapping in OTDS. The OpenText Information Archive roles are reflected as OTDS groups in a specially named, non-synchronized partition inside OTDS.

While it is possible to complete the mapping process using the OTDS Web Administration Client, it is much simpler to use IA Web App, where you map specific groups to the applicable user roles. This is because IA Server synchronizes with OTDS to record any changes in the group-role mapping. For more information, see Section 3.10.2.3 “Managing application permissions” in *OpenText Information Archive - Administration Guide (EARCORE-AGD)*.

7.2 Authorization

After user accounts are authenticated, OpenText Information Archive authorizes the user accounts to use one or more OpenText Information Archive resources.

Users are mapped to groups in Active Directory, LDAP, or in the file that manages example user accounts (`<IA_ROOT>/config/iawebapp/application-infoarchive.gateway.profile.AUTHENTICATION_IN_MEMORY.properties`). It is strongly recommended that you disable the in-memory profile once bootstrapping has been completed.

You should map your existing groups in Active Directory or LDAP to roles in OpenText Information Archive. When you map a group to a role, the role is assigned across all OpenText Information Archive applications. However, you can control which applications are visible to a group. By default, all applications are visible.

7.3 User roles and actions

OpenText Information Archive defines specific user roles that are usually involved in the application archiving process. The IA Web App user interface is built and pre-configured for these fixed roles:

- Administrator
- End User
- Developer
- Retention Manager
- Business Owner
- Auditor
- IT Owner

A user's role, which is inherited by membership in a group, determines the actions that the user can perform. Users can access the functionality on various tabs depending on the roles they are a member of, via the group membership. For example, an Administrator can see an **Administration** tab, and a Retention Manager can see a **Compliance** tab.

The following lists identify the actions that the seven user types can perform:

Administrator

Manage administration objects, such as storage systems or databases, that are required by the Developer to configure applications.

Manage export configurations not associated with an application (shared).

Set permissions for any application or retention policy.

Change the group to role to mappings.

Manage, schedule, and run jobs.

Manage high-level application configuration elements (such as spaces or stores) and manage cryptography objects.

Access to the License and Storage dashboards for all applications.

Manage which audit events are enabled.

Delete background requests, saved searches, and legal matters for all users.

End User

Run synchronous and asynchronous searches.

View search results, create saved searches, and manage own saved searches, including sharing and rerunning.

Export search results (only available if configured during search composition).

Manage own background requests.

Developer

Manage applications and their components, such as spaces, stores, holdings, tables, searches, and export configurations.

Ingest data.

Manage, schedule, and run data-related jobs.

View administration objects, such as storage systems or databases.

Manage cryptography objects.

Delete application data and delete an application (only for applications that are In Test).

Can perform all end user actions.

Retention Manager

Manage retention policies and holds.

Manage, schedule, and run retention-related jobs.

Apply retention to an application, table, or package.

Apply hold to an application, table, package, search results, saved search, or matter.

Manage disposition, such as viewing and approving purge lists.

View the Retention Dashboard for accessible applications.

View applied retention, which includes viewing retained sets and hold sets, and enriching saved searches and search results for compliance.

Can perform all end user actions.

Business Owner

View application information.

View applied retention, which includes viewing retained sets and hold sets.

Manage packages and tables (for example, see audits, invalidate a package).

View License and Storage dashboards for accessible applications.

Delete application data and delete an application (only for applications that are In Test).

Can perform all end user actions.

Auditor

View which audits are enabled for the system and all applications.

View License and Storage dashboards for all applications.

An Auditor cannot run searches or view background requests. An Auditor can see all applications but cannot navigate inside them, as they do not have access to the Application Info screen.

Note: To be able to search the audits, the Auditor needs to be a member of group that is mapped to the End User role. If there are group permissions on the Audit application, the user must be a member of at least one of the groups (which is mapped to a role that can do searches).

IT Owner

View the stores for all applications, independent of group permissions.

Use the Administration tab to view storage, database, data node information.

View the Storage Dashboard for all applications.

7.4 Configuring groups to access OpenText Information Archive

The previous section described roles. To configure access for OpenText Information Archive, Administrators must:

- Map groups to roles.
- Optionally, configure which groups should have access to which applications.

Note: In previous OpenText Information Archive releases, the Developer could configure the group to role mapping. While the Developer can still access the **mapping**, they are not able to update it.

If an application has no groups configured, every user can see the application. If one or more groups are configured, only users belonging to at least one of those groups can see the application.

OpenText Information Archive provides two modes for determining which roles a user has for an application they can see: with “restrict group permissions” enabled or disabled.

Independent of the mode, if a user is not in any of the application’s groups, they will not be able to access the application.

- If the restrict group permissions feature is disabled, the user’s permissions for an application are a union of all roles of all the groups they are a member of. For example, if a user is in two groups (GROUP_A and GROUP_B):

  – GROUP_A is mapped to the `END_USER` role, and

  – GROUP_B is mapped to the `RETENTION_MANAGER` role.

  The user has `END_USER` and `RETENTION_MANAGER` access to all applications, assuming that they can see the application at all. Even if the application has only GROUP_A configured, our user will still have the RETENTION_MANAGER role in the application, with all the associated permissions.

- If the restrict group permissions feature is enabled, the user only gets roles for that application based on which groups were set on the application. For example, if someone sets GROUP_A on an application, that user would only get the END_USER access for the application, even though they are in GROUP_B, too. If an Administrator maps GROUP_B to the application as well, the user regains their `RETENTION_MANAGER` access for the application. This example illustrates a typical use case for this feature: reducing the permissions for a powerful user in selected applications.

If there are no groups associated with the application, all users can perform actions allowed by their role permissions.

7.5 Auditing

OpenText Information Archive allows the auditing of many events. There are events related to the following:

- System
- Tenant
- Individual applications

OpenText Information Archive stores audited events in a PostgreSQL database, from where they can be archived in a SIP archive (see below). In the IA Web App, administrators can access the **Administration > Audit** tab to select which events to audit.

For more information about installing applications, see Section 3.3 “Installing applications for a demo configuration” in *OpenText Information Archive - Installation Guide (EARCORE-IGD)*.

7.5.1 Default retention on audits

Audits can be searched after you install the Audit application and run the Archive Audits job. The job ingests the unarchived audits from the PostgreSQL database into the Audit application as Lucene indexes, since the Audit application is a SIP archive.

Note: For new installations only, the default Audit-policy retention policy is disabled after installing the first time applications. Disabled retention policies allow the retention policy to be applied on ingest; however, the packages will not be eligible for purge lists until the retention policy is enabled.

7.6 Dashboards

OpenText Information Archive has an administration dashboard and a compliance dashboard which allow administrators and retention managers to view information related to applications, such as the number of retention policies, holds and purges.

The administration dashboard shows how much storage is in use and how much is remaining, as well as pricing information. The compliance dashboard gives information about retention coverage, as well as forecasts for disposition.

The information on the dashboards is updated by the Refresh Metrics job.

Chapter 8

Background processing – General concepts

OpenText Information Archive allows three types of background processing:

1. Jobs
2. Background requests
3. Scheduled Tasks

All background processing supports logging. Both jobs and background processes provide access to the logs through the IA Web App.

