

The Siemens logo is displayed in a white rectangular box in the top left corner. The background of the entire page is a low-angle photograph of a modern glass skyscraper reaching towards a blue sky with scattered white clouds.

SIEMENS

Active Workspace 5.2

Indexing and Search Deployment and Configuration

AW030 - 5.2

Contents

Indexing

Indexing deployment strategy	1-1
Indexer component	1-1
Hardware considerations for TcFTSIndexer	1-2
Indexing deployment mode	1-2
Optimize hardware for indexing	1-3
Determine the indexing strategy	1-6
Clone and merge reindexing	1-8
Multi-site indexing strategy	1-10
Install indexing	1-11
Installing indexing components	1-11
Installing Indexing Engine	1-12
Installing the Indexer	1-17
Patch indexing components	1-20
Configure object data indexing	1-23
Prepare to configure indexing	1-23
Configure indexing	1-26
Troubleshoot indexing	1-42
Find logs related to indexing	1-42
Troubleshoot indexing object data	1-47
Troubleshoot TcFTSIndexer	1-48
Troubleshoot indexing performance	1-50
Update Solr credentials	1-52
Troubleshoot Solr issues	1-53
Using indexing utilities	1-56
Indexing utilities	1-56
Customize indexing	1-68
Overview of indexer customization	1-68
Indexer customization prerequisites	1-68
TcFTSIndexer installation layout	1-69
TcFTSIndexer extensibility framework	1-69
Object data indexing steps	1-71
Run a sample search indexing flow	1-72
Adding search indexing steps	1-76
Adding new search indexing types	1-78
Deploy a new search indexing type	1-78
Modify an existing search indexing type	1-79
Adding search indexing flows	1-79
ObjData indexing support for saved queries	1-80
Indexing non-Teamcenter data	1-81

Search

Configure search	2-1
-------------------------	-----

Search configuration tasks	2-1
Configure search suggestions	2-1
Set the default search operator	2-3
Add wildcards to searches automatically	2-4
Configure searching for common words	2-4
Add synonyms to searches	2-5
Boosting search results	2-6
Configure results threshold	2-7
Configure revision rules	2-8
Define search prefilters	2-10
Configure search prefilters for object sets	2-14
Configure Filter panel behavior	2-15
Configure saved search	2-18
Configure column sorting	2-19
Configure export	2-19
Return the parent business object for a dataset	2-20
Configure types to return for global search	2-21
Index and search alternate IDs	2-21
Configure search for multiple sites	2-22
Filter object data with a custom user exit method	2-23
Configure Solr for a two-tier environment	2-25
Example of search results performance	2-26
Configure Advanced search	2-26
Return external business objects with a custom user exit method	2-27
Configure Quick search	2-28
Configuring shape search	2-28
Troubleshooting search	2-30
Find logs related to search	2-30
Troubleshoot search results	2-31
Troubleshoot search performance	2-31
Customize Solr	2-33
Solr URL single point-of-failure	2-33
Configure Solr for HTTPS	2-33
Solr cloud configuration	2-35

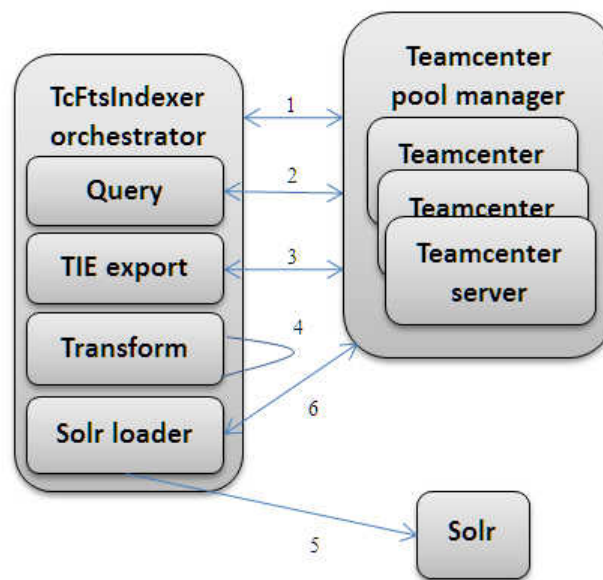


1. Indexing

Indexing deployment strategy

Indexer component

The TcFTSIndexer is a four-tier SOA client that exports Teamcenter data for importing into the Solr database. The TcFTSIndexer is the main component for indexing and is comprised of steps that run in sequential order. In this conceptual diagram, you can follow the flow for Query, Export, Transform, and Load.



1. The TcFTSIndexer connects to Teamcenter and does the initial validation. It reads the configuration for start and end time, and then breaks down the time into configurable time slices for query.
2. The TcFTSIndexer creates a new thread that connects to Teamcenter and gets the list of UIDs.
3. The TcFTSIndexer reads the output UID file and chunks the data into a manageable size, and then calls TIE export by connecting to the pool manager. This step creates the Teamcenter XML file.
4. The TcFTSIndexer converts the Teamcenter XML file to a Solr input XML file with the security read expressions.
5. The TcFTSIndexer loads the Solr input XML file into Solr.
6. The TcFTSIndexer connects to the server manager and calls to confirm export.

Hardware considerations for TcFTSIndexer

When planning your indexing deployment strategy, consider the hardware where you want to install the indexing components. The number of machines and the resources available on the machines help determine how you should configure the TcFTSIndexer environment.

Some of the hardware considerations for TcFTSIndexer are based on memory, storage type, and CPU usage by the TcFTSIndexer components:

- **TcFTSIndexer**
Runs a Java process that runs the query, TIE export, transform, and load operations in stand-alone mode. At high levels of parallelization, transform is more CPU and memory intensive.
- **Server manager**
For object data indexing, the TcFTSIndexer orchestrator connects to the server manager to run the query, TIE export, and load operations. These operations run on multiple **tcserver** instances and connect to the database for SQL calls.
- **Solr**
Solr is CPU-intensive during query calls and can be memory intensive during large dataset file indexing, as well as I/O intensive for reading and writing many documents.
- **Teamcenter database**
The database is intensive in all areas of hardware use due to the number of requests that it handles in a short time. These requests can be CPU-intensive, and the SQL calls require both reading and writing to tables, which is I/O and memory intensive. Careful consideration of the database configuration and hardware requirements should be a priority.

To achieve optimal performance, Siemens Digital Industries Software strongly recommends that these components are installed on separate machines designed for their particular use,

Indexing performance also relies on the number of parallel threads that can be simultaneously completed. The number of threads is controlled by the **tc.maxConnections** property, but it must be coordinated with the number of warmed-up servers in the server manager and the number of threads that the Teamcenter database can support. Optimizing **tc.maxConnections** avoids having too many threads or too few threads which can impact performance. TcServers are single threaded, so it is important to figure out how many CPUs and cores you need on server machines (Indexer, database, server manager) to configure and size the indexing process.

Indexing deployment mode

Standalone indexing mode is the only mode supported for object data indexing.

In this configuration, TcFTSIndexer runs indexing operations in a single Java process (TcFTSIndexer) that connects to the server manager and the Solr search engine. This Java process is a Java SOA client that tracks the number of **tcserver** connections and runs flows based on the configuration.

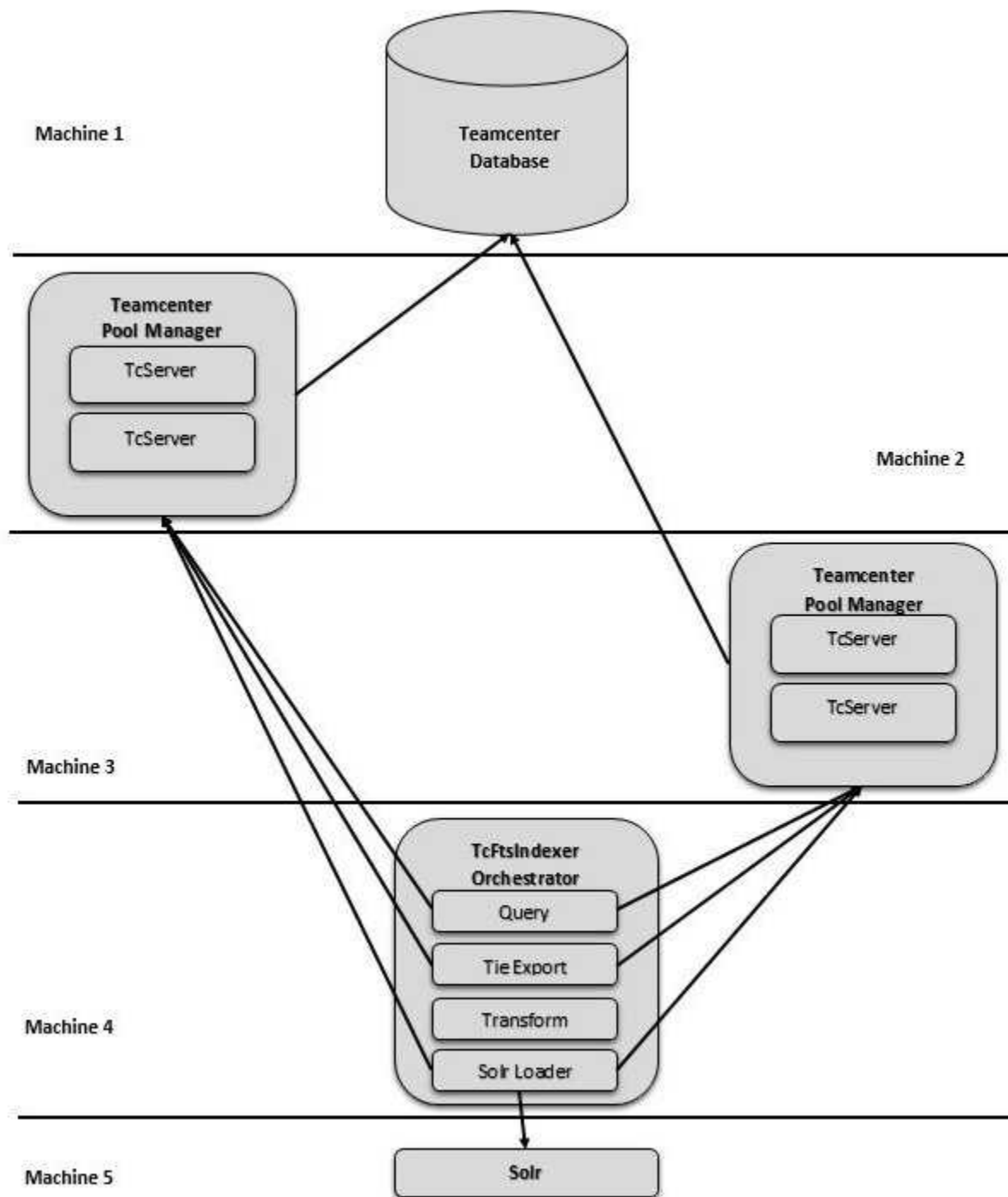
TcFTSIndexer components can be set up independently on different machines, or on a single machine for a simple environment. Configuration strategies need to consider:

- Number of machines available
- Machine specifications
- Available time for initial indexing
- Whether the component being installed is CPU/memory/disk intensive
- Long-term maintenance

Optimize hardware for indexing

Guidelines based on Siemens Digital Industries Software testing can help determine an approach to setting up your indexing environment. In our testing, each of the indexing components was configured on separate machines to better assess resource usage during indexing and identify the hardware bottlenecks associated with this process.

Indexing system layout example



Server setup specifications

Machine	CPU hyper-threaded cores	Core clock speed	RAM	Storage	Software	Operating System
Database	28	2.6 GHz	256 GB	2700 GB (8x800 GB SAS solid state drive in RAID 10)	Oracle 12c Enterprise Edition Release 12.2.0.1.0 64-bit	SUSE Linux Enterprise Server 11.4
Pool 1	12	2.0 GHz	64 GB	5400 GB (C, D, and E partitions. E is 5x1.2TB in RAID 5)	Teamcenter, rich client, TC_DATA location, pool manager, FSC Master, Oracle WebLogic Server 12.2.1.3.0	Microsoft Windows Server 2016 Enterprise
Pool 2	24	2.6 GHz	128 GB	4000 GB (C and D partitions. D is 6x600 GB in RAID 5)	Teamcenter, pool manager, FSC slave	Microsoft Windows Server 2012 R2 Standard
Indexer	12	3.5 GHz	64 GB	1500 GB	TcFTSIndexer	Windows 10
Solr	24	2.3 GHz	128 GB	256, 512 GB solid state disks	Solr	Windows 10 Enterprise

Explanation of machine choices

Machine	Explanation
Database	We chose high performance hardware as the database is the bottleneck for Query and Export tasks. All requests consume CPU, but parallelization allows processing several requests simultaneously. Query and Export requests consume I/O unless the buffer cache is large enough, then the requests are processed in memory. Our configuration has a 96% cache hit with 160 GB dedicated to the buffer

Machine	Explanation
	cache. We also selected SAS solid state drives to limit the I/O bottleneck of simultaneous read/write activities.
Pool 1 and Pool 2	We chose hardware for CPU and memory capacity. The number of warmed servers is based on the number of CPU cores/threads available. Pool 1 has 24 total threads and 22 warmed servers. Pool 2 has 48 total threads and 46 warmed servers. Memory must be adequate to support the servers.
Indexer	To maximize indexing performance, we use all available warmed servers by setting tc.maxConnections to 68. As the database and pool managers are properly tuned, Export tasks complete quickly. Because transform tasks are more CPU intensive, we chose a faster CPU clock speed. Processing a large number of concurrent transform tasks means CPU and memory are consumed in higher values.
Solr	We chose hardware for optimal indexing of file content. The solid state disks improve the I/O and the memory improves streaming files.

Software and configuration

This system used Teamcenter 12.3 with Active Workspace 4.3.

- Warmed servers: 46 in Pool 2 (Pool 1 not used in this test)
- **tc.maxConnections** set to **44**
- **baseBatchSize** set to **2000**

Indexing results

Indexing objects in Solr (object data only):

- 15,106,617 objects in 2.1 hours or 7,066,969 objects per hour
- Item revisions and datasets: no documents were indexed during this run.
- Properties: 28 indexed per object

Determine the indexing strategy

Indexing strategies for object data

You can choose an indexing strategy depending on whether you are performing an Active Workspace installation, upgrade, or patch to determine the type of indexing changes made in your system.

- A new installation requires a **full, initial index of the object data**.
- For an upgrade or patch, choose a reindexing approach:
 - **Perform a full reindex** if you have a wide variety of changes or a high volume of changes.
 - If you need a full reindex of your data, you may opt for **the clone and merge approach** to minimize downtime during the update window. Update your index in a parallel environment cloned from your production environment, and then merge the updated index back into the upgraded production environment. Index any intervening changes that were made after you cloned by indexing the delta of changes.
 - You may choose to **optimize reindexing using timeslices** for a subset of your object data. For example, you could index the data for only the previous year. Remaining data can be indexed after the system is in production by gradually adding more time slices to the index. Verify the performance impact on users if you plan to synchronize the index after rollout.

Note:

Active Workspace returns item revisions in search results rather than items. If a revision rule is applied to identify which revisions of the item are returned, and not all historical data is indexed, the revision rule may not be accurate in some searches.

- When Teamcenter data model changes occur between the previous version and the current version, you may choose between a full reindex or a delta index. The choice depends on the number and type of indexing changes.

Delta reindexing process

Determine whether the number of changes is small or large, based on the number of changes compared to the number of objects your indexing infrastructure can support. In some cases, a full reindex and a delta index may take about the same time, due to a high percentage of impacted type and property changes from the data model update. You must **merge the Teamcenter and Solr schemas**, and then evaluate the data model changes.

- If you have only object data type and property indexing changes, evaluate these changes to determine the scope:

Type changes New types or existing types that were not previously configured for indexing.

Property changes For types previously indexed, new properties or existing properties that were not previously configured for indexing (including referenced and compound properties).

Use **awindexerutil -delta -dryrun** to identify the delta of changes between the last version and the current version. It evaluates type and property changes to data model objects (including custom schema changes to objects made by the Business Modeler IDE). In the returned report, you can evaluate the scope of changes.

You can **index the delta of changes** by running **awindexerutil -delta**. After the index update, set up the synchronization flow using **the runTcFTSIndexer utility**.

Indexing when the Solr schema changes

Solr schema changes (for example, if the **fieldtype** definition is changed in the Solr schema file) can occur in a newer version of Active Workspace, causing the need for a full reindex.

Configure search

There are many **search behaviors you can configure for global search**.

Clone and merge reindexing

You can expedite the reindexing process using the clone and merge approach. Make a copy of your current production environment, upgrade the clone, perform the reindexing, and then merge the reindexed files into your updated production environment. Afterward, index any intervening changes that were made between when you cloned and when you merged.

- You must have existing indexed object data in a production environment.
- You must be familiar with using **the runTcFTSIndexer utility**. There are several **admin** flow actions for **runTcFTSIndexer** that you use to accomplish the clone and indexing actions.
- You need adequate disk space for copying the set of files that comprise the cloned environment.
- Do not make customizations to the cloned environment. Wait until after the cloned environment has been merged to the upgraded production environment.

Tip:

You can check the clone status in a production environment at any time by running:

```
runTcFTSIndexer.bat -task=admin:status
```

1. Mark the production environment as **Clone Ready** by running:

```
runTcFTSIndexer.bat -task=admin:cloneready
```

This action applies the clone state to the production environment in *SOLR_HOME\server\solr\admin*.

While an environment is marked **Clone Ready**, attempting to run **runTcFTSIndexer -task=objdata:clear** fails. This error prevents accidentally clearing the index during this process.

2. If you are running a synchronization flow in the production environment, run:

```
runTcFTSIndexer -stop -task=objdata:sync
```

3. Copy the production environment to the cloned environment (hereafter referred to as the cloned environment).

You must make an exact replica of the entire production environment, making sure to include the SOLR index and database.

```
SOLR_HOME\server\solr\collection-name\data
```

4. After copying the files, restart index synchronization in the production environment while the cloned environment is being updated. The delta of indexing changes are picked up after you merge the cloned environment back into the updated production environment.

5. Run the release update on the cloned environment. Follow the normal upgrade or patch procedures. Perform the same actions that you plan to apply to the production environment.

Do not apply customizations to the cloned environment. You may apply those after the cloned environment is merged with the updated production environment.

6. If the upgrade or update includes a new version of Solr (for example, if Solr is going from version 7.7 to 8.6), copy `SOLR_HOME\server\solr\admin\data` from the old version of Solr in the cloned environment to the same location in the new version of Solr in the cloned environment.
7. Update objects, properties, or other values in the cloned environment, and then begin reindexing the cloned environment:

```
runTcFTSIndexer -task=objdata:index
```

8. Go back to the production environment and apply the same upgrade or patch that you applied to the cloned environment. Follow the normal upgrade or patch procedures.

Do not to apply customizations to the production environment at this time. You may apply those after the cloned environment is merged with the updated production environment.

9. After indexing is complete in the cloned environment, copy the Solr collection data from the cloned environment to the same location in the updated production environment.

```
SOLR_HOME\server\solr\name-of-collection\data
```

Be sure to copy all the collections, including **admin**, **collection1**, **collection2**, **ProductScopeCollection**, and so on.

10. In the production environment, verify that the Solr data you copied to the production environment matches what is expected by running:

```
runTcFTSIndexer -task=admin:clonevalidate
```

If there is a date mismatch between Solr and the database, the error message provides instructions on how to resolve issues. This type of validation does not look for data conflicts.

11. After validation is completed and successful, update the production environment index with the delta of changes that were made after the system was marked as **Clone Ready**:

```
runTcFTSIndexer.bat -task=objdata:index clone
```

12. After the indexing picks up the rest of the changes with no failures, remove the **Clone Ready** state from the production environment:

```
runTcFTSIndexer -task=admin:clonecomplete
```

13. Resume the synchronization flow in the production environment.

Multi-site indexing strategy

Users can search for data available from other sites using Multi-Site Collaboration. Objects are published from their original site and then replicated at multiple sites, allowing users to search for a shared published object. Users can filter the results using **Search Site** to display **Local** or **Remote** results. **Local** results include data from shared sites. The **Remote** filter has a related sub-filter listing specific **Remote Sites** that users can choose.

The Object Directory Services (ODS) published objects must be configured for indexing. A published object, also known as a publication record, tracks updates to the master and determines if shared data needs to be updated at other sites.

Multi-site Collaboration information is available from the Teamcenter help.

Enable multi-site indexing

- Configure TcFTSIndexer to work with Multi-site Collaboration.
In the Teamcenter software kit for your platform, open the *soa_client.zip* and find *\soa_client\java\libs*. Copy the **TcSoaMultisiteStrong_version.jar** and **TcSoaMultisiteTypes_version.jar** files to the following directory on your Indexer server:

```
TC_ROOT\TcFtsIndexer\lib
```

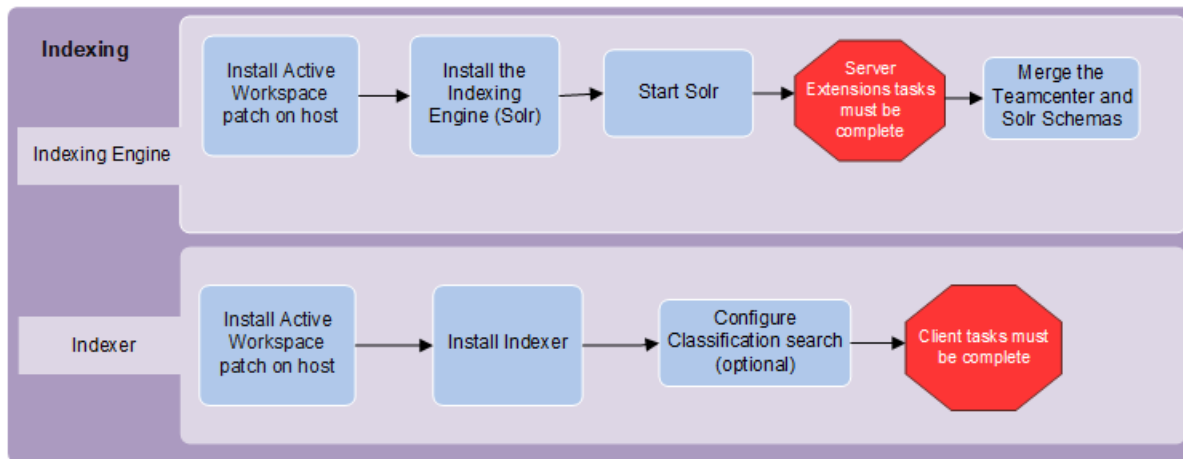
- Set the **ODS_site** preference to the Teamcenter site name designated as the default ODS site. Set the **ODS_searchable_sites** preference to the ODS sites that can be searched for published objects.
- **Indexing properties** for multi-site indexing are in the *TcFtsIndexer_multisite.properties* file located in *TC_ROOT\TcFTSIndexer\conf*. For example, it sets start and end times for indexing objects.
- To enable indexing published records, the **POM_owning_site** property can be indexed and filtered out of the box.

- **Index the published records** using the **runTcFTSIndexer** utility multi-site flow tasks.

Install indexing

Installing indexing components

Indexing installation overview



The Indexer and the Indexing Engine provide global search capabilities for the Active Workspace client.

- **Indexing Engine**

Installs the Solr enterprise search platform. The search engine stores indexed Teamcenter data for global search in Active Workspace.

Selected product data is indexed in Solr, an open source search platform from Apache. The master product data is not stored in Solr. It is always loaded from Teamcenter.

- **Indexer**

Installs a four-tier SOA client that exports Teamcenter data for merging into Solr. The Indexer manages overall indexing processes. The TcFTSIndexer manages initial indexing for object data. You can then schedule synchronization to run periodically for subsequent updates to object data or structure data indexes.

The TcFTSIndexer indexes external and Teamcenter objects into Solr. It connects to the server manager to query and extract Teamcenter data to be indexed into Solr.

There are two modes for installing the **Indexer**, standalone for object data and Dispatcher for asynchronous job processing and translator conversions.

Prerequisites for indexing deployment

Preparing to deploy indexing requires:

- A general understanding of the TcFTSIndexer and its available configurations. Look at configurations in the `TC_ROOT\TcFTSIndexer\conf` directory.

- An understanding of all options provided by the **runTcFTSIndexer utility**.
- Ensuring the Teamcenter database is on a dedicated machine.
- Ensuring the Teamcenter database is properly tuned and does not have slow SQL call messages in the **syslog** files.
- Ensuring Solr is installed on a secure machine and is only accessible through the Teamcenter server.

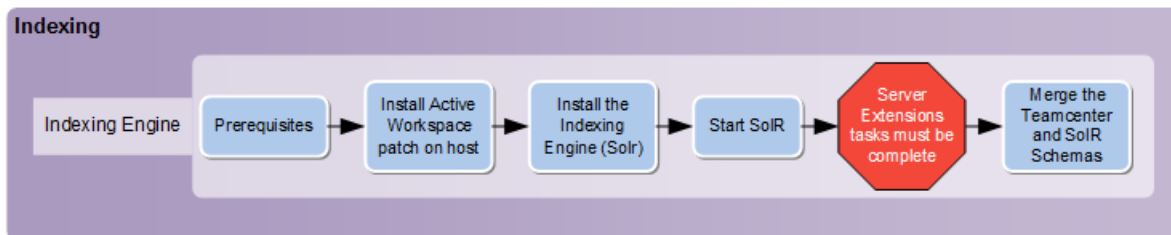
Hardware system requirements

For information on estimating system requirements for the Solr server, go to the Lucidworks web page **lucidworks.com** and look for the following:

- For hardware sizing, search for an article called **Sizing Hardware In The Abstract: Why We Don't Have A Definitive Answer**.
- For memory and Java heap requirements, search for an article called **Estimating Memory and Storage for Lucene/Solr**.

