Axcelerate 5.9.1

Project Setup with Structured Storage V2

Published: 2016-Dec-05

Contents

1 Project Setup with File-based Storages

In Axcelerate 5.9.1, a new structured storage method called *Structured Storage V2* is available. It comes with project- or pod-wide storage roots for nearly all storages and offers

- diverse file sharing options that reduce storage space.
- easy scalability through adding new storage roots when needed.
- independent matters. All stored files are accessible from the application, without dependencies on external sources or other applications.

This new structured storage method is used for new on Premise projects by default.

Note: Projects created in Axcelerate 5.9 or before do not use *Structured Storage V2*. If you create a new Axcelerate Review & Analysis application from an updated Axcelerate Ingestion application that does not use *Structured Storage V2* , the new application won't use *Structured Storage V2* either.

Structured Storage V2 is available if these templates are used:

For Axcelerate Ingestion application

documentHold.sytemTemplate-Axc5-v2

For Axcelerate Ingestion index engines

singleMindServer.documentHoldSystemTemplate-v3

For aAxcelerate Review & Analysis index engine

singleMindServer.axcelerateStandaloneSystemTemplate-Axc5-v3

1.1 Create an Application for File-based Storages

Usually, the only application you need to create manually, is Axcelerate Ingestion.

1. In CORE Administration, open a workspace and click on its name in the treeview.
2. From the **Actions** menu, select **Create application**.
   The **New Application Wizard** opens.
3. In the **Define application type** step, select **Axcelerate Ingestion**.
4. In the **Define template type** step, select **System template**.
5. In the **Define application template** step, select either **Single engine** or **Meta Engine with sub-engines**.
6. In the **Additional configuration options** step, keep the **Native storage options**.
7. In the **Application details** step, enter a unique display name for the application

and the **Client ID** and **Case Name**.

Client ID and **Case name** are shared with all index engines. They are used for data storage URIs. The Client ID must not be empty. Keep *default* if you do not want to specify a custom Client ID associated with this application.

8. Follow the wizard and finish it.

The Axcelerate Ingestion application is created.

1.1.1 Prerequisites

» All index engines using structured storage V2 must have access to the servers hosting the master and slave storage roots.

1.1.2 Storage Properties for Projects created with Axcelerate 5.9.1

» Storages are file-based. Most storage handlers are structured storage V2 storage handlers.

» Applications use a default central storage root, configured in the CORE master service during installation. A custom storage root per project (= Axcelerate Ingestion with all depending matters) is possible.

» Scalability: If the master storage root is full, another storage root can be added and made the master storage. The previous master storage is made a slave storage, and stored files can be retrieved and removed from it. New files are added only to the current master storage.

» Stored files are SQL-managed. SQL-management in combination with structured storage V2 offers:

  » single instance storage, i.e. file sharing within one storage handler

  » external cloning, i.e. file sharing between diffent storage handlers.

  » case-wide sharing of native files

  » compressed storage of native files, images and redactions

» Axcelerate Ingestion and Axcelerate Review & Analysis use primary storages by default, i.e., deleting an application does not affect other applications using the same stored files. Also, applications do not use an external storage by default.

What does this mean for native files and images?

» Native file copies are created for any type of data loading, including CSV data load. A reference to these native file copies is added to the Axcelerate Ingestion storage handler. During publishing, a reference to these native file copies is added to the Axcelerate Review & Analysis storage handler.

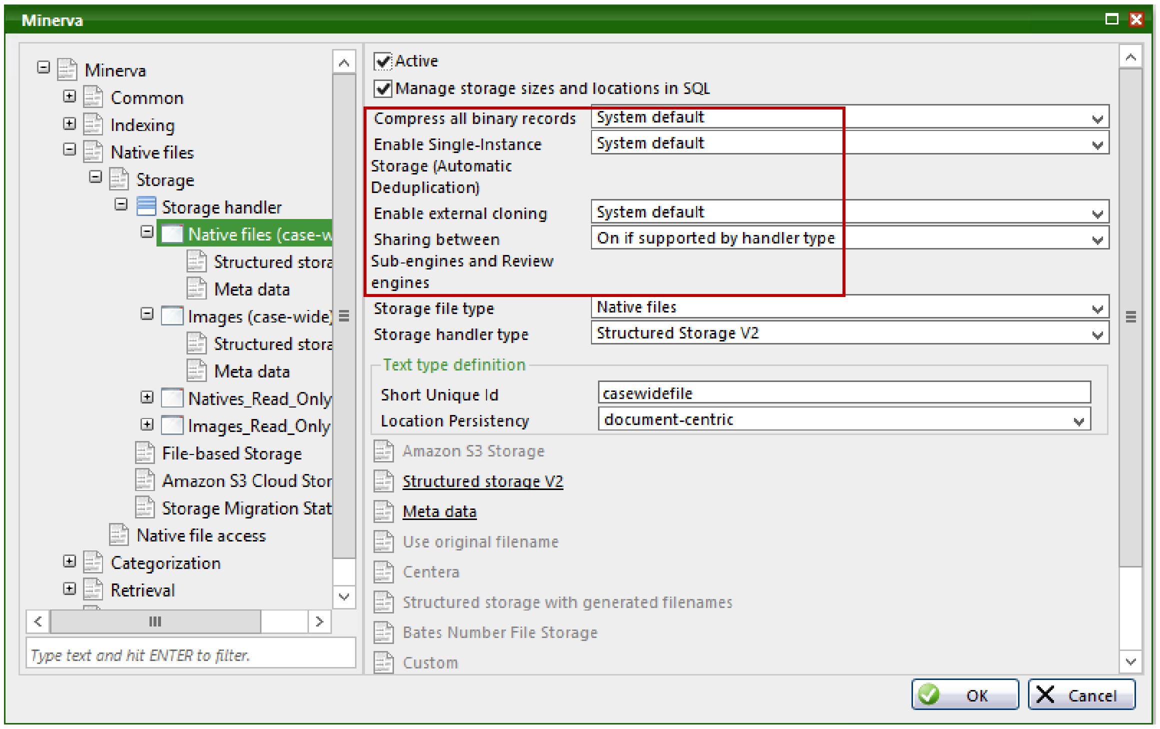
» Image copies are imported with CSV load or CSV merge. A reference to these image copies is added to the respective storage handler. During publishing images are copied to the Axcelerate Review & Analysis storage handler for images.

1.2 Storage Handlers in Ingestion

Native files (case-wide)

Structured Storage V2 storage handler. Manages native files copied from source documents. The storage handler supports sharing items of storage file type **Native files** on storage handler, index engine and case level. i.e. between an Axcelerate Ingestion application and all dependent matters.

