OPENTEXT

OpenText StreamServe 5.6.1

Microsoft® SQL Server®

Database Guidelines

Rev B

OpenText StreamServe 5.6.1 Microsoft® SQL Server® Database Guidelines
Rev B

Open Text SA
40 Avenue Monterey , Luxembourg, Luxembourg L-2163
Tel: 35 2 264566 1

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
Email: support@opentext.com
FTP: ftp://ftp.opentext.com
For more information, visit http://www.opentext.com

Disclaimer

No Warranties and Limitation of Liability

Every effort has been made to ensure the accuracy of the features and techniques presented in this publication. However, Open Text Corporation and its affiliates accept no responsibility and offer no warranty whether expressed or implied, for the accuracy of this publication.

Contents

About StreamServe repositories

This document provides an overview of how to install, maintain, and back up OpenText StreamServe repositories running on Microsoft® SQL Server®.

Intended audience

This document is intended for developers, for example OpenText consultants, who are also familiar with Microsoft SQL Server databases.

The document also contains some information that may be of interest for a database administrator, for example information about the tables with the highest growth rate and how jobs are deleted from the runtime repository.

Information about third party products

This document does not describe how to carry out configurations in third party products, for example in Microsoft SQL Server Management Studio. For such information, see *Microsoft SQL Server Online Books*.

In this section

Overview of StreamServe repositories

The following repositories are available:

- *StreamServe Enterprise Repository*
- *Runtime repository*
- *Document Broker Plus repository* (for Document Broker Plus)
- *Collector archive* (for StreamStudio Collector)
- *Web content repository* (for StreamStudio Composition Center)

All StreamServe repositories in the environment must use the same database vendor. For example, if the enterprise repository runs on SQL Server, the runtime repository, Collector archive, and web content repository must also run on SQL Server.

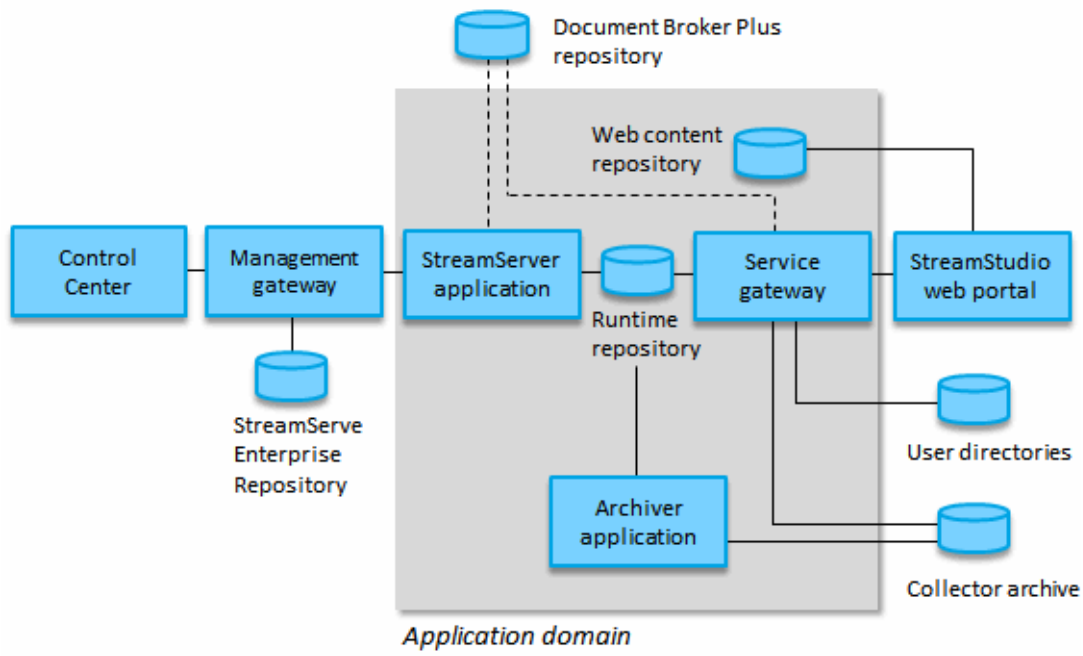


Figure 1 Overview, repositories and components

For detailed information about the components that interacts with the repositories, see *OpenText StreamServe Control Center User Guide*.

StreamServe Enterprise Repository

A StreamServe Enterprise Repository contains information about computers, StreamServe applications, and application domains for a company or organization. You use one central enterprise repository, installed on a specified database server.

If document types are used to categorize documents (for example, as invoices and orders), the enterprise repository is the master storage for these document types. The enterprise repository is also the master storage for any StreamStudio Composition Center template versions.

All communication with the enterprise repository is handled via a management gateway. For example, StreamServe Control Center communicates with the enterprise repository through a management gateway. Several management gateways can connect to the same enterprise repository.

Runtime repository

The runtime repository stores jobs and job related information in queues. The repository also contains security profiles and web access information for the StreamStudio web applications. Any persistent resources are also stored in the repository. For example, document definitions for StreamStudio Composition Center and exception rules to pause service-enabled Messages.

If you pause Messages, the Messages are stored in a separate queue called the Message storage. If you use the Document Broker Plus solution, the documents are stored in a Post-processing storage in the runtime repository.

Note: As an alternative to the default Post-processing storage in the runtime repository, a central Document Broker Plus repository can be used. For more information, see *Document Broker Plus repository* on page 8.

Each application domain requires a separate runtime repository. For example, you can use one repository for the applications in development, and another for the applications in production.

The runtime repository is shared by all the applications in the application domain and can be accessed by StreamServer, service gateway, Archiver, and Task Scheduler applications. When a StreamStudio user accesses the runtime repository, the requests and responses are sent through the service gateway.

Document Broker Plus repository

As an alternative to the default Post-processing storage in the runtime repository, a central Document Broker Plus repository can be used. This central repository can be linked to several application domains and can store documents generated from all these domains. Note that the central repository stores documents, but not jobs.

When the Document Broker Plus repository is linked to an application domain, the applications in this domain are automatically disconnected from the default Post-processing storage in the runtime repository and instead connected to the Document Broker Plus repository.

The Document Broker Plus repository can be accessed by StreamServer, service gateway, and Task Scheduler applications. When a DBQ Tool user searches for documents in the Document Broker Plus repository, all requests and responses are sent through the service gateway.

Collector archive

A Collector archive stores output documents and related metadata to be accessed from StreamStudio Collector. The Collector archive is optimized for searching and querying for documents.

Each application domain can access one single Collector archive. However, one Collector archive can be shared by several application domains.

An Archiver application transfers documents and related metadata from the runtime repository to the Collector archive according to schedules defined in Control Center. When the Collector user searches for documents in the Collector archive, all requests and responses are sent through the service gateway.

Web content repository

The web content repository is used by StreamStudio Composition Center for storing document definitions, resources, and rules during the document design phase.

When a document definition is approved, the document definition (together with its resources and rules), is set to published in the runtime repository, and is available to the StreamServer application that produces the document.

Composition Center is the only application that accesses the web content repository.

Tools for handling StreamServe repositories

When handling StreamServe repositories, we recommend that you use the tools, applications, and scripts provided by OpenText.

Priority order

Use the priority order below when deciding which tool to use:

1. Handle as much as possible using the configuration tools provided by OpenText, for example in StreamServe Design Center, StreamServe Control Center, and the StreamStudio web applications.
2. Use StreamServe Database Administration Tool only for tasks which cannot be performed in the configuration tools above. For information about this tool, see *OpenText StreamServe Database Administration Tool User Guide*.
3. Use the appropriate SQL Server tool, for example SQL Server Management Studio, only for tasks which cannot be performed using the tools provided by OpenText.

   When running external tools, you should primarily use the database scripts provided by OpenText. For example, the maintenance scripts referred to in this document.

Installing StreamServe repositories

This chapter contains information on how to configure a Microsoft SQL Server database for OpenText StreamServe repositories and how to create the repositories.

In this section

Hardware and software requirements

The requirements below applies for the database server where you install the runtime repository, Document Broker Plus repository, and the Collector archive.

In this section

Software requirements

Database

Microsoft SQL Server

- For supported versions, see *OpenText StreamServe Supported platforms and software Reference Guide*.
- For supported editions, see *Installing an SQL Server database* on page 14.