Installing Indexing Engine

Indexing Engine overview

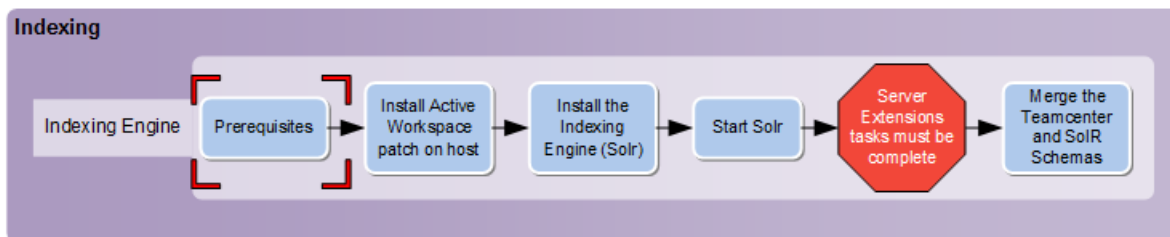


Install the **Indexing Engine** (Solr) search platform. Depending on whether you use TEM or Deployment Center, you may need to enter or verify the following information:

Indexing Engine Configuration		
Parameter	Where value is defined	Your value
Teamcenter machine name and installation path	Teamcenter installation	
TC_DATA	Teamcenter installation	
Java location	Indexing Engine installation	

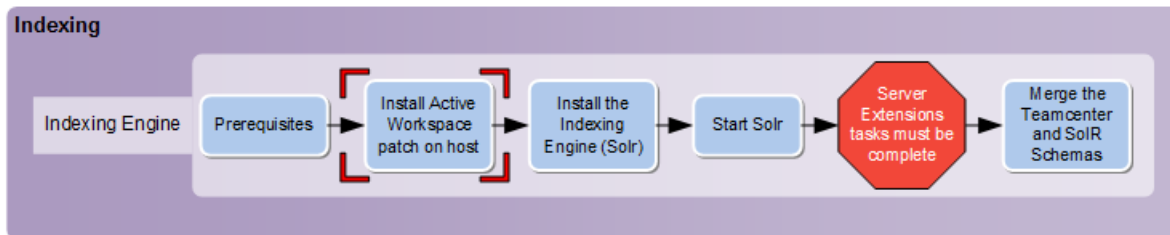
Indexing Engine Configuration		
Solr directory and credentials	Indexing Engine installation	
Solr URL (http://host:8983/solr)	Indexing Engine installation	
Indexing Engine credentials	Indexing Engine installation	

Indexing Engine prerequisites



Prerequisites for the Indexing Engine are a 64-bit operating system and either 64-bit JDK or 64-bit JRE.

Installing the Active Workspace patch on the Indexing Engine host

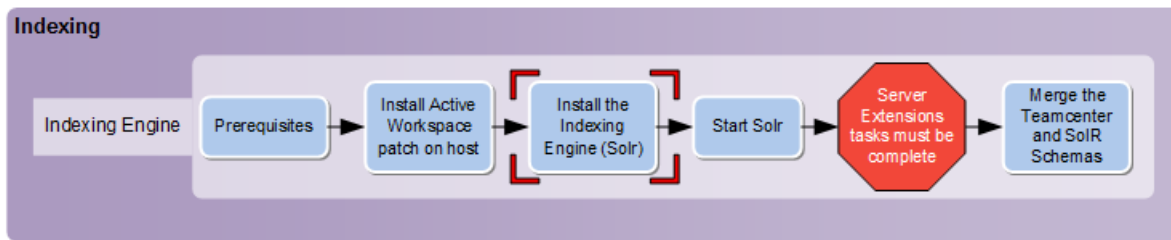


To install the Indexing Engine feature, you must install a supported version of Teamcenter or Teamcenter Rapid Start and Active Workspace. If Active Workspace is not part of the installation, the Indexing Engine feature is not available.

- Install Active Workspace in an existing environment that is patched to the supported Teamcenter or Teamcenter Rapid Start version by installing the Active Workspace patch on the Teamcenter servers.
- If you are using the **Reports** feature, you need to **set up the GraphQL microservice**.

You can install a new Teamcenter environment using TEM or Deployment Center. Add the Teamcenter or Teamcenter Rapid Start installation software kits and the Active Workspace patch software kit.

Install Indexing Engine (Solr)



The Indexing Engine feature can be installed in a new or an existing environment. Be sure you have supported Teamcenter or Teamcenter Rapid Start major and minor releases.

If you are installing the Indexing Engine feature into an existing environment, be aware of the following:

- You can use TEM for that Teamcenter configuration.
In the **Media Locations** panel, specify the locations of Teamcenter updates.
The Teamcenter updates must be listed before the Active Workspace 5.2 location.
- You can use Deployment Center to add Active Workspace to an existing Teamcenter environment.
In the **Selected Software** list, ensure the Active Workspace 5.2 software is selected.

The following provides information about installing the Indexing Engine (Solr). The order or field names may vary slightly, depending on whether you are using TEM or Deployment Center.

Note:

To deploy Solr in Solaris environments, you must use a supported version of Bash. For supported versions, see the Hardware and Software Certifications knowledge base article on Support Center.

1. If you are also installing Teamcenter/Teamcenter Rapid Start, specify the Teamcenter or Teamcenter Rapid Start release software.

Choose a deployment option.
2. Ensure that **Active Workspace** is added and the **Indexing Engine** component is available.
3. Begin configuring the **Indexing Engine**. You may need to provide the machine name.

Provide the Solr administrator's user name and password. These credentials must match the Indexer and the Active Content Structure Translator (if used).

Provide the operating system's user name and password.

Choose whether to install the Indexing Engine as a service on Windows. If this is not selected, you must start the Indexing Engine manually.

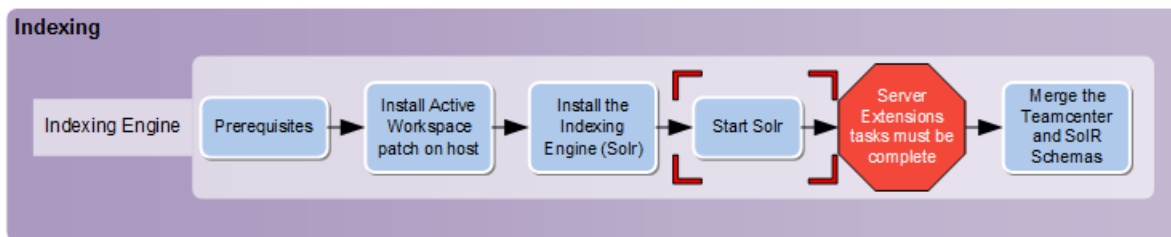
4. In Deployment Center, provide the location of your supported Java installation. Provide or verify your Teamcenter installation path.
5. In TEM, you may provide the SOLR schema files location. This allows the Solr and Teamcenter schemas to be automatically merged during installation of the Indexing Engine. The Solr schema files are created on the corporate server when installing the Server Extensions.

There are two schema files located in *TC_DATA\fts\solr_schema_files*:

- **TC_ACE_SOLR_SCHEMA.xml**
- **TC_SOLR_SCHEMA.xml**

If you have not yet installed Server Extensions or if you are using Deployment Center, you may complete the Indexing Engine installation and **merge the Solr and Teamcenter schemas** manually after completing the Indexing Engine and Server Extensions installations.

Start Solr



Solr is installed on the machine where the **Indexing Engine** is installed. It is located in *INDEXING-ENGINE-ROOT\solr-version*.

If you selected to install the Indexing Engine as a service, the **Active Workspace Indexing Service** was created and started during installation. Its **Startup Type** is **Automatic**.

If you did not install the Indexing Engine as a Windows service or if you are on a Linux system, you must start it manually. Run **runSolr.bat** or **runSolr.sh** from:

INDEXING-ENGINE-ROOT\solr-version

To verify that Solr is running, open a web browser and access the Solr page:

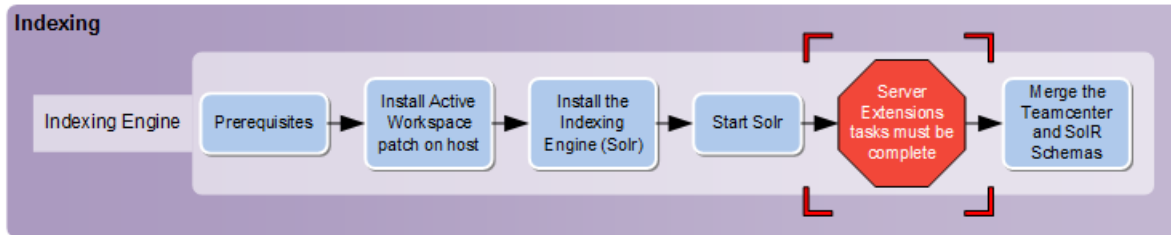
`http://host:port/solr`

host is the machine on which Solr is installed.

port is the port used by Solr. The default is **8983**.

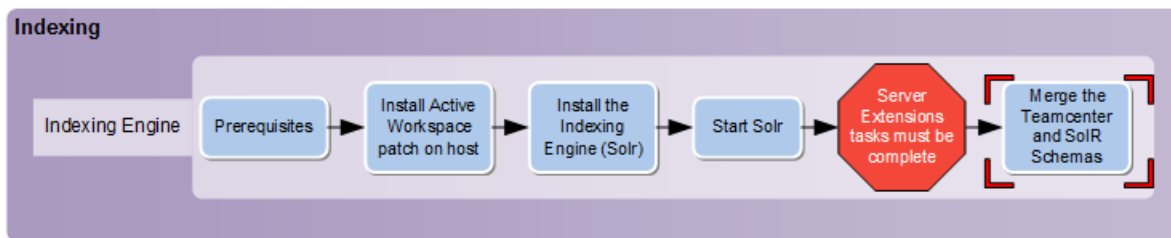
Sign in with the Solr administrator user name and password that you defined when installing the Indexing Engine.

Server Extensions tasks must be complete



Before you can proceed to the next task in the *Indexing Engine* taskflow, all tasks in the Server Extension taskflow must be complete.

Merge the Teamcenter and Solr schemas



Before indexing data, you must merge the Solr schema with the Teamcenter schema. If you did not merge the schema files (*TC_DATA\ftsi\solr_schema_files*) during Indexing Engine installation, you must merge the schemas manually.

You also have to merge the schemas in other situations, for example, if you have made changes in the Business Modeler IDE that affect indexing.

1. If Solr is running, stop it.

In your environment, Solr may be running as the **Active Workspace Indexing Service** or was launched using the **runSolr** command.

2. Locate the *TC_SOLR_SCHEMA.xml* and *TC_ACE_SOLR_SCHEMA.xml* files in the *TC_DATA\ftsi\solr_schema_files* directory on the corporate server. To manually merge the schemas, these files must be available on the machine where Solr is installed.

TC_ACE_SOLR_SCHEMA.xml is present only if the **Active Content Structure** feature is installed.

Solr is installed on the same machine as the **Indexing Engine** feature. If you installed this feature on a machine other than the corporate server, you must copy these files to a temporary location on the Solr host.

- Open a command prompt and change to the *SOLR_HOME* directory.

If you installed the indexing features in the corporate server configuration, the *SOLR_HOME* directory is the *TC_ROOT\solr-version* directory.

If you installed the indexing features on a separate machine, the *SOLR_HOME* directory is *indexing-engine-install-dir\solr-version*.

- Run this command to update the Solr schemas:

TcSchemaToSolrSchemaTransform.bat *LOCAL-DIR*

LOCAL-DIR is the local directory containing the *TC_SOLR_SCHEMA.xml* and *TC_ACE_SOLR_SCHEMA.xml* files.

Tip:

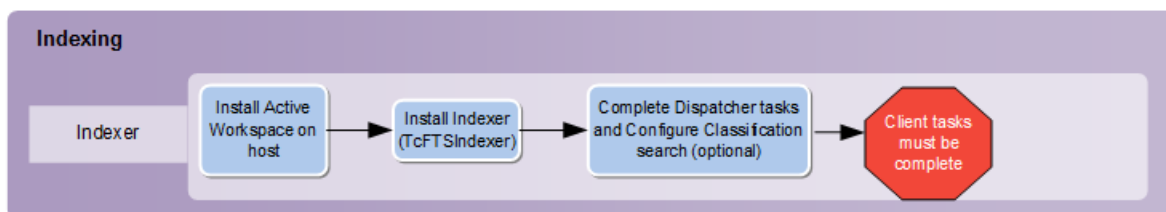
If this command gives the error "file not found", it cannot find the *JAVA_HOME* folder. This *.bat* file has a hard-coded value for *JAVA_HOME*. Verify whether it matches your environment, or remove this line if you have set the *JAVA_HOME* environment variable on your system.

- Restart Solr, either by restarting the **Active Workspace Indexing Service** or by running the startup command:

SOLR_HOME\runSolr.bat

Installing the Indexer

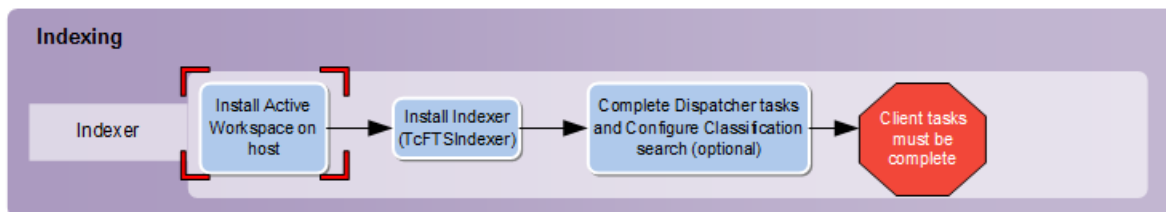
Indexer overview



The Indexer (TcFTSIndexer) feature can be installed in a new or an existing environment. Depending on whether you use TEM or Deployment Center, you may need to enter or verify the following information:

Indexer Configuration		
Parameter	Where value is defined	Your value
Indexing Engine user name and password	Indexing Engine installation	
Teamcenter four-tier URL	Teamcenter web tier deployment	
TcFTSIndexer directory	Install Indexer (TcFTSIndexer)	

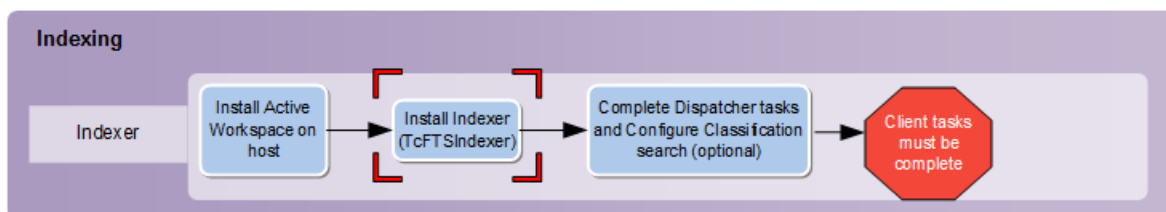
Installing Active Workspace on the Indexer host



To install the Indexer feature, you must install a supported versions of Teamcenter or Teamcenter Rapid Start and Active Workspace. If Active Workspace is not part of the installation, the Indexer feature is not available.

- Install Active Workspace in an existing configuration that is patched to the supported Teamcenter or Teamcenter Rapid Start version by installing the Active Workspace patch on the Teamcenter servers.
- You can install a new Teamcenter environment using TEM or Deployment Center. Add the Teamcenter or Teamcenter Rapid Start installation software kits and the Active Workspace patch software kit.

Install Indexer (TcFTSIndexer)



The Indexer feature can be installed in a new or an existing environment. Be sure you have supported Teamcenter or Teamcenter Rapid Start major and minor releases.

If you are installing the Indexer feature into an existing environment, be aware of the following:

- You can use TEM for that Teamcenter configuration.
In the **Media Locations** panel, specify the locations of Teamcenter updates.
The Teamcenter updates must be listed before the Active Workspace 5.2 location.
- You can use Deployment Center to add Active Workspace to an existing Teamcenter environment.
In the **Selected Software** list, ensure the Active Workspace 5.2 software is selected.

The following provides information about installing the Indexer (TcFTSIndexer) The order or field names may vary slightly, depending on whether you are using TEM or Deployment Center. You need the Indexing Engine user name and password.

1. If you are also installing Teamcenter/Teamcenter Rapid Start, specify the Teamcenter or Teamcenter Rapid Start release software.

Choose a deployment option.

2. Ensure that **Active Workspace** is added and the **Indexer** component is available.
3. Begin configuring the **Indexer**. You may need to provide the machine name.

Provide the Solr administrator's user name and password. These credentials must match the Indexing Engine and the Active Content Structure Translator (if used).

Choose the Indexer type:

- Standalone indexes Teamcenter data. This is the standard global search indexer.
- Dispatcher for Active Content structures.

Maximum Teamcenter Connections specifies the maximum number of connection between the Teamcenter server and the indexer that can be open at a given time. This number should not exceed the number of warm TcServers available in Teamcenter server manager pool and controls the performance of the indexing process using parallel steps. The default value is 3. The minimum value is 2.

Initially, consider the number of warm servers available in this environment and the percentage of them that are available for indexing only.

Teamcenter Retry Count specifies the number of tries to connect to the Teamcenter server. Minimum value is 1.

You may set the object data indexing start and end times.

- **Start Time**
All data modified after this date and time are extracted for indexing; data older than this is not extracted. This value is only used during first-time indexing or re-indexing.
- **End Time**

If selected, specifies the end date for extracting data. All data modified after this date will not be extracted for indexing. This value is only used during first-time indexing or re-indexing.

If no end time is specified, all data modified from the start time to the present is indexed.

You may also set:

- **Maximum Query Timespan**

Specifies the maximum span of a Teamcenter query in minutes. Maximum value is 50000; minimum value is 5000; default value is 20000.

- **Export Batch Size**

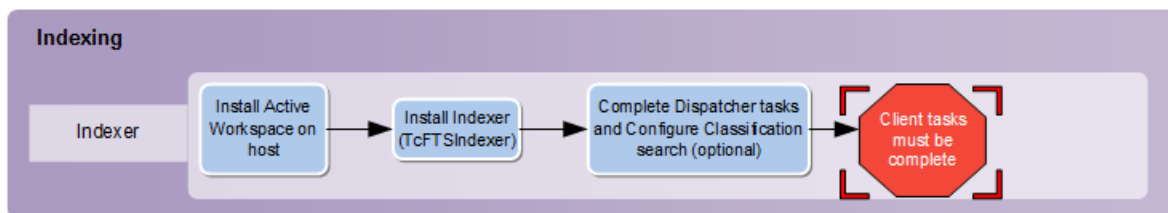
Specifies the maximum number of Teamcenter objects handled in one thread. Maximum value is 20000; minimum value is 1; default value is 1000.

4. In Deployment Center, provide or verify your Teamcenter installation path, and choose whether to install database triggers for indexing.
5. In TEM, provide the Teamcenter 4-tier URL.

`http://host:port/web-app-name`

6. After installing the Indexer, you must **optimize instances of TcFTSIndexer** by adjusting the maximum Teamcenter connections value, maximum query timespan, and export batch size as necessary.

Client tasks must be complete



Before you can proceed to the next task, all Active Workspace Client installation tasks must be complete.

Patch indexing components

Update Indexer settings

If you are using TEM, you must update the Indexer settings after patching Teamcenter and Active Workspace.

1. In the **Maintenance** panel, select **Configuration Manager**.

2. In the **Configuration Maintenance** panel, select **Perform maintenance on an existing configuration**.
3. In the **Old Configuration** panel, select the configuration for Indexer.
4. In the **Feature Maintenance** panel, under **Active Workspace Indexer**, select **Update Active Workspace Indexer settings**.
5. In the **Teamcenter Administrative User** panel, enter the password.
6. After confirming, click **Start**.

Indexing for a Solr upgrade

New version of Solr

In Active Workspace 5.2, you are automatically upgraded to a new version of Solr with the upgraded Indexing Engine. Your indexing strategy depends on which version of Active Workspace you are migrating from.

- If you are upgrading from Active Workspace 4.3 or later, you have the option to move your indexed data without reindexing. This approach assumes that you do not have new data to index.
- If you are upgrading Active Workspace from a version prior to 4.3, you must reindex your data.

Note:

In both cases, the previous version of Solr is not removed during an upgrade of Active Workspace. When the upgrade is complete, ensure that your system is operational using the upgraded version. Then, you can back up the directory where Solr was previously installed and remove it from your system.

Upgrading Solr from Active Workspace 4.3 or later

You can choose to reindex your data or you can move your existing indexed data.

If you opt to move your indexed data without reindexing, the new Solr configuration must match the previous Active Workspace configuration of Solr.

1. Copy the following directory from Solr:

`SOLR_HOME\server\solr\collection1\data`

2. Place the directory in the same path for the upgraded release of Solr, overwriting the existing directory created by the upgrade.

3. Repeat the previous steps for all Solr instances.

If the Active Workspace configuration used SolrCloud, be sure that the Solr upgrade is configured with the same number of shards as the previous version of Solr. Additionally, ensure that the shards match, for example, Solr *old-version* **Leader1** matches Solr *new-version* **Leader1**, and so on.

Upgrading Solr from a version prior to Active Workspace 4.3

To upgrade Solr from a version prior to 4.3, you must reindex your data so that field types remain synchronized and valid.

Remerge Solr and Teamcenter schemas and update the index

After moving to the target Active Workspace and Teamcenter versions, the Teamcenter and Solr schemas are not synchronized. You can merge the schemas and then choose the method for updating the index. You must **determine your indexing strategy** before planning your index update.

Evaluate the delta of object data changes

If your indexing changes are additions, modifications, and deletions for types and properties, you can perform a delta indexing update rather than a complete reindex.

1. **Merge the Teamcenter and Solr schemas.**
2. Stop synchronization by the indexer if it's running.

```
runTcFTSIndexer -stop
```

3. Determine the scope of the changes between the last indexing schema and the current schema.

Run **the awindexerutil utility** using **-delta -dryrun** to get a report of the expected delta of changes. For example:

```
awindexerutil -u=adminuser -p=password -g=group -delta -dryrun
```

The differences are output to the command window as well as to a log file.

4. After you evaluate the report, determine whether you want to use the delta of changes for reindexing. If so, run **awindexerutil** to index the changes from the report:

```
awindexerutil u=adminuser -p=password -g=group -delta
```

5. Test indexer connectivity by running the indexer test flow.

```
runTcFTSIndexer -task=objdata:test
```


- Restart the synchronization flow using **the runTcFTSIndexer utility**:

```
runTcFTSIndexer -task=objdata:sync -interval=seconds
```

Reindex your data if needed

Your index may have a high number of changes or other kinds of changes that are outside the scope of delta changes. If that is the case, reindex your data instead:

- Merge the Teamcenter and Solr schemas.**
- Test indexer connectivity by running the indexer test flow.

```
runTcFTSIndexer -task=objdata:test
```

- Reindex the data.**

Configure object data indexing

Prepare to configure indexing

Ensure that indexing is installed and running

You should be familiar with the **indexing component hardware specifications** on the servers for the Indexer, Solr, the Teamcenter database, and server managers.

Before you can configure indexing, you must ensure that search indexing is installed and tested:

- Ensure that the search indexing features are installed.**
- Ensure the Solr indexing engine is running.**
Start Solr using the following command:

```
INDEXING-ENGINE-ROOT\solr-version\runSolr.bat
```

Test that Solr is running:

```
http://host:port/solr/admin
```

The default port is **8983**.

- Test the indexer connectivity.**
Run the following command from the `TC_ROOT\TcFTSIndexer\bin` directory to test object data indexing:

```
runTcFTSIndexer -task=objdata:test
```

Test TcFTSIndexer connectivity

You need to ensure that the Teamcenter user running the **runTcFTSIndexer** utility can sign in to the database and that all the indexing systems are running.

1. The default user who runs the utility is defined in the **Tc.user** setting in the *TC_ROOT/TcFTSIndexer/conf/TcFtsIndexer.properties* file.

When running in a Security Services protected environment, this user must also be a valid user that Security Services can authenticate in the LDAP server.

- a. If you are using multiple TCCS Security Services application IDs, make sure they are configured correctly.

You can configure multiple application IDs using the **Environment Settings for Client Communication System** panel in Teamcenter Environment Manager (TEM).

Enter the required information to create the TCCS environment:

Value	Description
Name	Specifies the name of a the TCCS environment. This name is displayed in the TCCS logon dialog after configuration is complete.
URI	<p>Specifies the URI to the TCCS environment. This is the endpoint URI for the web tier deployment, for example, http://host:port/tc.</p> <p>Whether your network uses IPV6 (128-bit) or IPV4 (32-bit) addresses, use the hostname in URIs and not the literal addresses, so the domain name system (DNS) can determine which IP address should be used.</p>
Tag	<p>Specifies a string identifier for the TCCS environment.</p> <p>When installing a rich client, you can optionally provide a Client Tag Filter value to filter the list of environments displayed in the rich client to those environments that match the filter.</p> <p>For example, if the Client Tag Filter value is 9*, all TCCS environments with Tag values beginning with 9 are available to the client host. Environments with Tag values beginning with 10 are not available.</p>
SSO App ID	Specifies the ID of the Security Services application you use with TCCS.
SSO Login URL	<p>Specifies the URL to the Security Services application you use with TCCS.</p> <p>If you use Security Services in applet-free mode, include /tccs at the end of the URL, for example:</p>

Value	Description
	<code>http://host:port/app-name/tccs</code>

- b. Ensure that the user defined by the **Tc.user** setting in the `TC_ROOT\TcFTSIndexer\conf\TcFtsIndexer.properties` file is a valid user in the LDAP server and the Teamcenter database. Create a user in both if needed or select an existing valid active user to run the **runTcFTSIndexer** utility.

If you create a new user, create an encrypted password file by setting an environment variable, within the console, to the password value, for example:

```
set mytcenv=password
```

Then run the **encryptPass.bat/sh** utility with the **-tc** argument specifying the environment variable name created, for example:

```
encryptPass -tc mytcenv
```

Then run the utility for the LDAP password used by the TcFTSIndexer user. The **encryptPass.bat/sh** utility is located in the `TC_ROOT\TcFTSIndexer\bin` directory.

After creating the encrypted password file, remove the password variable.