Storage handler default settings



[✓] Active
[✓] Manage storage sizes and locations in SQL

Compress all binary records	System default
Enable Single-Instance Storage (Automatic Deduplication)	System default
Enable external cloning	System default
Sharing between Sub-engines and Review engines	On if supported by handler type
Storage file type	Native files
Storage handler type	Structured Storage V2

Text type definition

Short Unique Id	casewidefile
Location Persistency	document-centric

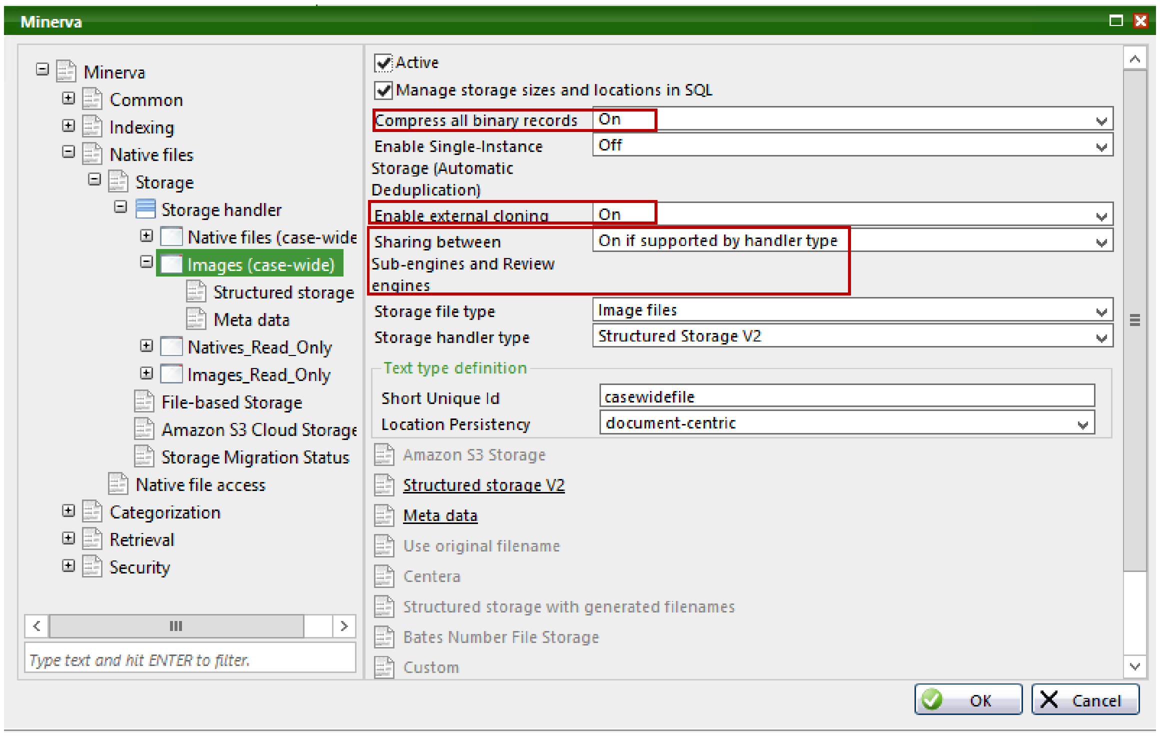
Setting	Set to	Result
Compress all binary records	System default	Upon storing, file is compressed to ZIP file, because the system default activates compression for storage file type **Native files**.
Enable external cloning	System default	Native files are shared between storage handlers, because the system default activates compressoin for storage file type **Native files**.

Setting	Set to	Result
Sharing between Sub-engines and Review Engines	On if supported by handler type	The storage handler type is Structured Storage V2. This handler type allows case-wide sharing.

Images (case-wide)

Structured Storage V2 storage handler. Manages image files copied during CSV load or CSV merge, or during standard file crawls that use CSV reference files. The storage handler supports sharing image files with other storage handlers of the same type between a case's applications, i.e. between an Axcelerate Ingestion application and all dependent matters.

Storage handler default settings



✓ Active
✓ Manage storage sizes and locations in SQL

Compress all binary records	On
Enable Single-Instance Storage (Automatic Deduplication)	Off
Enable external cloning	On
Sharing between Sub-engines and Review engines	On if supported by handler type
Storage file type	Image files
Storage handler type	Structured Storage V2

Text type definition

Short Unique Id	casewidefile
Location Persistency	document-centric

Setting	Set to	Result
Compress all binary records	On	Upon storing, file is compressed to ZIP file. **Note:** If the setting was System default, no mages would be compressed, as the system default only allows compression for storage file type **Native files**.

Setting	Set to	Result
Enable external cloning	On	Image files are shared between storage handlers **Note:** If the setting was **System default**, no images would be shared, as the system default only allows external cloning for storage file type **Native files**.
Sharing between Sub-engines and Review Engines	On if supported by handler type	The storage handler type is Structured Storage V2. This handler type allows case-wide sharing.

Natives_Read_Only

Local storage handler. Manages references to native files. This handler is required for copying native files to the **Native files (case-wide)** storage.

Images_Read_Only

Local storage handler. Manages references to images. This handler is required for copying images to the **Images (case-wide)** storage.

1.3 Storage Handlers for Matters

Note: If you create productions via script or in the legacy Axcelerate Analysis module, check the production and OCR text storage paths before the first production run. You cannot change storage paths for structured storage or Bates number storage once they are used for storing files.

Native files (case-wide)

Structured Storage V2 storage handler. Manages native files that were copied during ingestion. The storage handler supports compression and case-wide sharing of native files, i.e. between an Axcelerate Ingestion application and all dependent matters.

Images (case-wide)

Structured Storage V2 storage handler. Manages image files ingested during CSV load or CSV merge, or during standard file crawls that use CSV reference files. The storage handler supports sharing image files with other storage handlers of the same type between a case’s applications, i.e. between an Axcelerate Ingestion application and all dependent matters.

Redactions

Structured Storage V2 storage handler.Manages redactions and annotations applied to documents during review.

Productions

Bates Number File storage handler. Manages production results created in the Axcelerate Analysis module or via the `manageProduction.bat` script. Not used by production exports or export snapshots that users create in the Axcelerate 5 front end.
Bates Number File storage is a locally configurable structured storage. Locations can be different between index engines.

OCR Text

Bates Number File storage handler for extracted text files created during production. Manages production results created in the Axcelerate Analysis module or via the `manageProduction.bat` script. Not used by production exports or export snapshots that users create in the Axcelerate 5 front end.
Bates Number File storage is a locally configurable structured storage. Locations can be different between index engines.

Document View files

Structured Storage V2 storage handler. Manages conversion results.
By default, this storage handler does not use compression, nor any type of file sharing. Enabling compression or file sharing would not have any positive effect.