8.1 Jobs

Jobs allow you to process either ad hoc or on a schedule. OpenText Information Archive provides various jobs for performing various tasks for compliance, cleanup, and upgrade.

OpenText Information Archive has the concept of job definition, which defines what is the function of the job and the metadata that controls what exactly the job does and acts on. When a job runs or is scheduled, a job instance is created, which tracks what that one run of the job did.

Most jobs can be run ad hoc. Jobs that cannot be run ad hoc are spawned by other OpenText Information Archive tasks.

Jobs can be scheduled using either an interval or a cron expression. The schedule can be started by the `start schedule` CLI command.

There are different ways to stop the job from running:

- Inactivate the job
- Suspend the schedule
- Stop the schedule
- Edit the job schedule settings

Inactivating the job prevents the job running, which includes ad hoc runs. Scheduled instances are skipped, and the schedule continues. Activating the job is the only way to allow the job to run again.

Suspending the schedule prevents the job scheduled job instances from running, but will not prevent an ad hoc job instance from running. Resume the schedule when you no longer need it suspended. Consider suspending the schedule for jobs running on an interval before upgrading to a new version and then resume the schedule after the upgrade.

Stopping the schedule deletes any scheduled job instances. To resume a suspended schedule, first avoid having the started schedule from skipping the job instances.

Editing a job schedule setting to manual will also stop the schedule. If the job's schedule was changed, the job will use the new settings. In previous versions of OpenText Information Archive, changing the job schedule settings automatically stopped the schedule, so it had to be manually restarted.

Jobs can be cloned, and you can run them ad hoc. Cloning jobs makes sense for tasks that need to be scheduled, such as the Clean job, if you want to have a different schedule for different applications.

Some jobs may need to break the work down. These jobs may spawn order items that can also further be broken down. Order items that further break down work are called batched order items, and the logs for batches can also be viewed from the IA Web App.

8.1.1 Anatomy of a job definition

A job definition has the following:

- A concept of scope, which is either System scoped or for one or more applications. When a job is scheduled for more than one application, a separate job instance is scheduled for each application.
- A scheduling mode, which is Manual, Interval, or Expression.
- An optional list of parameters. Each job parameter has a type, a potential default value, and a name. Some job parameters may only accept a fixed list of values.

Job definitions also track who last modified the job and the state of the job. For example, it is possible to see if the job has been inactivated or suspended.

8.2 Background requests

Some operations can take a long time, such as a search. Users can allow these operations to run while they perform other tasks. Some background requests support cancel, which can be useful if you realized for example that your search criteria was returning too many records. Background requests that were canceled can often be retried.

Users can view their own background requests, and administrators can view all background requests. Normally, background requests are removed after a week. This setting is configurable via the Global Settings tab.

8.3 Scheduled tasks

The IA Server has scheduled tasks that typically run in the background, and the user cannot interact with these directly. These tasks typically perform maintenance operations, such as backups. It is possible to change the interval of these tasks using the IA Server's `application.yml` file.

Chapter 9

Compliance – General concepts

Compliance is an important feature because it allows you to retain data for a designated amount of time and specify when data can be disposed or purged. Disposition is vital because data stored in a repository increases cost. It is best to retain sensitive data only as long as the data is required.

Once an application is active (for example, no longer has the in-test status), the only way to purge stored data is through the disposition process.

Compliance in OpenText Information Archive consists of:

Retention policies

Retention policies define when data is eligible for disposition and can also be used for your applications.

OpenText Information Archive provides the ability to control how long data can be kept in the archive. You can choose how to apply retention to items in the archive. For instance, opting to use granular disposition to apply retention to individual records.

Data can be retained using the following methods:

- Retention Managers can use the IA Web App to apply retention and holds directly to applications, AIPs, and tables.
- Retention Managers can use the OpenText Information Archive search interface to apply holds directly to records (AIUs and table rows).
- Developers can configure the holding to apply retention automatically to content during ingestion (AIP and AIUs).
- Administrators can configure jobs to apply retention and holds to records (AIUs and table rows) via the use of rules.

Holds

Apply holds to prevent disposition. Data is never destroyed (disposed) until approval is given.

Disposition

Disposition, also referred to as purges, refers to the controlled process of what to do when the required aging has been met by the retention policies.

For applications that are in test, if no retention is applied, the data can be purged to clean up test systems. All retention and holds must be removed before deleting the application data. The retention manager can remove retention from all the packages in an application to facilitate this. This operation does not affect individual records that may have holds or retention applied. If retention was applied to records, you

have the option of using the Remove Policy job. To remove all holds, the retention manager must delete all the hold sets.

9.1 Compliance-related roles

Retention Managers perform the following compliance-related duties:

- Manage retention policies and holds.
- Apply and remove retention policies.
- Apply and remove holds.
- Approve or reject a purge candidate list (used during disposition).

9.2 The retention lifecycle

The following outlines the retention lifecycle:

1. A retention policy is applied to data.
2. Wait for time to elapse or an event to occur.
3. Items are bundled for disposition approval.
4. The Retention Manager approves the disposition of the purge candidate list.
5. Data is disposed.

9.3 Architecture

To manage items within the system, this is the data model:

Managed item
This references the item that is managed. It could be an application, package/table, or individual record (AIU or table row).

Policy Application (Retention Application
This is the instance of an application of a retention policy to a record. It contains metadata that shows when the managed item qualifies for disposition when using individual retention. If two retention polices were applied to the same record, there would be two policy applications.

Retained Set
Policy applications refer to the retained set. The set contains information about the policy that was applied and the qualification date for the records (if not using individual mode).

Retention Policy
Defines how long to keep the data. There are four types of policies: Fixed, Duration, Event and Mixed.

Hold Application
This is the instance of the application of a hold to a record. Each application of a hold will have an equivalent hold application. If two holds were applied to the same record, there would be two policy applications.

Hold Set
Hold applications refer to a hold set. A hold set represents the application of a hold to records.

Hold
Defines the information about the hold.

Purge Candidate List
Used to control what gets disposed. Purge candidate lists require approval before items get disposed. Holds can be placed on items to prevent disposition. Hold and policy applications are stored together with the managed item in a new object called the managed object.

9.4 What is a retention policy?

Retention policies define when the items in the repository should be disposed. If data is ingested into the system, and a retention policy is not applied, the records will remain in the archive indefinitely. Retention management supports a controlled disposition process for the Retention Manager.

A retention policy specifies the rules for how long to retain the data. It is a set of guidelines that describes what data will be archived, how long it will be kept, and other factors concerning the retention of the data. Data can only be disposed when the retention period expires. Retention policies can also be configured to disable disposition processing, even after the retention period has expired.

There are four different types of retention policies:

Fixed Date
The records will be retained until a particular date. This means that there is no aging and that all records associated to this policy will be eligible for disposition on the date specified in the policy. This policy is useful if the disposition date is known in advance.

For instance, you want to retain records until January 1, 2025. Records that have the retention policy applied after January 1, 2025, are immediately eligible for disposition.

All records using the same fixed date retention policy can share the same retained set. Individual retention should be set to false.

Duration
The information is kept for a period relative to a base date. For example, you opt to keep all transactions for 90 days from their creation date. For retention policies that use a cutoff, the calculation is based on UTC (Coordinated Universal Time) midnight.

The duration policy is a good fit when the age of the records is based on an attribute associated to the record (for example, creation date).