2. Ensure that the following are running:
 - Teamcenter database
 - Solr
 - Web application server hosting the Teamcenter web tier application
 - Web application server hosting the Active Workspace web application
 - Server manager
3. On the machine where the **Indexer** feature is installed, open a command prompt.
4. Navigate to the bin directory of the TcFTSIndexer, for example, `TC_ROOT\TcFTSIndexer\bin`.
5. To test connectivity for TcFTSIndexer, run the command:

```
runTcFTSIndexer -task=objdata:test
```

6. Verify that there were no errors.

Configure indexing

Indexing configuration tasks

Be sure you determine your **object data indexing strategy** and understand **the standard process for indexing object data**.

Evaluate **search configuration tasks** that you may need to perform for your site.

You can also configure structured content indexing.

Optimize instances of TcFTSIndexer

Before you start optimization

Before starting to optimize your TcFTSIndexer environment, be sure to:

1. Review the **hardware considerations**.
2. Review the **example indexing configuration**.
3. Test **TcFTSIndexer connectivity**.

Understanding optimization

Indexing optimization is an iterative process:

1. First, index a small set of data to understand how long it takes to index the whole set of data.
2. Then, analyze the consumption of CPU, memory, and I/O, using this information to determine if you need more or fewer parallel threads in your environment.

Evaluating this information helps to reconfigure TcFTSIndexer settings, as well as configure additional machines.

The TcFTSIndexer orchestrator supports query, TIE export, transform, and load as stand-alone processes that run in a sequential pattern. These processes have a single level of configuration in stand-alone mode, which is the maximum connections setting (**tc.maxConnections** property). This setting specifies the maximum number of Teamcenter connections that can be open at a given time. This number should not exceed the number of warmed **tcserver** instances available in the pool manager. The minimum value is **2** and the default value is **3**.

You can change the **tc.maxConnections** value in Deployment Center or Teamcenter Environment Manager (TEM).

Optimize timeslice and batch size

After the number of machines and the components on each machine are defined, determine the optimal timeslice and batch size. Then adjust the number of connections.

1. Identify a small list of objects to index. Indexing is based on time slices, so you need to identify a start and end time that contain around 5% of total objects. You can run queries for indexable objects in the Teamcenter rich client to get an approximate duration. Set the start and end time in the file:

```
TC_ROOT\TcFTSIndexer\conf\TcFtsIndexer_objdata.properties
```

2. Test that the indexing components are running and the TcFTSIndexer orchestrator can connect to them using the **runTcFTSIndexer** command:

```
runTcFTSIndexer -task=objdata:test
```

3. Start Windows Task Manager on each machine that has an indexing component installed, such as Solr, pool manager, TcFtsIndexer orchestrator, and database.
4. Run the following command to index the 5% set of objects:

```
runTcFTSIndexer -task=objdata:index
```

- Monitor memory consumption and CPU processes using the Task Manager on machines with an indexing component installed. You can evaluate whether there is room for more **tcserver** connections.
 - Monitor both the database and the machine it runs on to be sure you avoid overloading either one.
 - Record the overall time taken for indexing from the report generated at the end of the TcFTSIndexer run. The report is available from the console and stored in the **TcFtsIndexer.log** and **TcFtsIndexer_objdata.log** in *TC_ROOT\TcFTSIndexer\logs*.
5. After the initial run, if all machines used less than 90% of their resources, increase the **tc.maxConnections** value, as long as it does not exceed the maximum of warmed **tcservers**. If it does, then increase the number of warmed Teamcenter servers available in the pool before increasing the number of connections.

Tip:

You can dynamically modify the **tc.maxConnections** setting during an active indexing run:

- a. Open a new Teamcenter command window and navigate to the `TC_ROOT\TcFTSIndexer\bin` directory.
- b. Run the following **runTcFTSIndexer** utility command:

```
runTcFTSIndexer -maxConnections=connections
```

connections is the number of connections you want, but it must not exceed the number of **tcservers** available.

6. Iterate through the previous two steps until the **tc.maxConnections** setting is optimized by reaching more than 90% resource consumption but not always at 100%.
7. After you find the optimum connections setting, make this setting permanent for all indexing sessions on the indexing machine using Teamcenter Environment Manager (TEM).
 - a. Open Teamcenter Environment Manager (TEM), and click **Next** until you reach the **Feature Maintenance** panel.
 - b. Select **Update Active Workspace Indexer settings** and click **Next** until you reach the **Active Workspace Indexer Settings** panel.
 - c. Look for **Maximum Teamcenter Connections** and change the value.
8. After you optimize the **tc.maxConnections** setting, optimize the **exportbatchsizestep.baseBatchSize** property.

Note:

This step is optional for small environments with fewer than one million objects to index, but it is recommended for medium to large environments with more than 1 million objects to index.

The default **exportbatchsizestep.baseBatchSize** value is 1000.

- You can increase the value for **exportbatchsizestep.baseBatchSize** to improve performance, which increases the number of Teamcenter objects per thread.
 - Internal testing produced optimized results with the value set between 2000 and 5000 depending on the size of the index.
 - Increasing it to a value that is too large can adversely impact performance.
9. After the TcFTSIndexer is optimized and sized correctly, you can start a full index of the data.

Set up TcFTSIndexer scheduling

After running the initial index, you can set up indexing to synchronize at specified intervals.

1. Navigate to the `TC_ROOT\TcFTSIndexer\bin` directory on the machine where the Active Workspace Indexer is installed.
2. If you are using multiple types of indexing (for example, object indexing and structure indexing), the TcFTSIndexer process must be started in service mode as shown in this step. If you are using object indexing only, skip to the next step.

To start the TcFTSIndexer in service mode, use the **-service** option. Using the **-service** option keeps the TcFTSIndexer process running so that multiple types of indexing operations can run concurrently. For example:

```
runTcFTSIndexer -service
```

3. Enter **runTcFTSIndexer -task=objdata:sync -interval=x**, where x is the number of seconds greater than 0 to wait before rerunning the synchronization operation.

For example, to wait 25 seconds, type:

```
runTcFTSIndexer -task=objdata:sync -interval=25
```

If the **-service** option was previously used, the command is sent to the service and the service runs it. Otherwise, the action runs in the current process.

4. To stop the TcFTSIndexer process, choose one of the following:
 - To stop the process after all flows stop running, type **runTcFTSIndexer -stop**, and then type **runTcFTSIndexer -status**.
When these commands return that no flows are running, the process is stopped. You may need to wait a period and repeat this command to give the process time to stop. The service is still running.
 - To stop the service and then shut it down, type **runTcFTSIndexer -shutdown**.
After flows are allowed to complete, the service is stopped.
5. You can run the TcFTSIndexer service in a new window.

Start object data synchronization using the desired interval. In the following example, the interval is 25 seconds.

```
runTcFTSIndexer -task=objdata:sync -interval=25
```

Define index data and filters

By default, children of the **ItemRevision** business object are marked for indexing as well as specific properties on those business objects. In the Business Modeler IDE, you can create a custom template that defines business objects and properties to index. You can also define search filter behavior using global constants. For information about working with constants, see Introduction to business object constants in the Teamcenter documentation.

You can define only certain property types for search and filtering:

Properties supported for search and filtering	Properties not supported for search and filtering
Date	String properties with long string as storage
String	Properties required for search and filtering
Numeric	These properties are indexed by default and cannot be removed:
References	POM_application_object.last_mod_date
Logical	POM_application_object.creation_date
Array properties	WorkspaceObject.object_type

Note:

You can set up separate filtering for Changes and Inbox.

Tip:

When setting properties as filters, avoid using any property that has a high number of unique values, such as object IDs. When the user tries to use such a filter, the action can result in a **No Results Found** error message. For example, if a search uses the ***** wildcard and a filter specifies the ID property, a high number of returned items would cause the error when the user tries to filter on ID. When configuring filters, choose properties that have fewer unique values.

1. Import the Active Workspace (**aws2**) template into your Business Modeler IDE project. For details about importing templates, see Introduction to templates in the Teamcenter Business Modeler IDE help.
2. In the Business Modeler IDE, open the business object you want to index.
3. To define a business object for indexing, use the **Awp0SearchIsIndexed** business object constant. The indexing values are inherited in the hierarchy. For example, if you mark **ItemRevision** business objects for indexing, all item revisions under it (part revision, document revision, and so on) are

automatically marked for indexing. You can, however, choose to not index a lower level object by setting the property to **false** to override its inherited value.

4. To define a property for indexing, use the **Awp0SearchIsIndexed** property constant. The indexing values are inherited in the hierarchy. For example, if you mark **object_type** on **ItemRevision** business objects for indexing, object types for all item revisions under it (part revision, document revision, and so on) are automatically marked for indexing. You can, however, choose to not index a property on a lower level object by deselecting it manually.
5. The **AW_Indexable_File_Extensions** preference defines the file types that you want to index. It specifies file extensions for text content extraction during search indexing. Valid values are file extension names (for example, *.txt*, *.doc*, and so on). If no value is specified for this preference, the file extensions listed under the **Awp0IndexableFileTypes** global constant are used.
6. Define the search filters using the appropriate global constant, business object constant, or property constants. For example, to specify a property to display on the list of search results filters, use the **Awp0SearchCanFilter** property constant. To set the order that the property appears in the list of filters, use the **Awp0SearchFilterPriority** property constant. The indexing values are inherited in the hierarchy. For the filters show up correctly in the user interface, the **Awp0SearchCanFilter** and **Awp0SearchFilterPriority** property constants should be set for the property on its source business object.

Tip:

When you set the **Awp0SearchCanFilter** property constant to true on a property, that property only displays in the search results filtering list if all the objects returned also have that property. If all properties of the returned objects were displayed, the filter list would be far too long to be useful. Because owning user, last modification date, release status, and similar properties are common to all objects, they always appear in the search results filter list.

When the property is a **Table** type property, all valid properties of the referenced **TableRow** type are available as filters. All these table row properties get the same filter priority, so they appear together, vertically listed in the filter pane.

The filter category name for table properties includes the table property display name appended with a delimiter (:) and a table row property display name. For example, **Content** is the table property display name while **Cost**, **Processed Date**, and **Value** are the table row property display names.

7. By default, Solr displays a maximum of 100 values under each property in the filters list. You can remove the limitation and allow all objects to be listed. Open the `TC_ROOT\solr-version\server\solr\collection1\conf\solrconfig.xml` file, and search for `<requestHandler name="/tcfts" class="solr.SearchHandler">`. Add the following to the `<lst name="defaults">` section:

```
<requestHandler name="/tcfts" class="solr.SearchHandler">
  <!-- Request handler for Teamcenter search -->
  <lst name="defaults"
    <str name="echoParams">explicit</str>
    <int name="rows">10</int>
    <str name="df">TC_0Y0_FTS</str>
    <str name="q.op">OR</str>
    <str name="facet.limit">-1</str>
  </lst>
```

You can also remove the limitation for specific properties. For example, to allow all object types to be listed, add the following to the **<lst name="defaults">** section:

```
<str name="f.TC_0Y0_WorkspaceObject_0Y0_object_type.facet.limit">-1</str>
<str name="facet.sort">count</str>
```

8. Run the Indexer after updating the custom template.

Configuring search index properties

TcFTSIndexer search properties are stored in the **TcFTSIndexer_objdata.properties** file located in **TC_ROOT\TcFTSIndexer\conf**. Most search properties are defined during installation of the TcFTSIndexer component.

You can define other properties in the **TcFTSIndexer_objdata.properties** file, for example:

```
objdataloadstep.indexStandaloneDatasets
```

Indexing datasets

If dataset types are marked for indexing, standalone datasets and their file content are indexed. Dataset file content is indexed by setting the preference **AWS_FullTextSearch_Index_Dataset_File_Content** to **on**.

You can configure whether to **return the parent business object for a dataset**.

Run initial index of object data

Before running your initial index, be sure you **optimize your TcFTSIndexer machines**.

Note:

Siemens Digital Industries Software recommends that you do not exceed indexing 10 million objects in a single indexing run. If you must index more than 10 million objects in a single indexing run, you can increment the *N* value in **xmxNg** in the **TC_ROOT\TcFTSIndexer\bin\runTcFTSIndexer.bat** by 1 for each additional 10 million objects. (The initial default value is **3g**.)

1. If there were no errors resulting from the **TcFTSIndexer connectivity test**, you are ready to run the initial index. Ensure that the following are running:
 - Teamcenter database
 - Solr
 - Teamcenter web tier
 - Server manager
 - Active Workspace Gateway
2. On the machine on which the **Indexer** feature is installed, open a command prompt.
3. Navigate to the *bin* directory of the TcFTSIndexer, for example, *TC_ROOT\TcFTSIndexer\bin*.
4. For object data indexing, run the following command:

```
runTcFTSIndexer -task=objdata:index
```

The initial index may take some time to run if there is existing data in the database.

5. Determine whether index failures occurred. Check the report generated on the console and stored in *TC_ROOT\TcFTSIndexer\logs\TcFtsIndexer_objdata.log*.
6. If there are failures, run the **recover** option:

```
runTcFTSIndexer -task=objdata:recover
```

Run **objdata:recover** repeatedly until all indexable objects succeed, and the remaining errors fail repeatedly. Verify whether the index errors are resolved.

You can fix the errors or return at a later time to fix them. If you leave them, these errors are also logged as failures during synchronization.

The **runTcFTSIndexer** utility can run a variety of indexing tasks.

7. Ensure that the following are running:
 - Teamcenter database
 - Server manager
 - Teamcenter web tier

- Active Workspace Gateway

8. Test that indexing is working by running a search from Active Workspace. Open the URL to the Active Workspace Gateway.

Type a term that you know was indexed in the search box and click **Search**.

If you see results related to the term you typed and you do not receive any errors, the Active Workspace deployment is operating properly.

Change the user running the TcFTSIndexer

The user who runs the search Indexer (TcFTSIndexer) is set during installation of the Indexing Engine. Perform the following steps if you want to change the user. The indexing user must have read access to datasets and their associated files to index their text content.

1. Navigate to the `TC_ROOT\TcFTSIndexer\conf\TcFtsIndexer.properties` file and open it.
2. Change the user in the **Tc.user** setting to an authorized Teamcenter user with database privileges.
3. In the console, set an environment variable to the password value.

```
set mytcenv=password
```

4. Create an encrypted password file for this user by running the **encryptPass.bat/sh** utility in the `TC_ROOT\TcFTSIndexer\bin` directory.

Run the **encryptPass.bat/sh** utility with the **-tc** argument and specify the environment variable created in the previous step, for example:

```
encryptPass -tc mytcenv
```

5. After you create the encrypted password file, remove the environment variable value.

```
set mytcenv=
```

Run the indexer after updating a custom template

Any time you make a change to your custom template package, you must redeploy it to the database and run the Indexer.

1. Install your custom template package. For information about templates in Business Modeler IDE, see Introduction to deploying templates in the Teamcenter help.

Note:

If changes are done using Hot Deploy or Live Update, you must manually run a utility afterward. Run the following command in a Teamcenter command window that has **TC_ROOT** and **TC_DATA** defined.

```
bmode_modeltool.bat -u=username -p=password -g=group -tool=all
-mode=upgrade -target_dir="TC_DATA"
```

2. **Merge the Teamcenter and Solr schemas.**

3. **Re-index search data.**

Users can search in the master language set on the server, which is defined by the **SiteMasterLanguage** global constant. If you change the **SiteMasterLanguage** global constant value, you must run the Indexer again with the **objdata:index** option because the data needs to be indexed with the new set of analyzers for the language.

Display snippets from file content

You can display the location of search terms within file content attached to items. Global search results with matching content from attached dataset files display snippets of the text. The returned snippet is displayed as a phrase under the object. Displaying snippets can help the user determine where the search term is matched in the results, especially if the result does not match the name or description. Snippets obey the stemming principle for matching similar words when displaying search terms.

1. The Indexing Engine (Solr) requires that you set a global constant In Business Modeler IDE to ensure that dataset fields are stored for the Solr schema. Set the global constant **Awp0StoreDatasetContent** to **true**.

You must then reindex your data, including datasets, and restart Solr.

2. The **AWC_Search_Enable_Snippets** preference controls whether the highlighted phrases are displayed for global search, and it also checks whether datasets are indexed. This preference is set to **true** by default. The preference is enabled for the site, but the user may set it to **false** to increase the number of visible items in the results list.
3. You can control how much of the surrounding content is displayed in the snippet by setting the **AWC_Search_Trim_Snippets_Size** preference. Specify the number of words to be displayed on either side of the matched search criteria in the file content. The preference is applied only when the snippet is more than 400 characters. For example, if the returned snippet size is 450 characters, then the value of this preference is applied. The default value is 10 words to the left and right of the matching term.

If snippets are enabled but searching within attached datasets is disabled, the preferences controlling snippets are ignored.

If there is a large number of lengthy documents being indexed, search performance may be affected.

Configure localized display names

Display names may be localized and then indexed to make the localized property values available to users.

Be sure that the string properties you want to index have been marked as localizable.

1. In Business Modeler IDE, change the **Awp0IndexLocalizedValues** global constant to **true**.
2. From a Teamcenter command prompt, run:

```
bmidemodeltool -u=user -p=password -g=dba -tool=all -mode=upgrade
               -target_dir="%TC_DATA%"
```

3. **Merge the Teamcenter and Solr Schemas.**
4. Navigate to *TC_ROOT/TcFtsIndexer/conf/TcFtsIndexer_objdata.properties*. Open the file and confirm that the **objdatasaxtransformstep.supportedLocalesForIndexing** property contains the list of locales you want to index.
5. Run **runTcFtsIndexer -task=objdata:clear** to clear index data. Specify either option **1** or option **4** when prompted to enter an option.

Option **1** clears the indexer cache.

Option **4** clears the cache, the Solr index, and the object data indexing records from the Accountability table.

6. Re-index your data.

Example:

Set the supported languages in the *TcFtsIndexer_objdata.properties* file.

```
objdatasaxtransformstep.supportedLocalesForIndexing=
  en_US,de_DE,it_IT,ja_JP,cs_CZ,es_ES,fr_FR,ko_KR,pl_PL,pt_BR,
  ru_RU,zh_CN,zh_TW,en_US,de_DE
```

If the value is empty, only the master locale of the server is indexed.

Index right-to-left languages

You can index right-to-left text in file content for the RTL languages, such as Arabic and Hebrew, that are supported by Solr. Indexer and Solr configuration provide the ability to index datasets, such as PDFs, containing supported RTL languages.

Set up RTL language support for indexing

RTL language files provide indexing support. If you do not have language profile files, you can obtain them from your Solr installation.

1. In Solr's `\contrib\langid\lib\` directory, locate the Solr language profile JAR file *langdetect*. Copy it to another directory and extract the JAR file.

In the extracted *profiles* directory, find the profile file for your RTL language, for example, *he* for Hebrew.

2. Add the language profile files (either your own or Solr's) to the following directory:

`TC_ROOT/TcFTSIndexer/conf/languageProfiles`

3. Open `SOLR_HOME\server\solr\collection1\conf\schema.xml`.
4. Create the field type for your RTL language. This example sets it for Hebrew.

```
<fieldType class="solr.TextField" name="tc_text_he"
positionIncrementGap="0">
  <!-- Hebrew (he) -->
  <analyzer>
    <tokenizer class="solr.ICUTokenizerFactory"/>
  </analyzer>
</fieldType>
```

5. Create a field for your RTL language.

```
<field indexed="true" multiValued="true" name="TC_DS_file_content_he"
required="false" stored="false" type="tc_text_he"/>
```

6. Restart Solr.
7. Create the preference **AWC_Additional_Supported_Languages_Dataset_Indexing** to specify the set of RTL languages. Set it to the list of language values. The language values must match the language profile file name, for example, *he* for Hebrew.
8. Be sure to reindex the datasets to make their content searchable in RTL languages.

Configure the reporting microservice

If the **Reports** feature is installed, you can generate reports from **My Dashboard** that search the Indexing Engine (Solr) directly for the search criteria. These reports run more quickly as they return filter properties but they do not retrieve the objects themselves. This type of report requires the GraphQL microservice.

When installing or updating Active Workspace, you must install microservices and configure the GraphQL microservice to communicate with the Indexing Engine (Solr). The GraphQL microservice is available by default with Active Workspace.

Using TEM If you are installing or upgrading Active Workspace, configure the **GraphQL Microservice** feature by providing the Teamcenter web tier URL and the **Indexing Engine** user name and password.

If you are performing **Feature Maintenance** updates for Active Workspace, look for **Teamcenter GraphQL Service** and select **Update Settings**. You must provide the Indexing Engine user name and password.

Using Deployment Center If you are installing or upgrading Active Workspace, configure the default **Microservice Node** component, which includes the required **Teamcenter GraphQL Service**. You must also provide the **Indexing Engine** component user name and password.

Ensure you have the correct **Indexing Engine user name and password** as it is possible they were updated.

After completing the configuration, you may run the sample **My Modified Objects** or **My Group Released Objects** reports available from **My Dashboard** in Active Workspace to test the connection.

Re-index search data

You should re-index your existing data using the **runTcFTSIndexer** utility after any of the following occurs:

- The business objects or properties to be indexed are modified in the Business Modeler IDE and the template is redeployed.
- A new template is added.
- An existing template that affects indexing is updated.
- Data is imported using TC XML utilities.
- A change is made to the **Start Of Week** preference. You can change this preference in the rich client by choosing **Edit→Options→Calendar→Start Of Week**.
- A Classification attribute is added or removed from **search index**.
- A Classification attribute filter property is set to **True**.

You need to evaluate the types of indexing changes to **determine the best approach for updating the index**.

The **initial search index** is performed during installation of the client.

You can **merge the schemas and then choose the method for updating the index**.

To re-index your existing data:

1. If the **runTcFTSIndexer -task=objdata:sync -interval=seconds** command is running, temporarily shut it down by using the following command:

```
runTcFTSIndexer -task=objdata:sync -stop
```

2. Perform one of the following:

- To clear the existing full-text search data and perform a new index, open a Teamcenter command prompt, navigate to **TC_ROOT\TcFTSIndexer\bin**, and type the following commands:

```
runTcFTSIndexer -task=objdata:clear
runTcFTSIndexer -task=objdata:index
```

- To perform a new index without deleting the existing indexed data, type the following command:

```
runTcFTSIndexer -task=objdata:index
```

By default, the indexer uses the current date and time as an end point. To specify a different date and time, use the TcFTSIndexer settings in TEM.

Note:

To index data from the date defined in:

TC_FTS_INDEXER_HOME\conf\TcFtsIndexer_objdata.properties

You must delete **TcFtsIndexerSessionobjdata.cache** from *TC_FTS_INDEXER_HOME\cache*.

3. If you shut down the sync flow, you can now start it again.

Refreshing indexed objects using the **awindexerutil** utility is an alternative to a complete re-index. This utility marks indexed object data as needing a refresh, and then the objects are subsequently reindexed in a synchronization flow. The synchronization flow refreshes the indexed data in batches according to the property value of **objdatasyncstep.refreshBatchSize**. The default is 10,000.

Synchronize the TcFTSIndexer

After re-indexing, the synchronization operation can be run on the TcFTSIndexer at regular intervals to keep indexed data up-to-date, for example:

```
runTcFTSIndexer -task=objdata:sync -interval=seconds
```

Synchronization is run as a single Java SOA client that connects to the pool manager and Solr to achieve delta indexing. Synchronization takes an additional argument for the interval. This interval specifies the waiting period before the synchronization operation can be run again.

Synchronization performs the following steps:

1. Query for objects that failed to be indexed during the previous synchronization. If there are any failed objects, a warning message is written to the logs.

There are *NNN* failed objects. Some of these errors may get fixed in subsequent sync calls. If the errors persist, these errors have to be fixed manually and "indexsyncfailures flow" has to be rerun to index these failures.

In this case, run **TcFtsIndexer** with the **-task=objdata:indexsyncfailures** option. The next sync picks the failed objects and tries to recover them.

2. Query for all revision rule changed objects, and run the list of objects through TIE export and transform. Then load into Solr.
Only the revisions that have been previously indexed are synchronized.
For example, suppose that there are revisions A, B, and C and that only B and C are indexed. Any changes like creating new revisions, deleting existing revisions, or changing the status of existing revisions identifies all its siblings as revision rule impacted and must be synchronized. This step only synchronizes the siblings that exist in the index. In this case, only revisions B and C are synchronized.
3. Query for deleted objects, connect to Solr, and delete the objects.
4. Query for new objects, and run the list of objects through TIE export and transform. Then load into Solr.
5. Query for objects affected by an Access Manager rule change, run the list of objects through TIE export and transform, and load to Solr.
6. Query for modified objects, run the list of objects through transform, and load to Solr.
7. Refresh objects.
During this step the system queries for indexed objects that have exported dates older than the last processed date of the **:FTS_REFRESH** application ID. These objects must be indexed again.
The **refreshBatchSize** property value is configured in the **TcFtsIndexer_objdata.properties** file.
The maximum number of objects that are queried for and indexed is set to **10000** by default. If the number of objects to be refreshed exceeds 10,000 or the value of the limit in the file, the additional objects are handled in the next sync call.
8. Query for replication-pending objects.

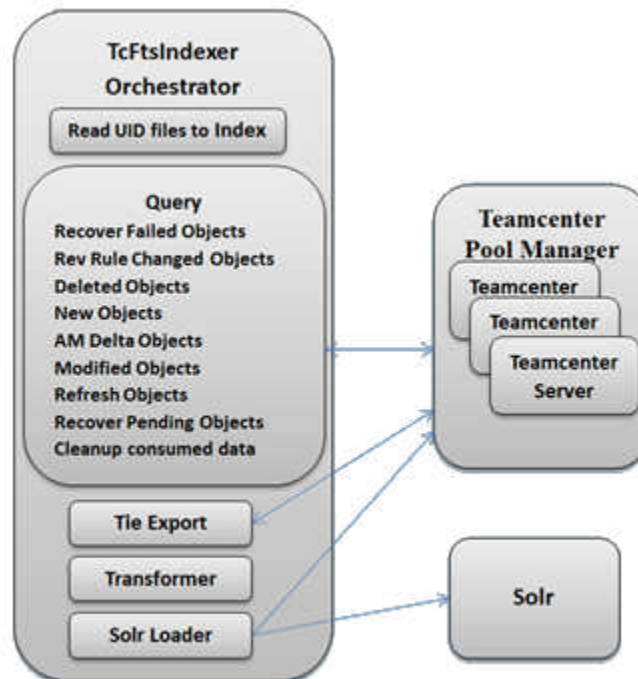
These are the objects that would have failed during TIE export, transform, or load during the previous steps.