Production export snapshots

Structured Storage V2 storage handler. Manages production export snapshots. By default, this storage handler is enabled for external cloning, i.e., it can reference native files managed by another storage handler.

Production exports

Structured Storage V2 storage handler. Manages production export ZIP files. By default, this storage handler is enabled for external cloning, i.e. it can use native file references managed by the production export snapshots storage handler, in order to copy them to the export ZIP file..

Images_Read_Only

Structured storage handler. Manages references to images. This handler is required for copying images to the **Images** storage during a CSV merge.

Natives_Read_Only_from_ECA_<namespace of the first index engine used for ingestion>

Structured storage handler. This handler is required for copying native files from Axcelerate Ingestion index engines to the **Native files (case-wide)**, if case-wide sharing is not active.

2 Storages

2.1 Storages Introduction

Several features require that files are stored. This is handled by storage handlers.

Each storage handler is responsible for exactly one storage file type. Storage file types are, for example, native files, document view files, or production files.

Each storage handler has a storage handler type. It defines the storage properties, and the access to stored data.

Storages can be managed in an SQL database. SQL-managed storages provide storage metrics and other, optional features.

2.1.1 Order of Storage Handlers

Storage handlers are configured in a specific order which must not be changed. The order of storage handlers determines how requests for stored files are treated.

The rule for file requests is simple: The storage handler placed higher than others in the index engine configuration is called first. If, in the document metadata, there is no access information for this storage handler, the storage handler of the next, lower level is called.

If there is a wrong access information for a storage handler, the next, lower level is not called. Each file type has its own storage handlers.

2.1.1.1 Configure Storages

The configuration settings in CORE Administration are shown in alphabetical order.

2.1.1.1.1 Active

Activate to enable this storage handler.

Location: Index engine: **Native files > Storage > Storage handler > <storage handler instance name>**

Allowed values:
- true
- false

Default value:
- true

2.1.1.1.2 Location Persistency

Defines whether storage location information is stored with document metadata or in a document-independent way.

Most storages are document-centric and location information is stored with the document. But storages for document collections like the production export and production export snapshots require a specific field for storage location information.

Location: Index engine: **Native files > Storage > Storage handler > <storage handler instance name>**

Allowed values:
- document-centric
- standalone (document-independent)

Default value:
- None

2.1.1.1.3 Short Unique ID

The unique ID distinguishes between multiple writable storages for the same storage file type in one index engine. If there is only one storage for a given file type, this field can be empty.

The index engine will not start if multiple writable storage handlers for the same storage file type have the same ID.

Location: Index engine: **Native files > Storage > Storage handler > <storage handler instance name> > Text type definition**

Allowed values: any string (a-z or 0-9)

Default value:
- None

Related:

"Storage file type" below

2.1.1.1.4 Storage backup folder

Define a backup folder for storage files used during reindexing. The folder must be located inside the index engine folder.

Relative file paths are relative to the `%MINDSERVER_PROJECTS%\singleMindServer.<index engine name>\Config` folder.

Location: Index engine: **Native files > Storage**

Allowed values: a file path

Default value:
- ../storageBackup

2.1.1.1.5 Storage file type

Type of files to be covered by this handler.

Location: Index engine: **Native files > Storage > Storage handler > <storage handler instance name>**

Allowed values:

- Native files
- Image files
- Production exports
- Redaction files
- Production files
- OCR text
- Native files staging
- Document view files
- Production export snapshots

Default value:

- None

2.1.1.1.6 Storage handler type

Select the type of storage.

Location: Index engine: **Native files > Storage > Storage handler > <storage handler instance name>**

Allowed values:

- Structured Storage V2
- Structured Storage
- Centera
- Original Filename
- Custom
- Readonly file
- Readonly Centera
- Bates Number File Storage
- Amazon S3 Storage
- Readonly Amazon S3 Storage
- Readonly Structured Storage V2
- Separate text type file

Default value:

- None

2.2 SQL-managed Storages

SQL-managed storages provide capabilities for binary data compression, storage item sharing and storage metrics capture.

SQL management can be set in CORE Administration, but only for newly created projects that do not contain any documents yet. SQL management can be set for all storages of an index engine, or for individual storage handlers, except for read-only storage handlers.

For each SQL-managed storage, storage metrics capture is automatically enabled.

The database used for storage management is the NGDB database that is installed with the Recommind software.

2.2.1 Storage Compression

Files can be compressed to ZIP format when they are added to a storage. The compressed files keep the initial file extension, which is needed for file processing.

Storage compression is especially useful for native file storage that often takes a lot of disk space. Image storage space can also be significantly reduced through compression.

Note: It is not recommended to use compression for storages that contain already compressed formats, such as XDL or SVG files. This would only lead to an overhead for the compression process.

2.2.2 Compression, Single-instance Storage and System Performance

Compression and single-instance storage reduce disk space usage considerably, for native files and for images.

However, they may reduce system performance to some extent. This depends on the files handled by a storage handler, and on the storage medium.

Some general tests showed that with compression and single-instance storage enabled, data loading is 5 to 20% slower. Review and document view performance is maximally 5% slower.

Performance is significantly better when files are stored on SSD (Solid State Drive).

2.2.3 Enable SQL-managed Storage for all Index Engine Storages

Required:

» For index engines of an Axcelerate database application: Data have not been loaded yet.

» For index engines of an Axcelerate Review & Analysis application: Data have not been published yet.

1. In the index engine configuration, open the **Native files > Storage** node and enable **Manage storage sizes in SQL**.
2. If you want to enable compression, for **System default: Compression support**, select **On**.

3. If you want to enable single-instance storage, for **System default: Single-Instance Storage support**, select **On**.
4. Make sure that for each storage handler, **Compress all binary records** and the **Enable Single-Instance Storage** are set to **System default**.
5. Restart the index engine.
6. If there are multiple index engines, do this for every engine.

All storage handlers, except the read-only handlers, use compression and single-instance storage.

2.2.4 Enable SQL-managed Storage where it is Recommended

The highest benefit of SQL-managed storage is for native file storage. Compression can also be very useful for redactions. The **Auto** setting enables SQL management for these storages, if they use the **System default** setting.

Required:

- For index engines of an Axcelerate database application: Data have not been loaded yet.
- For index engines of an Axcelerate Review & Analysis application: Data have not been published yet.

To enable SQL-managed storage where it is recommended:

1. In the index engine configuration, open the **Native files > Storage** node and enable **Manage storage sizes in SQL**.
2. If you want to enable compression, for **System default: Compression support**, select **Auto**.
3. If you want to enable single-instance storage, for **System default: Single-Instance Storage support**, select **Auto**.
4. Make sure that for each storage handler, **Compress all binary records** and the **Enable Single-Instance Storage** are set to **System default**.
5. Restart the index engine.
6. If there are multiple index engines, do this for every engine.