Device driver

- For Windows – Microsoft SQL Server Native Driver.
- For UNIX – Progress® DataDirect Connect® for ODBC

Hardware requirements

- 2 CPUs (minimum).
- 8 GB RAM (recommended minimum).
  We recommend you run 64-bit SQL Server.
- We recommend 2 physical hard drives (required to store the database file and the transaction log on separate disks).

  We recommend that you use one of the following alternatives:

  – Ultra SCSI (Small Computer System Interface) drives – either in an SAN (Storage Area Network) environment or with high quality controller cards. If write caching disk controllers are used, these must be specifically designed for use in a data critical transactional database management system (DBMS) environment.

  – SSD (Solid State Drive) disks, with the latest version of controller cards. To achieve fault tolerance, disk mirroring can be used.

Server configuration

- Always place data on striped disks. Note that the most important is the maximum number of I/O:s per second, rather than the disk throughput. We recommend that you use RAID 1+0 for data files.
- Use the RAID hardware. Optimize your RAID controller by setting the read/write cache. You can use Windows Performance Monitor to discover the read/write ratio on your disk sub-system and help set the controller to the optimal value.
- Identify your access patterns and learn how your user access data depends on these patterns, on the indexes, and on the read/write ratio. Based on this, you can decide whether to use multiple filegroups over multiple disks or just one single filegroup.
- Always store the transaction log on its own physical disk. Use the fastest available disks for the log files, and protect the log file disk.

Installing an SQL Server database

To create StreamServe repositories, there must be a database available.

- In a production environment, you should use the Standard or Enterprise edition of the database (Microsoft SQL Server Standard or Microsoft SQL Server Enterprise).

  In most scenarios, the Standard edition is sufficient. However, there are certain scenarios where the Enterprise edition is required. For example, if you use partitioning or in a High Availability scenario with more than two nodes. For comparisons between the different editions, see *Microsoft SQL Server Online Books*.
- In a development and testing environment, you can use the Express edition of the database (Microsoft SQL Server Express).

Database configuration

Configure the database according to the following:

- Enable TCP/IP.
- Specify Mixed Mode authentication.
- Specify a static TCP port number for accessing the database from all IP addresses.

  Note: Do not specify any dynamic TCP ports.
- Select a proper collation. See *Selecting collation* below.

You configure the database manually using the appropriate SQL Server tool, for example SQL Server Management Studio. For more information, see *Microsoft SQL Server Books Online*.

Snapshot isolation level

Read Committed Snapshot Isolation (RCSI) is used by default for the runtime repository and the Document Broker Plus repository. No further configuration is required.

Selecting collation

A collation specifies rules for how character strings are sorted and compared, based on the norms of particular languages and locales. The default setting is the collation from the computer where SQL Server is installed. For example, if you install SQL Server in Finland, a Finnish collation is suggested.

When you search for documents in StreamStudio Collector, the collation decides how the search result is sorted. If the collation is case sensitive, this also affects the search result.

To avoid conflicts in different StreamServe repositories, you must set the same collation for the database where you install the runtime repository and for the database where you install the Collector archive.

For information and recommendations, see *Microsoft SQL Server Books Online*.

Creating repositories

You use StreamServe Control Center to create the StreamServe repositories.

You can either create the repositories directly in Control Center, or you can generate the database scripts in Control Center and then run the scripts using an external tool. For example, if the company security policy prevents Control Center from connecting to the database, or if you want to have full traceability of the repository creation.

If Control Center is not available, you can carry out the corresponding procedures using the command line utilities. For more information, see *OpenText StreamServe Command line utilities Reference guide*.

Modifying repository users

In Control Center, default repository users are suggested. We recommend that you change these users before creating the repositories.

Related documentation

See *OpenText StreamServe Control Center User Guide* for information about how to:

- Configure the database settings for the StreamServe repositories and the connection settings to access the repositories.
- Create the StreamServe repositories directly in Control Center.
- Generate the database scripts in Control Center and then execute the scripts.

Checking repositories

After creating a repository, we recommend that you check the repository.

Check the log files for errors

After creating a repository in Control Center, you can check the short version of the log file in Control Center or you can open the full log file in a text editor from Control Center.

When executing the generated scripts using an external tool, you can use the logging features of this tool.

Perform a sanity test

You should always perform a sanity check to make sure the repository was created according to the configurations.

Adjusting repositories

An enterprise repository that you create in Control Center can be used for development and testing purposes and may also be sufficient for a production environment. If specific security or performance requirements apply, you may have to adjust the enterprise repository to fit the actual conditions.

The runtime repository, Document Broker Plus repository, and Collector archive that you create in Control Center are sufficient for development and testing purposes. However, before using these repositories in a production environment, you most likely have to adjust the repositories to fit the actual conditions. For example, configure number of disks, filegroups, set the size and growth parameters of the database file and the transaction log file, and index columns.

We recommend that you create the repositories using Control Center and then adjust the repositories using the appropriate SQL Server tool, for example SQL Server Management Studio.

In this section

Estimating the sizes for databases and transaction logs

To prevent the disk drives from running out of size, we recommend that you specify a maximum size for both the database file and for the transaction logs.

Underestimating the sizes of the database file and the transaction logs may result in automatic file growth and decreased performance.

In this section

Estimating the database size

You can estimate the size of the database file based on the sizes of the database tables.

You may be able to use the reporting function in the SQL Server tool to find out the actual size of each table. For example, the reporting function in SQL Server Management Studio. For more information, see *Microsoft SQL Server Online Books*.

If the reporting function is not available in your SQL Server tool, you can use a script provided by OpenText (`sp_spaceusedbytable.sql`) to find out the actual sizes. The tables sizes provided by `sp_spaceusedbytable` are in kilobytes (KB).

Note: A large table size does not necessary mean a large index size and vice versa. For example, in the runtime repository, the `BinaryObject` table has the biggest table size, but the `Part` table has biggest index size. If you have separated the data files into different filegroups, you must be aware of which files are the biggest ones.

To find out the size of tables and indexes (SQL Server Express)

1 Execute `sp_spaceusedbytable.sql` to create the stored procedure `sp_spaceusedbytable`. The script is located in:

- Windows:
  `<Base directory>\<Version>\root\config\database\<Version>\strsdata\sqlserver\maintenance`

  Where `<Base directory>` is the path specified for StreamServe Projects during the **Framework and StreamServer** installation.
  For example: `C:\ManagementGateway`
- UNIX:
  `<Project location>/config/database/<Version>/strsdata/sqlserver/maintenance`

  Where `<Project location>` is the Project location specified during the **Framework and StreamServer** installation.
  For example: `/opt/streamserve/root`

2 Run the following command to return the actual size of the tables and indexes:

```
EXEC sp_spaceusedbytable
```

Estimating the transaction log size

A general rule of thumb is that the transaction log size should be 20-25% of the database size.

However, the smaller the size of the database, the greater the size of the transaction log, and vice versa. For example, if the estimated database size is 10 MB, you set the size of the transaction log to 4-5 MB. If the estimated database size is over 500 MB, you set the size of the transaction log to 50 MB.

Creating new filegroups

If you are running a non-RAID disk set and the repository exceeds 100 MB, you may consider creating user-defined filegroups.

Even if you are using a RAID system, you may still benefit from using filegroups since maintenance and backups can be scheduled at different times for different filegroups. Filegroups also enable a more effective backup strategy for large databases.

If possible, we recommend one disk for each filegroup. If you use a filegroup with many files within, make sure to spread them across several disk drives. Performance will be improved as long as the hardware is sufficient enough to optimize reading and writing to multiple disk drives concurrently.

If indexes are placed into their own filegroup, the indexes and the data pages can be handled as separate read elements. If the associated filegroups are placed on separate physical disks, each disk can be read without interfering with the reading of the other. While an index is read in a sequential manner, the data can be accessed randomly, without the need for manipulating the physical arm of a hard drive back and forth from the index and the data. This can improve performance, and at the same time save the hardware.

Reducing the disk queue length

If the disk queue length (the number of data and log files in queue) on SQL Server has an average above three, we recommend that you reduce the number of files and filegroups.

You can use Windows Performance Monitor and the Disk Queue Length counter to determine the appropriate number of data and log files. Once you have made the changes, you must continue the monitoring to ensure that your disk I/O is optimized.