For example, the use case is that all trades entering the system must be kept for 120 days from their creation date. A duration policy of 120 days is applied to the transactions and the creation date is used as the base date. After 120 days, the trades will be eligible for disposition.

If all trades are using the same policy, you can group a day's worth of trades into one retained set and not set the age individually. This would mean that the date you are using will be stored on the retained set and not on the individual policy applications. This will make for more efficient updates if you need to change the duration of the policy (for example, to 150 days).

If you are applying a duration retention policy, all the records must have a value for the Date field. Otherwise, retention will not be applied. You will then have to refer to the logs to see which records are protected. For a duration retention policy, the system needs to know the base date to start aging from. If the base date is not known for records, consider using an event-based retention policy.

If you are applying a retention policy directly to tables or packages, depending on the storage type being used for unstructured content, you may not be able to change the duration of a retention policy once it has been set or apply additional retention policies. See the table for each supported storage option for more details.

Event

Aging only begins after a particular event has been fulfilled.

Specify a time period to retain the records from the moment an event has occurred. All records are grouped together using a context so that, when the event is triggered, all records using that context, regardless of their policy, start aging.

Event policies are recommended when a record will be retained until a specific condition is met. For instance, you want to retain records until the date the employee leaves the company. The context that you would use to group all the employee's records could be the employee ID.

For example, Bill Steele (Customer ID BILLS01) cancels all his policies with the company. The company wants to keep all the policy documents generated for 5 years from the date Bill cancels his policy and all correspondence Bill had with the company for 3 years. Bill cancels his policy on December 15, 2007. The event was fulfilled on December 15, 2007. All records belonging to Bill have a context of BILLS01. The insurance policy records have a 5-year event policy applied to them, and all the correspondence records have a 3-year event policy applied to them.

When the event is triggered for Bill's records, the insurance policy documents will be eligible for disposition on December 15, 2012, and the correspondence records will be eligible for disposition on December 15, 2010.

Event policies can only be applied using the jobs.

Mixed Mode

This policy is a combination of the Duration and Event policy types. It means that the records will age like a duration policy unless an event happens. If the event happens, then the record will age based only on the event date and will qualify for disposition even though the duration qualification date is shorter.

If the event date causes a qualification date that is later than the duration-based date, the duration-based date is used.

For example, there is a 10-year retention policy, and the aging started in January 2001. If the event is set to be July 1, 2018, the qualification date would remain as January 2010 (assuming no aging was required after the event was fulfilled). If, however, you have a 10-year retention policy, and the aging started today, but the event is fulfilled a week later, the record is immediately eligible for disposition.

Mixed mode policies can only be applied using the jobs.

If you are applying a mixed retention policy, all the records should include the Date field.

If you are using rules to apply the policy, and the base date is not specified, retention is applied but the base date is not set. Aging commences only if the event is set.

In the case of the original Apply Policy to Records job, if the retention policy is mixed mode and a date field is not specified, the retention policy will not be applied to anything. If the date field is specified, but the record specifies a null date, then the record will never begin aging.

At the end of the retention period, all data will be destroyed. The Retention Manager can disable disposition for a retention policy, which means that data governed by the policy will not be added to a purge candidate list or disposed. Even if the **Disable Disposition** is not selected, disposition will not automatically occur until the Retention Manager approves a purge candidate list.

9.4.1 What is a retention set?

A retention set is a logical container that references data under retention, including:

- Whether or not items in the set are aging together
- The type of items in set (for example, application, package, table, or record).

When a retention policy is applied to data, that data is held in a retention set.

Retention sets are created and managed by the Retention Manager.

To view an application's retention sets in IA Web App, select an application, and access the **Retention Sets** tab.

9.4.2 Applying retention

Now that you have your retention policies defined, you will want to apply those polices to records. This section will give some strategies for how to apply retention, when individual retention should be used, when to share retained sets and how to use rules to determine retention policies.

9.4.2.1 Deciding where to apply retention

To determine whether to apply a policy to the packages (AIP) or to the records (AIU) depends on how you have structured your AIPs. If all records within a package can age together based on the package date, then applying the policy to the package would be advisable. It is always better to try and use package retention as much as possible. Mixed use of package and record retention can be implemented.

> **Important**
> However, it is advisable not to mix package- and record-level retention for records in an AIP if retention is applied to each record. It makes understanding when a record should be disposed more complicated. The same logic above also applies to tables/table rows.

If the records age using a different base date or event date within the package, record-level (granular) retention would be required.

You also have the option of applying policy directly to an entire application. This means that:

- During disposition, everything in the application (and associated content and objects) will be disposed.
- Once an application has been moved to production, applying the policy and running disposition is the only way within OpenText Information Archive to delete an application.

If configured, the managed item associated to a record is backed up whenever a policy is applied to or removed from it. The configuration for the retention backups is set at the holding (AIP) or at the database (table) level.

There are three main options on how to apply retention:

Per package or table

This option allows the package or table to be disposed as a unit and is often a reasonable approach if all the items in the package or table should be disposed at the same time.

This is not always practical because it would impact how the packages or tables are organized. If you want to use hardware-based retention, this is the only strategy that pushes a date to the hardware. If this is your choice, retention can be applied directly to the package or table. For SIP archiving, you can use the holding wizard to configure retention to be applied automatically during ingestion.

Per record (AIU or table row)

This gives the maximum flexibility but does have the largest retention footprint.

You can configure this option by using jobs or by using the holding wizard (for SIP applications only).

When applying retention per record, a search must be configured that is used to determine which fields to show.

When IA Web App displays records under retention or hold, it uses an associated search to determine what information to display. When you want to apply retention per record, you must specify that search.

If you plan to apply to table rows (records), the following additional steps are required:

- The search must be associated to a single table that will be protected.
- The query must specify the row ID for the same table.
- For at least one field in the result list, set the binding to one of the values specified in the table definition (from **Edit Column**, **Result List** tab). If you do not map any fields, then only the row identifier will be shown in purge lists, which will make it difficult for retention managers to determine if the record can be approved for disposition. Fields from supporting tables cannot be shown in the purge list.

It is advisable to create a dedicated search and/or search sets for searching of records to be placed under hold or retention. These fields can be used for rules evaluation and are shown to the retention manager for determining whether to approve the purge list.

When running the apply retention jobs, the search should not be constructed so that it causes all the AIPs to be cached in or that content is retrieved from storage that has additional cost (for example, Glacier).

On the application

For very simple needs, applying to the application works well if decommissioning a legacy application and the application is only required for say, x years.

This option for applying retention has the smallest retention footprint. The amount of space to store the retention information is the least of the three options. If this is your choice, apply retention directly to the application using the IA Web App.

9.4.2.2 Applying retention to records

Retained sets

When applying retention to records using the jobs, it is possible to specify the name of the set to use. This allows you to reuse the same set over multiple runs of the job. If the name is not specified, a set will be created when the job was run. The idea of a retained set is to represent a group of similar records (for example, same policy, same base date, *etc.*).

While you can always add to existing sets (through the apply policy jobs), there are some basic rules that must be followed:

- All records must use the same retention policy.
- All records must use the same retention strategy (age together or age individually).
- If the policy is Duration or Mixed, and you are not using an age individually strategy, the base date must be the same.

  If the above rules are broken during the apply call, an exception will be thrown by the system.