Result:

Compression is active for native storage handlers, except the read-only handlers, and for redactions. Single-instance storage is active for native storage handlers, except the read-only handlers.

2.2.5 Enable SQL-managed Storage for a Single Storage

Required:

- For index engines of an Axcelerate database application: Data has not been loaded yet.

» For index engines of an Axcelerate Review & Analysis application: Data has not been published yet.

To enable SQL-managed storage for a single item:

1. In the index engine configuration, open the **Native files > Storage** node and enable **Manage storage sizes in SQL**.
2. For the respective storage handler, set **Compress all binary records** or the **Enable Single-Instance Storage** to **On**.
3. Restart the index engine.
4. If there are multiple index engines, do this for every engine.

Result:

Compression and single-instance storage are active for the storage handler. This overrides the system defaults for compression and single-instance storage set in the **Native files > Storage** node.

2.2.6 Migrate File-based Storages to SQL in Axcelerate 5

It is possible to migrate storage items to SQL. Usually, this type of migration is only needed for projects created in Axcelerate 5.5 or earlier. SQL management is enabled by default for native file storages in Axcelerate 5.6 and later versions.

2.2.6.1 What happens during migration?

During migration, files are copied to a *new* storage folder, and file metadata is transferred to the NGDB database. Migration separates the existing storages of Axcelerate Review & Analysis and Axcelerate Ingestion, i.e., the native file storage previously been shared exists both in Axcelerate Review & Analysis and Axcelerate Ingestion after migration.

2.2.6.2 How to proceed

You need to migrate the storages of an Axcelerate Ingestion index engine and the depending Axcelerate Review & Analysis index engines synchronously. This is due to the fact that Axcelerate Review & Analysis native file storage references the Axcelerate Ingestion native file storage.

Important: Synchronous migration means that you have to do each of the steps detailed below first for Axcelerate Ingestion and then for Axcelerate Review & Analysis. Only continue with the next step when the preceding step is finished for all involved index engines.

Note: Users can continue to work with Axcelerate 5 and the Axcelerate Ingestion user module during migration.

1. Create a new, SQL-managed storage handler in all index engines

In the index engine configuration, create a new storage handler for native files above all other native file storage handlers. Specify a new storage location. Enable **Manage storage sizes in SQL** for the index engine. Restart the index engine.

2. Copy files from a non-SQL to an SQL-managed storage handler for all index engines

Make sure that all index engines are correctly configured and restarted.

Copy the files from each involved old storage to the new storage using this script command:

```
migratestorage -project <index engine.identifier> [-
host <hostname>] [-port <portNumber]-op copy -
sourceStorageHandlers NATIVE -storageHandlers
NATIVE:<target storage handler name> [-wait] [-
priority <number>] -user <username> -password
<password>
```

Tip: The optional `-wait` parameter prompts you when the job is finished. If you do not use it, go to the **Jobs** tab to check whether the job is finished.

By default, the job priority is 10. This means it has the same priority as other multiple-documents jobs.

Example for a script run on the index engine host:

```
migratestorage -project singleMindserver.minerva -op
copy -sourceStorageHandlers NATIVE -storageHandlers
NATIVE sqlnativestorage -wait -user "Ann Smith" -
password gS93!
```

This copies the complete storage to the SQL-managed storage handler called *sql-nativestorage*.

On the CORE Administration**Jobs** tab for the respective application, check that the copy job is succesfully finished.

3. Remove duplicate storage files for all index engines

To avoid double storage space, remove all files from the source storage that exist in the target storage using this script command:

```
migratestorage -project <index engine.identifier> [-
host <hostname>] [-port <portNumber]-op
```

```
removeFromOutDatedStorage -storageHandlers <NATIVE> -
user <username> -password
```

Example for a script run on the index engine host:

```
migratestorage -project singleMindserver.minerva -op
removeFromOutDatedStorage -StorageHandlers NATIVE -
user "Ann Smith" -password gS93!
```

This removes duplicate files from all storages that are not referenced in the first modifiable storage handler for the same file type. The first modifiable storage is the one that occurs higher in the storage handler order of the index engine configuration.

4. Check that files in the new storage are accessible

In Axcelerate 5, in the document viewer, click **Download** to download a native file. If this works as expected, the new storage is accessible for all documents.

2.2.7 Configure SQL-managed Storages

The configuration settings in CORE Administration are shown in alphabetical order.

2.2.7.1 Compress all binary records

Activates zip compression for every binary data stored with the respective storage handler. The choice **System default** configures the storage according to the option **System default: Compression support**.

Requires that **Manage storage sizes in SQL** is enabled.

Location: Index engine: **Native files > Storage > Storage handler > <storage handler instance name>**

Allowed values:
- System default
- Off
- On

Default value:
- System default

Related:

"System default: Compression support" on the next page

2.2.7.2 Manage storage sizes in SQL

If enabled, storages can be managed in an SQL database. The database supports single instance storage, compression, and storage metrics capture.

Location: Index engine: **Native files > Storage**

Allowed values:

- true
- false

Default value:

- Axcelerate Ingestion: true for new projects created in version 5.6 or later
- Axcelerate Review & Analysis: true for new matters created in version 5.6 or later
- Axcelerate ECA & Collection, if not part of Axcelerate 5: false
- Decisiv Search: false

2.2.7.3 System default: Compression support

The system-wide default for zip compression of binary storage data. This setting controls how storage handlers with compression set to **System default** are configured.

Auto enables compression for the storage file type **Native files** or **Redaction files**.

Auto also enables compression for the storage file type **Images**, but only for **structured storage V2** storage handlers.

Requires that **Manage storage sizes in SQL** is enabled.

Location: Index engine: **Native files > Storage**

Allowed values:

- Auto
- On
- Off

Default value:

- Auto

2.3 Primary Storages

Primary storages make sure that each application has its own, independent native file and image storages.

With primary storages, an Axcelerate Review & Analysis application is completely independent from the Axcelerate Ingestion application from which it was created. Any stored objects are directly accessible from Axcelerate Review & Analysis, i.e., from the published matter.

You can remove an Axcelerate Ingestion application without any impact on published matters.

Applications are also independent from external sources (except if external storages are explicitly configured). For example, during CSV Load or CSV Merge, binary copies of referenced native and image files are stored by the application's storage handlers.

You can remove an external source or deactivate the connection to an external source without any impact on your project.

Note: Primary storage can have different matter publishing results. For example, if it is combined with case-wide sharing for native files, a reference count for the respective native files will be added to the NGDB database. If it is not combined with case-wide sharing for native files, native files will be physically copied to the native file storage of the Axcelerate Review & Analysis application. In both cases, links to native files in the published matter will not be broken when the Axcelerate Ingestion is deleted. Native file storages are completely independent.