Placing log files on different disks

Place the SQL Server log files on different physical disks than the data files. As a minimum requirement, the disks containing the log files must have sufficient disk I/O performance, since logging is very a write-intensive action.

Indexing

For most tables, performance will be improved by indexing one or several of the columns. You should consider indexing columns that are used when filtering out a relatively small amount of rows from the tables.

Indexing columns improves the speed of the data retrieval operations, but at the cost of increased storage space and slower writes.

If a table is subjected to lots of inserts, increasing the number of indexes may have negative effect on the insert performance. You must take this into consideration when deciding whether to index or not.

You index columns using the appropriate SQL Server tool, for example SQL Server Management Studio. For more information, see *Microsoft SQL Server Books Online*.

In this section

Related topics

- *Running index maintenance* on page 39

Columns suitable for indexing

The columns most suitable for indexing are usually the ones with user defined metadata, configured in the Design Center Project.

You should adjust your indexes according to the queries being used. For example, the searches used in the Collector archive, or the DBQs/PPQs used when accessing the Document Broker Plus repository.

Consult the end-users for information about which document types and which metadata are most frequently used and index the corresponding columns.

Indexing columns in the runtime repository

If the runtime repository contains Message storages or Post-processing storages, you should consider indexing these storages.

Message storage

When a Message is paused by an exception rule, the Message is stored in the Message storage. The Messages can then be invoked via service calls, for example web service calls from Ad Hoc Correspondence or Correspondence Reviewer.

If a large amount of rows are stored in a Message storage, service call performance may be improved by indexing one or several of the columns. The columns most suitable for indexing are the ones for user defined metadata (that is, configured with the **Message** context in Design Center).

Example 1 Indexing a column in the Message storage

In this example, an index is added to the column `COL_Surname` in a Message storage called `cxMESSAGE_Letter`.

The column is indexed using the following command:

```
CREATE INDEX ixLCOL_Surname ON stc.cxMESSAGE_Letter(COL_Surname);
```

Post-processing storage

When Document Broker Plus is used, the documents are by default stored in a Post-processing storage. Metadata is used when searching for, retrieving, and post-processing the documents.

If a large amount of rows are stored in the storage, search performance may be improved by indexing one or several of the columns. The columns most suitable for indexing are the ones for user defined metadata (that is, configured with the **Post-processing** context in Design Center).

Example 2 Indexing a column in the Post-processing storage

In this example, an index is added to the column `COL_Surname` in the Post-processing storage called `cxPOST_Letter`.

The column is indexed using the following command:

```
CREATE INDEX ixLCOL_Surname ON stc.cxPOST_Letter(COL_Surname);
```

Indexing columns in the Document Broker Plus repository

When Document Broker Plus is used, the documents can be stored in a central Document Broker Plus repository instead of in the default Post-processing storage in the runtime repository.

Metadata is used when searching for, retrieving, and post-processing documents in the Document Broker Plus repository.

If a large amount of rows are stored in the repository, search performance may be improved by indexing one or several of the columns. The columns most suitable for indexing are the ones for user defined metadata (that is, the ones configured with the **Post-processing** context in Design Center).

Example 3 Indexing metadata in the Document Broker Plus repository

In this example, an index is added to the column `COL_Surname` in the Document Broker Plus repository called `cxPOST_Letter`.

The column is indexed using the following command:

```
CREATE INDEX ixLCOL_Surname ON stc.cxPOST_Letter(COL_Surname);
```

Indexing columns in the Collector archive

When an end-user searches for documents in StreamStudio Collector, metadata is used as search criteria.

If a large amount of rows are stored in the metadata tables, search performance may be improved by indexing the columns for this metadata. The columns most suitable for indexing are the ones for user defined metadata (configured with the **Archive** context in Design Center).

Example 4 Indexing metadata

In this example, an index is added to the metadata column `Customer Name` for the document type table `META_Letters`.

```
CREATE INDEX ixLCustomerName ON stc.META_Letters("Customer
Name");
```

Using ODBC script functions with 32-bit StreamServe and 64-bit SQL Server

You can install and run OpenText StreamServe in either 32-bit mode or 64-bit mode.

Why run 32-bit StreamServe on 64-bit Windows?

If you run the 64-bit (x64) version of Microsoft Windows and you want to use a StreamServe feature for which there is no 64-bit support, you must run StreamServe in 32-bit mode. For example, to use Document Broker for FastObjects.

Note: For detailed information about supported features when running the StreamServe software in 32-bit and 64-bit mode, see *OpenText StreamServe Supported platforms and software Reference Guide*.

StreamServe ODBC script functions

In order for StreamServe ODBC script functions to work properly with 64-bit versions of Microsoft SQL Server, you must manually define a 32-bit ODBC data source (DSN) on the StreamServe host machine.

Note: As an alternative, you can use the StreamServe DataDirect driver.

To manually add a 32-bit DSN on 64-bit Windows

1. Open `Odbcad32.exe`, located in `C:\Windows\SysWoW64`. The 32-bit version of the Microsoft ODBC Data Source Administrator opens.
2. In the Data Source Administrator tool, create the DSN, pointing to the 64-bit SQL Server you want to use. Remember to test the DSN.
3. Use the new DSN in all ODBC-related scripts.

Uninstalling repositories

When you uninstall the StreamServe software, you can choose to remove the components. This uninstalls the files and services for the repositories, but does not delete the repositories or the repository users.

Under certain circumstances, you may want to drop a repository. For example, if you upgrade to a later StreamServe release, and want to create a new enterprise repository instead of upgrading the existing one.

When you drop a repository, all data within the repository is lost.

Prerequisites

Before dropping a repository, the following must be fulfilled:

- There must be no active sessions running against the repository. You can use the Activity Monitor in SQL Server Management Studio to find out if there are any active sessions. For more information, see *Microsoft SQL Server Online Books*.

To drop a repository

- You can drop the repository directly from SQL Server Management Studio. For more information, see the *Microsoft SQL Server Online Books*.

Maintaining StreamServe repositories

General maintenance tasks

You maintain the OpenText StreamServe repositories using standard Microsoft maintenance procedures. For recommended maintenance activities, including error handling and performance improvement strategies, see *Microsoft SQL Server Books Online*.

OpenText StreamServe specific maintenance tasks

If you experience performance bottlenecks related to the database, you can tune the maintenance tasks below to optimize your environment.

Note: If you do not experience any performance bottlenecks, you should keep the default setup and the default scheduling.

- **Job, Message, and document deletion**
  You can optimize the way in which expired jobs, Messages, and documents are deleted from the runtime repository and how expired documents are deleted from the Document Broker Plus repository.
- **Job status update**
  To be deleted from the runtime repository, the status of a top job must be set to completed. You can optimize the way in which the statuses of top jobs are updated.
- **Index maintenance**
  When data is modified in indexed table columns, indexes become fragmented and the speed of the data retrieval operations deteriorates. To preserve performance, you can set up an index maintenance plan for your indexed repositories.

Running the maintenance tasks charges the database and may affect database performance.

This chapter contains some recommendations regarding setting up and scheduling the maintenance tasks. Rather than being absolute truths, these recommendations are to be considered as starting points for trying out the most optimal settings for your specific job setup and environment.

Tuning of maintenance tasks is a continuous assignment. You must monitor the jobs in the runtime repository to ensure that the tasks are correctly scheduled and that the database does not grow over time.

You should always schedule the maintenance tasks in a way that ensures minimum impact on any other database activities.

Maintaining StreamServe repositories

In this section

Top jobs, input jobs, and output jobs

This section describes the concept of top jobs. You must be aware of this concept when scheduling the tasks for job deletion and job status update.

Top jobs, input jobs and output jobs

StreamServe jobs are divided into top jobs, input jobs, and output jobs.

A top job is created when a StreamServer application receives input, for example from an ERP system. Each top job generates one input job processed by the StreamServer application. When the application processes an input job, it produces one or more output jobs.

This means a top job generates one input job, and the input job can be divided into several output jobs. All jobs that belong to the same top job are identified by the same Tracker ID.

A top job is completed when all included input and output jobs are completed.