Indexing these objects is retried three times. If they continue to fail they are marked as *import failed* and are addressed in next sync call.

9. Clean up the object data that has been consumed by all the applications that have subscribed.

The number of objects being synchronized are output on the console and in the `TC_ROOT\TcFTSIndexer\logs\TcFtsIndexer.log` file. The interval of running synchronization must be defined based on the typical number of objects being handled by the synchronization case. If the number of objects is high, reduce the synchronization interval. Also consider how fast users need to get access to these changed objects.

If a synchronization operation is in progress and it is time for the next scheduled synchronization based on the interval, the system skips the next scheduled synchronization and starts when the current synchronization operation completes.



Index multi-site published objects

You can index shared published records by running the **runTcFTSIndexer** utility. Use the **task=multisite** flow actions of the **runTcFTSIndexer utility** to index published objects. Duplicate published records are removed during indexing.

1. **-task=multisite:test**

Verify whether the environment is set up correctly prior to indexing.

2. **-task=multisite:index**

Run indexing for the time period specified in *TcFtsIndexer_multisite.properties* without clearing the existing index.

3. **-task=multisite:recover**

Run indexing on the failed multi-site objects.

4. **-task=multisite:sync**

Update the multi-site index for changes that happened between the previous run and the current time.

The optional **-interval** argument repeats the **sync** action at the specified interval in seconds. To run **sync** only once, omit the **-interval** argument.

5. **-task=multisite:indexsyncfailures**

Run indexing on the multi-site sync failures which required manual intervention.

6. **-task=multisite:clear**

Clears the existing multi-site publication record indexed data and the cached files.

7. **-task=multisite:indexuids**

Index the UIDs for a specific Object Directory Services (ODS) site. Specify the ODS site name and the path to the *uid.txt* files, for example:

```
runTcFTSIndexer -task=multisite:indexuids ODS_Site_name D:\UID_dir
```

Be sure to **configure search for multiple sites** to make the multi-site replicated data available to users.

Troubleshoot indexing

Find logs related to indexing

Logging is available from several locations for different types of operations being performed. You can set a variety of logging specifications to debug errors and issues.

Note:

To avoid causing performance issues, revert your logging levels and variables back to their previous settings after you complete your debugging. After you finish troubleshooting, be sure to also restart the system you are troubleshooting.

TcFTSIndexer logs

These logs are located in the `TC_ROOT\TcFTSIndexer\logs\` directory. The logs contain messages for object data and structure data as well as framework. These logs provide basic information about the Indexer.

- *TcFTSIndexer.log* contains all the messages related to framework and all the TcFTSIndexer flows.
- *TcFTSIndexer_objdata.log* contains only object data messages related to indexing flows.
- *TcFTSIndexer_structure.log* contains only structure data messages related to flows.

You can set debug logging by changing the logging level.

1. Open the `TC_ROOT\TcFTSIndexer\conf\log4j.properties` file.
2. Set the **DEBUG** logging level:

```
log4j.logger.com.siemens.teamcenter.ftsi=DEBUG
```

3. Restart TcFTSIndexer.

tcserver logs

TcFTSIndexer is a Java SOA client that makes server calls to index data. If an SOA call fails, the **tcserver** and TcFTSIndexer provide limited messages about the failure. When investigating errors:

- Set the logging level to **DEBUG** and run the use case causing the error. Follow the instructions as explained previously in *TcFTSIndexer logs*.
- Check the **tcserver syslog** files. For more information about a server-side failure, find the **syslog** file associated with the SOA call.
The **syslog** files are located in the *Temp* directory. If you do not see them there, check that **TC_KEEP_SYSTEM_LOG** is set to true.
- For debug logging, open the file:

```
TC_LOGGER_CONFIGURATION\logger.properties
```

Change the logging level to **DEBUG** for:

```
logging.rootLogger
logging.logger.Teamcenter
logging.logger.Teamcenter.Soa.Communication
```

- For SQL debug logging, you can set an environment variable.
 1. Open the *tc_profilevars* file.
 2. Set **TC_SQL_DEBUG=PT**.
 3. Restart the Pool Manager and the **tcserver**.
 4. Check the **syslog** file for **DEBUG** messages.
- Check the **Journal** file to get a low level analysis of the **tcserver**. You can trace all the methods that were called by setting the following environment variables:

```
TC_JOURNAL=FULL
TC_JOURNALLING=ON
TC_JOURNAL_LINE_LIMIT=0
```

- Check the *.pjl* file. You can get performance feedback by setting an environment variable:

```
TC_JOURNAL_PERFORMANCE_ONLY=true
```

Restart the Pool Manager and the **tcserver**.

These files can become very large, so run only the use case you are investigating. Journal logging creates a large amount of data that can be difficult to analyze. Restore normal operation by resetting the **TC_JOURNAL_PERFORMANCE_ONLY** environment variable.

Solr logs

If you have Solr failures, find the logs in *TC_ROOT\solr-version\server\logs\solr.log*.

TcFTSIndexer object data staging logs

The staging log files contain information about data issues. These files are located in:

```
Staging_Directory\TcFtsIndexer_objdata
```

The staging directory is defined by the **Staging.Dir** property in:

```
TC_ROOT\TcFTSIndexer\conf\TcFtsIndexer.properties
```


By default, the staging directory location is:

```
TC_ROOT\TcFTSIndexer\working\TcFtsIndexer_objdata
```

- *Tcxml*.xml* files contain the low-level TC XML import export data exported from Teamcenter.
- *Solr_pool_*.xml* files contain the exported data in Solr format to be loaded into Solr.

By default, generated staging files are not deleted if there are errors. You may keep the files when there are no errors:

Open *TC_ROOT\TcFTSIndexer\conf\TcFtsIndexer.properties*, and set:

```
Indexer.keepGeneratedFiles=always
```

TcFTSIndexer object data indexing failures in syslog files

Find object data indexing errors in **syslog** files or in the *TcFtsIndexer_objdata.log*.

- Get the list of TcFTSIndexer **syslog** files by running **runTcFTSIndexer -debug**.
- Open *TC_ROOT\TcFTSIndexer\logs\TcFTSIndexer_objdata.log*.
Search for statements containing **ERROR** –. Look for **syslog** file locations like the following:

Example:

```
2017-02-09 01:55:19,977 ERROR - Please see the log file
'C:\tc_logs\tcserver.exe59d00782.syslog'
on the server 'xyz' for more information.

2017-02-09 01:55:19,978 ERROR - The request XML:
<?xml version="1.0" encoding="UTF-8"?>
<ExportObjectsInput tcGSMMessageId="TCpool-1-thread-104_196181486623316423"
synchronize="false"
xmlns="http://teamcenter.com/Schemas/Internal/GlobalMultiSite/2007-06/
ImportExport"
2017-02-09 01:55:19,978 ERROR - The response XML:
2017-02-09 01:55:19,978 ERROR - SYSLOG FILE: C:\tc_logs\tcserver.exe59d00782.syslog

2017-02-09 01:55:19,981 ERROR - TieExport failed. Files are located in
D:\tc\tcroot\TcFTSIndexer\working\TcFtsIndexer_objdata\
U159c8055a56130e1761114495\result
```

Generated indexing logs

Each indexing operation has a task ID, for example, **U159c8055a56130e1761114495**. If the task ID is created for a failing operation, you can locate the subdirectory within *Staging_Directory* \TcFTSIndexer_objdata to find the generated files.

The error message provides the directory location if there is an error, for example:

```
2017-02-09 01:55:19,981 ERROR - TieExport failed. Files are located in
D:\tc\tcroot\TcFTSIndexer\working\TcFtsIndexer_objdata\U159c8055a56130e1761114495\result
```

If there are no failures, but you still want to see the generated files:

1. Restrict the amount of data by using a smaller time slice for an **index** flow, or by running a **sync** flow after objects that need to be indexed are added, modified, or deleted.
2. Open `TC_ROOT\TcFTSIndexer\conf\TcFTSIndexer.properties`, and set:

```
Indexer.keepGeneratedFiles=always
```

3. Run the **index** flow or **sync** flow.
4. Look for the generated files in `Staging_Directory_path\TcFTSIndexer_objdata` with the date and time corresponding to when the flow ran.

Query error logs

Get the list of TcFTSIndexer **syslog** files by running `runTcFTSIndexer -debug`. Look at the **syslog** file information in the `TcFTSIndexer_objdata.log` file.

For SQL debug logging, open the `tc_profilevars` file.

1. Set `TC_SQL_DEBUG=PT`.
2. Restart the Pool Manager.
3. Check the **syslog** file for **DEBUG** messages.

Export error logs

Get the list of TcFTSIndexer **syslog** files by running `TcFTSIndexer -debug`. Look at **syslog** file information in the `TcFTSIndexer_objdata.log` file.

Transform error logs

Look for errors in `TC_ROOT\TcFTSIndexer\logs\TcFTSIndexer_objdata.log`. If an error refers to a generated file directory, evaluate those log files.

Load error logs

Look for errors in `TC_ROOT\TcFTSIndexer\logs\TcFTSIndexer_objdata.log` and `TC_ROOT\solr-version\server\logs\solr.log` files.

Troubleshoot indexing object data

Check the indexing process

Before troubleshooting problems with object data indexing, be sure you followed the basic indexing process:

1. **Run the initial index flow for object data** using **the runTcFTSIndexer utility**, for example, **runTcFTSIndexer -task=objdata:index**.
2. If there are failures, run the **recover** flow (**runTcFTSIndexer -task=objdata:recover**).
3. Fix the errors by repeating the **recover** flow until most of the failures are addressed.
4. Run the **sync** (synchronization) flow and set the interval, for example, **runTcFTSIndexer -task=objdata:sync -interval=seconds**.
5. After completing the **sync** flow the first time, all the object data that was not indexed is marked **Failed**.

Index failed objects and obtain the logs

Investigate the failed objects from the **sync** flow and the **recover** flow by **looking at the related staging and log files**.

1. Copy files in *Staging Directory\TcFtsIndexer_objdata* and *TC_ROOT\TcFTSIndexer\logs* to a separate directory.
2. Open *TC_ROOT\TcFTSIndexer\conf\TcFtsIndexer.properties*, and set:


```
Indexer.keepGeneratedFiles=always
```
3. Choose the action for your flow:
 - If you are using the **recover** flow, run **runTcFTSIndexer -task=objdata:recover**.
 - If you are using the **sync** flow, run **runTcFTSIndexer -task=objdata:sync -interval=seconds**. After it completes, process failed objects by running **runTcFTSIndexer -task=objdata:indexsyncfailures**.
4. **Find the logs related to TcFTSIndexer generated logs, syslog files, and Solr logs** for evaluation.

Obtain the list of failed objects

After all object data is processed and in a final state (either success or failure), run **the runTcFTSIndexer utility** using the **show** flow to get a list of failed objects. Specify a code for the object state and an output directory.

```
runTcFTSIndexer -task=objdata:show 3 d:\TcFTSIndexer\uid_output
```

In this example, state **3** specifies **indexing failed** and the output directory contains the *uid.txt* and *uid_prop.txt* files listing the objects:

- *uid.txt*
Contains the UIDs of the objects. You can synchronize the objects again using **objdata:sync uid_file_dir**, for example:

```
runTcFTSIndexer.bat -task=objdata:sync d:\TcFTSIndexer\uid_output
```

- *uid_prop.txt*
Contains UIDs in the output format *UID|object_string|object_type*. For example:

```
TRa3S5qd$DyB | Breaker Panel Anchor Plate/AP02-A | Physical Part Revision
```

Troubleshoot TcFTSIndexer

Login errors

You may encounter an error if the environment is configured for SSO when you run **the runTcFTSIndexer utility**. If you get an invalid user ID or password error, check the following:

- Teamcenter client communication system SSO App IDs may not be configured correctly. You can configure multiple application IDs using the **Environment Settings for Client Communication System** panel in Teamcenter Environment Manager (TEM).
- Ensure that the user running the **runTcFTSIndexer** utility is authorized in your authentication system and has read access to the datasets and their associated files to be indexed.
- Check whether the user is defined in the **Tc.user** setting in the *TC_ROOT\TcFTSIndexer\conf\TcFtsIndexer.properties* file.
- If you are using an encrypted password, be sure it was created correctly. You can **change the user running the TcFTSIndexer**.

Indexer access errors

- The Indexer is not accessible or the schema is not correct.
Merge the Teamcenter and Solr schemas.

`SOLR_HOME\TcSchemaToSolrSchemaTransform.batTC_DATA\ftsi\solr_schema_files`

- Posts to Solr are returned with HTTP response code **401**. Solr credentials must match for Teamcenter, Indexing Engine (Solr), and the TcFTSIndexer. You can update Solr credentials if needed.

Database table errors

- If you find indexing **errors in the syslog files** that refer to errors like the following examples, you have an outdated global temporary table called **TIE_CLSF_DATA** in the Teamcenter database from an earlier release.
- The syslog reports a problem with an extra column in the database global temporary table called **ICM0UNITSYSTEM**.
- The syslog reports an error like the following:

```
#####inside EIM_not_ok#####

sqlca_error_code is -904
EIM_db_error_code is 545001
ROLLBACK;
```

You can correct the issue by removing the **TIE_CLSF_DATA** table from the database. Stop the indexing or synchronization process, remove the table from the database, and then resume indexing. A new temporary table is correctly created.

- The syslog reports an error during indexing synchronization for email notification like the following:

```
Error running Cleanup Scratch Table
Unable to open session with mail server
Failed to send OS Mail notification
```

The error is related to a change to the database subscription table during synchronization. However, the error only applies to email notification. You may take one of the following actions:

- Safely ignore the error, as it doesn't interfere with the synchronization operation.
- Choose to suppress email notifications (for example, if no email server or email recipient is configured at your site), by setting **Subscription_Table_Notify_Level=2** (see the Teamcenter help topic *Maintain data synchronization through site consolidation*).

Dataset download errors

- If the dataset is missing files, run **bmide_modeltool** to generate the files, and upload them to the dataset. If the files are not generated or uploaded, consult the **bmide_modeltool** logs. For information, see *Business Modeler IDE* in the Teamcenter help.

- File Management System issues can cause problems in downloading files. Make sure that FMS is configured correctly. For information, see *File Management System* in the Teamcenter help.

Dataset text file encoding errors

If text file content is not indexed, check the file encoding. Text files using UTF-8 encoding are indexed properly. If the text file uses another type of encoding, *Shift-JIS* for example, convert the file to UTF-8.

Dataset files causing memory errors

If you get out of memory errors when indexing an unusually large file, you can **check the TcFTSIndexer logs** for messages related to the Tika Parser. If your error seems related to the Tika Parser, you can set the maximum number of Tika parsers that run during indexing.

In *TcFTSIndexer\conf\TcFtsIndexer_objdata.properties*, set **objdataloadstep.maxTikaParserCount=n**.

Set a maximum number to reduce the amount of data that is parsed at one time. You can iteratively reduce the value until the data parses without causing an out of memory error. The default, which is no value, means no limit.

Troubleshoot indexing performance

You can gather and analyze timing information. You can also evaluate the connection between Teamcenter and the Indexer.

Collect performance information

Initial indexing provides a summary of the duration for each operation. For example:

```

2017-03-01 18:23:07,358 INFO - Step Summary
2017-03-01 18:23:07,358 INFO -   objdataquerystep
2017-03-01 18:23:07,359 INFO -     Status: Created: 0   Started: 0   Done: 189
Error: 0
2017-03-01 18:23:07,359 INFO -     Total Time   68.32   Total Count 90899

2017-03-01 18:23:07,359 INFO -   objdataexportstep
2017-03-01 18:23:07,359 INFO -     Status: Created: 0   Started: 0   Done: 125
Error: 0
2017-03-01 18:23:07,359 INFO -     Total Time 19092.57   Total Count 90899

2017-03-01 18:23:07,360 INFO -   objdatasaxtransformstep
2017-03-01 18:23:07,360 INFO -     Status: Created: 0   Started: 0   Done: 125
Error: 0
2017-03-01 18:23:07,360 INFO -     Total Time   213.68   Total Count 90899

2017-03-01 18:23:07,360 INFO -   objdataloadstep
2017-03-01 18:23:07,361 INFO -     Status: Created: 0   Started: 0   Done: 125
Error: 0
2017-03-01 18:23:07,361 INFO -     Total Time   133.21   Total Count 90899

```

You can add the **-status** argument when you run **runTcFTSIndexer -status -task=objdata:index** to report the performance and status for all the operations in the **index** flow.

Running the **runTcFTSIndexer -task=objdata:sync** flow reports the times for each query and for indexing the objects found during the query.

Optimize the Indexer

The connections between Teamcenter and the Indexer are set when you **install Indexer**. You set the connections between Teamcenter and the Indexer after installation when you **optimize instances of TcFTSIndexer**.

You can also adjust the maximum number of connections in the **Tc.maxConnections** property in the **Tcftsindexer\conf\TcFtsIndexer.properties** file.

You can tune the number of warmed servers in the pool using the **PROCESS_WARM** parameter in the **serverPool.properties** file. For more information, see *System Administration* in the Teamcenter help.

Check preference values

The **AW_FTSIndexer_skip_modifications_via_relations** preference controls whether the TcFTSIndexer can find modifications by running relation queries during the TcFTSIndexer synchronization flow. The default value is **true**, which skips the queries. Check whether the preference value is set to **false**. For example, if **alternate ID on Item is being indexed** or **a dataset object attached to an object of Document Revision type** (or any of its subtypes) is already indexed, the preference value could be set to **false**.

Slow synchronization after upgrade from a patch

After you upgrade the Teamcenter platform from a previous patch version, the first indexing synchronization of object data may take a long time to complete if the system is not configured properly. The issue may occur because:

- The synchronization operation relies on a threshold value specified by the **TC_TIMESTAMP_THRESHOLD** POM parameter of the **install** utility. This is the time that the **POM_timestamp** table holds timestamps. The default is 96 hours if not set. (Object data indexing relies on the TC XML export that occurs on the server.)
- If the time between two synchronization operations exceeds the threshold value (for example, when a system is down during an upgrade):
 - The synchronization logic uses the POM object table to identify the synchronization candidates that may take longer if the size of the POM object table is large.
 - In the case of an upgrade, If the time between two synchronization operations exceeds the threshold, you may see the issue.

- If the time between synchronizations is less than the threshold (for example, normal operating conditions):
 - The synchronization logic uses the **POM_timestamp** table because it is expected to be smaller and to run more quickly.
 - Typically, the synchronization interval is set to 300 seconds at the start of synchronization, which would be less than the threshold. Therefore, you would not encounter the issue.

You can resolve this issue when upgrading by temporarily setting the threshold value higher before the upgrade.

1. View the current value by running **install -ask_pom_param**.
2. Set a higher value by running **install -set_pom_param**.
3. After the first synchronization operation, reset the parameter to its previous value.

For directions about how to run the **install** utility, refer to the Utilities Reference in the Teamcenter help.

Update Solr credentials

If you need to update the user name and password for the Solr indexing engine, you can use Teamcenter Environment Manager (TEM) in maintenance mode to make the changes. The Solr user name and password must be updated in several locations. Credentials must match for the Solr indexing engine, the TcFTSIndexer, and Server Extensions.

1. Launch TEM from your installed environment.
2. In the **Maintenance** panel, select **Configuration Manager**.
3. In the **Configuration Maintenance** panel, select **Perform maintenance on an existing configuration**.
4. In the **Old Configuration** panel, select the configuration where the Active Workspace components are installed.
5. The options in the **Feature Maintenance** panel are based on what is installed in the selected configuration.

Select **Active Workspace Indexing Engine**→**Update Indexing Engine user credentials** to update the credentials for Solr.

Select **Active Workspace Indexer**→**Update search engine user credentials** to update the credentials for TcFTSIndexer.

Select **Data Model**→**Update Indexing Engine user credentials** to update the Solr credentials for each Teamcenter server with Server Extensions features installed.

Troubleshoot Solr issues

Solr logs are located in *TC_ROOT\solr-version\server\logs\solr.log*.

Solr authentication

If an error occurs during JSON parsing, you may see an `Unknown value type` error. Verify that the Solr password is set correctly in **tcservers**.

Solr credentials must match for Teamcenter, Solr Indexing Engine, and TcFTSIndexer. **Update Solr credentials**.

Solr schema issues

If the Solr schema is outdated, the *solr.log* contains information about the field having issues. Be sure you **merge the Teamcenter and Solr schemas**.

```
SOLR_HOME\TcSchemaToSolrSchemaTransform.bat TC_DATA\ftsi\solr_schema_files
```

You can verify the Solr schema:

1. Examine the Solr schema in the Solr administrative panel and find the Teamcenter schema that starts with **TC_0Y0_***.
2. Run the **bmide_extractor** utility to extract the data model and verify the data model changes.
3. Run the **preferences_manager** utility to extract the preferences and verify the Active Workspace preferences.
4. Be sure to restart Solr.

Missing filters or other schema issues

Check for the following:

- Be sure that the data was indexed.
- Ensure Solr was started after **merging the Teamcenter schema with the Solr schema**.
- Ensure that all of the necessary data model changes were made in the Business Modeler IDE.

- If data model changes were deployed to Teamcenter using Hot Deploy or Live Update, ensure that the **bmide_modeltool** utility ran in a Teamcenter command window that has **TC_ROOT** and **TC_DATA** defined. For example:

```
bmide_modeltool.bat -u=username -p=password -g=dba -tool=all
-mode=upgrade -target_dir="TC_DATA"
```

Solr memory issues

Be sure you understand the [prerequisites for indexing deployment](#) and [how to optimize your hardware for indexing](#).

If you encounter Solr memory issues, you can increase the Java heap size. You can calculate a maximum value as one quarter of the server memory capacity. For example, if the Solr server has 32 GB of memory, you could set the maximum heap size to 8 GB.

Open the script:

```
SOLR_HOME\bin\bin\solr.in.sh or cmd
```

Set the Java memory value as follows:

```
SOLR_JAVA_MEM="-Xmssize -Xmxsize"
```

-Xms specifies the initial Java heap size. **-Xmx** specifies the maximum heap size.

Specify the *size* value in megabytes using **m** or **M** or in gigabytes using **g** or **G**.

Example:

```
SOLR_JAVA_MEM="-Xms8g -Xmx10g"
```

If you are running Solr as a Windows service with AdoptOpenJDK, you may run into memory issues. For the Solr Windows service, the heap size is not set and defaults to 256 MB. You can increase the maximum heap size using **-Xmx**.

Open the script:

```
SOLR_HOME\solrWinService.bat
```

Find the line that begins with:

```
windowsService.exe %PARAM1% -service=%SERVICENAME%
```

Add the **-Xmxng** specification to the end of the line, for example, **-Xmx8g** for 8 GB. Save the file and restart the service.

Note:

Oracle OpenJDK is not affected by this issue.

Solr Java version issue

You may encounter a Java version issue for Solr during startup on Solaris. You can update the version referenced in the startup script if **runSolr.sh** returns an error such as the following:

```
awk: can't open /version/ {print $2}
Your current version of Java is too old to run this version of Solr.
```

Open `SOLR_HOME\bin\solr.sh` and find the **JAVA_VER_NUM** line. It appears as follows:

```
JAVA_VER_NUM=$(echo $JAVA_VER | head -1 |
awk -F "" '/version/ {print $2}' | sed -e's/^1\./' | sed -e's/[._-].*$//')
```

Replace the value with the Java version number:

```
JAVA_VER_NUM=Java version
```

Be sure to enter the major Java version number (such as **8**). Do not enter the version string (such as **1.8.0**).

SolrCloud

If SolrCloud does not start when you upload the `schema.xml` file to an external Zookeeper, you may see the error:

```
org.apache.solr.cloud.SolrZkServer ZooKeeper Server ERROR java.io.IOException:
Unreasonable length = 1070263 message
```

The unreasonable length refers to the maximum size Zookeeper allows for the `schema.xml` file. By default, the maximum size is 1 MB.

The error occurs when `SOLR_HOME\server\solr\collection1\conf\schema.xml` is larger than the limit allowed by the **jute.maxbuffer** system property,

To resolve the issue, increase the Zookeeper data limit. The following example sets it to 2 MB.

1. Stop Zookeeper and delete the `zoo_data` cache.
2. Change the limit. In the example, **-Djute.maxbuffer=2097152** sets the limit to 2 MB.

```
call %JAVA% -Djute.maxbuffer=2097152 "-Dzookeeper.log.dir=%ZOO_LOG_DIR%"  
"-Dzookeeper.root.logger=%ZOO_LOG4J_PROP%" -cp "%CLASSPATH%" %ZOOMAIN% "%ZOOCFG  
%" %*
```

3. Change the value of **jute.maxbuffer** in the `ZOOKEEPER_HOME\bin\zkServer.cmd` script.

Using indexing utilities

Indexing utilities

Use the **runTcFTSIndexer** utility to index data using task flows.

Use the **awindexerutil** utility to refresh the index of indexed objects when you run the synchronization flow.

awindexerutil

Refreshes indexed objects if you make changes to them and you want the synchronization flow to refresh the index for those objects. The **awindexerutil** utility marks those objects to be picked up during the next synchronization flow batch. This allows you to update the index without the downtime of a full index flow.

You can refresh your index for only the delta of changes since the last completed synchronization. You can index changes for types and properties that have been added, modified, or removed, by performing a delta update when you **remerge Solr and Teamcenter schemas and update the index**.

Running **awindexerutil** does not interfere with current synchronization flows.

Run the utility from the *TC_ROOT\bin* directory (*TC_BIN* if it's set).

SYNTAX

awindexerutil -u=user-id -p=password -g=group [-refresh] [-delta [-dryrun] [-daterange]] -h

ARGUMENTS

-u

Specifies the user ID. The user needs administration privileges.

Note:

If Security Services single sign-on (SSO) is enabled for your server, the user and password arguments are authenticated externally through SSO rather than against the Teamcenter database. If you do not supply these arguments, the utility attempts to join an existing SSO session. If no session is found, you are prompted to enter a user ID and password.

-p

Specifies the password.

-g

Specifies the group associated with the user.