2.3.1 Configure Primary Storages

The configuration settings in CORE Administration are shown in alphabetical order.

2.3.1.1 Copy storage items into primary storages during indexing

During data load, all files referenced in indexed documents (even documents ingested via CSV Load, or native or image files added via CSV Merge) will be copied into index engine storages.

During a publish, all files referenced in indexed documents are copied, too, or a reference count is added to the SQL database, depending on case-wide sharing settings.

This allows direct file access for all applications, independently of other applications or sources.

Location: Index engine: **Native files > Storage**

Allowed values:

- true
- false

Default value:

- Applications mainly using Amazon S3 storages: true
- Applications mainly using structured storage V2: true
- Appications mainly using structured storage: false

2.4 Single-instance Storage

Single-instance storage is used to store identical items only once per storage handler. Single-instance storage is available for SQL-managed storages.

The system default is set to **Auto** for the complete storage system. This setting means that single-instance storage is enabled for items with the storage file type **Native files** in writable storages. The storage of these items often takes a lot of disk space, especially in Axcelerate Ingestion, where usually a lot of duplicates exist.

You can remove the **Auto** setting for individual storage handlers.

Besides for native files, single instance storage may also be useful for:

- storage file type **Production exports**
- storage file type **Production export snapshots**

Single instance storage does not make sense for:

- storage file type **Image files**
- storage file type **Document view files**
- storage file type **Redaction files**
- storage file type **Production files**

You can enable single-instance storage even after ingestion or publishing. The system then ignores items that were stored before single-instance storage was enabled.

How does single-instance storage work?

For identifying duplicate items, the system uses a SHA-2 hash sum that is calculated from the binary data. The storage handler only stores the first of the duplicate items. This item is referenced in the metadata of all documents that use one of the duplicates.

The reference numbers for duplicates are stored in the database, too. Whenever a document in a project is deleted, or replaced in a way that the hash sum changes, the number of duplicates is reduced. The actual item is only removed from the storage if there is no document with a reference to this item anymore.

Note: Do not confound single-instance storage with document de-duplication or duplicate detection during publishing. You can use single-instance storage and, at the same time, publish duplicates.

2.4.1 Single-instance Storage Prerequisites

- SQL management is enabled.

Related:

"SQL-managed Storages" on page 13

2.4.2 Configure Single-instance Storage

The configuration settings in CORE Administration are shown in alphabetical order.

2.4.2.1 Enable Single-Instance Storage (Automatic Deduplication)

Activates binary object de-duplication for the respective storage handler. Set to **System default** to take over the system default setting for the index engine.

Requires that **Manage storage sizes in SQL** is enabled.

Location: Index engine: **Native files > Storage > Storage handler > <storage handler name>**

Allowed values:

- Off
- On
- System default

Default value:

- System default

Related:

"System default: Single-Instance Storage support (Automatic Deduplication)" below

2.4.2.2 System default: Single-Instance Storage support (Automatic Deduplication)

The system-wide default for single-instance storage. This setting controls how storage handlers with single-instance storage set to **System default** are configured.

Auto enables single-instance storage automatically for all native file storage handlers that are no **Read-only file** storage handlers.

Requires that **Manage storage sizes in SQL** is enabled.

Location: Index engine: **Native files > Storage**

Allowed values:

- Auto
- On
- Off

Default value:

- Auto

2.5 External Cloning

External cloning is storage item-sharing between storage handlers of a single index engine.

One example for storage item sharing betwee storage handlers is native file production. For an item referenced by the **Axcelerate Review & Analysis Native files** storage handler, a reference is added to the SQL database when you produce the respective document.

Without external cloning, the native file would be copied to the production export snap-shots storage. External cloning saves the time and space needed for file copies that are only temporary.

Note: External Cloning always includes single instance storage, even if single instance storage is disabled for a storage handler in the index engine.

System default

You can set external cloning per storage handler or use the system default . If the system default is set to **Auto**, external cloning is enabled for these storage handlers:

Sharing of storage file type Native

- Native files
- Native files staging
- Production files
- Production export snapshots
- Production exports (Uses references of production export snapshots to create native file copies that should occur in the export ZIP file.)

When is external cloning for images recommended?

- If images are imported via CSV features. External cloning is the prerequisite for case-, wide sharing. If you want to publish images from Axcelerate Ingestion, you need case-wide sharing, and therefore external cloning.
- If imported images are used for production export.

2.5.1 External Cloning Prerequisites

- SQL-management is enabled for storages.
- Storage handler type is **Structured Storage V2**.
- Storage file types are identical.

2.5.2 Configure External Cloning

The configuration settings in CORE Administration are shown in alphabetical order.

2.5.2.1 Enable external cloning

Activates external binary object cloning: if an object is cloned (duplicated) from one storage to another, only one of them will be stored (with some reference mechanism for the other). This requires managed SQL locations. The choice 'System default' configures the storage according to the option 'System default: support binary cloning' (see above). For read-only handlers, the choice 'System default' expands to 'Off'.

Location: Index engine: **Native files > Storage > Storage handler > <storage handler name>**

Allowed values:
- » System default
- » On
- » Off

Default value:
- » On

2.5.2.2 System default: support binary cloning

The system-wide default for external cloning of binary streams, i.e. among different storage handlers. This setting controls how storage handlers with 'Enable external cloning=System default' are configured. Enabling this allows algorithms to use clones in order to reduce storage costs. This requires managed SQL locations. The choice 'Auto' enables the feature for storages of file types Native, Native files staging, production, production export snapshots, and ocr."

Location: Document model/Application/Index engine/Data source: (Insert one of the menuitem snippets.)

Allowed values:
- » Auto
- » On
- » Off

Default value:
- » Auto

2.6 Case-wide Sharing

Case-wide sharing is storage item sharing between storage handlers of an Axcelerate Ingestion application and all dependent matters.

Stored items are stored once. For each occurrence in the project, one reference to the stored item is added to the SQL database.

Case-wide sharing requires that external cloning is enabled, which automatically includes single-instance storage.

Example: Native files and case-wide sharing

A native file is copied during ingestion. This creates an entry in the SQL database with one reference.

A duplicate file is loaded. This augments the reference count to two.

The corresponding documents are published. This augments the reference count to four.

Both documents are part of the production export snapshot. This augments the reference count to six.

Both documents are part of the production export. As production exports automatically delete the corresponding snapshots, the reference count goes down to four.

2.6.1 Case-wide Sharing Prerequisites

- SQL-management is enabled for storages.
- Storage handler type is **Structured Storage V2**.
- Storage file types are identical.
- External cloning is active.

2.6.2 Configure Case-wide Sharing

The configuration settings in CORE Administration are shown in alphabetical order.