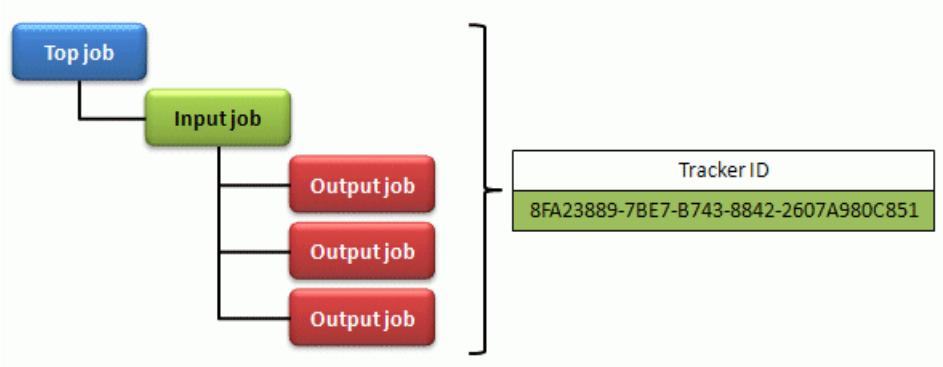


Figure 2 Top job, input job, and output jobs

Deleting expired jobs from the runtime repository

Expired top jobs and job status information must be deleted from the runtime repository. The job deletion also includes expired Messages/documents in Message/Post-processing storages in the runtime repository.

If a central Document Broker Plus repository is used, any expired documents in this repository are also included in the job deletion.

Two ways of deleting expired jobs

In this documentation, the following ways of deleting expired jobs are described:

- Schedule the StreamServer applications to delete jobs (default).
- Schedule one or several Task Scheduler applications to delete jobs.

  Note: When using Task Scheduler applications, the deletion of Messages and documents are not included in the job deletion. Instead you configure separate deletion tasks for jobs, Messages, and documents.

Low volume installation with a single StreamServer application

By default, each StreamServer application is set up to perform data deletion tasks every hour. Successfully completed job data is kept in the repository for 5 days. Failed job data is kept for a month.

The deletion is regulated in the `repositorymanager.xml` configuration file for each StreamServer application. This deletion is suitable for less complex, low volume installations with a single StreamServer application. To optimize the delete performance, you may have to update the schedule.

High volume installations with several StreamServer applications

For medium to high volume installations, where several StreamServer applications try to follow the instructions in `repositorymanager.xml` at the same time, we recommend that you let one or several StreamServe Task Scheduler applications delete the expired jobs, Messages, and documents instead. Using Task Scheduler applications may improve performance by centralizing the status updates and enabling more complex and controlled schedules.

In this section

Job deletion process

This section describes the order in which expired top jobs and expired Messages/documents in Message/Post-processing storages are deleted. If a central Document Broker Plus repository is used, expired documents in this repository are also included in the deletion.

If you schedule StreamServer applications to delete expired objects, the stored procedures are automatically executed in the order below. If you schedule Task Scheduler applications to delete expired objects, you should make sure that the tasks are executed as described below.

Step 1 – Expire jobs

In Design Center, expiry times for successfully processed and failed top jobs are configured. The jobs remains in the queues for the expiry time and are then ready to be handled by the job deletion process.

Step 2 – Delete expired Messages and documents

The `pc_deleteexpireddocabstraction` stored procedure deletes any expired Messages/documents from Message/Post-processing storages.

If a central Document Broker Plus repository is used, the stored procedure deletes any expired documents from this repository instead of from the Post-processing storage.

Step 3 – Mark jobs for deletion

The `pq_deletemarkexpiredtopparts` stored procedure searches for expired top jobs that are ready to be deleted and marks these jobs for deletion.

Step 4 – Delete marked jobs

The `pq_pickdeleteevent` stored procedure deletes any expired top jobs marked for deletion. Only top jobs without any related Messages/documents in Message/Post-processing storages can be deleted.

Note: An archiving job cannot be deleted unless all related documents are first transferred to the Collector archive.

Prerequisites and recommendations

For optimal performance when deleting jobs and job status information, you must consider both the Design Center configurations and the scheduling of the job deletion.

In this section

Design Center configurations

- We strongly recommend that you configure the input and output queues to store both successful and failed jobs (**Store information and job** should be selected for successful and failed jobs in the **Manage Queues** dialog box).

  Note: Both information and jobs are always initially stored in the runtime repository. If the options to store nothing or store information only are used, expired input and output jobs are continuously deleted from the queues. This results in an increased number of delete transactions and a decreased delete performance.

- To delete successful jobs, the deletion process must be allowed to delete successful jobs (**Delete successful jobs** must be enabled in the **Configure Platform** dialog box).

  We recommend a short expiry time for successfully processed jobs (configured in the **Allow deletion after** setting in the **Configure Platform** dialog box). As a rule of thumb, you should keep the number of top jobs marked for deletion to a minimum. This has a great impact on the delete performance, especially if each top job generates one single output job. However, you must also consider how long the customer wants to access successful jobs in the queues.

- To delete failed jobs, the job deletion process must also be allowed to delete failed jobs (**Delete failed jobs** must be enabled in the **Configure Platform** dialog box).

  An expiry time for the failed top jobs must be configured (the **Allow deletion after** setting in the **Configure Platform** dialog box). You must consider how long the customer wants to access the failed jobs in the queues.

- If possible, avoid using notifications (**Use notifications** should be cleared in the **Configure Platform** dialog box).

For more information about the options, see *OpenText StreamServe Design Center User Guide*.

Job deletion schedule

We recommend that you schedule the job deletion in the following way:

- Primarily, you should run the job deletion at a time period when the StreamServer applications are idle or the job throughput is low. For example, after scheduled batch jobs or when the average CPU usage for the database falls below a specified value and remains below this level for a specified time period.

  Note: You must make sure that the available time period is longer than the time interval required to complete the job deletion after a peak load.

- If the available time periods are too short or if the workload is continuous, you should start the job deletions at an available time period and then schedule the remaining deletions in the following way:
  - In general, schedule a continuous job deletion with a high frequency.
  - **Exception** – If most of your top jobs generate many output jobs, you may receive a better delete performance by running the job deletion less frequent or (if possible) after each top job is successfully completed.
- If a large number of jobs are deleted in each job deletion, any defragmentation or re-indexing should run after the job deletion in order to avoid index fragmentation. See *Running index maintenance* on page 39.

Scheduling StreamServer applications to delete jobs

By default, the deletion is scheduled in the `repositorymanager.xml` file for each StreamServer application.

Every 20 minutes, each StreamServer application searches for and marks any expired top jobs that are ready to be deleted. Every 60 minutes, each StreamServer application triggers the deletion of marked jobs, expired Messages, and expired documents.

You can edit the default schedules to fit the actual conditions.

Location of repositorymanager.xml

You can either edit the configuration in:

- The template file (used for all new StreamServer applications), located in:

  Windows:
  `<StreamServe installation>\Applications\Management\<Version>\etc\config\<Version>\STRSCS\`

  UNIX:
  `<StreamServe installation>/applications/managementgateway/etc/config/<Version>/STRSCS/`

  The changes that you make to a template file only apply for new applications, created after the changes are done. They do not apply for already created applications.

- The configuration file for a specific StreamServer application, located in:

  Windows:
  `<Base directory>\applications\<Application>\<Layer>`

  UNIX:
  `<Project location>/applications/<Application>/<Layer>`

Prerequisites and recommendations

- See *Prerequisites and recommendations* on page 30

To edit the schedule for job deletion

1. Open the `repositorymanager.xml` file.
2. Edit the following lines:

```
<deleteevent schedule="T II * * MH * * 60" />
<!-- heartbeatevent schedule="T II * * MH * * 20"
interval="PT20M" / -->
<deletemarkevents>
   <toppart use="default" schedule="T II * * MH * * 20" />
</deletemarkevents>
```

For syntax, see *Appendix A - Time scheduling syntax* on page 59.

Scheduling Task Schedulers to delete jobs

You can add a StreamServe Task Scheduler application and configure tasks for deleting expired top jobs, expired Messages, and expired documents. To ensure deletion even if the Task Scheduler goes down, you can add several Task Scheduler applications.

Note: When using Task Scheduler applications, you configure separate deletion tasks for jobs, Messages, and documents.