-refresh

Starts the synchronization flow according to the batch size.

-delta

Updates the index for the delta of changes to object types and properties since the last completed synchronization. Be sure to run **-delta -dryrun** to evaluate the changes before running the **-delta** index update.

Use this approach to remerge Solr and Teamcenter schemas and to update the index.

-dryrun

Use with **-delta** to compare the changes in object types and properties for delta reindexing. No indexing is performed. The differences are output to the command window and a log file. The *.log* file path is returned after the operation is complete.

-daterange

Use with **-delta** or **-delta -dryrun** to set a date range for object types and properties that can be indexed in the updated schema. To set the date range, use the following year-month-day form:

YYYY/mm/dd HH:MM:SS-YYYY/mm/dd HH:MM:SS

The time (**HH:MM:SS**) is optional. You can specify multiple ranges in a comma-separated list. If the specified date range contains spaces, enclose the entire specification in quotation marks.

-classification

Used together with the **-delta** argument, evaluates the schema files and marks the changes in classification data for indexing in the next scheduled indexing synchronization run. Run the utility with the **-classification -delta -dryrun** arguments to receive a list of changes that will be marked for indexing when the **awindexerutil** is run.

-h

Displays help for this utility.

EXAMPLES

- Start the synchronization flow to refresh data objects:

```
awindexerutil -u=admin -p=pwd -g=group -refresh
```

- Start the dry run comparison of the delta of indexing changes. All instances of new, modified, and deleted types and properties are identified. For newly indexed types, only those instances that were changed during the specified date range (2012/12/12 - 2016/12/31) are included. Only a report is returned. No indexing changes are made.

```
awindexerutil -u=admin -p=pwd -g=group -delta -dryrun
-daterange=2012/12/12-2016/12/31
```

- Start reindexing the delta of changes. Updates all new, modified, and deleted types and properties.

```
awindexerutil -u=admin -p=pwd -g=group -delta
```

- Start reindexing the delta of changes. Updates all new, modified, and deleted types and properties. For types that are newly added, only those objects last modified in the specified date range are indexed.

```
awindexerutil -u=admin -p=pwd -g=group -delta -daterange=  
  "2015/12/12 15:30:00 - 2015/12/31 22:00:00, 2016/02/01 07:00:00  
  - 2016/12/31 14:00:00"
```

- Mark changes in classification data for indexing:

```
awindexerutil -u=admin -p=pwd -g=group -delta -classification
```

runTcFTSIndexer

Indexes data into the Solr indexing engine. Run this command from the *FTS_INDEXER_HOME* directory, for example, *TC_ROOT\TcFTSIndexer\bin*.

SYNTAX

runTcFTSIndexer -debug -maxconnections -status -stop -service -shutdown -task=[objdata | multisite | structure | fourgd]:flow-action -h

Operating system considerations:

Linux A *.sh* extension is required, for example, *./runTcFTSIndexer.sh*.

If a UID contains a special character, you must enclose it in straight, single quotation marks. UIDs can contain **A-Z, a-z, 0-9, \$, and _**.

Windows The *.bat* extension is not required for **runTcFTSIndexer**.

For UIDs on Windows, do not enclose in straight, single quotation marks.

ARGUMENTS

-debug

Reports a summary of the flow in progress, including connections and the logs associated with them, to the command window and to the TcFtsIndexer logs in **TcFTSIndexer\logs**. Run **-debug** in a separate command window.

```
runTcFTSIndexer -debug
```

-maxconnections

Sets a new value for maximum **tcserver** connections to use at any given time, for example:

```
runTcFTSIndexer -maxconnections=5
```

-status

Checks the status of the indexer and reports the flows running in the indexer. For example, the following shows the status for all the flows:

```
runTcFTSIndexer -status
```

This argument can also be used with the **-task** argument. For example, the following command shows the status of the **objdata:index** flow:

```
runTcFTSIndexer -status -task=objdata:index
```


-stop

Stops indexing flows that use an interval.

```
runTcFTSIndexer -stop
```

This argument can also be used with the **-task** argument. For example, the following command stops the **objdata:sync** flow with intervals:

```
runTcFTSIndexer -stop -task=objdata:sync
```

-service

Starts the indexer as a Java Remote Method Invocation (RMI) service in a console, for example:

```
runTcFTSIndexer -service
```

This argument can also be used with the **-task** argument to run a flow when starting the service, for example:

```
runTcFTSIndexer -service -task=objdata:index
```

-shutdown

Shuts down the service after stopping all the flows.

```
runTcFTSIndexer -shutdown
```

-task

Runs an indexing task in this format:

```
-task=type:flow-action
```

Flow actions are specific to the type of task.

objdata	Indexing actions to support object data indexing
admin	Administrative actions to support clone and merge indexing
fourgd	Indexing actions to support 4GD
multisite	Indexing actions to support multi-site published records
structure	Indexing actions to support structured content indexing

-h

Displays the help for this utility.

OBJDATA FLOW ACTIONS

- **objdata:clear**

Clears existing indexed data, records, and cached files. This option is most often used prior to running **objdata:index**. When you run **-task=objdata:clear**, specify one of these options when you are prompted:

1 clears the indexer cache.

2 clears the Solr index.

3 clears the object data indexing records from the Accountability table.

4 clears the data covered by options 1, 2, and 3.

5 clears the UIDs passed in from the Accountability table.

When you specify **5**, provide the full path to the text file containing the UIDs.

6 cancels the clear flow.

7 cleans up the scratch table records, where the last saved date precedes the last processed date of the subscription table.

If you run **-task=objdata:clear** in an environment that is set to **Clone Ready**, it fails in order to prevent accidentally clearing the index.

Example:

To clear a list of UIDs, run the command:

```
runTcFTSIndexer -task=objdata:clear
```

At the prompt:

```
Please provide a clear option from above list [1-7]:
```

Enter **5**. The prompt returns:

```
Provide full path to input uid text file.
```

Enter the full path and file name:

```
c:\my_uid_directory\uids.txt
```

- **objdata:errors**

Returns the errors for indexing failures to a specified directory. The directory must be empty. You can optionally specify the number of errors to be returned. The default is 50 errors. The error directory contains a directory for each UID of the failed object. Each UID directory contains the following information:

- Properties such as object name and type provide information on the failed objects.
- The TC XML and Solr XML files are generated and saved for debugging.
- Syslog information.
- TcFtsIndexer log files with the errors.

In this example, the first 100 errors are sent to the specified output directory:

```
runTcFTSIndexer -task=objdata:errors d:\TcFTSIndexer\errors 100
```

- **objdata:index**

Performs indexing for the time period specified in *TC_ROOT\TcFTSIndexer\conf\TcFtsIndexer_objdata.properties* without clearing the existing index.

For a clean start, first use **objdata:clear** to clear indexed data and cached files. However, if you run it in a **Clone Ready** environment, it fails in order to prevent accidentally clearing the index.

- **objdata: index clone**

Indexes the specified time period without clearing the existing index. The start time is set to when the **Clone Ready** state was set. The end time is set in the *confobjdata.properties* file.

This indexing flow supports the clone and merge indexing approach, which is a method to manage upgrades and patches more efficiently by reindexing a cloned environment and then merging it to an updated production environment.

The **objdata: index clone** task picks up the delta of indexing changes in the production environment that occurred while the cloned environment was being indexed. This flow depends on the **administrative flow actions**.

- **objdata:indexsyncfailures**

Indexes synchronization failures that required manual intervention.

- **objdata:recover**

Recovers failed indexed objects.

If failures are reported during initial indexing, run the recover flow action after initial indexing completes. Recover failures from the initial index by running **objdata:recover** repeatedly until there are no failures reported or the failures are consistent and need to be fixed.

You can choose to fix the errors or leave them for later and proceed with the synchronization flow. If you leave them, these errors are logged as failures during synchronization. You can return at a later time to fix them. The recover flow action operates by time slice. The recover flow action processes all failed objects together.

- **objdata:show**

Returns objects that are in a specified indexing state in two text files. Specify the state code and a directory for the output files using the form **-task=objdata:show** *n uid_file_dir*. The **show** number to specify the state is **1**, **2**, or **3**:

1 returns objects in the replication pending state

2 returns objects in the indexing complete state

3 returns objects in the indexing failed state

In this example, the output files contain the objects that failed indexing:

```
runTcFTSIndexer -task=objdata:show 3 d:\TcFTSIndexer\uid_output
```

The output returns two text files:

- *uid.txt*
Contains the UIDs of the objects that match the specified state. Each UID is on one line. You can synchronize the objects again using **objdata:sync** *uid_file_dir*. Specify the directory containing the *uid.txt* file, for example:

```
runTcFTSIndexer.bat -task=objdata:sync d:\TcFTSIndexer\uid_output
```

- *uid_prop.txt*
Contains UIDs in the output format *UID | object_string | object_type*, for example:

```
TRa3S5qdSDyB | Breaker Panel Anchor Plate/AP02-A | Physical Part Revision
```

- **objdata:sync**
Updates the index with changes to data between the previous run and the current run.
- The **sync** action can take the **-interval=seconds** argument.
For example, to synchronize object data using the stand-alone indexer every 300 seconds (5 minutes):

```
runTcFTSIndexer -task=objdata:sync -interval=300
```

The value must be greater than **0**.
To run **sync** once, omit **-interval=seconds**.

- The **sync** action can take a *file path* to a *uid.txt* file.
You can synchronize the objects again using **objdata:sync** *filepath*. Specify the directory containing the *uid.txt* file, for example:

```
runTcFTSIndexer.bat -task=objdata:sync d:\TcFTSIndexer\uid_output
```

- **objdata:test**

Verifies whether the environment is set up correctly prior to running the indexer.

ADMINISTRATIVE FLOW ACTIONS

- **admin:cloneready**
Sets the production environment to **Clone Ready** in preparation for creating a cloned environment. After this action is applied, the cloned environment can be upgraded or patched and then indexed using **objdata:index**.
- **admin:clonevalidate**
Confirms that the indexing information in the upgraded production environment and the upgraded cloned environment are the same.
Run this flow action to determine whether the production environment is ready for the indexing system to be merged.
After the merge, pick up the delta of indexing changes in the upgraded production environment using **objdata:index clone**.
- **admin:clonecomplete**
Removes the **Clone Ready** setting on the indexing system in the upgraded production environment. The environment must have been previously set to **Clone Ready**.
- **admin:status**
Provides the means to verify the current clone status of the production environment.

MULTI-SITE FLOW ACTIONS

- **multisite:clear**
Clear indexed data and cached files for existing multi-site published records.
Use prior to running **multisite:index**.
- **multisite:index**
Without clearing the existing indexed multi-site published records, index for the time period specified in the `TC_ROOT\TcFTSIndexer\conf\TcFtsIndexer_multisite.properties` file. For example:

```
runTcFTSIndexer -task:multisite:index
```

During indexing, duplicate published records are removed.

For a clean start, use **multisite:clear** to clear indexed data for multi-site published records and cached files.

- **multisite:indexuids**
Index the UIDs for a specific Object Directory Services (ODS) site.
Specify the ODS site name and the directory containing the `uid.txt` file, for example:

```
runTcFTSIndexer -task=multisite:indexuids ODS_Site_name D:\UID_dir
```

- **multisite:indexsyncfailures**

Index synchronization failures that required manual intervention.

- **multisite:sync**

Updates the index with changes to published record data between the previous run and the current run.

The **sync** action can take the **-interval=seconds** argument to repeat the **sync** action at the specified interval. For example, to synchronize multi-site published record data every 300 seconds (5 minutes):

```
runTcFTSIndexer -task=multisite:sync -interval=300
```

The value must be greater than 0.

To run **multisite:sync** once, omit **-interval=seconds**.

- **multisite:recover**

Recovers failed indexed published record objects.

If some published records fail during indexing, run the recover flow action after multi-site indexing completes. Recover failures by running **multisite:recover** repeatedly until there are no failures reported or the failures are consistent and need to be fixed.

- **multisite:test**

Verify whether the environment is set up correctly prior to running the multi-site indexer.

STRUCTURE FLOW ACTIONS

- **structure:recoverfailures**

Changes all product configurations with failed states to the **ReadyToIndex** state or the **MarkedForDeletion** state. For example:

```
runTcFTSIndexer -task=structure:recoverfailures
```

- **structure:reset *product-configuration-UID***

Resets the given product configuration UID setting to the **ReadyToIndex** state or the **MarkedForDeletion** state. For example:

```
runTcFTSIndexer -task=structure:reset 'goZRkWxoqd$DyB'
```

- **structure:resetall**

Downloads the latest transform and schema files, resets all active product configurations to the **ReadyToIndex** state, and resets all deleted product configurations back to the **MarkedForDeletion** state. For example:

```
runTcFTSIndexer -task=structure:resetall
```

- **structure:show**

Shows a summary of all configured product configurations, for example:

```
runTcFTSIndexer -task=structure:show
```

- **structure:sync**

Queues synchronization and delete actions for all product configurations, for example:

```
runTcFTSIndexer -task=structure:sync
```

- **structure:syncone *product-configuration-UID***

Queues synchronization and delete actions for a single product configuration UID, for example:

```
runTcFTSIndexer -task=structure:syncone goZRkWxoqd12DyB
```

- **structure:test**

Verifies that the environment is set up correctly prior to running the indexer, for example:

```
runTcFTSIndexer -task=structure:test
```

4GD FLOW ACTIONS

The 4GD product structure is initially indexed using the **bomindex_admin** utility.

- **fourgd:reset *product-configuration-UID***

Resets the given 4GD product configuration UID setting to the **ReadyToIndex** state. For example:

```
runTcFTSIndexer -task=fourgd:reset goZRkWxoqd$DyB
```

- **fourgd:resetall**

Downloads the latest transform and schema files, resets all active 4GD product configurations to the **ReadyToIndex** state. For example:

```
runTcFTSIndexer -task=fourgd:resetall
```

- **fourgd:show**

Shows a summary of all 4GD configured product configurations, for example:

```
runTcFTSIndexer -task=fourgd:show
```

- **fourgd:sync**

Queues synchronization and delete actions for all 4GD product configurations, for example:

```
runTcFTSIndexer -task=fourgd:sync
```

- **fourgd:syncone *product-configuration-UID***

Queues synchronization and delete actions for a single 4GD product configuration UID, for example:

```
runTcFTSIndexer -task=fourgd:syncone goZRkWxoqd12DyB
```

Customize indexing

Overview of indexer customization

TcFTSIndexer is a Java application that can run types, flows, and steps.

TcFTSIndexer:

- Is an SOA client that connects to Teamcenter to extract data and index the data into Solr.
- Allows modification of any existing steps and flows to meet customer requirements.
- Can be customized to extract external system data and index into Solr.
- Provides utilities that can be used in step customization.
Details of these utilities are in Javadocs available in the `TC_ROOT\TcFTSIndexer\docs\javadocs` directory.

Indexer customization prerequisites

- A working TcFTSIndexer Installation in stand-alone mode.
- A high-level understanding of TcFTSIndexer architecture.
- An understanding of input and output objects associated with each step in a flow that is being customized.
- An understanding of properties associated with the flow.
- Review sample code for steps in the `TC_ROOT\TcFTSIndexer\sample` directory.
- Refer to Javadocs for published methods and classes discussed.
- Refer to the `TC_ROOT\TcFTSIndexer\sample\TcFtsIndexer_sample1.properties` files and the `TC_ROOT\TcFTSIndexer\conf\TcFtsIndexer_objdata.properties` files for example configurations.
- For the new requirements, create a high-level design of the functionality and:
 - Create a list of steps with their input and output objects defined. Check if there are existing steps that can be reused.
 - Chain these steps to create a new flow or modify an existing flow.
 - Determine if the flow is part of an existing type or a new type.

TcFTSIndexer installation layout

TcFTSIndexer installation has the following directory structure (located at *TC_ROOT\TcFTSIndexer*):

- **bin**
Contains scripts to start up TcFTSIndexer.
- **cache**
Stores all relevant cache files.
- **conf**
Contains all the configuration files.
- **docs**
Contains Javadocs for published APIs.
- **lib**
Contains all the JAR files needed to run TcFTSIndexer.
- **logs**
Contains the indexer log files.
- **sample**
Contains sample files to help with customizations.

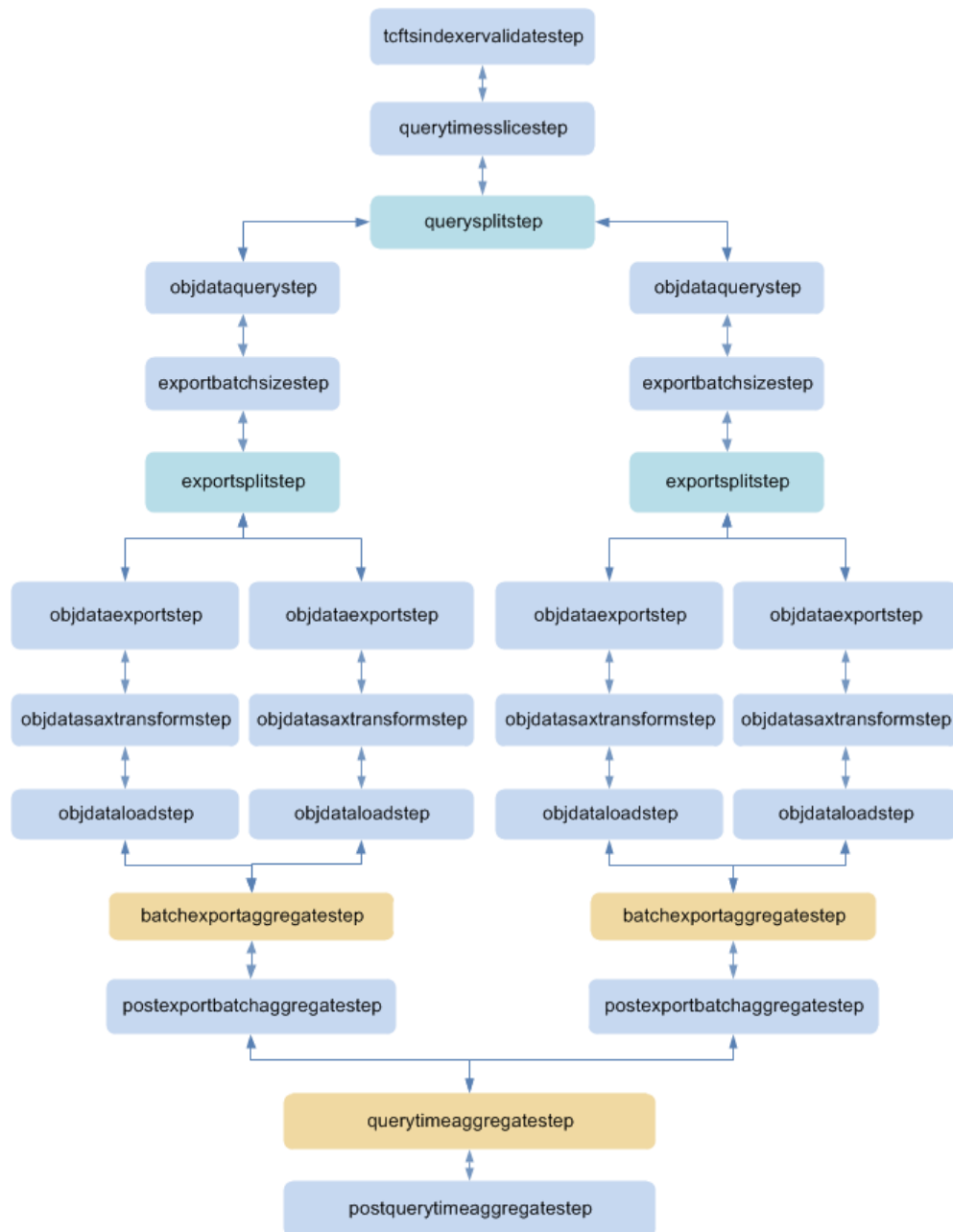
TcFTSIndexer extensibility framework

TcFTSIndexer is a Java application that runs types, flows, and steps.

- **Types**
Represent the different integrations or customizations into TcFTSIndexer, for example, object data and structure. Types contain flows.
- **Flows**
Represent the supported operations for a given type. For example, some flows for object data (**ObjData**) are clear, index, recover, and synchronization. Flows contain steps that are chained together.
- **Steps**
Contain methods that define a certain behavior. Each step should have the input and outputs defined. These steps can be reused across different flows and types based on the input and output requirements. Steps are run in sequence as defined in a flow. Output of one step becomes the input of the next step.
For example, the object data (**ObjData**) index flow has query, TIE export, transform, and load steps. There are three types of steps:

- Simple step
Runs a step based on the input and returns the output data for the next step to process.
- Split step
Splits a list of input data into multiple simple steps based on the size of the list. In the case of object data (**ObjData**), the query is split into time slices using this step. A flow can have multiple split steps.
- Aggregate step
Waits on all the split steps to finish and combines all the output data from the processing of split steps. In a flow, every split step has to have an associated aggregate step.

Object data indexing steps



Following are steps in the ObjData index flow. These steps are reused in the **ObjData** recover and synchronization flows. Input and output objects related to each step are available in Javadocs.

- **tcftsindexvalidatestep**
Validates the configuration, connects to pool manager, Solr, and Dispatcher in dispatcher mode.
- **querytimesslicestep**

Breaks down the start and end time into smaller configurable time slices and creates a list of time slices.

- **quersplitstep**
Is a split step that uses the list from the **querytimeslicestep** step and creates multiple **objdataquerystep** steps. The number of **objdataquerystep** steps matches the list size.
- **objdataquerystep**
Queries for all indexable objects in the time slice and returns a list of UIDs to index.
- **exportbatchsizestep**
Breaks down the list of UIDs to a configurable batch size and creates a list of configured batch size UIDs.
- **exportsplitstep**
Splits a list of configured batch size UIDs into multiple **objdataexportstep** steps.
- **objdataexportstep**
Calls TIE export using the UIDs from the **exportsplitstep** step and creates the Teamcenter XML file.
- **objdatasaxtransformstep**
Converts the Teamcenter XML file to a Solr XML file.
- **objdataloadstep**
Loads the Solr XML into Solr and calls to confirm export.
- **batchexportaggregatestep**
Waits on all steps to be processed that are created by the **exportsplitstep** split step. Essentially it waits on all UIDs for a given time slice to be indexed.
- **postexportbatchaggregatestep**
Updates the time slice as success in the cache file.
- **querytimeaggregatestep**
Waits on all time slices to be finished. It waits on all splits created by the **quersplitstep** step to be processed.
- **postquerytimeaggregatestep**
Creates the Solr suggestions after all the objects are indexed.

Run a sample search indexing flow

TcFTSIndexer comes with a sample flow. This sample flow can be run using the following steps:

1. Copy the `TC_ROOT\TcFTSIndexer\sample\TcFtsIndexer_sample1.properties` file to the `TC_ROOT\TcFTSIndexer\conf` directory.

2. Run the command:

```
runTcFTSIndexer -task=sample1:sampleflow1
```

This sample flow contains:

- Four simple step implementation classes (**SampleStep1.java**, **SampleStep2.java**, **SampleStep3.java**, and **SampleStep4.java**). Code is available in the `TC_ROOT\TcFTSIndexer\sample` directory.
- One split step.
- One aggregate step.
- The **SampleStep4.java** file that is reused to create a new **samplestep5** step.

This flow is chained as follows:

1. **samplestep1**

This simple step creates a list of lists (**List<List<String>>**) and adds it to the message data. This message data list has four lists that contain sample strings of size 3. This message data list is set as tracking information in the **StepInfo** message to show status. Following is the sample step:

```
[Batch 1 MD 1, Batch 1 MD 2, Batch 1 MD 3,
Batch 2 MD 1, Batch 2 MD 2, Batch 2 MD 3,
Batch 3 MD 1, Batch 3 MD 2, Batch 3 MD 3,
Batch 4 MD 1, Batch 4 MD 2, Batch 4 MD 3]
```

2. **samplesplitstep**

This is a split step that creates four threads and these threads call the **samplestep2** step with message data input as the inner list of strings of size 3.

3. **samplestep2**

This step is called four times using separate threads with inner list of size 3:

```
Thread1 -> [Batch 1 MD 1, Batch 1 MD 2, Batch 1 MD 3]
Thread2 -> [Batch 2 MD 1, Batch 2 MD 2, Batch 2 MD 3]
Thread3 -> [Batch 3 MD 1, Batch 3 MD 2, Batch 3 MD 3]
Thread4 -> [Batch 4 MD 1, Batch 4 MD 2, Batch 4 MD 3]
```

4. **samplestep3**

This step is called by each of the four threads and the same message objects are sent from the previous step. An additional string is added to it.

```
Thread1 -> [Batch 1 MD 1, Batch 1 MD 2, Batch 1 MD 3, Message Data for Step3]
Thread2 -> [Batch 2 MD 1, Batch 2 MD 2, Batch 2 MD 3, Message Data for Step3]
Thread3 -> [Batch 3 MD 1, Batch 3 MD 2, Batch 3 MD 3, Message Data for Step3]
Thread4 -> [Batch 4 MD 1, Batch 4 MD 2, Batch 4 MD 3, Message Data for Step3]
```

One thread execution of this step is returned as **NoData** status and the other throws an exception. This stops further processing of the steps on these threads and only the remaining two threads move on to the next step.

5. **samplestep4**

This step is called by only two threads and the message associated with these threads is passed from the previous step.

```
Thread2 -> [Batch 2 MD 1, Batch 2 MD 2, Batch 2 MD 3, Message Data for Step3]
Thread3 -> [Batch 3 MD 1, Batch 3 MD 2, Batch 3 MD 3, Message Data for Step3]
```

6. **sampleaggregatestep**

This step waits on all four threads to finish and aggregates the message data associated with successful threads. Because one thread failed and another had no data, the combined message data created is a list of two entries, and these two entries are the lists from previous successful messages.

```
[[Batch 2 MD 1, Batch 2 MD 2, Batch 2 MD 3, Message Data for Step3]
 [Batch 3 MD 1, Batch 3 MD 2, Batch 3 MD 3, Message Data for Step3]].
```

This step retrieves the message object before the **samplestep1** split step message object and passes it to the next step.