2.6.2.1 Prepare for case-wide sharing

If this option is active, a storage handler is prepared for case-wide sharing.

Setting this option to active replaces the Engine ID with **shared** in the S3 base URI or with casewidefile in the the Structured storage V2 URI.

Note that **Prepare for case-wide sharing** alone merely prepares the paths; it does not share records.

Location: Index engine: **Native files > Storage > Storage handler > <storage handler instance name> Amazon S3 Storage (or Structured storage V2)**

Allowed values:
- On
- Off
- Auto (sets option to **On**, but only for storages with storage file type **Native files** and **Image files**).

Default value:
- On

2.6.2.2 Sharing between Sub-engines and Review engines

Applies binary deduplication on case-level: items inserted into one sub-engine or one review-engine are deduplicated among all engines belonging to the case.

Only available if **Enable external cloning** is active.

Location: Index engine: **Native files > Storage > Storage handler > <storage handler name>**

Allowed values:

- On if supported by handler type
- Off

Default value:

- On if supported by handler type

2.7 File-based Storage Handler Types

2.7.1 Structured Storage V2

Structured storage V2 is a file-based structured storage that can be shared across index engines.

Each storage handler using structured storage V2 has a **Structured Storage V2** node.

> **Note:** Do not modify this node. Filepaths are automatically set, and the folder structure is automatically adapted, if needed.

A structured storage is a folder on a file system. The root folder contains a sub-folder structure that depends on how files are shared. Below this structure, there is the actual structured storage with generated folder and file names.The folder structure is numeric, e.g., folder names are 000, 001, 002, sub-folder names are also 000, 001,002 etc. For structured storage V2, this structure is automatically enlarged if the default number of folders is not sufficient. It is not configurable.

As the same storage root must be used by all **Structured Storage V2** storage handlers, these storage handlers use common settings.

You find them in the index engine configuration, in the **Native files > Storage > File-based Storage** node.

When you create an Axcelerate Review & Analysis application for a matter, it takes over the settings made for Axcelerate Ingestion.

One of the big advantages of structured storage V2 is scalability. You can easily add another storage root for additional storage space.

2.7.1.1 Structured Storage Version 2 Prerequisites

- All index engines using structured storage V2 have access to the servers hosting the respective storage roots.
- The **Shared Client ID** and **Shared Case Name** must be identical for all applications and index engines of a project, i.e. for an Axcelerate Ingestion application and all dependent matters. None of the must be empty. They may not be changed once ingestion has started.

2.7.1.2 Configure Storage Root on Master Service

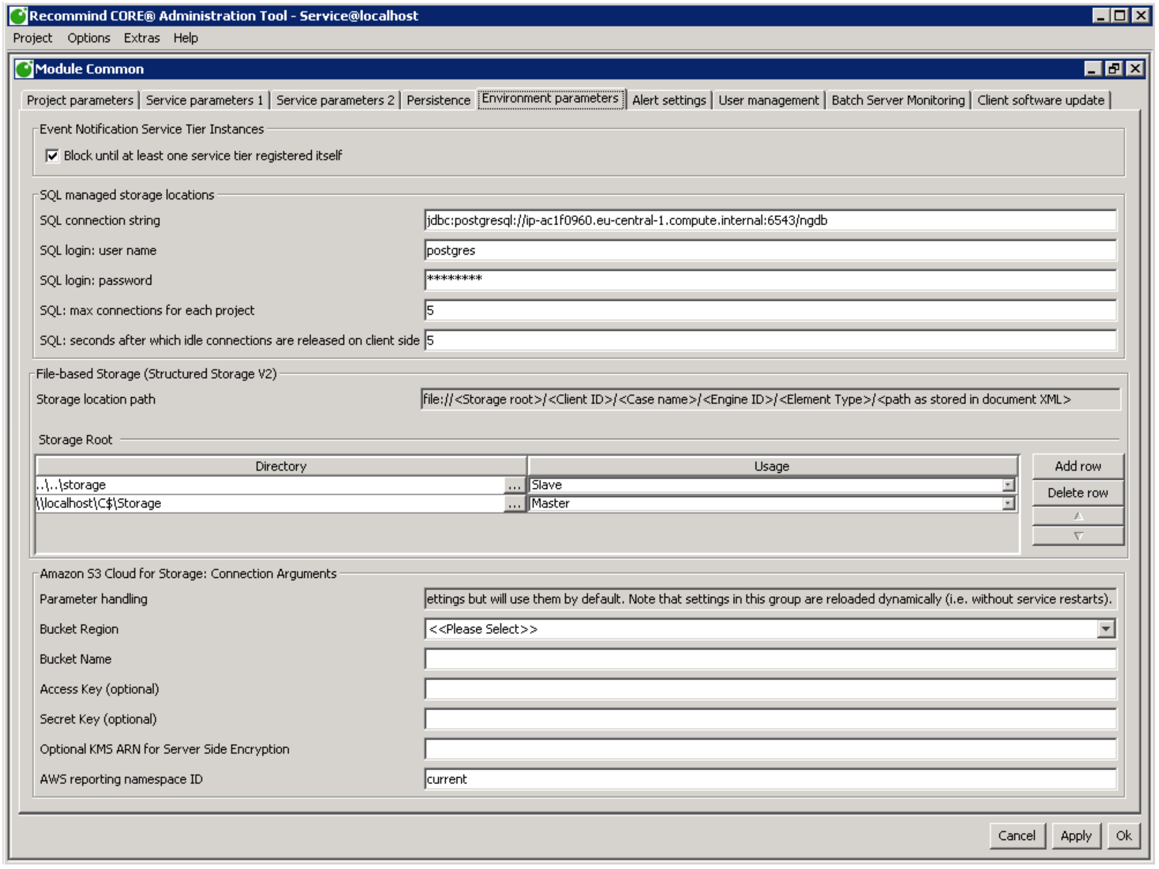
Required:

- The storage root folder has been created.

To configure the storage root on the master service:

1. On the master service host, run `mindserver.bat` in a command prompt.
2. From the **Project** menu, select **Service**.
3. Click the **Common** button.
4. In the **File-based Storage (Structured Storage V2)** section, you see the location path structure.

   Tip: If you don't see the **Storage Root** table below, enlarge the Window size.