Prerequisites and recommendations

- See *Job deletion process* on page 29
- See *Prerequisites and recommendations* on page 30

Post requisites

- We recommend that you comment out the corresponding lines for job deletion in the `repositorymanager.xml` files for the StreamServer applications, see *Scheduling StreamServer applications to delete jobs* on page 32.

To schedule a Task Scheduler to delete jobs, Messages, and documents

1. In Control Center, right-click the application domain and select **New Application**. The New Application dialog box opens.
2. Configure the application properties for the new Task Scheduler.

   Note: You cannot use the name `Task Scheduler` if you run an application on a Windows host, since this name is used by a Windows service.

3. Click **OK**. The Configuration dialog box opens.
4. Select the **(Item list)** field and click the button to the right of the field. The Service Configuration dialog box opens.
5. Add and configure the required tasks:

Delete expired jobs	Select to mark expired top jobs for deletion and delete these expired jobs from the runtime repository at the scheduled interval. **Threads** – The maximum number of threads to be used when deleting expired jobs marked for deletion. Several treads enables the application to delete several jobs in parallel. Note that only the first thread searches for and marks jobs for deletion. **Note:** Each thread consumes system resources. Several threads may result in deadlocks.
Delete expired Messages	Select to delete expired Messages from Message storages in the runtime repository at the scheduled interval.

Delete expired documents	Select to delete expired documents from Post-processing storages in the runtime repository or from the Document Broker Plus repository at the scheduled interval.

For more information, see *OpenText StreamServe Control Center User Guide*.

Updating top job statuses in the runtime repository

A successful top job cannot be deleted from the runtime repository unless:

- The status of the top job is updated to completed.
- There are no documents waiting for being archived.

  Note: If the **Documents from successful jobs** setting is selected on the **Archiving** tab in Design Center, documents cannot be archived until the top job status is updated to completed.

A failed top job cannot be deleted from the runtime repository unless the status is updated to aborted.

Two ways of updating job status

In this documentation, the following ways of updating top job statuses are described:

- Schedule the StreamServer applications to update statuses (default).
- Schedule one or several Task Scheduler applications to update statuses.

Low volume installation with a single StreamServer application

By default, all StreamServer applications update top job statuses every minute. The ability to update statuses and the time interval are regulated in the `repositorymanager.xml` configuration file for each StreamServer application.

This way of updating top job statuses is suitable for less complex, low volume installations with a single StreamServer application. To optimize the performance, you may have to update the schedule.

High volume installations with several StreamServer applications

For medium to high volume installations, where several StreamServer applications try to follow the instructions in `repositorymanager.xml` at the same time, we recommend that you let one or several StreamServe Task Scheduler applications update the job statuses instead. Using Task Scheduler applications may improve performance by centralizing the status updates and enabling more complex and controlled schedules.

Exception – If you want the same StreamServer application that created a top job to complete the top job, the top job statuses must be updated by the StreamServer applications as described in *Low volume installation with a single StreamServer application* above (that is, not by Task Scheduler applications). For example, in a notification scenario, where both the `beginbatch` and the `endbatch` notifications are required by the same StreamServer application. To enable this, you must add a `private="true"` attribute to the `repositorymanager.xml` files for the StreamServer applications.

In this section

Job status update process

Both steps below must be completed to fulfill the job status update process. That is, you must make sure that the status is both updated *and* reported.

Step 1 – Update status

When updating a status, the `pq_pickreadystatusevent` stored procedure is used to check and update the status of the top jobs.

A top job is completed when all included sub jobs are completed. The top job is aborted if any of the included sub jobs has failed the maximum number of retries.

Step 2 – Report status

When the top status is updated, one application in the application domain must report the status. For example, in the application log file. After being reported, the status is consumed and is no longer available to the other applications.

This means that one application can check and update the status, and another application can report and consume the status. Of these two operations, the status update is the most resource intensive.

If StreamServe Status Messenger is used to generate reports in the application domain, you must make sure that the application that runs the Design Center Project for Status Messenger is the only one that reports the status.

Recommendations

We recommend that you schedule the job status update in the following way:

- The job status update task must be scheduled in relation to the job deletion tasks and any archiving tasks. The job status update task should run more frequent than these tasks. See *Deleting expired jobs from the runtime repository* on page 28.
- In general, the following applies:
  - If most of the top jobs generate one or a few output jobs, we recommend a continuous job status update with a high frequency.
  - If most of the top jobs generate many output jobs, we recommend a job status update with a lower frequency.
- Since updating statuses is more resource intensive than reporting statuses, you should let only one application (or, if you use several applications for redundancy, as few as possible) perform the update operation.

Scheduling StreamServer applications to update job statuses

By default, every StreamServer application updates top job statuses every minute. The ability to update the statuses and the time interval are regulated in the `repositorymanager.xml` configuration file for each StreamServer application.

You can edit the default schedule to fit the actual conditions.

Location of repositorymanager.xml

You can either edit the configuration in:

- The template file (used for all new StreamServer applications), located in:

  Windows:
  <StreamServe installation>\Applications\Management*<Version>*\etc\config*<Version>*\STRSCS\

  UNIX:
  <StreamServe installation>/applications/managementgateway/etc/config/*<Version>*/STRSCS/

  The changes that you make to a template file only apply for new applications, created after the changes are done. They do not apply for already created applications.

- The configuration file for a specific StreamServer application, located in:

  Windows:
  <Base directory>\applications*<Application>**<Layer>*

  UNIX:
  <Project location>/applications/*<Application>*/*<Layer>*

Prerequisites and recommendations

- See *Job status update process* on page 36
- See *Recommendations* on page 36

To edit the schedule for updating job statuses

1. Open the `repositorymanager.xml` file.
2. Edit the following line:

   ```
   <readystatusevent schedule="T II * * MH * * 1" update="true"
   report="true" />
   ```

   Note: If Status Messenger is used to generate status reports, you must keep `report="true"` for the StreamServer application that runs the Status Messenger Project and set `report="false"` for all other applications.

   For syntax, see *Appendix A - Time scheduling syntax* on page 59.
3. Optional – To enable a top job to be completed by the same StreamServer application that created the top job, add the `private="true"` attribute:

   ```
   <readystatusevent schedule="T II * * MH * * 1" update="true"
   report="true" private="true" />
   ```

Scheduling Task Schedulers to update job statuses

You can add a StreamServe Task Scheduler application and configure a task to update the statuses of top jobs. To ensure job status updates even if the Task Scheduler application goes down, you can add several applications.

Note: You must make sure that at least one application in the application domain (StreamServer or Task Scheduler) is configured to report status.

Prerequisites and recommendations

- See *Job status update process* on page 36
- See *Recommendations* on page 36

Post requisites

You must disable the status update in the `repositorymanager.xml` files for the StreamServer applications. You can either comment out the corresponding line or you can change to `update="false"` but keep `report="true"`.
See *Scheduling StreamServer applications to update job statuses* on page 37.

Note: As an alternative, you can override the status update and the status report in the configuration file by using the startup argument `-statusevent 0`. This argument must be applied on each Design Center Project for which you want to override the settings. See *OpenText StreamServe Startup Argument Reference Guide*.

If Status Messenger is used to generate status reports, you must keep the configuration in `repositorymanager.xml` (with `update="false"` and `report="true"`) for the StreamServer application that runs the Status Messenger Project.

To schedule a Task Scheduler to update job statuses

1. In Control Center, right-click the application domain and select **New Application**. The New Application dialog box opens.
2. Configure the application properties for the new Task Scheduler.

   Note: You cannot use the name `Task Scheduler` if you run an application on a Windows host, since this name is used by a Windows service.

3. Click **OK**. The Configuration dialog box opens.
4. Select the **(Item list)** field and click the button to the right of the field.
5. Add and configure the task for updating job status:

Update job status	Select to update and report the statuses of top jobs according to the scheduled interval. **Update status** – Select to let the application update the statuses of top jobs. **Report status** – Select to let the application report updated statuses. **Note:** If Status Messenger is used to generate status reports, you must keep the **Report status** option clear.

Running index maintenance

When data is modified in indexed table columns, indexes become fragmented and the speed of the data retrieval operations deteriorates. To preserve performance, you must set up an index maintenance plan for your indexed repositories. For example, for the runtime repository, the Document Broker Plus repository, and the Collector archive.