7. **samplestep5**

This step gets the message object from the previous aggregate step with message data.

```
Thread 5 -> [[Batch 2 MD 1, Batch 2 MD 2, Batch 2 MD 3, Message Data for Step3]
 [Batch 3 MD 1, Batch 3 MD 2, Batch 3 MD 3, Message Data for Step3]].
```

The tracking information set on the message data is the same as the one set in the **samplestep1** step.

This sample flow outputs the message data at every step and provides a consolidate status at the end:

```
-----samplestep1-----
-----
      TaskId          Time      Status      StepInfo
U147194098092927a28dd0002  0.01      Done  {MessageList=[Batch 1 MD 1, Batch 1 MD 2,
Batch 1 MD 3,
      Batch 2 MD 1, Batch 2 MD 2, Batch 2 MD 3, Batch 3 MD 1, Batch 3 MD 2, Batch 3 MD 3,
      Batch 4 MD 1, Batch 4 MD 2, Batch 4 MD 3], Count=12}
-----
Status: Created: 0   Started: 0   Done: 1   Error: 0
Total Time    0.01   Total Count 12

-----samplestep2-----
-----
      TaskId          Time      Status      StepInfo
U14772590a0f6927a28dd0006  0.03      Done  {MessageList=[Batch 1 MD 1, Batch 1 MD 2,
Batch 1 MD 3,
      Message Data for Step3], Count=3}
```

```

U146c0e08d2d1927a28dd0008    0.03    Done    {MessageList=[Batch 2 MD 1, Batch 2 MD 2,
Batch 2 MD 3,
    Message Data for Step3], Count=3}
U1471b1a19662927a28dd0010    0.02    Done    {MessageList=[Batch 3 MD 1, Batch 3 MD 2,
Batch 3 MD 3,
    Message Data for Step3], Count=3}
U1478ed730b77927a28dd0011    0.02    Done    {MessageList=[Batch 4 MD 1, Batch 4 MD 2,
Batch 4 MD 3,
    Message Data for Step3], Count=3}

```

```

-----
Status: Created: 0    Started: 0    Done: 4    Error: 0
Total Time    0.10    Total Count 12

```

-----samplestep3-----

```

-----
TaskId          Time    Status    StepInfo
U1470d4f69e32927a28dd0012    0.14    Done    {MessageList=[Batch 3 MD 1, Batch 3 MD 2,
Batch 3 MD 3,
    Message Data for Step3], Count=4}
U146e788e7a93927a28dd0013    0.14    Done    {MessageList=[Batch 4 MD 1, Batch 4 MD 2,
Batch 4 MD 3,
    Message Data for Step3], Count=0}
U146f3ecaa056927a28dd0014    0.15    Error    {MessageList=[Batch 1 MD 1, Batch 1 MD 2,
Batch 1 MD 3,
    Message Data for Step3], Count=0}
U146eee0c1848927a28dd0015    0.14    Done    {MessageList=[Batch 2 MD 1, Batch 2 MD 2,
Batch 2 MD 3,
    Message Data for Step3], Count=4}

```

```

-----
Status: Created: 0    Started: 0    Done: 3    Error: 1
Total Time    0.57    Total Count 8

```

-----samplestep4-----

```

-----
TaskId          Time    Status    StepInfo
U1473287849e4927a28dd0016    0.01    Done    {MessageList=[Batch 3 MD 1, Batch 3 MD 2,
Batch 3 MD 3,
    Message Data for Step3], Count=4}
U14739f5f7070927a28dd0018    0.02    Done    {MessageList=[Batch 2 MD 1, Batch 2 MD 2,
Batch 2 MD 3,
    Message Data for Step3], Count=4}

```

```

-----
Status: Created: 0    Started: 0    Done: 2    Error: 0
Total Time    0.03    Total Count 8

```

-----samplestep5-----

```

-----
TaskId          Time    Status    StepInfo
U146fc0312df1927a28dd0020    0.00    Done    {MessageList=[Batch 1 MD 1, Batch 1 MD 2,
Batch 1 MD 3,
    Batch 2 MD 1, Batch 2 MD 2, Batch 2 MD 3, Batch 3 MD 1, Batch 3 MD 2, Batch 3 MD 3,
    Batch 4 MD 1, Batch 4 MD 2, Batch 4 MD 3], Count=12}

```

```

-----
Status: Created: 0    Started: 0    Done: 1    Error: 0
Total Time    0.00    Total Count 12

```

```

Step Summary
samplestep1
  Status: Created: 0   Started: 0   Done: 1   Error: 0

samplestep2
  Status: Created: 0   Started: 0   Done: 4   Error: 0

samplestep3
  Status: Created: 0   Started: 0   Done: 3   Error: 1

samplestep4
  Status: Created: 0   Started: 0   Done: 2   Error: 0

samplestep5
  Status: Created: 0   Started: 0   Done: 1   Error: 0

Total time for all Steps 0 sec
Overall Time 0.307 sec
Done processing Type: sample1 FlowAction: sampleflow1

```

Adding search indexing steps

- Implementation class
Each step must be associated with an implementation class. This class must have clearly defined input and output parameters so that it can be reused in other flows if needed. Steps can be defined in the property file as:

```
step-name=full-class-name
```

For example:

```
querytimeslicestep=com.siemens.teamcenter.ftsi.objdata.steps.query.QueryTimeSliceStep
```

There are three choices based on types of steps:

- Simple step
For a simple step that converts an input object to an output object, implement a class that extends the **TcFtsIndexerAbstractStep** class. (Refer to the Javadocs for details.) This requires implementation of the **process** method:

```
public ITcFtsIndexerStep.Status process( IStepInfo zStepInfo, IMessage zMessage )
    throws Exception
```

The **process** method for each step is called in sequence for all steps defined in a flow. Output of the process is sent as input to the next step. The process takes an **IStepInfo** class as an argument. This class tracks all the statuses associated with the step being processed. Steps can set tracking information and also use helper methods to access the staging directory using the **IStepInfo** class. All **StepInfo** objects are printed as a status at the end of indexing or during **-status** command execution.

The **IMessage** object is a message object that holds on to input/output objects and is accessible across steps in a flow. The message object contains a data object that can be any Java object. In the case of simple steps, the same message object is passed around.

TcFTSIndexer parses all the arguments passed in through the command line and creates a list of arguments that are not framework related. These arguments are considered type-specific arguments. This list is sent to the first step in the flow for processing as message data. Steps can get this message data object and update it. They can also set new message data objects. Steps can be stopped from further processing by throwing an exception or returning the **NoData** status.

See `TC_ROOT\TcFTSIndexer\sample` directory for sample steps.

- Split step

This step splits a list of objects as the message data input to the individual elements and creates a new message object for each element in the list. TcFTSIndexer creates parallel threads equal to the size of the input message data list and runs the next step with these new message objects. The split step can be defined as:

```
step-name=com.siemens.teamcenter.ftsi.core.TcFtsIndexerSplitStep
```

For example, object data indexing splits a list of lists (**List<List<String>>**). This list contains a list of UUIDs of specific batch size. This step breaks the list of lists into multiple threads processing each batch size list of UUIDs.

- Aggregate step

This step waits on all split threads to complete and combines the successful split step messages data into a new message object and passes it to the next step. The number of aggregate steps must match the split steps. The aggregate step can be defined as:

```
step-name=com.siemens.teamcenter.ftsi.core.TcFtsIndexerAggregateStep
```

For example, the object data aggregate step waits on all the threads created by the split step to process a batch size of UUIDs. After all the split steps are aggregated, the data is committed in Solr in the next step.

- Steps in a flow

TcFTSIndexer runs the steps in the order specified. The sequence of steps defined in a flow must be defined in the property file as:

```
internal-flow-name.steps=step-name-1, step-name-2, step-name-n
```

For example:

```
reindexflow.steps=tcftsindexervalidatestep,querytimeslicestep,
  querysplitstep,objdataquerystep,exportbatchsizestep,exportsplitstep,
  objdataexportstep,objdatatransformstep,objdataloadstep,batchexportaggregatestep,
  postexportbatchaggregatestep,querytimeaggregatestep,postquerytimeaggregatestep
```

- Status of steps

By default, the **-status** argument outputs the status of all steps. To control the output to only certain important steps, the following property format can be used:

```
internal-flow-name.status=step-name-1, step-name-4, step-name-7
```

For example:

```
reindexflow.status=objdataquerystep,objdataexportstep,objdatatransformstep,
objdataloadstep
```

Adding new search indexing types

A type is defined by creating a property file that includes all flows and step configurations. Create this property file in the `TC_ROOT\TcFTSIndexer\conf` directory. The file name must have the following syntax: **TcFTSIndexer_TypeName.properties**. TcFTSIndexer reads the property file name and determines the type name associated with these properties.

The type uses the default behavior unless specific behavior is required. In this case a type class can be defined as follows in the **TcFTSIndexer_TypeName.properties** file:

```
type=full-class-name
```

For example:

```
com.siemens.teamcenter.ftsi.objdata.TcFTSIndexerObjDataType
```

This custom implementation class must extend the **TcFTSIndexerType** class and typically requires the override of the following method:

```
public void initialize( Object zData )
```

This method is called during the execution of the type and helps in any initialization of objects specific to this type or in the validation of inputs. For example, the object data type implementation class overrides the initialize method to make sure only one object data type runs at any given time.

Be sure to set up logging for the custom type in the `TC_ROOT\TcFTSIndexer\conf\log4j.properties`, similar to **objdata**.

Deploy a new search indexing type

1. Ensure the TcFTSIndexer is installed and working in stand-alone mode.
2. Create a new **TcFTSIndexer_type-name.properties** file and add the associated properties to the `TC_ROOT\TcFTSIndexer\conf` directory.
3. Implement the classes for types, flows, and steps as defined in the property file.
4. Compile the code by adding JAR files in the `TC_ROOT\TcFTSIndexer\lib` directory to the classpath.
5. Create a custom JAR file with all the custom implementation classes.
6. Copy the new custom JAR file into the `TC_ROOT\TcFTSIndexer\lib` directory.

7. Run the **runTcFTSIndexer -status** command to check if the new custom type and flows appear in the console.
8. Fix any configuration and implementation related issues based on the output console messages.
9. Set up logging for the custom type in the `TC_ROOT\TcFTSIndexer\conf\log4j.properties`, similar to **objdata**.
10. Run the flow using the following command:

```
runTcFTSIndexer -task=type:flow-action
```

Modify an existing search indexing type

Back up the existing type before any modifications; work with a copy of the existing type.

1. Make sure TcFTSIndexer is installed and working in stand-alone mode.
2. Copy `TC_ROOT\TcFTSIndexer\conf\TcFtsIndexer_type-name1.properties` to `TC_ROOT\TcFTSIndexer\conf\TcFtsIndexer_type-name2.properties`.
3. Change any type-specific definitions to the new name in the file.
4. Delete unnecessary flows and steps for the new type.
5. Add, modify, or delete any steps associated with a flow.
6. Implement the new steps based on the property file definition.
7. Create a custom JAR file with all the implementation classes.
8. Copy the new custom JAR file into the `TC_ROOT\TcFTSIndexer\lib` directory.
9. Run the **runTcFTSIndexer -?** command to check if the new custom type and flows appear in the console.
10. Fix any configuration and implementation related issues based on the output console messages.
11. Run the flow using the following command:

```
runTcFTSIndexer -task=type:flow-action
```

Adding search indexing flows

A flow has an internal name and an associated flow action that are run by end users as commands. The mapping between the flow action and the internal flow name is defined as:

```
internal-flow-name.action=flow-action
```

For example:

```
reindexflow.action=reindex
```

Flow uses the default behavior unless there is a need to support flow-specific behavior. A custom flow class can be defined by creating a property as:

```
internal-flow-name=full-class-name
```

For example:

```
reindexflow=com.siemens.teamcenter.ftsi.objdata.TcFtsIndexeObjDataFlow
```

This custom implementation class must extend the **TcFtsIndexerFlow** class and typically requires an override of the following method:

```
public void initialize( Object zData )
```

This method is called during the execution of the flow and helps in any initialization of objects specific to this flow or in the validation of inputs. For example, the object data flow implementation class overrides the initialize method to cache Teamcenter preferences in a cache file for performance reasons.

Users can type **runTcFTSIndexer.bat/sh -status** command to get information of all supported flows. The flow description can be provided as follows:

```
internal-flow-name.description=description-text
```

For example:

```
reindexflow.description=Clears the existing indexed data and performs index of data.
```

ObjData indexing support for saved queries

A sample **savedquery** flow action is added to the **ObjData** type to support queries for UIDs through saved queries. Sample code for the **savedquerystep** step is available in the **TC_ROOT\TcFTSIndexer\sample** directory. This sample step is integrated into the saved query flow. Flow and step configurations are available in the **TC_ROOT\TcFTSIndexer\conf\TcFtsIndexer_objdata.properties** file. Properties related to sample saved queries are named **savedquerystep.***. See the properties description for usage and details.

Run this saved query:

```
runTcFTSIndexer -task=objdata:savedquery
```

Additional saved query requirements can be handled by making changes to sample code and creating a new step. This new step can replace the existing **savedquerystep** step.

Indexing non-Teamcenter data

Create external system objects

Before performing this step, be sure you know the structure of the objects you are creating (that is, property name and type).

1. In the Business Modeler IDE, create a new template project. Make sure you select **foundation** and **aws2** as the **Dependent Templates**.
For more information, see [Create a Business Modeler IDE template project in the Teamcenter help](#).
2. In the newly created project, create a new business object using the **Awp0AWCExternalSystemObject** business object as the parent object.
3. Add the required properties as **Runtime** properties.
The following attribute types are supported: **Boolean**, **Date**, **Double**, **Integer**, and **String**.
4. Set the **Awp0SearchIsStored** property constant to **true** for the object property content to display in Active Workspace.
5. Set the **Awp0SearchIsIndexed** property constant to **true** to indicate that the field is searchable.
6. Open the **Awp0AWCExternalSystemObject** business object and set the **Awp0SearchIsIndexedExt** business object constant to **true** to indicate that external business objects are indexed for searching.
7. Deploy your template to the Teamcenter server.
For information about templates, see [Introduction to deploying templates in the Teamcenter Business Modeler IDE help](#).
8. Create the XRT files used to format and display the non-Teamcenter data in Active Workspace.
For each object type added, you must create two datasets with XRT files for the proper display of objects in Active Workspace: one for the **Summary** panel and one for the **Info** panel (you can review **Awp0ItemRevSummary** and **Awp0ItemRevInfoSummary** for an example of the syntax).
9. Add the following Teamcenter preferences to link these XRT files to the external system objects added:
 - *external-system-object-name*.**CellProperties**
 - *external-system-object-name*.**INFORENDERING**
 - *external-system-object-name*.**SUMMARYRENDERING**
 - *info-XRT-dataset-name*.**INFO_REGISTEREDTO**

- *info-XRT-dataset-name.SUMMARY_REGISTEREDTO*

Note:

If you add an **AWC_** prefix to the new preferences, they only apply to Active Workspace; other Teamcenter clients are unaffected.

Add data indexing to TcFTSIndexer

Before starting on the external objects indexing, understand how **ObjData** indexing works. Also ensure you have a working TcFTSIndexer installation for **ObjData** indexing. After determining the details of how to get the data from your external system, create a **type**, **flow**, and **steps**. This flow reuses some of the steps defined in the **objdata:savedqueryFlow** flow.

1. Create a new type by adding a new property file to the `TC_ROOT\TcFTSIndexer\conf\` directory. For example, create a `TcFtsIndexer_externaldata.properties` file. (This creates an **externaldata** type.)
2. Create a flow action, for example, `extdataflow.action=extdata`
3. Copy the steps to reuse from the **objdata:savedquery** flow found in the `TC_ROOT\TcFTSIndexer\conf\TcFtsIndexer_objdata.properties` file. Steps that can be reused are defined as follows. Details for these steps are in Javadoc associated with each class.

```
tcftsindexervalidatestep=com.siemens.teamcenter.ftsi.objdata.steps.TcFtsIndexerValidateStep
exportbatchsizestep=com.siemens.teamcenter.ftsi.objdata.steps.export.ExportBatchSizeStep
exportsplitstep=com.siemens.teamcenter.ftsi.core.TcFtsIndexerSplitStep
objdataloadstep=com.siemens.teamcenter.ftsi.objdata.steps.load.ObjDataLoadStep
batchexportaggregatestep=com.siemens.teamcenter.ftsi.core.TcFtsIndexerAggregateStep
postexportbatchaggregatestep=com.siemens.teamcenter.ftsi.objdata.steps.postprocess.PostBatchExportAggregateStep
postquerytimeaggregatestep=com.siemens.teamcenter.ftsi.objdata.steps.postprocess.PostQueryTimeAggregateStep
```

The exact steps and details of the flow to create are data and system dependent, for example:

- In a system where there are large numbers of objects to index, **return data as IDs in query calls**. This allows for greater control over how the objects are gathered from the system and how large are the chunks of data to process.
- In an environment where there is already an existing XML export, **return data as XML in export calls**.
- In other cases, a **direct export of data into the Solr input XML format** may be preferable.

Return external data as IDs in query calls to the external system

1. Define a flow with the steps as follows.

```
extdataflow.steps=tcftsindexervalidatestep,extquerystep,
    exportbatchsizestep,exportsplitstep,extexportstep,exttransformstep,
    objdataloadstep,batchexportaggregatestep,postexportbatchaggregatestep,
    postquerytimeaggregatestep
```

The three steps in bold (**extquerystep**, **extexportstep**, and **exttransformstep**) must be created in the property file and implemented.

2. Implement a step class that connects to external systems and returns the IDs of objects to index as a list in message data. Look at the `TC_ROOT\TcFTSIndexer\sample\TcFtsIndexerSavedQueryStep.java` file for sample code that runs a saved query to return a list of UIDs in message data.

The output is a list of IDs, for example:

```
extquerystep=com.siemens.teamcenter.ftsi.externaldata.steps.query.QueryStep
```

3. Implement an export step that connects to an external system and exports an XML file for the input IDs. This step must return the full path to the XML file as a string in the message data. The input is a list of IDs and the output is a string that is the full path to the external system XML file, for example,

```
extexportstep=com.siemens.teamcenter.ftsi.externaldata.steps.export.ExportStep
```

4. Implement a step that takes the XML file from the previous step and converts it to a Solr input XML file. The full path to the XML file that was output from the export step is sent as input to this step as message data. This step must return the full path to the Solr XML file as a string in the message data object. The input is a string that is the full path to the external system XML file. The output is a string that is the full path to the Solr XML file, for example:

```
exttransformstep=com.siemens.teamcenter.ftsi.externaldata.steps.transform.TransformStep
```

Return external data as XML in export calls to the external system (no query)

1. Define a flow with the steps as follows:

```
extdataflow.steps=tcftsindexervalidatestep,extexportstep,exttransformstep,
    objdataloadstep,postexportbatchaggregatestep,postquerytimeaggregatestep
```

The two steps in bold (**extexportstep** and **exttransformstep**) must be created in the property file and implemented.

2. Implement an export step that connects to an external system and exports the XML file for all objects to be indexed. This step must return the full path to the XML file as a string in the message data object.

The output is a string that is full path to the external system XML file, for example:

```
extexportstep=com.siemens.teamcenter.ftsi.externaldata.steps.export.ExportStep
```

- Implement a step that takes the XML file from the previous step and converts it to a Solr input XML file. The full path to the XML file that was the output of the export step is sent as input to this step as message data. This step must return the full path to the Solr XML file as a string in the message data object.

The input is a string that is the full path to the external system XML file, and output is a string that is the full path to the Solr XML file, for example:

```
exttransformstep=com.siemens.teamcenter.ftsi.externaldata.steps.transform.TransformStep
```

Return external data as Solr input XML from external system (no query and export)

- Define a flow with the steps as follows:

```
extdataflow.steps=tcftsindexervalidatestep, exttransformstep,
objdataloadstep, postexportbatchaggregatestep, postquerytimeaggregatestep
```

The **exttransformstep** needs to be created in the property file and implemented.

- Implement a step that connects to the external system and creates a Solr input XML file for all objects to index. This step must return the full path to the Solr XML file as a string in the message data object.

The output is a string that is the full path to the Solr XML file. The basic format of the Solr XML to generate is as follows:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<add>
  <doc>
    <field name="property name">Value of Prop</field>
    ...
    <field name="property name">Value of Prop</field>
  </doc>
  ...
  <doc>
    ...
  </doc>
</add>
```

Each object is represented by a **<doc>** element. The properties of the object are represented by the **<field>** element. The name attribute represents the property name defined in the Business Modeler IDE and is in the following format: **TC_0Y0_name-of-the-object_0Y0_propertyname**. The following elements are required:

- id**
Specifies the unique ID of the object. To prevent object ID collisions, prefix the ID with a unique identifier.
- TC_GROUP_FIELD_ID**
Represents the ID. This is an internally used value; it should be the same as **id**.

- **TC_0Y0_Awp0AWCEXternalSystemObject_0Y0_awp0ExternalSysObjClassName**
Identifies the name of the object defined in the Business Modeler IDE used to properly generate the object type in Active Workspace.
- **TC_0Y0_Awp0AWCEXternalSystemObject_0Y0_awp0ExternalSystemName**
Identifies the name of the external system from which the object came.
- **TC_0Y0_Awp0AWCEXternalSystemObject_0Y0_awp0ExternalSystemURI**
Defines the URI in which to display the object.
- **TC_0Y0_Awp0AWCEXternalSystemObject_0Y0_awp0PropertyNameList**
Specifies the element for each Solr name of the object properties.
The value format is: **TC_0Y0_object-name_0Y0_property-name**
- **TC_PRIV_am_rule_str**
Defines the permission read expression.
Permission information is indexed with each object. The base security read expression string to allow an object to be searchable by all users is **EAKT(EACT+)EAYT+**. Detailed permission information may require assistance from Siemens Industry Software Inc.
- **TC_0Y0_WorkspaceObject_0Y0_object_type**
Allows objects of this class name to be filtered using the **Type** category in the Active Workspace client. The value must be the same as the **TC_0Y0_Awp0AWCEXternalSystemObject_0Y0_awp0ExternalSysObjClassName** field.

An example of the general XML follows:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<add>
  <doc>
    <field name="TC_0Y0_Awp0AWCEXternalSystemObject_0Y0_awp0External
      SysObjClassName">The object class name defined in the BMIDE</field>
    <field name="TC_0Y0_Awp0AWCEXternalSystemObject_0Y0_awp0External
      SystemName">The name of the external system</field>
    <field name="TC_0Y0_Awp0AWCEXternalSystemObject_0Y0_awp0External
      SystemURI"><field name="TC_0Y0_Awp0AWCEXternalSystemObject_0Y0_
      awp0PropertyNameList">TC_0Y0_A4_address_0Y0_
      a4configuration</field>
    ...
    <field name="TC_0Y0_Awp0AWCEXternalSystemObject_0Y0_
      awp0PropertyNameList">TC_0Y0_A4_address_0Y0_ a4type_
      displayName</field>
    <field name="id">A4_FD004</field>
    <field name="TC_Group_Field_Id">A4_FD004</field>
    <field name="TC_0Y0_A4_address_0Y0_a4configuration">Feature Dictionary
      Revision</field>
    ...
    <field name="TC_0Y0_A4_address_0Y0_a4type_displayName">Address</field>
    <field name="TC_PRIV_am_rule_str">EAKT(EACT+)EAYT+"</field>
  </doc>
```

SOA method calls to Teamcenter are not allowed in this method.

Configure TcFTSIndexer for external data

1. Add JAR files in the *TC_ROOT\TcFTSIndexer\lib* directory into the classpath and compile the implemented code and package it into a *.jar* file.
2. Copy the *.jar* file created to the *TC_ROOT\TcFTSIndexer\lib* directory.
3. Copy the **TcFTSIndexer_type.properties** file created when you **added data indexing to TcFTSIndexer** the *TC_ROOT\TcFTSIndexer\conf* directory.
4. Run the **runTcFTSIndexer -task=type:flow-action** command to run the flow, for example:

```
runTcFTSIndexer -task= externaldata:extdata
```

2. Search

Configure search

Search configuration tasks

Be sure that objects have been configured for indexing before initializing or synchronizing the indexed data.

- **Define index data and filters.**
- Configure Business object and properties indexing.

You can configure many aspects of search, including:

- **Configure search suggestions.**
- **Define search prefilters.**
- **Configure Filter panel behavior.**
- **Configure saved search.**
- **Configure revision rules.**
- **Configure export to Microsoft Excel**
- **Enable Advanced search.**
- **Configure Quick search.**

Configure search suggestions

Configure search suggestion threshold

In a global search, you can configure suggested search terms as users type search criteria. By default, keywords that make up at least .5 percent of the indexed data display as search suggestions. For example, if there are 1000 indexed items, a keyword that exists at least five times is used as a suggestion.

You can configure the threshold for presenting keywords as search suggestions.

Change the threshold percentage

1. Find and open the **solrconfig.xml** file in your Solr installation directory.

`TC_ROOT\solr-version\server\solr\collection1\conf\solrconfig.xml`

2. In the **solrconfig.xml** file, look for the following:

```
<float name=threshold">0.005</float>
```

3. To display more search suggestions, decrease the value (for example, change it to **0.003** for .3 percent). To display fewer search suggestions, increase the value (for example, change it to **0.008** for .8 percent).
4. Save the file.
5. Reboot Solr.

Turn search suggestions off

1. Open the Solr **solrconfig.xml** file.
2. In the **solrconfig.xml** file, find the **tcftssuggest** configuration section.

```
<requestHandler
class="org.apache.solr.handler.component.SearchHandler"
name="/tcftssuggest">
  <!-- Request handler for Teamcenter search suggestions -->
  <lst name="defaults">
    <str name="spellcheck">true</str>
    <str name="spellcheck.dictionary">tcftssuggest</str>
    <str name="spellcheck.onlyMorePopular">true</str>
    <str name="spellcheck.count">5</str>
    <str name="spellcheck.collate">true</str>
  </lst>
  <arr name="components">
    <str>tcftssuggest</str>
  </arr>
</requestHandler>
```

3. Comment out the line:

```
<str>tcftssuggest</str>
```

Save and close the file.