Recommind CORE® Administration Tool - Service@localhost

Module Common

Environment parameters

Event Notification Service Tier Instances

☑ Block until at least one service tier registered itself

SQL managed storage locations

SQL connection string	jdbc:postgresql://ip-ac1f0960.eu-central-1.compute.internal:6543/ngdb
SQL login: user name	postgres
SQL login: password	********
SQL: max connections for each project	5
SQL: seconds after which idle connections are released on client side	5

File-based Storage (Structured Storage V2)

Storage location path	file://<Storage root>/<Client ID>/<Case name>/<Engine ID>/<Element Type>/<path as stored in document XML>

Storage Root

Directory	Usage
..\..\storage	Slave
\\localhost\C$\Storage	Master

Amazon S3 Cloud for Storage: Connection Arguments

Parameter handling	ettings but will use them by default. Note that settings in this group are reloaded dynamically (i.e. without service restarts).
Bucket Region	<<Please Select>>
Bucket Name	
Access Key (optional)	
Secret Key (optional)	
Optional KMS ARN for Server Side Encryption	
AWS reporting namespace ID	current

5. In the **Storage Root** table, change the **Master** according to your needs and click **Apply**.

   This is the storage root used for structured storage V2.

   Note: There may be a slave storage, too. This slave storage may result from updated projects or installation. It is used for the other, non-V2 structured storage type that may still be required. Leave this entry as it is.

6. If index engines have already been started, right-click the document models in CORE Administration and select **Reload index configuration**.

2.7.1.2.1 If the master storage is full

If there is little space left in the master storage, create a new one. Do not delete the old one, but make it slave storage.

Required:

» There is no active process writing to storages.

» The new storage root folder has been created.

If the master storage is full, proceed as follows:

1. In the **Storage Root** table, in the **Usage** column, select **Slave** for the full storage.

   Caution: Never delete full storage root rows! This will cut the access to the stored files.

2. Click **Add row**.

   This automatically adds a new **Master** row.
3. Enter the path to the new storage root in the **Directory** column.
4. Click **Apply**.
5. In CORE Administration right-click the document models of all index engines that use the master service storage root and select **Reload index configuration**.

Result: New files are stored to the new master storage root from now on. Existing files can still be read or deleted from all slave storages.

2.7.1.3 Configure Storage Root for a Project

If you want to use a master storage root for a single Axcelerate Ingestion and all its dependent matters, configure it in the Axcelerate Ingestion index engines.

Required:

» The storage root folder has been created.

To configure the storage root for a single project:

1. In CORE Administration, click an index engine and, from the **Actions** menu, select **Configure**.
2. Navigate to the **Native files > Storage > File-based Storage** node.
3. Deactivate the **Copy Settings from Master Service** check box.
4. In the **Storage Root** section, click the **Master** row and modify the path to the storage rot as needed.

   Note: There may be a slave storage, too. This slave storage may result from updated projects or installation. It is used for the other, non-V2 structured storage type that may still be required. Leave this entry as it is.

5. Click **OK**.
6. Do this for all Axcelerate Ingestion index engines.
7. Right-click the document model and select **Reload index configuration**.

To configure the storage root on the master service:

2.7.1.3.1 If the master storage is full

If there is little space left in the master storage, create a new one. Do not delete the old one, but make it slave storage.

Required:

- There is no active process writing to storages.
- The new storage root folder has been created.

If the master storage is full, proceed as follows:

1. In the **Storage Root** table, in the **Usage** column, select **Slave** for the full storage.

   Caution: Never delete full storage root rows! This will cut the access to the stored files.

2. Click **Add row**.

   This automatically adds a new **Master** row.
3. Enter the path to the new storage root in the **Directory** column.
4. Click **OK**.
5. Right-click the document model and select **Reload index configuration**.

Result: New files are stored to the new master storage root from now on. Existing files can still be read or deleted from all slave storages.

2.7.1.4 File Storage Paths for Structured Storage V2

The storage paths follow this pattern:

```
<storage root>\<client id>\<case name>\<unique
ID>\<storage file type>\<generated filepath including
a timestamp>
```

Example for a storage without case-wide sharing

```
\\myserver\c:\McMuster\MCMvsDE\minerva\DOCUMENT_
VIEWS\_default_\2016-10-18_17.36.19.907\0\0\0.pdf
```

This is a path to a document view file handled by index engine `sin-glemindserver.minerva`. The document view files storage handler does not use case-wide sharing.

Example for a storage enabled for case-wide sharing

```
\\myserver\c:\McMuster\MCMvsDE\casewidefile\NATIVE\_
default_\2016-10-18_17.36.19.907\0\0\0.doc
```

This is a path to a native file. By default, native file storage handlers use case-wide sharing. Instead of the index engine name, you see *casewidefile* as unique ID. This is the default name. The name is arbitrary.

2.7.1.4.1 How do I find a specific file?

In the document metadata, you only see the native file type and the generated filepath for the respective file, e.g., `NATIVE\_default_\2016-10-18_17.36.19.907\0\0\0.doc`.

To find the first part of the path look up these entries:

Storage root

In the index engine configuration, see **Native files > Storage > File-based Storage.**
The master in the **Storage Root** list shows either the master service settings or the storage root used for a single .Axcelerate Ingestion application with dependent matters.

Tip: If your project has slave storage roots, the file may be located in one of them, or in the master storage root.

Shared case name

In the index engine configuration, see **Common > Project parameters**

Shared Client Identifier

In the index engine configuration, see **Common > Project parameters**

Short Unique ID

In the index engine configuration, see **Native files > Storage > Storage handler > <storage handler name>**

Note: From the file extension, you cannot tell whether a file is compressed or not. Compression keeps the original extension, although the compressed file is a ZIP file. To access the file from the file system, copy it to a place outside the storage and replace the extension with *zip*.

2.7.1.5 Storage Backup

Required:

» Index engines are stopped.

To run the backup:

1. Backup the SQL database.
2. Backup the complete storage, i.e., master and slave storages.
3. Backup the other storages that do not use the master storage root, i.e., that do not use structured storage V2.

2.7.1.6 Configure Structured Storage V2

The configuration settings in CORE Administration are shown in alphabetical order.

2.7.1.6.1 Copy Settings from Master Service

Defines where to read and write files. The table can contain more than one storage root in order to handle the case of full drives. The document's XML will contain locations relative to the base URI (see above for a definition of the base URI).

Location: Index engine: **Native files > File-based Storage**

Allowed values:
- true
- false

Default value:
- true

2.7.1.6.2 Storage Root (if not copied from Master Service)

Defines where to read and write files. The table can contain more than one storage root in order to handle the case of full drives. The document's XML will contain locations relative to the base URI (see above for a definition of the base URI).

Location: Index engine: **Native files > File-based Storage**

Directory

The base directory for all storages of type **Structured Storage V2**. Must be accessible from all hosts

Allowed values: absolute path to storage root

Default value:
- none

Usage

Defines how to use the path. There can be at most one item with **Master**; it will be used to store new records.

Allowed values:
- Master (writable storage root)
- Slave (storage root from where items can be read or removed)

Default value:
- Master

2.7.2 Structured Storage

Note: Do not confound structured storage and structured storage V2. They are both file-based, but behave differently.

A structured storage is a folder on a file system. It is configured per storage handler.