The index maintenance plan must include the following:

- Update of repository statistics.
- Defragmentation and rebuilding of indexes.
- Re-optimization of queries by recompiling the underlying stored procedures.

Index maintenance via SQL Server (recommended)

We recommend that you use the index maintenance functionality available for SQL Server when setting up your index maintenance plan. For example, you can use the Microsoft Maintenance Plan Wizard.

OpenText provides an index maintenance package which you can use when setting up index maintenance via Microsoft SQL Server Agent.

Index maintenance via StreamServe Task Scheduler

If you use the SQL Server Express edition in a testing or development environment, SQL Server Agent is not available. You can then use a StreamServe Task Scheduler to run index maintenance on a runtime or Document Broker Plus repository.

We do not recommend that you run index maintenance via a Task Scheduler in a production environment.

If you already use a Task Scheduler for index maintenance in your production environment, we recommend that you disable this index maintenance and instead create an index maintenance plan via SQL Server.

In this section

- *Most fragmented tables* on page 40
- *Recommendations* on page 40
- *Running index maintenance via SQL Server Agent* on page 42
- *Running index maintenance via a Task Scheduler* on page 44

Related topics

- *Indexing* on page 20

Most fragmented tables

Repository	Most fragmented tables (according to OpenText tests)
Runtime repository	• `FixedMetaData` • `QStatusReport` • `TrackerMap` • `relPartResourceStore` • `ResourceStore` • `BinaryObject` • `BlobInfo` • `Part` • `VariableMetaData` • `cxMESSAGE_<DocumentType>` **Note:** Only if Messages are paused in Message storages. • `cxPOST_<DocumentType>` **Note:** Only if documents are stored in Post-processing storages.
Document Broker Plus repository	• `cxPOST_<DocumentType>`
Collector archive	• `Meta_<Document type>`

Recommendations

Maintenance plan recommendations

- We recommend that you use the index maintenance functionality available for SQL Server when setting up your index maintenance plan.
- When setting up the index maintenance plan, you must consider the overall amount and frequency of data inserts and deletes.

  The required index maintenance depends on the type of scenario you are running. For example, a batch scenario (where multiple input files are sent to the StreamServer applications in batches) needs different maintenance compared to an Ad Hoc Correspondence on-demand scenario (where multiple web service requests quickly populates previously empty tables).

  The type of repository also affects the index maintenance. For example, a runtime repository needs different maintenance compared to a Collector archive. For the runtime repository, data is in general highly volatile, which means frequently modified tables with fragmentation as a result.
- You must continuously monitor table sizes, index usage, and the disk I/O and update your index maintenance plan accordingly.

Scheduling recommendations

Running index maintenance charges the database and may affect the database performance. We therefore recommend that you schedule the index maintenance in the following way:

- When the StreamServer application is idle or when the job throughput is low. For example, after scheduled batch jobs or when the average CPU usage for the database falls below a specified value and remains below this level for a specified time period.
- If the workload is continuous, without any periods of low throughput, the indexes must be rebuilt online (that is, while other processes are accessing a table). This is only supported for SQL Server Enterprise Edition.
- After deletion of expired jobs. In general, if you use a continuous job deletion, you can run the index maintenance less frequent than the job deletion (but still when the database activity on the runtime repository is as low as possible). See *Deleting expired jobs from the runtime repository* on page 28.

General recommendations

- Make use of the automatic SQL Server processes for creating and updating statistics. For example, in most cases you will benefit from using the option for automatically creating and updating statistics.

  Note: Do not rely entirely on automatic processes. For example, additional manual updates of the statistics might also be required.

- If you experience deteriorated query performance and you do not have time for a thorough analysis, you can try performing an update statistics only for the involved tables or indexes. If this does not enhance performance, you can try updating distinct statistics with full scan.

  Note: Watch out for any differences between actual and estimated row-counts in the actual execution plan. If there are considerable differences, this could be an indication of non-conforming statistics.

- If required, rebuild fragmented indexes. During the rebuild, indexed-linked statistics are automatically updated with full scan.

  Note: Do not update index-linked statistics immediately after rebuilding indexes since this might result in degraded quality of the statistics.

Running index maintenance via SQL Server Agent

Via SQL Server Agent, you can run an OpenText index maintenance package on the runtime repository, Document Broker Plus repository, and Collector archive.

Index maintenance process

The steps below are included in the OpenText index maintenance package. The maintenance plan runs daily at a time that you specify when running the batch file. For each step, there are three retries (with an interval of one minute).

Step 1 – Update statistics

The statistics for any modified table columns is updated.

Step 2 – Rebuild indexes

The indexes are defragmented and rebuilt. Note that rebuilding indexes online is only supported for SQL Server Enterprise Edition.

Step 3 – Drop and create the index

The indexes are dropped and recreated.

Step 4 – Recompile all stored procedures

Queries (based on stored procedures) are optimized only when they are compiled. As a repository changes, for example when an index is rebuilt, the compiled stored procedures lose efficiency. By recompiling all stored procedures based on the updated statistics, the queries are re-optimized.

Scheduling SQL Server Agent to run index maintenance

The maintenance package (`Maintplan.bat`) is located in:

`<Base directory>\<Version>\root\config\database\<Version>\strsdata\sqlserver\maintenance`

Prerequisites and recommendations

- See *Recommendations* on page 40.
- Make sure no other index maintenance runs on the repository. For example, via a Task Scheduler or via the Microsoft Maintenance Plan Wizard.
- SQL Server Agent must be available and running.
- The sqlcmd utility must be available.

To set up SQL Server Agent to run index maintenance

1. Double-click `Maintplan.bat` to run the file.
2. When prompted for, enter the required parameters. For example, `sysadmin`, `dbname`, `scheduled time`, etc.

   Note: When you are prompted to enter the path and name to the Microsoft SQL Server Query File, you must enter the path and name to `Maintplan.sql`. For example:
   `C:\ManagementGateway\5.6.1\root\config\database\5.61.0\strsdata\sqlserver\maintenance\Maintplan.sql`

Post requisites

- When the maintenance plan is triggered, you can check the status in the log files to verify that the maintenance task works as expected. For example, in the `C:\maintstep.log` file on the computer where SQL Server runs.

Running index maintenance via a Task Scheduler

Only for testing or development environments where SQL Server Agent is not available.

Via a Task Scheduler, you can run index maintenance on a runtime or Document Broker Plus repository. To ensure that indexes are maintained even if the Task Scheduler application goes down, you can add several applications.

We do not recommend that you run index maintenance via Task Schedulers in a production environment.

If you already use Task Schedulers for index maintenance in your production environment, we recommend that you disable this index maintenance and instead create an index maintenance plan using the SQL Server functionality.

Index maintenance process

The index maintenance process includes rebuilding indexes, updating statistics, and recompiling stored procedures.

Scheduling a Task Scheduler to run index maintenance

Prerequisites and recommendations

- See *Recommendations* on page 40.
- Make sure no other index maintenance runs on the repository. For example, via SQL Server Agent or via the Microsoft Maintenance Plan Wizard.
- The executing user must have permissions at least corresponding to the `db_owner` database role for the repository aimed for the maintenance.

Post requisites

- In the task configuration, you assign a connection profile to access the repository. Before you can run the maintenance, you must edit the default connection profile to fit the actual conditions. See *Creating a connection profile* on page 45.
- You can check the Platform log in Control Center to verify that the maintenance task works as expected.

To schedule a Task Scheduler to run index maintenance

1. In Control Center, right-click the application domain and select **New Application**. The New Application dialog box opens.
2. From the **Application host** drop-down list, select the host.
3. Configure the application properties for the new Task Scheduler.

   Note: You cannot use the name `Task Scheduler` if you run an application on a Windows host, since this name is used by a Windows service.

4. Click **OK**. The Configuration dialog box opens.
5. Select the **(Item list)** field and click the button to the right of the field. The Service Configuration dialog box opens.

6 Add and configure the required task:

Run database maintenance	Select to perform database maintenance on a specified repository according to the specified schedule. **Connection profile** – The connection profile to use. The default profile connects to the runtime repository in the Task Scheduler application domain.

For more information, see *OpenText StreamServe Control Center User Guide*.

Creating a connection profile

A connection profile describes which runtime repository to connect to and as which user you will connect.