  If the above rules are broken during the apply call, an exception will be thrown by the system.

Individual retention

Individual retention is used when records have a Duration or Mixed retention policy, and are using a unique date (for example, the creation date of the record). When individual retention is used, when applying retention, each record needs to specify a base date. The qualification date will be updated on the policy application associated to each record.

If you have a lot of records that have the same retention policy and share the same base date (for example, records created on the same day), it is better to age them together. Updates to the retention policy to change the duration would be easier because only the retained set would need to be updated instead of all the policy applications.

Event or Mixed policies will likely use individual retention unless they share the same event context and base date (Mixed retention only).

Age together

Age together, for example, setting age individually to false, implies that all the records that are applied using the same retained set all age using the same date.

If you are using Fixed date policies, you should always age together, since the date is the same for all records.

If all records share the same date, you can age them together.

9.4.2.3 Storage considerations

Although the licensing pricing does not change depending on where the retention policy is applied, the retention footprint can be quite significant if each record is aging individually especially when dealing with over 100,000,000 records. If the packages can be organized in a way that most of the records will mostly have the same retention, it is possible to apply retention to the package and only put additional longer retention on records meeting certain criteria using rules to avoid having to manage every record separately.

9.4.2.4 Applying retention automatically

There are two strategies for applying retention automatically:

- **During ingestion (only SIP, and only on package and records)**: Retention can be applied when ingesting packages. The holding wizard allows you to configure if retention is applied to the package or to records. If applying retention to records, you can only use a duration-based retention policy and there must be a date field available on the record. The customer must ensure that the field is always available (not null) or else the application of retention will fail.

  Note that if you are applying retention to records, this does have an impact on ingestion performance.

  There is a procedure defined in Section 9.4.2 "Updating configuration to apply retention classes dynamically on records" in *OpenText Information Archive - Configuration Guide (EARCORE-CGD)* for updating a configuration to apply retention classes dynamically on records. This procedure allows you to use values from records to apply different duration-based retention policies to the records, without having to use rules.

- **After ingestion (records only)**: Retention can be applied by a job to records after all the ingestion has been completed. If event or mixed retention policies are required, then the job is the only option.

  Note that it is possible for SIP applications to limit the jobs to only evaluate newly ingested packages rather than evaluating every record that was ever ingested.

The following table contains a summary of options for applying retention to records:

Mechanism	At ingestion	Constraints	Notes
Holding wizard configuration	Yes	Restricted to a single duration retention policy to all records in holding	Simplest option, avoids post processing.
Modifying holding wizard configuration manually	Yes	Restricted to only duration retention policies	Avoids using rules, avoids post processing

Mechanism	At ingestion	Constraints	Notes
Job applying single policy to all records matching search criteria	No	Restricted to a single retention policy	Best option for applying retention to table records if applying the same policy.
Job with rules deciding which policy if any to apply	No	No constraints, rule can decide not to apply retention	Most complicated, rules can use any information on an evaluated record

9.4.2.5 Using jobs to apply retention

The Apply Retention Policy to Records and Apply Retention Rule to Records jobs are used to apply a policy to records. Both jobs require job parameters to determine which search and search set to use, as well as the criteria to evaluate. The difference is that Apply Retention Rule to Records job uses rules to determine per record which, if any, retention policy should be applied. The Apply Retention Policy to Records job applies the one retention policy to all the records that matched the search.

For performance reasons, it is advisable to try and use criteria to narrow down the results that will be evaluated using the rule. While the rule does have powerful evaluation capabilities, it does take time to evaluate the rule for each record. If the number of records that need to be evaluated via a criteria list can be reduced, it would be advantageous from a performance perspective.

When writing rules that apply retention to records, if each record will age individually, do not specify a retained set to improve performance. The system will automatically generate and name the set required and put all records for that policy in the common set.

For SIP applications, if you plan to run jobs regularly, there are job parameters that allow the system to only evaluate records from newly ingested packages. Both the Apply Policy and Apply Hold jobs support this option.

Caution

Always ensure with the search that at least one criterion is used that has an index to avoid performance issues. The format for the search criteria is different between SIP and table applications.

Type	Sample search criteria
SIP	<data> <criterion> <name>CustomerId</name> <operator>EQUAL</operator> <value>10</value> </criterion> <criterion> <name>TransactionTime</name> <operator>LESS_EQUAL</operator> <value>2010-01-01T00:00:00.000000000Z</value> </criterion>
Table	<data> <customerId >1</customerId > <transactionTime>2010-01-01T00:00:00.000000000Z</transactionTime> </data>

Make sure to use the correct format for your application. For example, if you specify the search for a table application using the SIP format, the result is that every record would be acted on which likely not what was intended.

White spaces and carriage returns were added for readability, but are not needed. For tables, the SQL query would need to be written to use the specified criteria to the expected records.

To ensure that you use the correct format, run the search from the IA Web App, use the browser's developer tools view, and lookup the expected request payload in the network tab for the REST request. The search form may be inserting an additional criterion that is necessary for the search to return the expected results.

9.4.2.6 Configuring retention options on the holding after ingestion

It was previously possible to configure retention options when you were using the holding wizard to create a new holding. The ability to edit retention has now been added.

Configuring disposition options on holdings

When ingesting a package, any unstructured content associated with records is stored in a structure called the CI container. This container is created for each package with at least one record that has unstructured content.

If a package would be pruned because only some of the records in the package can be disposed, the customer can configure on the holding to potentially reclaim space for unstructured content specific to disposed AIUs (records). The unstructured content option can be set to either 'Shrink' or 'Blank and Shrink' to reduce the size of the CI container. When the shrink ratio is reached, the CI container is refactored. These settings can be updated by Edit Retention on the holding.

The blank or shrink options have no effect if you are using hardware-based retention for the Unstructured Content Store (or the Common store) because the content cannot be modified. For more information about these settings and how to modify them, see Section 4.2.1 "Creating a holding with the holding wizard" in *OpenText Information Archive - Configuration Help (EARCORE-H-CGD)*.

9.4.2.7 Applying multiple retention policies

It is best to avoid applying multiple retention policies. If multiple policies have been applied, the longest retention policy is used. For example, if a Duration retention policy is applied for five years from now, and a Fixed retention policy was applied to retain until 2050, the item will not be eligible for disposition until 2050.

If a shorter retention policy is applied, a new date will not be pushed to the hardware (Isilon). For Dell EMC PowerScale storage, dates can only be pushed further into the future.

The ability to apply multiple policies is not supported by ECS.

Retention Manager have a complete view of all retention policies and holds governing a record from search results using the **Compliance** tab. This tab shows a summary and details about each source of retention or hold. Sources can be the application, table/package, or record.

A table or package will not be completely disposed if one of its children cannot be disposed. The following outlines the effect of applying multiple retention policies from different sources:

- It is recommended that you pick one retention strategy instead of mixing strategies.
- If retention is applied to a package (or table) and records, the following is the behavior:
  - Different purge lists are created for both the package and records.
  - If a package is approved, the approval of purge lists for the records is ignored.
  - If records either have longer retention or holds, the package is re-factored.
  - If the purge list for the package is approved, eligible records will be disposed, even if the purge lists for those records are not approved or rejected.
- If specific records need to be kept, apply holds before approving the purge list for the package instead of rejecting the purge list for records.