The path to the structured storage folder is defined per index engine. Sharing between index engines is not possible.

Below the handler's root folder, there is an additional storage folder level for the structured storage. The name is generated from a time stamp (`yyyy-mm-dd_hh-mm-ss-ms`). This folder is created when storing the first document in the storage. The name is generated when the index engine starts. It uses the time the index engine starts. With each restart a new time stamp folder is generated. All files are saved in the configurable subfolder structure below this folder.

2.7.2.1 Configure Structured Storage

The configuration settings in CORE Administration are shown in alphabetical order.

2.7.2.1.1 Auto generate filename

If active, the stored files are numbered consecutively. Otherwise, a name must be provided.

Location: Index engine: **Native files > Storage > Storage handler > <storage handler instance name> > Structured storage with generated filenames**

Allowed values:
- true
- false

Default value:
- true

2.7.2.1.2 Folder size limit [in MB]

Enter the maximum size for first-level-folders in Megabyte.

Location: Index engine: **Native files > Storage > Storage handler > <storage handler instance name> > Structured storage with generated filenames**

Allowed values: 0 – 2147483647

Default value:
- None

2.7.2.1.3 Limit folder size

If active, the system checks for each first-level folder that the content does not grow beyond the configured limit.

Tip: You can use this feature to make sure you can store each folder on a medium of a certain capacity. For example, set a limit to not exceed the size of a DVD.

Location: Index engine: **Native files > Storage > Storage handler > <storage handler instance name> > Structured storage with generated filenames**

Allowed values:
- true
- false

Default value:
- false

2.7.2.1.4 Maximum number of files in every leaf directory

Maximum number of files that are stored in each leaf folder.

For example, a storage with two folder levels, maximum 100 subfolders and maximum 1000 files can store up to 100*100*1,000=10,000,000 files.

Tip: Some file systems become slow if there are two many files in a folder. This will impact your system performance. Choose these values with the limitations of your file system in mind.

Location: Index engine: **Native files > Storage > Storage handler > <storage handler instance name> > Structured storage with generated filenames**

Allowed values: 100 – 10,000

Default value:
- 1,000

2.7.2.1.5 Maximum number of subdirectories

Maximum number of subfolders that are created in every folder.

For example, when set to two levels with maximum 100 subfolders the storage handler creates the folders `<storagedir>\0\0\` to `<storagedir>\99\99\`. Together with **Maximum number of files in every leaf directory** this defines the maximum number of files the storage handler can store.

Location: Index engine: **Native files > Storage > Storage handler > <storage handler instance name> > Structured storage with generated filenames**

Allowed values: 1 – 1000

Default value:
- 100

2.7.2.1.6 Number of directory levels

Number of folder levels that are created below the storage directory.

For example a value of 2 will create the path `<storagedir>\0\0\0.doc` for the first document.

Location: Index engine: **Native files > Storage > Storage handler > <storage handler instance name> > Structured storage with generated filenames**

Allowed values: 0 – 50

Default value:
- 2

2.7.2.1.7 Storage location

This is the location where the files will be stored.

Location: Index engine: **Native files > Storage > Storage handler > <storage handler instance name> > Structured storage with generated filenames**

Allowed values: a relative or absolute path to a folder in operating system or UNC notation

Default value:
- path name generated during application creation

2.7.2.1.8 Store relative locations

Activate if relative locations shall be stored. The storage location is relative to the path stated above. For each storage location the part following the absolute base path is persisted, e.g. 0\0\0.doc

Location: Index engine: **Native files > Storage > Storage handler > <storage handler instance name> > Structured storage with generated filenames**

Allowed values:
- true
- false

Default value:
- true

3 Contact Us

About Recommind

Recommind provides the most accurate and automated enterprise search, automatic classification, and eDiscovery software available, giving organizations and their users the information they need when they need it.

Visit us at http://www.recommind.com.

Support

For support issues on Recommind products, visit the Recommind Ticketing System at https://rts.recommind.com.

Documentation

Find Recommind product documentation, Knowledge Base articles, and more information at the Recommind Customer Portal at https://supportkb.recommind.com. For login access to the site, contact your product support:

- For : SearchSupport@recommind.com
- For : eDiscoverySupport@recommind.com

The Recommind Documentation team is interested in your feedback.

For comments or questions about Recommind product documentation, contact us at documentation@recommind.com.

4 Terms of Use

Disclaimer

This document, as well as the products and services described in it, is furnished under license and may only be used or copied in accordance with the terms of the license. The information in this document is furnished for informational use only, is subject to change without notice, and should not be construed as a commitment by Recommind, Inc., including its affiliates and subsidiaries (collectively, "Recommind"). Recommind assumes no responsibility or liability for any errors or inaccuracies that may appear in this document or any software or services that may be provided in association with this document.

Except as permitted by such license, no part of this document may be reproduced, stored in a retrieval system, or transmitted in any form or by any means without the express written consent of Recommind. Information in this document is provided in connection with Recommind's products and services. No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted by this document.

EXCEPT AS PROVIDED IN RECOMMIND’S SOFTWARE LICENSE AGREEMENT OR SERVICES AGREEMENT FOR SUCH PRODUCTS OR SERVICES, RECOMMIND ASSUMES NO LIABILITY WHATSOEVER, AND RECOMMIND DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF RECOMMIND PRODUCTS OR SERVICES INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT. RECOMMIND MAKES NO WARRANTIES REGARDING THE COMPLETENESS OR ACCURACY OF ANY INFORMATION, NOR THAT THE PRODUCTS OR SERVICES WILL BE ERROR FREE, UNINTERRUPTED, OR SECURE. IN NO EVENT WILL RECOMMIND, THEIR DIRECTORS, EMPLOYEES, SHAREHOLDERS AND LICENSORS, BE LIABLE FOR ANY CONSEQUENTIAL, INCIDENTAL, INDIRECT, SPECIAL OR EXEMPLARY DAMAGES INCLUDING, BUT NOT LIMITED TO, LOSS OF ANTICIPATED PROFITS OR BENEFITS.

Recommind may make changes to specifications, and product and service descriptions at any time, without prior notice. Recommind's products may contain design defects or errors known as errata that may cause the product or service to deviate from published specifications. Current characterized errata are available on request. Whilst every effort has been made to ensure that the information and content within this document is accurate, up-to-date and reliable, Recommind cannot be held responsible for inaccuracies or errors. Recommind software, services and documentation have been developed and prepared with the appropriate degree of skill, expertise and care. While every effort has been made to ensure that this documentation contains the most up-to-date and accurate information available, Recommind accepts no responsibility for any damage that may be claimed by any user whatsoever for the specifications, errors or omissions in the use of the products, services and documentation.

Trademarks and Patents

Copyright