You must edit the default connection profile (`maintenance.xml`) in the `securityprofiles` folder in the working directory of the Task Scheduler application to fit the actual conditions.

```
...
<securityprofiles xmlns="..." owner="...">
  <connectionprofile type="..." name="...">
    <configuration>
      <sql xmlns="...">
        <vendor>sqlserver</vendor>
        <hoststring>myhost</hoststring>
        <port>1433</port>
        <dbname>StrsData</dbname>
        ...
        ...
      </sql>
    </configuration>
  </connectionprofile>
  <authenticationprofile type="..." name="...">
    <configuration>
      <simple xmlns="...">
        <name>User</name>
        <password>MyPassword</password>
      </simple>
    </configuration>
  </authenticationprofile>
</securityprofiles>
```

Figure 3 Connection information (bold) in connection profile

To create a connection profile

1. Open `maintenance.xml`, located in:
   - Windows:
     `<Base directory>\<Version>\root\applications\<Task Scheduler>\wd\securityprofiles`
   - UNIX:
     `<Project location>/applications/<Task Scheduler>/wd/securityprofiles`
2. Edit the connection information. The user that you assign must have sufficient permissions to perform database maintenance.
3. Start the Task Scheduler application.

Deleting expired documents from the Collector archive

By default, archived documents remain in the Collector archive for an unlimited time or until being manually removed by a database administrator.

If required, the `SetOutputExpireDate` scripting function can be used in the Design Center Project to set expiring dates on documents. The function creates an `ExpiringDateTime` for each document by adding a date offset to the `CreationDateTime` (that is, when the document is created in the output queue). The information is passed to the Collector archive together with the document.

After being expired, a document is ready to be deleted. You must then run the `dt_deleteexpired` stored procedure to delete the expired document.

Note: The `dt_deleteexpired` stored procedure deletes all documents that are expired on a certain day, regardless of time of the day.

In this section

Related topics

- For detailed information about the `SetOutputExpireDate` scripting function, see *OpenText StreamServe Scripting Reference Guide*.

Setting expiring dates to already archived documents

We recommend that you use the `SetOutputExpireDate` scripting function in the Design Center Project to set the expiring dates on documents.

However, if there are already archived documents in the Collector archive without expiring dates, you can run a `setexpiringdate.sql` script to set an absolute expiring date (`ExpiringDateTime`) in retrospect. You can either add the expiring date to all the documents in the Collector archive, or only to certain document types.

To set expiring dates for already archived documents

1. In SQL Server Management Studio, open the `setexpiringdate.sql` script. The script is located in:

   `<Base directory>\<Version>\root\config\database\<Version>\strsarchive\sqlserver\maintenance`

   Where `<Base directory>` is the path specified for StreamServe Projects during the **Framework and StreamServer** installation.
   For example: `C:\ManagementGateway`

2. Update the expiring date. For example:

   ```
   SET @day_to_expire = '2020-02-20'
   ```

3. Update the document types to be expired.
   For example, to expire only the document types `Order` and `Invoice`:

   ```
   SET @meta_table_name = 'Meta_Order,Meta_Invoice'
   ```

4. Execute the script.

Scheduling the database to delete expired documents

You can schedule the database to delete expired documents from the Collector archive using the `dt_deleteexpired` stored procedure. The task should be scheduled in a way that ensures minimum impact on any other database activities.

You create the job scheduling task using the appropriate SQL Server tool, for example SQL Server Management Studio.

Note: The `dt_deleteexpired` stored procedure deletes all documents that are expired on a certain day, regardless of time of the day.

Prerequisites and recommendations

- Expiring dates must be set on the documents in the Collector archive. We recommend that you use the `SetOutputExpireDate` scripting function to set the expiring dates.
- The deletion of expired documents must be scheduled in relation to the backup routines, ensuring that the Collector archive is backed up before the expired documents are deleted.
- If many expired documents are deleted at the same time, the deletion may charge the database. To avoid performance bottlenecks, we recommend a regular deletion of a limited number of expired documents.

To schedule the database to delete expired documents

1. In SQL Server Management Studio, start SQL Server Agent and create a new job.
2. Give the job a descriptive name, for example `DeleteDocuments`.
3. Create a new step for the job:
   - Select **Transact SQL-script (T-SQL)** as the type of command.
   - Select the database, for example **StrsData**
   - Enter the following Transact SQL-command:

   ```
   EXEC dt_deleteexpired
   ```

   Optional – When calling `dt_deleteexpired`, you can specify the maximum number of database rows to be removed in each deletion (by default, no limit), and the maximum number of minutes to run the deletion (by default, no limit). For information about the required syntax, open the `dt_deleteexpired` stored procedure in your SQL Server tool.
4. Schedule when to run the job.

Monitoring jobs in the runtime repository

This section contains some useful tips when monitoring jobs in the runtime repository. For example, when monitoring top job statuses (completed, failed, removed, etc.) in order to get an overview the job deletion process.

When monitoring jobs and job statuses, we recommend that you primarily use the tools provided by OpenText.

As a complement to the OpenText StreamServe tools, you can also run queries against the repository. For example, to receive a summary of the total number of top jobs in each processing status.

For more information, see:

- *Queries when monitoring jobs and job statuses* on page 50
- *Queries when monitoring delete performance* on page 54

StreamServe Database Administration Tool

You can use Database Administration Tool to monitor jobs and job statuses. For example, to get an overview of the statuses of the current top jobs and the included input and output jobs. Based on the information, you can then use the tool to further administer the jobs. For example, expire and delete top jobs, and cancel sub jobs.

For more information, see *OpenText StreamServe Database Administration Tool User Guide*.

StreamStudio Reporter

In Reporter, you can administer jobs that are received, processed, and produced by StreamServer applications. For example, you can view job status, resend jobs, and delete jobs from the queues.

For more information, see *OpenText StreamServe Reporter User Guide*.

StreamServe Status Messenger

You can use Status Messenger to generate status reports for jobs. For example, if a job fails, an email can be generated and sent to a system administrator who can take the proper precautions.

For more information, see *OpenText StreamServe Status Messenger User Guide*.

To to use Status Messenger, you must enable notifications in the **Configure Platform** dialog box (the **Use notifications** setting). Enabling of notifications always affects performance.

Queries when monitoring jobs and job statuses

This section contains some queries which you can use to monitor jobs and the processing statuses for the jobs in the runtime repository.

Run the queries as a DBA user in the appropriate SQL Server tool, for example in SQL Server Management Studio.

Available processing statuses

A processing status describes the last completed processing status of a job. The following statuses are available:

Status	Description
`0`	N/A – A processing status is not applicable (for example, when a job is being created). No processing state has yet been finished.
`1`	Completed – The job is completed, with or without errors.
`2`	Cancelled – The processing of the job is cancelled.
`3`	Aborted – The job has failed the maximum number of retries.
`4`	Removed – The job is marked for deletion and will be removed.
`5`	Failed over – Another StreamServer application has taken over the job due to failure of execution.

StructureTypeID

In the queries, `<PartType_Top_Empty>` is the following **Structure Type ID** for top jobs, as described in the `dbo.StructureType` table in the runtime repository:

`AB04C620-0668-400A-B78B-7139722AA194`

In this section

Retrieving a summary of the current top job statuses

To retrieve the number of current top jobs for each processing status with a certain error code, run the query below.

```
SELECT    count(*), ProcessingStatus, ErrorCode
FROM      Part
WHERE     StructureTypeID='<PartType_Top_Empty>' AND
          ExpiringDateTime < getutcdate()
GROUP BY  ProcessingStatus, ErrorCode ;
```

Retrieving a summary of events waiting for processing

To retrieve the total number of events waiting for processing, run the query below.

```
SELECT    count(*), StatusCodeEvent
FROM      QStatusReport
GROUP BY  StatusCodeEvent ;
```

Retrieving the queues that hold events waiting for processing

To retrieve information about which queues hold the events waiting for processing, run the query below.

```
SELECT    count(*), p.StructureTypeID, aq.QueueName
FROM      QStatusReport q inner join Part p on
          q.PartID = inner join Queue aq on
          q.QueueID = aq.QueueID
WHERE     StatusCodeEvent = 1
GROUP BY  p.StructureTypeID, aq.queuename ;
```