The following further describes how disposition works with packages and tables:

- If a package no longer has any records after disposition, the package is marked for purge.
- Tables are no longer destroyed if retention is applied directly to them. Only the records are destroyed.
- If the table is no longer needed, consider applying retention to the application once everything is removed to destroy the application.
- Tables are no longer destroyed if retention is applied directly to them. Only the records are destroyed.

- If the table is no longer needed, consider applying retention to the application once everything is removed to destroy the application.

9.5 What is a hold?

A hold blocks deletion or disposition either temporarily or indefinitely. OpenText Information Archive uses holds to prevent data from being deleted, even if the retention policy allows for the data to be included in a purge candidate list. Holds can be applied to applications, archival information packages (AIPs)/tables, or archival information units (AIUs)/records, depending on whether you are working with a SIP- or table-based archive. Holds are created and managed by the Retention Manager.

There are two types of holds:

- Legal: The intent is that the hold will eventually be removed once the legal case has been settled.
- Permanent: The intent is that the hold will not be removed.

9.5.1 What is a hold set?

Like a retention set, a hold set is a logical container that references data with a hold applied against it. A hold set is created when a hold is applied to one or more items.

To easily view an application's hold sets in IA Web App, select an application, and access the **Hold Sets** tab.

9.5.2 Legal Matters

Holds can be used in combination with matters to form “legal matters”. Adding a hold to a matter implies the hold applies to each record in each of the saved searches of the matter. One hold set will be created per saved search per hold.

Legal matters are often constructed as part of a legal investigation where evidence is collected (in saved searches) and preserved (with holds). Hence the term legal matter.

Legal matters are created and managed by the Retention Manager. Retention Manages can access and manage legal matters created by other Retention Managers.

Legal matters can also be shared with other groups. However, if the group does not have the Retention Manager role in all of the application of a matter’s saved searches, its members will only have viewing permissions for the matter, even if it was shared for editing. If at some point, the legal matter’s holds are removed, it is no longer a legal matter and sharing applies again as explained in Search results and saved searches.

Before a legal matter can be deleted, all its holds must be removed.

Applying holds to and removing them from a matter triggers background activity, which can get canceled, fail, have conflicts with other background activity, etc. If a

user has reason to believe that as a result of all this too many or too few holds have been applied on behalf of a matter, the holds can be completely reapplied. This triggers the system to go through all search results of all the matter's saved searches and remove redundant holds, and (re-)apply required holds. This is a costly activity and should be triggered no more than strictly necessary.

9.5.3 Where to apply a hold

As with retention policies, there is the choice of where to apply holds.

Often the choice will be to apply the hold on the records directly (using a search). If you have done a cross application search, you can apply a hold to the result of the delegate searches. It is not possible to apply holds to any items in a table application that has been cached out.

If there are records that should never be disposed, consider applying a hold of type Permanent to indicate that these records should never be disposed.

For table applications, you are not allowed to apply a hold to the search result if the search is not associated with a table, or if the table is not a primary table.

When applying holds to records, the Retention Manager has the following options:

Apply hold to a search result

Applying the hold to a search result is the simplest way to apply a hold to records. After running your search, it is possible to apply a filter, and the filter impacts which records have the hold applied.

If you want to apply holds to search results over time, the following options are available:

- You can configure the Apply hold rule to records job to run on a schedule.
- You can schedule the saved search to run automatically and, if the saved search has a hold, which could be inherited from a legal matter, when the search is rerun, any new records matching the saved search criteria will be protected.

> **Important**
> Searches that are either manually rerun or scheduled to rerun will not honor removing items from the saved search. For example, if the Retention Manager edited and removed some items from the saved search, the next time that search reruns, those items will be put under hold.

Apply hold to a saved search

A hold can also be added to an individual saved search. The hold is then applied to each record of the saved search. One hold set will be applied per hold added to the saved search. This can be done by editing the saved search. The advantage of this approach is that it is possible to rerun the search and have only the records associated with search have the hold applied. Before associating a hold to the saved search, it is also possible to remove items from the saved search.

Only a Retention Manager can add a hold to a saved search, after which other Retention Managers can also access and manage the saved search.

Saved searches with holds can also be shared with other groups. However, if the group does not have the Retention Manager role, its members will have only viewing permissions for the saved search, even if it was shared for editing. If at some point all its holds are removed, sharing applies again as explained in Search results and saved searches.

A saved search with holds cannot be deleted. To delete the saved search, all its holds must be removed, and the saved search then needs to be removed from any legal matters.

If you do remove items from the saved search and apply a hold, if the search is re-run, the items that were removed from the saved search will be part of the hold.

Associated a hold and saved search to a matter

When a hold is associated with a matter, the matter is considered a legal matter. Legal matters allow the Retention Manager to apply holds to records in multiple applications in one operation.

Note: If you are planning to create legal matters with saved searches in multiple applications that have group permissions, it is strongly recommended that you ensure that Retention Managers are granted access to those applications.

Retention Managers need to pay attention to the background request that is created to update the holds when editing a legal matter. Holds will not be applied to saved searches that are not associated with a table or are not associated with a primary table.

Apply hold using rules job

It is possible to apply holds to records in an application using this job. The search criteria can be used to limit the records evaluated by the rules.

9.5.4 Rerunning and removing items of a saved search with holds

If holds are added to a saved search or its matter(s), then, after removing individual items from it, or rerunning it, the associated hold sets are automatically updated to contain exactly the resulting records in the saved search.

9.6 Granularity

Retention policies and holds can be applied to:

- Applications
- Archival information packages (AIPs) or tables
- Archival information units (AIUs) or table rows also identified as records

Retention policies can be applied to applications via the **Application Info** tab. The **Application Info** tab allows the Retention Manager to apply or remove retention policies/holds from an application. The Business Owner role can also access the **Application Info** tab but cannot perform any actions other than review the details of the selected application.

Retention policies can be applied to packages and tables from the package and table screen. Retention policies can be applied to records via the jobs. Furthermore, retention can be applied to items via the IA Shell and REST API.

Retention and holds applied to applications, AIPs, and tables are referred to as “non-granular”. Retention and holds applied to AIUs and table records are called “granular”.

Non-granular compliance data, indicating what policies and holds are applied to which applications, AIPs, and tables, is stored in the System Data (PostgreSQL). Similar to SIP structured data, granular compliance data is stored in Lucene index files.

9.7 What is disposition?

Disposition refers to the controlled process of what to do when the required aging has been met by the retention policies. Currently, the only disposition action is to destroy the item. Disposition is only done after the required approvals have been given.

How retention is applied controls what is shown in the purge lists. For example, if package-based retention is applied, the packages will be displayed in the purge list rather than the records. If retention is applied directly to the records, the records will be displayed in the purge list.

Items under retention will eventually qualify for disposition. Qualification is the process that determines when an item qualifies for disposition. It is possible that a qualification date cannot be calculated. For example, for event-based retention, if the event has not been fulfilled, a qualification date will not be set. To qualify for disposition:

- The retention period must have expired, as required by the retention policy.
- There must be no holds applied against the item that had retention directly applied to it.

Note: For packages or tables, any holds applied to the records within will not prevent disposition but cause the package or table to be pruned. For applications, the application will show up for disposition, but will be skipped if anything within the application either has longer retention or has a hold applied.