4. Restart Solr.

Configure search phrase suggestions

In a global search, you can configure suggested phrases that users can match in their search criteria as they type. Suggested phrases are limited to eight words, and the top five suggested phrases that match the search criteria are displayed in the list of suggested phrases.

The threshold configuration doesn't apply to search phrase suggestions.

1. Enable search phrase suggestions by setting these preferences:
 - Set **AW_search_suggestion_handler** to **phrase**. The default is **spellcheck**.
 - Add a list of properties to the **AW_IndexedProperties_for_Phased_Search** preference. The name and description properties **WorkspaceObject.object_name** and **WorkspaceObject.object_desc** are configured as search phrases by default.
2. From a Teamcenter command prompt, run **bmide_modeltool.bat** with the **tc_solr_schema_gen** tool:

```
bmide_modeltool -u=user -p=password -g=dba -tool=tc_solr_schema_gen
                 -mode=upgrade -target_dir="%TC_DATA%"
```

3. **Merge the generated schema and the Solr schema.**
4. Restart Solr.
5. If you want phrase suggestions to apply to data that is already indexed, reindex all your data using the **objdata:index** task of the **runTcFTSIndexer** utility.

Otherwise, phrase suggestions are incrementally synchronized for new or updated data (**objdata:sync**).

Caution:

These suggested phrases are readable by all users regardless of their access privileges.

Set the default search operator

Use the **AWS_Default_Query_Operator** preference to set the default search operation to apply to searches for multiple terms. By default, if multiple terms are typed in the search box, the default operator applied is **AND**.

Therefore, searching for **HDD 0500** returns results with both **HDD** and **0500** in them, for example, **HDD 0500** or **HDD 050055**.

However, if the **AWS_Default_Query_Operator** preference is set to **OR**, the same search returns either **HDD** or **0500**, for example, **HDD 123** or **ABC 0500**.

Add wildcards to searches automatically

To return a greater number of results, the ***** wildcard is added by default to the end of search terms by the **AWC_search_automatic_wildcard** preference. By default, the preference is set to **1**, which adds the asterisk (*****) character as a suffix.

If you want to restrict or expand the search term, you can change the preference setting.

0 leaves the search criteria unaltered.

2 adds the asterisk wildcard as a prefix.

3 adds the asterisk wildcard as a prefix and a suffix.

The preference is ignored if either of the following occurs:

- The user enters search text enclosed in double quotation marks (for example, **"part"**).
- The user saves their search. When the user runs a saved search, the value of the preference at the time of reuse is applied.

If the user explicitly enters an asterisk in the search text, the wildcard actions are combined. For example, if the preference value is **2** (wildcard as a prefix) and the user enters **part***, the search text becomes ***part***.

Configure searching for common words

If a search contains common words, such as *the*, *and*, and *a*, they may be ignored in searches. These words, known as stop words, are not indexed by default to reduce the overall size of the search index. You may want to allow users to search for some or all of these stop words. The list of ignored words is found in the Solr documentation:

<https://wiki.apache.org/solr/>

Search for **AnalyzersTokenizersTokenFilters** and locate **solr.StopFilterFactory**.

A search phrase with stop words enclosed in quotation marks has the following effects on the query:

- If a stop word occurs between keywords in a search phrase enclosed in quotation marks, it is replaced by a wild card in the query.
- If a stop word occurs at the beginning or end of a search phrase enclosed in quotation marks, it is ignored

If an exact match is required for a search that contains a stop word, you need to either disable the Solr *stop words* function or edit the Solr stop word list.

Caution:

Disabling the stop words function may make the search index much larger and can degrade search performance.

When you make any changes to stop words, you must index again.

1. Navigate to the Solr **schema.xml** file, for example, `TC_ROOT\solr-version\server\solr\collection1\conf\schema.xml`.
2. If you want to include all stop words in searches, search for lines containing `stopwords_language.txt` and comment them out. For example:

```
<!--filter class="solr.StopFilterFactory" ignoreCase="true" words="lang/stopwords_en.txt"/-->
```

3. If you want to edit the stop word list, open the `stopwords_language.txt` files and edit the list. Save and close the files.
4. Reboot Solr.
5. Index again using the **runTcFTSIndexer** utility.

Add synonyms to searches

Solr allows you to search for synonyms. For example, if you search for **MB**, search results also are returned for **megabyte**. Information about Solr use of synonyms can be found in the Solr documentation:

<https://wiki.apache.org/solr/>

Search for **AnalyzersTokenizersTokenFilters** and locate **`solr.StopFilterFactory`**.

You can add words to the synonyms file to enable additional synonym searching.

1. Navigate to the Solr **synonyms.txt** file, for example, `TC_ROOT\solr-version\version\solr\collection1\conf\synonyms.txt`.
2. Add synonyms on a single line separated by commas, for example:

```
GB,gig,gigabyte,gigabytes
```

3. Reboot Solr.

Boosting search results

You can affect the relevance of search results by configuring some data characteristics to appear higher in the list of results. When boosting is calculated and applied to matching results, many factors are considered together. Although some matching results may be boosted higher in the list based on these factors, all results matching the search criteria are returned. You should test your configuration to determine whether your boosting strategy has the desired effect on search results.

You can alter the relevance of object data attributes based on:

- Business properties
- Ownership based on user, group, and project
- Last modified or creation date
- Word proximity

Properties

Configure boosting based on properties specified in the **AWS_PREFERRED_ATTRIBUTES** preference. Specify the object properties to be considered for boosting. When search criteria matches object properties, the returned objects with the matching properties may appear higher in the list of search results. Specify a business object and a property following this format:

business-object-type.property-name

Specify multiple properties separated by commas. By default, the Item ID is considered for boosting:

ItemRevision.awp0_Item_item_id

You may append one of the boost priority values **LOW**, **MEDIUM**, or **HIGH**. If a boosting priority is not specified for a property, the default is **MEDIUM**. For example:

ItemRevision.awp0_Item_item_id:MEDIUM

Business objects and properties specified by this preference must be **marked as indexable** using the **Awp0SearchIsIndexed** property constant.

Ownership

Configure boosting based on ownership using the **AWS_PREFERRED_ATTRIBUTES** preference. Set a user, group, or project to be considered for boosting in the list of search results.

You may assign one of the boosting priority values **LOW**, **MEDIUM**, or **HIGH** to the property. If a boosting priority is not specified for a property, the default is **MEDIUM**. Ownership matches may appear higher in the list of results.

Current or latest user Specify a valid user name or the variable **\$ME** (the default) for **owning_user** or **last_modified_user**. To assign multiple users, create multiple entries for the same property and specify a user for each one.

POM_application_object.owning_user:LOW:*user-name*

POM_application_object.last_modified_user:HIGH:\$ME

Current user's group Specify a valid group name or the variable **\$MY_GROUP** (the default) for **user_group**. To assign multiple groups, create multiple entries for the same property and specify a group for each one.

POM_application_object.owning_group:HIGH:*user-group*

POM_application_object.owning_group:HIGH:\$MY_GROUP

Current user's current project Specify a valid project name or the variable **\$MY_PROJECT** (the default) for **user_project**. To assign multiple projects, create multiple entries for the same property and specify a project for each one.

WorkspaceObject.project_list:LOW:*user-project*

WorkspaceObject.project_list:LOW:\$MY_PROJECT

Date

Configure results boosting based on date using the **AW_Search_Results_Recency_WeightAge** preference. Set a numeric value progressively higher to boost the most recently created or modified objects. Matches with more recent creation or modification dates may appear higher in the results. Set the value to **0** (turn off boost by date) through **6** (maximize boost by most recent date), The default value is **1**.

Proximity

Configure results boosting based on the proximity of search terms to each other using the **AWC_Search_Results_Word_Proximity** preference. Set a numeric value for the maximum proximity of individual search words. Matches with search words occurring within the proximity setting may appear higher in the results. Set the value to **0** (turn off boost by proximity) through **100**. The default value is **10**.

For example, if the value is **5**, the results where the terms are no more than five words apart may appear higher in the results.

Configure results threshold

Users can enter search terms that return an extremely high number of results, which can cause long wait times. When the results exceed the limit for long lists, the user can refine their search criteria to

return fewer results. You can limit the number of returned results by specifying the maximum number of results in the **AWC_Search_Threshold_Value** preference.

If the number of results exceeds the limit, no results display. A message tells the user that the results exceed the limit. Other search actions, such as boosting, security access, revision rule checking, or charts, are not applied to the results. Users can narrow their search by changing their search criteria or choosing a filter.

Specify an integer for the threshold limit or **0** (the default) to disable it.

If the user runs a saved search or applies a filter from the filter panel, the threshold limit is not applied to the results. By default, **Category** and **Type** filters are always displayed.

Configure revision rules

Define the list of revision rules using a set of preferences that populate the list for global search users. The available revision rules are managed in Teamcenter; see the Teamcenter help for revision rule information.

Provide a version that meets the user's permission

When a user's search includes a revision rule, sometimes the user can't access an item that matches the search criteria revision rule. In this case, the user may still want to see the latest version of an object. You can configure global search to provide a previous version that satisfies the user's access permission.

For example, an engineer may want to view the latest revision of an object that has two revisions. The first revision is released and readable to all users, but the second is in progress and only available to designers. If the engineer is not in the designer group, search won't return the object because the revision rule was applied before the access rule.

However, you can configure search to first apply the access rule on search data and then apply the revision rule to the subset of results. This process requires setting a preference and a threshold. Use the following guidance to determine when to apply revision rules after read access permissions. The threshold is applied before any search postprocessing

- If the number of search results is more than the threshold, all permitted revisions of all the items that match the search criteria are returned in the search result. The search result breadcrumb above the result displays the message that multiple revisions of each item are in the search results.
- If the number of search results is less than or equal to the threshold, search applies the user's access permission first and then applies the revision rules configured by **AW_Search_check_read_access_for_RevRule**. This means that search looks for a previous revision permitted to the user and return that revision of the item.

You can specify which revision rules apply below the threshold. Determine which revision rules to apply and then specify them in the **AW_Search_check_read_access_for_RevRule** preference (empty by default).

When you specify revision rules, you also activate the search behavior that applies access rules before applying the specified revision rules. If you activate this method for searching, reindex your data unless you have already done so using your current version of Active Workspace.

Caution:

The revision rule search behavior is not supported by the sharding capability in SolrCloud. This search behavior does not affect the replication functionality of shards in SolrCloud.

Set a threshold value for the number of results that determine which behavior applies. Then re-index the objects.

1. Configure the threshold in `TC_ROOT\solr-version\server\solr\collection1\conf\solrconfig.xml`:

```
<int name="RevisionRuleFilter.Threshold">20000</int>
```

The threshold is set to **10000** by default.

2. Stop any synchronization flows in progress and stop Solr.
3. Make a backup copy of `TC_ROOT\solr-version\server\solr\collection1\conf\schema.xml`.
4. Update the **field** entries in **schema.xml** to set **docValues="true"**:

```
<field indexed="true" multiValued="false" name="TC_RevisionAnchor" required="false"
  docValues="true" stored="false" type="string"/>
<field indexed="true" multiValued="false" name="TC_PRIV_am_rule_str"
  required="false"
  docValues="true" stored="true" type="string"/>
<field indexed="true" multiValued="true" name="TC_PRIV_matchingRevisionSelectors"
  docValues="true" required="false" stored="false" type="string"/>
```

5. Restart Solr.
6. Start indexing:

```
runTcFTSIndexer -task=objdata:index
```

If object data was indexed with the same version of Active Workspace for other reasons, you can skip this step.

7. After indexing completes, start synchronization.

```
runTcFTSIndexer -task=objdata:sync -interval=300
```

Effectivity dates in revision rules

You can provide a revision rule that applies an effectivity date to a global search for item revisions or other similar objects. Effectivity dates are defined on a compound property of the item revision. If no effectivity date is present, the default is **Today**.

Follow this process to implement effectivity for a revision rule:

1. In Business Modeler IDE, create a compound property to store the effectivity. Make sure to mark it as indexable.
2. In Business Modeler IDE, set the value for the global constant **Awp0EffectivityDatesPropertyName** to the compound property name you created.
3. In the *TcFtsIndexer_objdata.properties* file, set the value for **objdatasaxtransformstep.EffectivityDatePropertyName** to the compound property name you created.
4. Create or edit the revision rule for the item, and define the **Date** entry. Specify the dates when the rule condition is in effect. To aid the global search user in selecting effectivity, include the effectivity in the rule name.
5. In Business Modeler IDE, regenerate the Solr schema file, and merge the changes. Restart Solr.
6. Reindex to update it for the compound property.

If multiple effectivities are active for a single item revision (for example, due to multiple statuses) only the latest one is indexed.

7. Ensure that the revision rule is specified in the **AWC_Rev_Rule_List** preference to make it available for global search.

Define search prefilters

Search prefilters appear next to the global search box and are properties used to filter data when a user performs a global search. Click a prefilter to see a list of constraints to filter search results. The default global search prefilter is **Any Category**, which applies prefiltering based on categories of objects.

In the following example, **Any Owner** and **Any Category** are search prefilters. A prefilter can only specify a single filter type. The **Any Owner** search filter results from setting the **AWS_SearchPreFilter_Property1** preference with the following value:

```
POM_application_object.owning_user
user-name ( user-id )
```

When specifying *user-name*, use the form:

LastName, FirstName (username)

For example, *Karia, Pamela (pkaria)*

Note:

The spaces in the example are required. Place a space following the comma after last name, following the first name, and bracketing the username inside the parentheses.

When a user applies a prefilter to a search, the selections are saved to the **AWS_SearchPreFilter_Property1_SelectedValue** and **AWS_SearchPreFilter_Property2_SelectedValue** preferences. When the user logs off and then back on, the prefilters are retrieved from these preferences and displayed in the client.

You can use keywords in the preferences as follows:

- Use **\$TODAY**, **\$THIS_WEEK**, and **\$THIS_MONTH** for date properties.
- Use **\$ME** for properties that store the values as *user-name (user-ID)*.
- Use **\$MY_GROUP** for properties that store the group name.
These keywords are substituted with the correct value when performing the search.

The following example specifies the **Last Modified Date** prefilter. The default options are **\$TODAY**, **\$THIS_WEEK**, and **\$THIS_MONTH**:

```
POM_application_object.last_mod_date
```

The following example specifies the **Engineering** and **Admins** groups as options for the **Group ID** prefilter:

```
POM_application_object.owning_group
Engineering
Admins
```

Customize the category prefilter

The values in the **Any Category** prefilter list are groupings of related business objects as defined by the administrator.

You can customize categories using the Teamcenter Business Modeler IDE.

1. Open your site's business template in the Business Modeler IDE.

- To assign an object to a category, set the value of the object's **Awp0BusinessObjectCategories** business object constant to the category name. Categories are created when one or more objects specify them with this constant.

To assign an object to more than one category, list each category name separated by commas (with no separating spaces). For example, setting an object's **Awp0BusinessObjectCategories** value to **Documents,Files** assigns it to the **Documents** and **Files** categories.

To remove an object from a category, remove the category name from the object's **Awp0BusinessObjectCategories** value.

Any child objects of the object are automatically assigned to the same categories. You can override it for the children by updating their **Awp0BusinessObjectCategories** values.

The following objects are assigned the following categories by default:

Files	Documents	Parts	Changes
MSEcelX	DocumentRevision	DesignRevision	ChangeltemRevision
MSPowerPointX	RequirementSpec	PartRevision	
MSWordX	Revision		
PDF			
Text			
JPEG			

- If you are also adding a category, add the category name to the prefilter preference for categories. For example, if you are using the preference **AWS_SearchPreFilter_Property2** to define **Any Category** as a search prefilter, add the category name to the list of categories defined in the preference.
- Once your customizations are complete, deploy the template to your site.
- Run the **bmide_modeltool.bat** utility.

```
bmide_modeltool.bat -u=username -p=password -g=dba -tool=all -mode=upgrade
-target_dir="TC_DATA"
```

There is another method for customizing categories without the need to deploy a new template or run the **bmide_modeltool.bat** utility. Siemens Digital Industries Software recommends extreme caution when using this method to update your site. Ensure you back up your settings before attempting this procedure.

Children of the objects categorized using this method do not automatically belong to the categories defined using this method. You must manually update each child object to change its category. Users must restart their sessions to see any changes.

1. Open the Teamcenter rich client.
2. Choose **Edit**→**Options** and locate the **AW_FullTextSearch_TypeCategories** preference. Set the **Values** field to the full value of all categories. For example, the **Files**, **Documents**, **Parts**, and **Changes** categories default values could be set as follows:

```
Files:MSEXcelX,MSPowerPointX,MSWordX,PDF,Text,JPEG
Documents:DocumentRevision,RequirementSpec Revision
Parts:DesignRevision,PartRevision
Changes:ChangeItemRevision
```

Warning:

This method of changing categories may cause unexpected results. Internal names of objects are cryptic, and the **Value** field content is not checked for spelling or other errors.

Category names are not localized.

Improve performance when generating results

You can use the **AWS_Search_disable_exclude_prefilter** preference to generate only the results for the selected prefilter from a prefilter category. The selected prefilter is the only filter that is displayed for the prefilter category in the **Filters** panel.

Set the preference to **false** (the default) to generate all facets for a prefilter category in the **Filters** panel that match the search results, regardless of the user's prefilter selection. This can take longer for a large number of returned results (such as entering the ***** wildcard) but provides the user with more flexibility.

Set the preference to **true** to apply the prefilter selection to the search results before generating facets for the **Filters** panel. This improves performance for a large number of results but limits the scope to the selected prefilter for the category.

Example:

The user chooses **Part Revision** from the list of **Type** prefilters and enters a search query.

If **AWS_Search_disable_exclude_prefilter** is set to **true**, search returns only the results that match the **Part Revision** prefilter for the **Type** category.

If **AWS_Search_disable_exclude_prefilter** is set to **false**, search returns all results that match any of the prefilters from the **Types** category.

Keep the global search Type prefilter

If you customized your global search **Type** prefilter, or prefer the **Type** prefilter included in previous versions of Active Workspace, you can keep that prefilter when updating Active Workspace:

1. Run the Teamcenter **preferences_manager** utility, exporting the **AWS_SearchPreFilter_Property2** preference. For example:

```
preferences_manager -u=user -p=password -mode=export
                    -preferences=AWS_SearchPreFilter_Property2 -out_file=c:\temp\prefilter.xml
```

2. Perform the Active Workspace upgrade.
3. Run the Teamcenter **preferences_manager** utility, importing the **AWS_SearchPreFilter_Property2** preference. For example:

```
preferences_manager -u=user -p=password -mode=import
                    -preferences=AWS_SearchPreFilter_Property2 -out_file=c:\temp\prefilter.xml
                    -action=OVERRIDE
```

Configure search prefilters for object sets

You can configure a prefilter on an object set to apply to an in-context search. When the Active Workspace user adds to the object set, the configured prefilter uses configured properties and their values to filter search results. The user can also easily remove the prefilters and continue to use other filters.

The prefilter **searchFilter** parameter is retrieved from the parameters that are set for the command that launches the **Add** object panel.

Specify `parameter name="searchFilter"` on the command of the object-set XRT. Specify the **searchFilter** parameter value as a combination of properties and values of the secondary object using the format:

Type.Property1=value1 Operator Type.Property2=value2

Type and *Property* are case-sensitive internal names.

Operator:

AND is a keyword that uses both prefilters.

TO is a keyword that connects start and end values for date and numeric ranges.

Leading and trailing spaces between prefilters are permitted.

Example:

```
<objectSet>
  <command id="AddNew">
    <parameter name="searchFilter"
      value="WorkspaceObject.object_type=Fnd0LogicalBlockRevision
      AND POM_application_object.owning_user=Zhu, Ray ( zhur )"/>
```



```
</command>
</objectSet>
```

Supported filter types are **string**, **date**, **date range**, **numeric**, and **numeric range**.

- **String example**

```
POM_application_object.owning_user = Engineer,Ed ( ed )
```

- **Date example**

```
POM_application_object.last_mod_date_0Z0_year= 2016
```

```
POM_application_object.last_mod_date_0Z0_year_month= September 2016
```

```
POM_application_object.last_mod_date_0Z0_year_month_day=
Sunday - Sep 25, 2016
```

- **Date range example**

```
POM_application_object.last_mod_date= 2016-08-07T20:00:00-04:00 TO
2017-08-09T19:59:59-04:00
```

```
POM_application_object.last_mod_date= 2016-08-07 TO 2017-08-09
```

```
POM_application_object.last_mod_date= * TO 2017-08-09
```

```
POM_application_object.last_mod_date= 2016-08-07 TO *
```

- **Numeric example**

```
WorkspaceObject.s2clAverageRatingFmSy=2
```

- **Numeric range example**


```
WorkspaceObject.s2clAverageRatingFmSy=2 TO *
```

```
WorkspaceObject.s2clAverageRatingFmSy=* TO 100
```

Configure Filter panel behavior

The search **Filters** panel displays properties and values related to items in the global search results. Users can choose filter categories and values to narrow their results. Administrators can initially configure the behavior of filters to make it easier for users to find and apply filters. Users can subsequently change some of these preferences themselves using the **Search Settings** panel.

Configure type-ahead behavior

When a user types in a filter search field, you can configure how the search behaves. Use the **AW_DisableTypeAheadFacetSearch** preference to configure whether the user initiates a filter search by clicking  or whether filter results are returned as the user types in the search term. The default value **false** displays the results as the user types.

You can set the **AW_TypeAheadFacetSearchDelay** preference to control the time between when the user starts typing in the search field and when results begin to appear. Specify a positive integer in milliseconds. The default is **500**.

Configure wildcard behavior for matching filter values

You can configure how searching for filter matches behaves with wildcards using the **AWC_search_filter_wildcard** preference.

- 0** specifies an exact match in the filter values and no wildcard character is applied.
- 1** specifies matching the search criteria as a suffix in the matching filter values.
- 2** specifies matching the search criteria as a prefix in the matching filter values.
- 3** (the default) specifies matching the search criteria anywhere in the matching filter values.

If the preference is not set, the **AWC_search_automatic_wildcard preference** setting applies.

Configure the display of unassigned values

You can hide the display of filter categories with unassigned values from the **Filter** panel using the **AWC_dynamicPriority_hideUnassignedValueFacetCategories** preference. Set to **true** to hide filters with only unassigned values. Set to **false** to display unassigned filter categories.

Limiting expanded filters takes precedence over hiding unassigned filters because limiting filters improves performance for search results. This means that if you hide unassigned values and enable **AWC_Limited_Filter_Categories_Expanded**, some filters may still display only unassigned values.

Specify the set of expanded filters

You can configure a list of filters that expand automatically when search results are returned. Reducing the number of expanded filters makes it easier to view a long list or view a set of expanded filters that are more commonly used.

In the **AWC_Limited_Filter_Categories_Expanded** preference, specify one or more properties to expand automatically, separated by commas.

Example:

```
AWC_Limited_Filter_Categories_Expanded=POM_application_object.owning_user,
POM_application_object.last_mod_user
```

Category and **Type** filter categories are expanded by default.

The **AWC_Limited_Filter_Categories_Enabled** preference enables or disables the behavior. The default is **false**.

This configuration determines which filters are available in the user's **Search Settings** panel. The user can choose to limit the displayed filter list in the **Filters** panel. When **Limit filters to expand** is selected, the user can also choose a subset of the configured filters list to apply to the results from **Filters to Expand**.

Specify the maximum number of displayed filter values

You can limit the number of filter values that display per specified category. Fewer values make it easier to view a long list of filters. If more filter values are available, users can click **More** at the bottom of the list to expand it.

Use the **AWC_Category_Filter_Show_Count** preference to specify how to display filter values.

ALL	Displays all the filters and values relevant to the search results in the Filters panel.
Positive integer	Returns the number of filters specified. Remaining filters are available by using More in the filter value list. This setting also controls the number of additional filters displayed by clicking More , eventually displaying the entire list.
	The order of returned categories is based on the priority set on each property in Business Modeler IDE.
Hidden	Hides the filter and its values in the Filters panel.

Example:

```
AWC_Category_Filter_Show_Count=POM_application_object.owning_user=20,
POM_application_object.owning_user=hidden
```

The default value for all filters is **50**. Invalid specified filters are ignored, which means the default value is applied.

Prioritize filter category display

You can change how filter categories display by setting their priority using the following preferences:

- **AWC_dynamicPriority_facetCategories**

Specifies the filter categories that are used to determine whether dynamic prioritization is required. Valid property values are defined in the **Awp0SearchCanFilter** property constant in the Business Modeler IDE, and take the form *business-object.property*. The value for this preference can be any property of a business object that is displayable in a filter, for example, **Mdl0ModelElement.mdl0model_object** or **Classification.1000**.

The default value is **Lbr0LibraryElement.lbr0Ancestors**.

This preference works in conjunction with the **AWC_dynamicPriority_threshold** preference. For example, consider a situation where the search result returned 1000 objects for a given search criteria and the **AWC_dynamicPriority_facetCategories** preference is set to the default value. If the number of objects returned in this category is 600 and the **AWC_dynamicPriority_threshold** preference is set to 50%, then the dynamic prioritization of filter categories is enabled.

If the value for this preference is set to a **WorkspaceObject.object** type (the most common type), then the dynamic prioritization for filter categories is always enabled as the result is always 100%.

- **AWC_dynamicPriority_hideSingleValueFacetCategories**

Hides the display of all single value filter categories from the filter panel when set to **true**. Setting to **false** displays all single-value filter categories.

This preference does not apply to single unassigned value facet categories. These are controlled by the **AWC_dynamicPriority_hideUnassignedValueFacetCategories** preference.

- **AWC_dynamicPriority_preferredNumberOfFacets**

Specifies the preferred number of filters that can be used in dynamic prioritization. The default value is **4**.

- **AWC_dynamicPriority_threshold**

Specifies the percentage of library objects returned by search that trigger filter category reprioritization. Valid values are between 1 and 100. The default value is **50**.

Therefore, if more than 50% of the search results are library elements, filter categories are reprioritized.

Configure saved search

When a saved search is shared, you can control which users have access to it. Set an access rule that uses the class for the type of shared search. The recommended permission settings are to share only with the owning user's group.