Retrieving the total number of failed top jobs

To retrieve the total number of failed top jobs, run the query below.

```
SELECT    count(*)
FROM      Part
WHERE     ((ErrorCode < o) OR (ErrorCode >= 0)) AND
          ProcessingStatus NOT IN (0,4) AND
          StructureTypeID='<PartType_Top_Empty>' ;
```

Retrieving all error messages

To retrieve the last error messages for all failed jobs, run the query below.

```
SELECT    distinct LastErrorMessage
FROM      FixedMetaData
WHERE     LastErrorMessage IS NOT null ;
```

Retrieving a summary of each type of error message

To retrieve the number of each type of last error message for all failed jobs, run the query below.

```
SELECT    count(*), LastErrorMessage
FROM      FixedMetaData
WHERE     LastErrorMessage IS NOT null
GROUP BY  LastErrorMessage ;
```

Retrieving failed jobs with error messages

To retrieve all failed jobs and the last error message for each job, run the query below.

```
SELECT    PartID, LastErrorMessage
FROM      FixedMetaData
WHERE     LastErrorMessage IS NOT null ;
```

Retrieving expired top jobs with error codes and statuses

To retrieve all expired top jobs and the corresponding error codes and processing statuses, run the query below.

```
SELECT    PartID, TrackerID, ErrorCode, ProcessingStatus
FROM      Part
WHERE     StructureTypeID='<PartType_Top_Empty>' AND
          ExpiringDateTime < getutcdate()
ORDER BY  ProcessingStatus, ErrorCode ;
```

Retrieving current processing statuses translated into textual meaning

You can retrieve and list the current processing states for all parts (that is, entities in the database, such as jobs, input files, and output files) in the runtime repository, for all parts belonging to a top job, or for a specific part.

In the result, status codes are translated into textual meaning according to *Available processing statuses* on page 50.

All parts

To retrieve the current processing statuses for all parts, run the query below.

```
SELECT    *
FROM      udf_partstate(NULL) ;
```

Parts belonging to top job

To retrieve the current processing statuses for all parts belonging to a certain top job, run the query below.

```
SELECT    *
FROM      udf_partstate('<Tracker_ID>') ;
```

One specific part

To retrieve the current processing status for a specific part, run the query below.

```
SELECT    *
FROM      udf_partstate('<Part_ID>') ;
```

Queries when monitoring delete performance

This section contains some queries which you can use to monitor the performance of the job deletion process.

Run the queries as a DBA user in the appropriate SQL Server tool, for example in SQL Server Management Studio.

In this section

Retrieving a summary of the spread of job deletion

When the prescribed Design Center setting **Store information and job** is selected for successful and failed jobs, the job deletion is carried out on top job level. This is the recommended setting, resulting in the most optimal delete performance.

However, if the **No** or **Store information only** setting is selected instead, the job deletion is spread among input and output jobs as well. This results in an increased number of delete events and a decreased delete performance. A typical scenario where this can occur is if you run an upgraded Project with old settings. For example, an upgraded StreamServe 4.x Project or a StreamServe Persuasion Project, upgraded from a version older than SP4.

If you experience a decreased delete performance, you can run the query below to find out if and how the job deletion is spread over top jobs, input jobs, and output jobs.

```
SELECT    count(*), StructureTypeID
FROM      QStatusReport q inner join Part p on q.PartID = p.PartID
WHERE     q.StatusCodeEvent=-1
GROUP BY  StructureTypeID ;
```

To interpret the result, you must look up the `structuretypeid` values in the `dbo.StructureType` table in the runtime repository:

- Delete event on top jobs – **PartType_Top_Empty**
- Delete event on input jobs – **PartType_Inputdata_Entity**
- Delete event on output jobs – **PartType_Data_Entity**

If the job deletion is spread over input and output jobs, we strongly recommend that you select the **Store information and job** setting in Design Center.

Retrieving the number of generated top jobs

To decide if the job deletion is scheduled frequently enough, you can run a query to find out the number of top jobs generated during a specified time span.

For example, you can run the query below to find out the number of top jobs generated each 10 minutes since 2012-12-01. If the number of generated top jobs exceeds 5000, the job deletion should be run more frequently than every 10 minutes.

In the query, *<PartType_Top_Empty>* is the following **Structure Type ID** for top jobs, as described in the `dbo.StructureType` table in the runtime repository:

`AB04C620-0668-400A-B78B-7139722AA194`

```
SELECT   count(*), CreDate
FROM     (SELECT  SUBSTRING(CONVERT(VARCHAR(40),
                  CreationDateTime, 20), 0, 16) CreDate
         FROM     Part
         WHERE    StructureTypeID = '<PartType_Top_Empty>' AND
                  CreationDateTime >= '2012-12-01') a
GROUP BY CreDate ;
```

Retrieving the number of notifications

Too many notifications may result in decreased delete performance, especially if each top job generates a single (or a few) output jobs.

If you experience decreased performance, you can run the query below to find out the total number of notifications in the repository.

```
SELECT    count(*)
FROM      Notification ;
```

If the number notifications is extensive, you may consider reducing the notifications. For example, by using the `-reducenotifications` startup argument in Design Center. For more information, see:

- *OpenText StreamServe Startup Arguments Reference Guide*
- *OpenText StreamServe Design Center User Guide*

Performing backup of StreamServe repositories

OpenText does not provide a separate backup package for the OpenText StreamServe repositories.

You back up the repositories using standard Microsoft backup procedures. For information and recommendations, see *Microsoft SQL Server Books Online*.

Simple recovery model recommended

When selecting recovery model, we recommend that you use a simple recovery model, which recovers the database to the most recent backup. The simple recovery does not take care of the transaction history, and therefore does not let the transaction log file grow out of control.

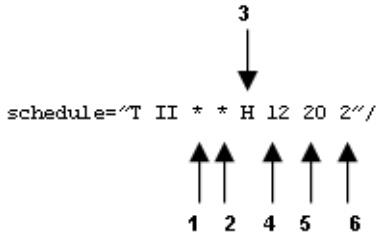
Recovery up to the minute

Carry out the following backups to achieve a recovery up to the minute:

- If possible, run complete backups twice a week.
- Use nightly differential backups.
- Back up the recovery transaction log every 10 minutes during business hours.
- Do not use the enable truncation of the log on checkpoint option (for example, in SQL Server Management Studio), as this will make it impossible to recover some of the transactions.
- Whenever possible, use multiple backup devices to improve the backup restore speed. Avoid backing up the databases on tape media since this is slower than disk.

Appendix A - Time scheduling syntax

The time scheduling syntax looks like the following:



1. The overall start time for the schedule. The time value is always in local time for the computer that is parsing it. The number is formatted in the following way: `YYYYMMDDhhmmssfff`, where:
   - `Y`: year
   - `M`: month
   - `D`: day
   - `h`: hour
   - `m`: minute
   - `s`: second
   - `f`: millisecond

   If no start time exists, the time can be replaced with *.
2. The overall stop time for the schedule, formatted as described above.
3. The type of time interval that the scheduling concerns:
   - `Y`: Year
   - `MY`: Month
   - `WY`: Week of year
   - `WM`: Week of month
   - `DY`: Day of year
   - `DM`: Day of month
   - `DW`: Day of week
   - `H`: Hour
   - `MH`: Minute
   - `S`: Second
   - `MS`: Millisecond
4. The start value for the time interval. If no start value exists, the value can be replaced with *.
5. The stop value for the time interval. If no stop value exists, the value can be replaced with *.
6. The step value for the time interval.

Note: The start and stop values will include the specified values. For example, if you specify `"T II * * H 3 15 2"/`, the scheduled event will run at 03:00, 05:00, 07:00, 09:00, 11:00, 13:00, and 15:00.

Example 5 Scheduling deletion of marked jobs

To delete marked jobs every year:

```
<deleteevent schedule="T II * * Y * * 1"/>
```

To delete marked jobs every second week:

```
<deleteevent schedule="T II * * WY * * 2"/>
```

To delete marked jobs every second hour between 12:00 and 20:00 every day:

```
<deleteevent schedule="T II * * H 12 20 2"/>
```

To delete marked jobs every five minutes between minute 10 and minute 50 in every hour:

```
<deleteevent schedule="T II * * MH 10 50 5"/>
```