To qualify for disposition, the retention period must have expired. Holds effectively block disposition until removed.

Once data qualifies for disposition, data can be disposed. For the purposes of this guide, disposition, purging, and data destruction are synonymous.

You must run jobs to determine which data qualifies for disposition and is added to a purge candidate list, and to then actually dispose that data. The two main jobs are:

- Generate Purge Candidate List: This job creates purge candidate lists for items that are eligible for disposition, meaning that the retention period has expired and the items are not under hold. By default, this job runs on a weekly schedule.

  Ensure that the Dispose Purge Candidate List job runs on a similar schedule. If the Generate Purge Candidate List job runs before the Dispose Purge Candidate List job, purge candidate lists with the status of Approved and Under Review will be marked as Cancelled.

- Dispose Purge Candidate List: This job runs the disposition of approved purge candidate lists. By default, this job runs on a weekly schedule.

  You must schedule this job because, when an application is active, this job is the only way to remove content from archive.

Before data is disposed, its disposal must be approved by the Retention Manager.

The following outlines the disposition lifecycle:

1. Data qualifies for disposition.
2. The Generate Purge Candidate List job runs. The purge candidate list only has the status of Under Review. There is an option to configure the job to automatically approve the purge list, which means someone does not have to approve the purge list. It is also possible for an auto-approval for a Retention Manger to reject the purge list before the Dispose Purge Candidate List job is run.
3. At this point, the Retention Manager can approve or reject the purge candidate list.
4. The Dispose Purge Candidate List runs. All approved purge candidate lists are disposed.

Warning

After disposition runs, users will no longer be able to view or download the content.

9.7.1 What is a purge candidate list?

During the disposition process, purge candidate lists are generated and used to track approvals.

A purge candidate list contains:

- Information about the data that qualified for disposition when the list was generated. Even though the item is under hold, it still technically qualifies for disposition, it just will not be disposed until all holds are removed. This information is available until either the purge list is disposed, or another purge list was created, and the items were put into the new purge list.
- The count of items that were in the purge list (this never changes).
- The state of the purge list.

Purge candidate lists are always associated with an application. Lists are created per application and type of object.

To easily view an application's purge candidate lists in IA Web App, select an application, and access the **Purge Lists** tab.

Once a purge candidate list is generated, the Retention Manager reviews the contents. The Retention Manager exports the purge list if it contains records or can export the packages and tables. When exporting from the purge list screen, any records under hold or no longer eligible will not be exported.

The Retention Manager can ignore, reject, or approve the list:

- If the purge candidate list is ignored, the items in the purge candidate list may move to a new list the next time the Purge Candidate List Generation job runs.
- If a purge candidate list is rejected, items in the purge list will not be eligible for inclusion in new lists until the state of the rejected list is changed.
- If the purge candidate list is approved, items in the list will be disposed the next time the Disposition job is run for that application. If a hold is subsequently placed on items in the list or a longer retention policy applied to items in the list, those items will not be disposed, even though approval was given.

9.7.2 Disposition processing

Purge lists

Purge lists contain items that are eligible for disposition.

The type of items in the purge list depends on where the retention was applied. For example, if retention was applied to the package, then packages are shown. If retention was applied to each record, the records are shown.

If a package (or table) is eligible for disposition, any records that are under hold will not be disposed and, when the purge list is approved, those packages will be pruned (removing the records that were not under hold). If the package has a hold applied to it after the purge list was created, the package will not be disposed.

Approving purge list

Purge lists must be approved before any disposition is done. It is possible to specify a reason and attach/upload content when approving the purge list. After approving, this content can only be retrieved through a related audits search (after the Audit job has run).

Exporting purge lists

Prior to granting approval for disposition, OpenText Information Archive provides the ability to export information about the records. Depending on the type of items in the purge list, different options are available. The following chart summarizes the changes.

Action	Can use export settings	Notes
Export purge list containing SIP records	Yes	Export configuration based on search set used when applying retention
Export purge list containing table records	No	
Export package	Yes	Retention Manager chooses an existing search and search set based on the holding of the package. After choosing the search set, they select an export configuration. The possible export configurations are defined at search composition.
Export table	No	

When exporting a purge list, records that are no longer eligible for disposition will be excluded from the export.

If package-based retention was applied, it is now possible to use an export configuration to export the records that would be disposed form the package.

Records within that package that are no longer eligible for disposition will no longer be exported.

It is also possible to view the records for a single package and export from the package page, but that export differs from the export from the purge list page in that records under hold would be exported.

Automatically approving purge lists

Customers now have the option of approving purge lists when the purge list is generated. This can be configured on the job parameters for the Generate Purge List job. The property is called `autoApprove` and the default is `false`.

There is no longer a mechanism for using rules to define how records are put into purge lists.

Controlling the size of purge lists

By default, to reduce the number of items put into a purge list, partitioning information is used in addition to both the type of items in the purge list and longest retention policy. This can be configured on the job parameters for the Generate Purge List job.

There is new job property called `restrictPurgeListSize` and its default value is `true`. Setting the value to `false` reverts to the previous behavior, which can create purge lists with large numbers of items. It is recommended to set this value to `false` if the packages were ingested before 16EP5.1 and package retention was used; otherwise, each purge list will contain one package.

Avoiding pruning for disposition

If you do not want your packages to be refactored due to disposition, instead of applying holds to records, consider applying the hold to a package. This results in the entire package being excluded from disposition.

Determining what will not be disposed

Retention managers can view the hold sets for an application.

> **Important**
> The default filter for viewing hold sets does not show hold sets created because of legal matters or holds applied to saved searches. The filter can be changed to show all sources of hold sets.

9.7.3 What is granular disposition?

Granular disposition, also known as record-based retention, allows the customer to specify retention on individual records so that records can be disposed independently. This provides more customer control, as they can choose to use package or table retention as an alternative or can decide to use both.

When applying retention to records, the search criteria must identify what to protect as well as which fields to show when viewing the record. Customers can either use the Apply Policy to Record job or the Apply Retention Rule to Records job.

It is recommended to use the same search set and search when applying retention. If different searches or search sets are used, a different purge list will be created for each search defined, as well as which retention policy was applied.

If a table or a package is up for disposition, and some of their records cannot be disposed, the table or package will be re-factored. In the case of a package, the package will be modified.

Customers are encouraged to apply retention to the records or apply retention to the packages (using the holding configuration or the document class).

If the purge list for the package is approved, disposition can be done on the records even if the purge list for the records was not approved.

Tip: It is recommended that you put holds on the records that should not be disposed and then approve the purge lists.

9.7.3.1 Timing of applying holds when using granular retention

Retention can be applied to records using one of the jobs and ensure that the records are eligible for disposition. You can use one of the following jobs:

- Apply Retention Policy to Records job
- Apply Retention Rule to Records job

If a hold is applied before the Generate Purge Candidate List job runs, the records are not included the purge candidate list.

If you remove the hold from the records and then run the Generate Purge Candidate List, the old purge candidate list will be cancelled, and the items will appear in the new purge list.

If you do the following, the purge candidate list displays the held items:

1. Apply the hold to the records again
2. Approve the purge candidate list
3. Run the Disposition job

However, when viewing the details of the purge candidate list in the side panel, some of the items are not displayed.

Note: It does not matter if the records were table rows or AIUs, granular disposition works the same for table and SIP archives.

If the items are eligible for disposition before a hold is applied, the item is included in a purge candidate list. Applying a hold keeps the item in the purge candidate list (causing it to be skipped if the purge list is disposed).

9.7.4 Disposition and unstructured content

In all cases, whenever disposition is run, the disposed records are no longer displayed in search results.

- For package-based retention, when there are no holds or additional retention policies applied to records, an AIP's phase is changed to Purge.

  If the content is offline and a purge confirmation requires information, a request is made to bring the content online. The Confirmation job is required to complete the process.

  If the confirmation is sent, the AIP is removed, and no backup is completed.

- For package-based retention, when there are holds on individual records, the AIUs refer to pieces of content in the PDI file. There is an `RI index` file (`XML`) that indicates the index in the CI Container with the content.

  In the structured data, the metadata is removed for records that can be disposed. A backup is taken. That backup is put on the store defined on the holding.

  If there is unstructured content associated with any records, a new `RI XML` is created and pushed to the storage. If the previous `RI XML` is not under storage retention, the content is marked orphaned and deleted by the Clean job. If the store supports partial updates, and the CI container is not under retention, the CI container is pruned with a blank section.

  For ECS storage, for example, the store would not be updated, as a retention date was set. A new RI (table of contents) would be put into the content store, but the old RI is not cleaned up till the entire package can be disposed.

- For granular retention, in the structured data, the metadata is removed for records that can be disposed. A backup is taken, and put on the store defined on the holding.

  If there is unstructured content associated with any records, a new `RI XML` is created and pushed to the storage. The content is marked as orphaned and will be deleted by the Clean job. If the store supports partial updates, the CI container is pruned with a blank section.

For example, for ECS storage, if you are using package-based retention, the CI container will not be updated because the date was pushed to the hardware. If using granular-based retention, because ECS supports partial updates, the CI container will be pruned with a blank section.

9.7.5 Table disposition and storage footprint

After running disposition on a table application, many records may have been deleted from their PostgreSQL database. PostgreSQL will not actually return the corresponding disk space to the operating system until you (your PostgreSQL DB admin) run `VACUUM FULL`. Until then, you will not see PostgreSQL's disk usage decrease after disposition. Refer to PostgreSQL documentation on how to use `VACUUM FULL`.

9.8 Event-based retention

Event-based retention is based on the principle that retention aging does not commence until an event happens. Usually an event has a context associated to it. For example, a retention policy may want to protect employee data associated but not start the retention aging until the employee leaves the organization.

To achieve this, OpenText Information Archive stores events separately from a retention application. This separation allows the ability for multiple retention policies to share the same event and not require the customer to set an event date for each application. This means that if a retention policy is applied, and the event has already been fulfilled, retention aging immediately commences.

You can apply event based retention using the following mechanisms:

- Using the Apply Retention Rule to Records job
- Via ingestion

9.8.1 What are compliance events?

The compliance system provides the ability for a retention policy to specify a condition. For example, a condition could be `EmployeeTermination` or `TradeCloseDate`.

When the retention policy of this type (event or mixed) is applied, the system needs to know what the context will be. For example, for an employee termination, we need to know which Employee ID or, if this was a trade, close the trade transaction identifier.

If you apply a retention policy to a record, the event has a name and a context so that the system can differentiate between employee A versus employee B leaving the company. It is important to note that since multiple records could be associated with that employee, the same event name and context could be used for multiple records so the event only must be fulfilled once and then all records can start aging.

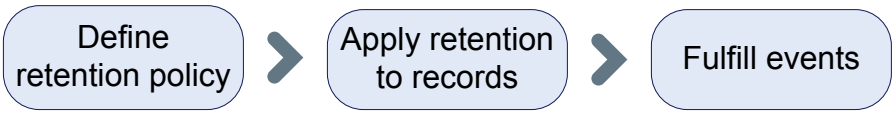
Events are modeled at the tenant level, which means events can be shared across applications.

Events are automatically created when applying retention if the retention policy is mixed or event based. Event or mixed retention policies can only be applied to records by using jobs (either using rules or not).

If you are using rules, you must specify the event context , and configure the rule to specify how to define the retention sets. The retention parameters on the bean can be set to either age individually or together. If multiple records are put into the same set and the set is aging together, one event governs the event that will be used as the base date.

If the Apply Policy to Records job is used, there are parameters that dictate how to determine, based on metadata in the record, what the event context is. Typically, the record will have a field that defines the context such as Employee id). This job always creates retention sets that age individually and the name of the retention policy to use is a mandatory parameter for the job.

9.8.2 Basic flow



Define retention policy

The retention policy must be either event or mixed, and a condition must be specified.

Apply event-based retention to records

There are two ways to apply event-based retention to records:

- Apply retention via the Apply Retention Policy to Records job
- Apply retention via the Apply Retention Rule to Records job

You can also configure that the event has been fulfilled when running the Apply Retention Policy to Records job.

Note: When applying retention, because there is only one condition, setting the event context is sufficient.

Fulfill events

Event policies age based on an event. The event that triggers the aging process may not happen until after the policy is applied to the records. The event that triggers the event might come from an external system.

For example, the following rule can look to an external system to get the event for a customer whose account has been closed. The context is set to the `customerId` and the date for the event is the date the account was closed:

```
// Rule for triggering events for SIP records
package com.emc.ia.retention.rules
//list any import classes here.
import com.emc.ia.retention.rules.beans.TriggerEventBean;
import com.mypackage.POJO_Class;

import org.slf4j.Logger;

//declare any global variables here
global Logger logger;
rule "Trigger Event" when
then
        TriggerEventBean $triggerEventBean1 = new TriggerEventBean();
        $triggerEventBean1.setContext(getCustomerId());
        $triggerEventBean1.setEventDate(getEventDate());
        $triggerEventBean1.setCondition("MyCondition"); insert($triggerEventBean1);

        logger.info("Trigger Event for Records");
end

function String getEventDate(){
  if(isAccountClosed(getCustomerId()))
   return getCloseDate(;
else
   return null;
}

function boolean isAccountClosed(String customerId) {
        POJO_Class myClass = new POJO_Class();
        return myClass.isAccountClosed(customerID);
}

function String getCloseDate(String customerId) {
        POJO_Class myClass = new POJO_Class();
        return myClass.getAccountClosedDate(customerID);
}

function String getCustomerId() {
POJO_Class myClass = new POJO_Class();
return myClass.getCustomerId();
}
```

Important

Once events have been fulfilled, it is necessary to run the Process Events job to update the qualification date for the policy applications and update the projected disposition for the managed item (representing the record).

There are two ways to fulfill events:

- Trigger events using the Trigger Event Policy job
- Trigger events using the Trigger Event Rule job

The first option uses an XML file to define which events have been fulfilled:

```
<?xml version="1.0"?>
<triggers>
 <event>
  <context>Morgan</context>
  <triggerdate>2016-02-28</triggerdate>
  <condition>EmployeeTermination</condition>
 </event>
<event>
  <context>Steve</context>
  <triggerdate>2015-02-28</triggerdate>
  <condition>EmployeeTermination</condition>
```

```
 </event>
</triggers>
```

For the Trigger Event job, the job runs a search and decides programmatically which events to fulfill. The PhoneCallsGranular sample application provides a sample rule for triggering events. See the `<IA_ROOT>/examples/applications/PhoneCallsGranular/config/application-config/rules/rule-triggerEventRule.drl` file.

9.8.3 How does aging work?

For a mixed retention policy, fulfilling the event is optional and the aging starts using the base date. If the event does happen, it takes priority and chronological path. For a mixed retention policy, a base date must be provided.

For an event retention policy, the aging does not start until the event is fulfilled. Base dates specified at policy application time are ignored when applying event-based retention policies.

The following are examples of how aging works:

- You can specify an event retention policy in which the policy is to start the aging after the employee leaves the organization.
- You can specify a mixed retention policy in which, if the event does not happen, OpenText Information Archive keeps the records for 7 years. If the event does happen, you can have the record immediately eligible for disposition (no aging) or extend the aging.

9.8.4 Audits

If you want to enable audit event types for only certain applications, the audit must be disabled at the tenant-level and then enabled only for an application. If the audit is at the tenant-level, it will be checked first before the application audit is checked (for example, application audits do not override tenant audits).

Consider modifying the audit application to choose an appropriate retention policy for your audits. The default of 89 days may not be correct for your organization.

Make sure that before you deploy into production, ensure that the correct audit levels for all operations are set. Enabling audits for AIU and table rows can have an adverse effect on performance. Therefore, care should be taken when enabling these audits. You may want to consider disabling the dispose audit event for AIUs and table rows to improve performance.

For more information about how to configure an audit, including compliance audits, see Section 9.18 “Audits” in *OpenText Information Archive - Configuration Help (EARCORE-H-CGD)*.

It is recommended to not run the Archive Audit job when ingesting either SIP or table data. It should be run regularly (the default is once a day). Depending on the volume of audits, you may want to run weekly.

9.9 Rules

The rules within the system are stored as repository objects with content (rule DRL file). These are named objects and are associated to an application. When specifying the rule in jobs that take a rule file as a parameter (apply policy, apply hold, trigger event) the name of the rule object should be used. This field is optional and, if not specified, will run every rule (for the operation type) associated to the application. Each DRL file can have multiple rules within it. When the rule system evaluates a record against a DRL file (for example, rule object), all rules within a DRL file will fire and be evaluated against the record. If the record matches more than one rule, then multiple operations (for example, apply retention) can be performed.

If you do not want all rules to fire within the DRL file, then separating your rules into multiple named rule objects (separate DRL files) would give you more control over which rules to use within the rule jobs.

For debugging, use global logger, as in the following example. Logs will be put into `rules.log` for the rules.

For example, the following DRL file has two rules in it:

```
// Rule for applying policy to SIP records
package com.emc.ia.retention.rules
//list any import classes here.
import com.emc.ia.retention.rules.beans.ApplyRetentionBean;
import com.emc.ia.retention.rules.beans.AiuRecordBean;
import org.slf4j.Logger;

//declare any global variables here
global Logger logger;

rule "Apply policy to customer ID 000103 and 000147"
when
    $aiuRecordBean:AiuRecordBean();
eval($aiuRecordBean.getRecordRows().get("CustomerID").equals("000103") ||
    $aiuRecordBean.getRecordRows().get("CustomerID").equals("000147"))
then
    ApplyRetentionBean $applyRetentionBean = new ApplyRetentionBean();
    $applyRetentionBean.setRetentionPolicy("Policy B");

    // setting the base date for an event based retention policy is not necessary,
    // doesn't hurt but is not relevant
    $applyRetentionBean.setBaseDate($aiuRecordBean.getRecordRows().get("CallStartDate"));

    // setting the event content is required for event and mixed retention policies

$applyRetentionBean.setEventContext($aiuRecordBean.getRecordRows().get("CustomerID"));
    $applyRetentionBean.setTriggerEvent(false);
    $applyRetentionBean.setIndividualRetention(true);

    // set to true if the base date calculation is different
    $applyRetentionBean.setReplaceRetention(true);
    // uncomment out to remove named retention policy if replacing, policy must exist
    // $applyRetentionBean.setRetentionPolicyToRemove("Legacy Patent Duration Policy");

    // optional to set retained set name, a new set will be created when the job runs
    // $applyRetentionBean.setRetainedSetName("Policy B : " +
```

```
    // $aiuRecordBean.getRecordRows().get("CustomerID"));
    $applyRetentionBean.setId($aiuRecordBean.getId());
    insert($applyRetentionBean);
    logger.debug( "Apply Policy B for records for customer ID : " +
    $aiuRecordBean.getRecordRows().get("CustomerID") + " AIU Id : " +
    $aiuRecordBean.getId() + " Retention Policy : " +
    $applyRetentionBean.getRetentionPolicy());
end
```

```
rule "Apply policy to customers that start with A" when
    $aiuRecordBean:AiuRecordBean();
eval($aiuRecordBean.getRecordRows().get("CustomerFirstName").startsWith("A"))
then
    ApplyRetentionBean $applyRetentionBean = new ApplyRetentionBean();
    $applyRetentionBean.setRetentionPolicy("Policy A");
    $applyRetentionBean.setBaseDate($aiuRecordBean.getRecordRows().get("CallEndDate"));

$applyRetentionBean.setEventContext($aiuRecordBean.getRecordRows().get("CustomerID"));
    $applyRetentionBean.setTriggerEvent(false);

    // generally on a mixed retention policy, you should always set the individual
    // retention to true as each record has a different base date
    $applyRetentionBean.setIndividualRetention(true);
    $applyRetentionBean.setRetainedSetName("Policy A : "
      +$aiuRecordBean.getRecordRows().get("CustomerID"));
    $applyRetentionBean.setId($aiuRecordBean.getId());

    insert($applyRetentionBean);
    logger.debug( "Apply Policy A for records for customer ID : " +
        $aiuRecordBean.getRecordRows().get("CustomerID") + " AIU Id : " +
        $aiuRecordBean.getId() + " Retention Policy : " +
        $applyRetentionBean.getRetentionPolicy());
end
```

Both rules will be evaluated for the records and, if a record matched both rules, (for example, customer ID is either 000103 or 000147 or the name begins with “A”) Policy A and Policy B will be applied to the record.

If your intent is to ensure only one rule applies a retention policy for a record, consider looking at the DROOLS salience and activation-group constructs.

```
rule "5 Year Policy For Technology stocks“
salience 3
activation-group "matchrule"
```

Note: The remove retention flag has two uses. If the calculation in the rule has been changed for base date or trigger dates, set to `true`. If you do not set this flag, running the job causes the retention policy to be applied again with the new base date (which is not desired since, if the new date is earlier, the qualification date will not change).

The second use is to set a second attribute on the bean to remove a named policy to avoid having to remove the old retention policy from all records. See the Patent sample application `README` file for details.

9.9.1 Accessing external systems

Within the rule file, the rules system has the capability to use external methods located in a JAR file. If the JAR file is on the CLASSPATH of the IA Server application, the rule will be able to use the methods from the JAR file.

First, you will need to import your class in:

```
import com.mypackage.POJO_Class;function Boolean isAccountClosed(String customerId) {
     POJO_Class myClass = new POJO_Class();
     return myClass.isAccountClosed(customerID);
}
```