Set the condition, value and access rule name like the following:

Example:

For the global search class **Awp0FullTextSavedSearch**:

```
Has Class ( Awp0FullTextSavedSearch )
Has Attribute( Awp0FullTextSavedSearch:awp0is_global_shared=1 )
SharedSavedSearchACL
```

Set the permissions as recommended:

Owning User: grant READ, WRITE, EXPORT
 Owning Group: grant READ, deny WRITE, EXPORT
 World: deny READ, WRITE, EXPORT

For the advanced search class **SavedSearch**:

Has Class (SavedSearch)
 Has Attribute(SavedSearch:shared=1)
 SharedSavedSearchACL

Set the permissions as recommended:

Owning User: grant READ, WRITE, EXPORT
 Owning Group: grant READ, deny WRITE, EXPORT
 World: deny READ, WRITE, EXPORT

Because location in the rule structure determines how the rule is evaluated, be sure to check where the rule is placed.

Configure column sorting

In search results, you can configure whether to view global search results in a table in order of relevance or in the order determined by a specific table column.

Add the **AWC_SearchUseUIConfigDefinedDefaultSortInplaceOfRelevance** preference and specify the setting:

- **true**
Displays search results according to the specified table column that you configure using the **import_uiconfig** utility.
- **false**
Displays search results in relevance order as determined by the number of times the search string matches an object, the date the object was created or last modified, and other factors.

Configure export

When users export search results, they can select **As Shown**, which exports either selected rows or all rows of search results to Microsoft Excel. If the user chooses **All Results**, all rows are exported, up to the maximum number configured.

Set the maximum number of rows using the **AW_Search_Results_Export_Max_Rows** preference. The default is **1000**.

Return the parent business object for a dataset

Users may want to see which parent business objects are related to dataset content that matches a global search. If the search criteria matches dataset file content that is attached to a business object, you can configure the business object to be returned in the search results. When a business object is indexed, the file content of associated datasets are indexed with it. When a search matches the file content, the business object is returned instead of the dataset. This behavior is configured by a business object constant which can be applied to a specific type level or to the entire type hierarchy.

File content can be in any UTF-8 language. The filter panel can display common properties of the business object and the dataset for user filtering.

- A business object or any of its subtypes must have the same relation type to one or more datasets. The dataset can contain one or more files.
- The **Awp0SearchDatasetIndexingBehavior** constant defines whether datasets are indexed inline with their business object.
- The **Awp0DatasetTypeToBeIndexedInline** constant identifies the datasets to be indexed inline with the business object. If datasets are marked as indexable, standalone datasets are indexed along with their file content. If you prefer to use only the inline indexing method, don't mark dataset types as indexable.
If you choose inline indexing for datasets, you may want to avoid indexing referenced datasets that are shared among a large number of types. For example, a PDF describing a specific material may be attached to a thousand parts that use it. To improve efficiency and performance, refine the indexing to include only the datasets and relations that best represent the type.
- If you want to index dataset files, set the **AWS_FullTextSearch_Index_Dataset_File_Content** preference to **on**.
- **AW_FTSIndexer_skip_modifications_via_relations** controls whether TcFTSIndexer can find modifications to a type by running relation queries during the synchronization indexing flow. For example, if a dataset object is attached to a **DocumentRevision** type or any of its subtypes and is already indexed, set the preference to **false** before running a synchronization indexing flow. The default is **true**, which skips the queries.

Factors to consider:

- If you want to use inline indexing for dataset file content, you must reindex the entire set of dataset files.
- User permission is validated on each object. User permission can filter out a dataset but still return a document revision.
- For datasets with content you do not want indexed, on their own or as part of a parent business object, you can create a custom filter to filter them out.

Example

Return PDF documents that have a specified attachment relation to the type using **Awp0DatasetTypeToBeIndexedInline**:

```
INHERIT:TC_Attaches:PDF
```

When indexing a dataset with a PDF, the primary business object UID referenced by the **TC_Attaches** relation type is indexed. When a search matches content in the PDF, the PDF dataset and its parent business object referenced by the **TC_Attaches** relation type are returned.

You can specify both **INHERIT** and **NO_INHERIT** rules for the same object.

```
INHERIT*:MSWORDX, NO_INHERIT:IMAN_specification:PDF
```

```
NO_INHERIT:TC_Attaches:PDF, INHERIT:IMAN_specification:HTML~MSWordX
```

Note that **INHERIT** rules apply to the type where it's defined and its subtypes. **NO_INHERIT** rules apply only to the type where it's defined.

Configure types to return for global search

You can limit global search results so that they do not include specific object types. The types must be specified in the **AWC_Global_Search_Suppress_Types_From_Results** preference. The search omits the specified object types and their filter properties from the search results.

Specify one or more internal names for the object types you want to suppress in the preference. Type hierarchy applies to specified object types. For example, if one of the preference values is **Dataset**, dataset types (such as PDF, Microsoft Word, and text), their respective child types, and filter properties are suppressed from the search results.

Index and search alternate IDs

Alternate identifiers store information (such as part numbers and attributes) about the same part from different perspectives. They allow different types of users to display an item according to their own rules rather than according to the rules of the user who created the object. Only Item business objects or its children use alternate IDs.

By default, alternate IDs cannot be indexed for searching in Active Workspace because the **altid_list** property on the **ItemRevision** business object is a run-time property, and run-time and relation properties are not supported for indexing. Only attribute, compound, table, and reference properties are indexed.

To index and search alternate IDs, you must define a compound property on the business object type that add the alternate IDs. The compound property uses a reference by relation to get to the source property.

For example, assume you want to find **IdentifierRev** objects. By default, **Identifier** and **IdentifierRev** business objects have the **Identifier** storage class. So, you must define a compound property as follows:

```
<TcCompoundPropertyRule destTypeName="ItemRevision" destPropertyName="dev3AltID"
  sourceTypeName="Identifier" sourcePropertyName="idfr_id" isReadOnly="false"
  pathToSource="REFBY(altid_of,Identifier).REF(suppl_context,Identifier).idfr_id"
  description=""/>
```

This does a backward reference to find the **IdentifierRev** objects through the **altid_of** relation and then finds all the references of **Identifier** objects through the **suppl_context** reference property. The **idfr_id** property is fetched from these **Identifier** objects.

Now assume there is a custom **Identifier** business object like **MyIdentifier** and **MyIdentifierRev**:

```
Identifier
|- IdentifierRev
|- MyIdentifier
|- MyIdentifierRev
```

Define the compound property as follows:

```
<TcCompoundPropertyRule destTypeName="ItemRevision" destPropertyName="dev3AltID"
  sourceTypeName="MyIdentifier" sourcePropertyName="idfr_id" isReadOnly="false"

  pathToSource="REFBY(altid_of,MyIdentifierRev).REF(suppl_context,MyIdentifier).idfr_id"
  description=""/>
```

Searching with the alternate ID returns the **ItemRevision** objects matching the keyword. A property-specific search for the compound property is also supported.

Note:

There is a limitation with the Business Modeler IDE. You cannot add these types of compound properties from the Business Modeler IDE. They must be added manually to the template, and you must validate the template by reloading it in the Business Modeler IDE.

The **AW_FTSIndexer_skip_modifications_via_relations** preference controls whether TcFTSIndexer can find modifications by running relation queries during the synchronization flow. The default value is **true**, which skips the queries. Set the value to **false** to index alternate IDs on Items.

Configure search for multiple sites

Multi-Site Collaboration enables sharing objects among multiple sites. These Object Directory Services (ODS) objects are published and replicated across multiple sites. The published objects can then be indexed for Active Workspace users. The **Filter** panel on the **Results** tab displays local results or remote results. The published object tracks updates to the master and determines if shared data needs to be updated at other sites.

1. You need to perform the setup and indexing tasks that **enable multi-site indexing**.
2. Enable multi-site search for users by setting the **AWC_Search_DisplayODSContent** preference to **true**. **AWC_Search_DisplayODSContent** allows the **Filter** panel to display the **Search site** category.

If there are no published records available, then only local results are returned. Choosing **Remote** to filter displays no objects.

Multi-site Collaboration information is available from the Teamcenter help.

Filter object data with a custom user exit method

An administrator can implement a user exit action to filter certain objects and prevent them from being extracted to the search index. The user exit action provides a way to specify criteria for an exclusion, such as ID values or objects that have certain property values. This procedure shows how to implement a custom method, register it against a provided message, and add custom filtering to the implementation.

1. Create a new library or use an existing one to implement custom indexer filter logic. For example, create **libIndexerFilter** and add this library name to the **TC_customization_libraries** preference.
2. Within the library, define a *library-name_register_callbacks* method and export it.
3. In this method, register custom implementations against the **AWS2_filter_object_indexing_user_exit** and **AWS2_filter_dataset_object_indexing_user_exit** user exit messages:
For example:

- Register the **AWP0CUSTOM_skip_objects_to_index** method against the **AWS2_filter_object_indexing_user_exit** message.
- Register the **AWP0CUSTOM_skip_dataset_objects_to_index** method against the **AWS2_filter_dataset_object_indexing_user_exit** message.

```
#define AWS2_filter_dataset_object_indexing_msg
    "AWS2_filter_dataset_object_indexing_user_exit"

#define AWS2_filter_object_indexing_msg
    "AWS2_filter_object_indexing_user_exit"

extern "C" DllExport int libawp0custom_register_callbacks()
{
    CUSTOM_register_exit( "libawp0custom", AWS2_filter_object_indexing_msg,
        (CUSTOM_EXIT_ftn_t) AWP0CUSTOM_skip_objects_to_index );

    CUSTOM_register_exit( "libawp0custom", AWS2_filter_dataset_object_indexing_msg,
        (CUSTOM_EXIT_ftn_t) AWP0CUSTOM_skip_dataset_objects_to_index );
    return 0;
}
```

4. Implement a **AWPOCUSTOM_skip_objects_to_index** custom method. This method is called when nondataset UIDs are indexed. For example:

```

/*
Custom function to filter out a list of UIDs. The UIDs on indices marked
by this function will not be indexed.
*/
extern int
  AWPOCUSTOM_skip_objects_to_index( int* decision, va_list args )
{
  va_list largs;
  va_copy(largs, args );

  // List of UIDs to be filtered
  const char** uidsToFilter = va_arg( largs, const char** );

  // Type of each UID in uidsToFilter
  const char** uidsType = va_arg( largs, const char** );

  // Size of UID list being passed
  int listSize = va_arg( largs, int );

  // Output parameter - list of indices of UIDs to be filtered/skipped
  unsigned int** skippedUidsIndices = va_arg( largs, unsigned int** );

  // Size of the output UID list
  int* sizeofIndicesToSkip = va_arg( largs, int* );

  va_end( largs );

  // Set the decision flag to show the custom method has been reached.
  *decision = ONLY_CURRENT_CUSTOMIZATION;

  std::vector< unsigned int > indicesToSkip;
  for( int indx = 0; indx < listSize; indx++ )
  {
    // Sample filtering logic goes here
    // For any UID in the uidsToFilter list that needs to be filtered,
    // put the associated index into the indicesToSkip vector.
    // The vector will be copied to output paramter skippedUidsIndices
    // at the end of this function.

    // Filtering can be done based on UID value, its type, or any object
    // property value. Filtering based on property value needs the
    // associated object loaded first.

    // Filtering based on type, for example, filtering Design Revisions
    if( strcmp( uidsType[indx], "Design Revision" ) == 0 )
    {
      indicesToSkip.push_back( indx );
      continue;
    }

    // Filtering based on UID value
    if( strcmp( uidsToFilter[indx], "UID_VALUE3" ) == 0 ||
        strcmp( uidsToFilter[indx], "UID_VALUE4" ) == 0 )
    {
      indicesToSkip.push_back( indx );
    }
  }
}

```

```

    }

    // Copy the indicesToSkip vector to the skippedUidsIndices array
    // The calling function will free any allocated memory,
    // so memory shouldn't be freed here
    *sizeOfIndicesToSkip = indicesToSkip.size();
    if( *sizeOfIndicesToSkip > 0 )
    {
        *skippedUidsIndices = static_cast<unsigned int*>( MEM_alloc(
            *sizeOfIndicesToSkip * sizeof( unsigned int ) ) );
        memcpy( *skippedUidsIndices, &indicesToSkip[0],
            *sizeOfIndicesToSkip * sizeof( unsigned int ) );
    }
    return 0;
}

```

5. Implement another custom method for **AWP0CUSTOM_skip_dataset_objects_to_index**. Follow the same pattern shown in the previous step for **AWP0CUSTOM_skip_objects_to_index**. This method is called when dataset UIDs are indexed.
6. Enable filtering in these preferences for the **AWP0CUSTOM_skip_dataset_objects_to_index** method:
 - In **AWC_Search_Enable_UserExit_Datasets**, enable dataset filtering.
 - In **AWC_Search_Enable_UserExit_NonDatasets**, enable nondataset filtering.

Note:

Enabling **AWC_Search_Enable_UserExit_NonDatasets** may degrade performance.

Configure Solr for a two-tier environment

If you use Active Workspace with a two-tier rich client, such as when running with NX, then you may need to set up Solr so users can search in this environment.

1. Open the *tc_profilevars.bat* file for the two-tier environment.
2. Change the location of the password file for the **TC_INDEXING_ENGINE_PASSWORD_FILE** environment variable. For example:

```
TC_INDEXING_ENGINE_PASSWORD_FILE=path_to_TEAMCENTER_solr_admin.pwf.
```

3. Set **TC_INDEXING_ENGINE_USER=solr_admin**.
4. Restart the two-tier **tcserver**.

Example of search results performance

As part of indexing configuration, you should have **optimized hardware for indexing**. Our testing provides the following global search results for query performance:

15,106,617 objects in Solr	Search value	Objects found	Overall Time (seconds)	Solr Time (seconds)
String searches	motor*	14,587	4.82	.08
	engine*	43,452	5.09	.11
	car*	133,474	5.19	.19
	data*	606,741	5.54	.61

Configure Advanced search

You can enable the Advanced search feature for Active Workspace users. Column sorting in results tables supports specific types of properties.

Enable Advanced Search

The **AW_Advanced_Search_Visibility** preference settings:

- **true** (the default) displays the **Advanced Search** link beneath the global search box. **true** also displays the **Advanced** page in the **Search** page header area.
- **false** removes the **Advanced Search** link and the **Advanced** page.

AW_Advanced_Search_Visibility is a site-level preference that can be overridden for specific users.

Note:

Use Query Builder to create advanced search queries. See *Creating queries for Advanced Search* in the Teamcenter documentation for details about working with search queries. When modifying and deploying existing queries, users must close and reopen their session to use the updated queries.

Sorting columns in Advanced Search results tables

Column sorting support in results tables is determined by type of property.

Supported Users can sort on primary, persistent property types such as **int**, **double**, and **string**.

Users can sort on single reference types for primary persistent properties. For example, you can sort on **owning_user.user_id** but not **Item.owning_user.user_id** because the latter is a compound reference.

For local queries, the query flags **QRY_RUN_BY_TC** and **QRY_RUN_BY_TC_PLUS_PROCESS** can support column sorting.

Not supported Users cannot sort on arrays, run-time properties, compound properties, and table properties. Callback queries do not support column sorting.

Return external business objects with a custom user exit method

Advanced search can return objects from an alternate data source such as from an ERP system. You need to implement a custom user exit query method to retrieve external objects. The external object is treated as a run-time business object.

The objects are retrieved through an **Advanced** search query in Active Workspace. The query is created in the rich client Query Builder.

Create a custom user exit query

This procedure explains how to create the custom user exit query. If you already have a custom query for an **Advanced** search, you can skip to the next procedure.

1. Create a new library or use an existing one to implement the custom external object logic. For example, you might use **libawp0custom.dll**, or the sample source file *TC_ROOT\sample\examples\user_query.c*. Create a custom method for your query. Then compile the library and put it in *tc_bin*.
2. Create a query with a name that matches the custom library method. You can export a saved query to an XML file, and then import the query definition XML file. For information on query definitions and Query Builder, refer to Teamcenter help.
3. Run the command **tc_set_query_where_run**. Specify the query name of the XML file (such as **-query=ExternalObject_exit_query**) and set the query flag to **-run=user**.
4. Add the name of the custom library that you created to the preference **TC_customization_libraries**.

Set the scope to **Site**, the context to **Teamcenter**, and **Values** to the library name (such as **libawp0custom**).

Configure what is displayed for the object

After your query is created and compiled, set up which properties display for the returned object in **Advanced** search results.

1. Create a preference to link the object properties to the table cells in the search results. Create a *query-name_AW.CellProperties* preference and specify which properties you want to display for the external object.

Use the same query name as the user exit query name. For a query named **EXTERNAL_exit_query**, then the preference name would be **EXTERNAL_exit_query_AW.CellProperties**.

2. Restart the server manager.

The user chooses the custom query object type *query-name_AW* from the **Advanced** search panel in Active Workspace. Then the user enters the query criteria and searches for the results.

Configure Quick search

Create Advanced search queries in Teamcenter Query Builder. Use the following preferences to configure the Quick search feature for users.

- **Quick_Access_Queries** specifies which queries appear in the **Quick** search list.
- **Quick_Access_Queries_Attribute** specifies the criteria attribute displayed for a query.

See *Quick search preferences* in the Teamcenter documentation for information about working with Quick searches. When existing queries are modified and deployed, users must close and reopen their current sessions to use the updated queries.

Configuring shape search

If you installed the **Shape Search** feature, you can configure it using the following preferences:

- **GeolusServer**
Defines the URL used for communication between shape search and Geolus.
- **AWC_ShapeSearch_Max_Result_Count**
Improve performance by setting the maximum number of search results. The default is 3000, and the maximum is 5000.
- **SS1_DASS_enable**
Enables and disables shape search.
- **SS1_DASS_shape_default**
Specifies the default shape similarity for shape search.
- **SS1_DASS_size_default_max**
Specifies the default upper range limit a user can specify when applying a size filter.
- **SS1_DASS_size_default_min**

Specifies the default lower range limit a user can specify when applying a size filter.

- **SS1_DASS_size_lower_limit**
Specifies the smallest lower range limit a user can specify when applying a size filter.
- **SS1_DASS_size_upper_limit**
Specifies the highest upper range limit a user can specify when applying a size filter.

To best plan implementing search for your users, see the *Administering Search Guidemap* in the Teamcenter help.

Set the GeolusServer preference

The **Shape Search** feature in Active Workspace uses the Geolus shape search engine. To enable communication between shape search and Geolus, you must create the **GeolusServer** preference.

Note:

This step is required only if you installed **Shape Search**.

Using **Preference Management** in Active Workspace or **Organization** in the rich client, create the **GeolusServer** preference with the following properties:

- **Name:** **GeolusServer**.
- **Category:** **DecisionApps.ShapeSearch.Preferences**.
- **Description:** Type a useful description for the preference.
- **Value:** Type the Geolus server URL, in the following format:
protocol://gServer:gPort/gContext

protocol can be **http** or **https**.

gServer is the machine name or IP address of the machine running the Geolus server. It must be accessible by all Teamcenter clients that need to connect to it.

gPort is the port number that the server uses to handle HTTP or HTTPS requests.

gContext is the context root of the Geolus server.

Troubleshooting search

Find logs related to search

The Active Workspace search operation makes calls to the server to return the results. In the case of a service-oriented architecture (SOA) or Solr error, the Active Workspace client has limited information about the error. Investigate information in the following logs.

Be sure you revert your logging levels and variables back to their previous settings after you complete your debugging, and restart the system you are troubleshooting.

tcserver logs

- Check the **tcserver syslog** files. Find the **syslog** file associated with the user and the SOA call for more information about a server-side failure.
Find the correct **syslog** files by searching for the user name to narrow the list of applicable **syslog** files, and then search for the string **performSearch** in the logs associated with the user.
The **syslog** files are located in the *Temp* directory. If you do not see them, be sure that **TC_KEEP_SYSTEM_LOG** is set to true.
- For debug logging, open the file:

```
TC_LOGGER_CONFIGURATION\logger.properties
```

Change the logging level to **DEBUG** for:

```
logging.rootLogger
logging.logger.Teamcenter
logging.logger.Teamcenter.Soa.Communication
```

- Check the **Journal** file to get a low level analysis of the **tcserver**. You can trace all the methods that were called by setting the following environment variables:

```
TC_JOURNAL=FULL
TC_JOURNALLING=ON
TC_JOURNAL_LINE_LIMIT=0
```

Solr logs

If you get Solr failures, find the *TC_ROOT\solr-version\server\logs\solr.log* files.

Web browser console logging

When performing a search, the user may see a message that acknowledges a failure with a brief explanation. For server errors, a log statement with more information is also provided in the console of

the web browser. Web browser console windows are usually available through the web browser developer tools.

Troubleshoot search results

Object data not returned

If object data was added, modified, or deleted in Teamcenter, but a search in Active Workspace does not return results from those changes, check that:

- The **runTcFTSIndexer utility** synchronization flow is running.
- The database triggers are installed.

Incorrect search counts

In `solr-version\server\solr\collection1\conf\solrconfig.xml`, change the value of **debugoptions**:

```
<str name="debugoptions">2</str>
```

Restart Solr from the console and capture all the logging for troubleshooting.

You can also examine the results coming back from the server using Fiddler or Firebug.

Search results count in Active Workspace and Teamcenter rich client don't match

In Active Workspace, only the revisions matching the current revision rule are returned, so that count may not match the count returned by the Teamcenter rich client. To see all revisions:

- In Active Workspace, set the revision rule to **Precise Only**, which returns all revisions for object data.
- Be sure that all failed and pending objects are indexed. To determine the failed and pending objects, use the **runTcFTSIndexer utility**:

```
runTcFTSIndexer -task=objdata:show n uid_file_dir
```

For the argument *n*, specify **1** to get the pending objects or **3** to get the failed objects.

Troubleshoot search performance

You can generate logging and analyze search performance issues. You can try to isolate a performance issue using one method or you can combine methods for the most informative analysis.

Syslog debugging for performance

Set log levels for *logger.properties* to **DEBUG**. The logging provides a hierarchy of the search call. Search for **fnd0performSearchBase** to locate the beginning and end point of the search call.

Performance journal logging

Check the *.pjl* file. You can get performance feedback by setting an environment variable:

```
TC_JOURNAL_PERFORMANCE_ONLY=true
```

Restart the Pool Manager and the **tcserver**.

These files can become very large, so run only the use case you are investigating. Journal logging creates a large amount of data that can be difficult to analyze. Restore normal operation by resetting **TC_JOURNAL_PERFORMANCE_ONLY**.

Log SQL performance

You can set debug logging for slow SQL queries using environment variables. The logging can slow the **tcserver** performance.

1. Open the *tc_profilevars* file.
2. To log any SQL statement that takes longer than one second, set **TC_SLOW_SQL=1**.
3. To send all SQL statements and their timing to the **syslog** file, set **TC_SQL_DEBUG=PT**.
4. Restart the Pool Manager and the **tcserver**.
5. Check the **syslog** file for **DEBUG** messages.

HTTP Archive (HAR) files for network calls

Create an HTTP archive file (HAR) for logging web browser performance of network transactions. HAR files can help investigate network calls made by Active Workspace.

Creating HAR files can vary among web browsers, but the essential process is the same. Using the web browser developer tools, find the **Network** menu or tab and capture the network traffic. Run only the browser request you are investigating. Be sure you choose the action to save the HAR file. (For example, in Google Chrome, when you hover over the logged entry, you can choose **Save as HAR with content** from the context menu. Internet Explorer exports XML files.)

Customize Solr

Solr URL single point-of-failure

TcFtsIndexer and **TcServer** can access alternate Solr URLs in case of connection issues with any Solr URL. If one Solr URL fails, **TcFtsIndexer** can continue indexing and the client can search to find all objects.

Following are the prerequisites:

- SolrCloud is required and is configured to support fail-over.
- The leaders and zookeepers are configured on multiple machines.
For information about SolrCloud leaders and zookeepers, see:

<https://cwiki.apache.org/confluence/display/solr/SolrCloud>

In a fail-over case, even if one machine goes down:

- **TcFtsIndexer** can connect to an alternate Solr URL to index the data. Objects indexed on the failed URL should still be accessible from the alternate URL.
- **tcserver** can connect to an alternate URL to access all the objects indexed.

Multiple Solr URLs are configurable through an install panel in TEM. To support multiple Solr URLs, **AWS_FullTextSearch_Solr_URL** is a multi-value preference. The process is as follows:

1. **TcFtsIndexer** and **tcserver** try to connect to the Solr URL using this preference.
2. Access to the URL is tried three times and, if it fails, the next available Solr URL in the preference list is used. This continues until access is successful or the entire list has been tried.
3. **TcFtsIndexer** and **tcserver** output warning messages for the failed URL until the issue is resolved.

Configure Solr for HTTPS

By default, Solr uses HTTP, but you can configure Solr to use HTTPS for SSL. **TcServer** and **TcFtsIndexer** are programmed to accept self-signed certificates as well as Certificate Authority (CA) signed certificates.

1. Choose a certificate signed by a trusted CA for Solr, or create a self-signed certificate using Java **keytool** or **openssl**.

Be sure that your security certificates are available before you proceed. Store the generated certificates in a keystore.

2. Stop Solr.
3. Copy the certificates, for example, *IdentityKeystore.jks*, *TrustKeystore.jks* or *private.cer*, to the *SOLR_HOME\server\etc* directory.
4. Update the *SOLR_HOME\server\etc\jetty-ssl.xml* file with the certificate names:

```
<Set name="keyStorePath"><Property name="solr.jetty.keystore"
  default=". /etc/IdentityKeystore.jks" /></Set>
<Set name="TrustStorePath"><Property name="solr.jetty.truststore"
  default=". /etc/TrustKeystore.jks" /></Set>
```

5. Uncomment the following statements in *SOLR_HOME\bin\solr.in.cmd* and update them with the full path to their respective certificates. For example:

```
set SOLR_SSL_KEY_STORE=
  D:\SolrVersion\Leader1\solr-version\server\etc\IdentityKeystore.jks
set SOLR_SSL_KEY_STORE_PASSWORD=secret
set SOLR_SSL_TRUST_STORE=
  D:\SolrVersion\Leader1\solr-version\server\etc\TrustKeystore.jks
set SOLR_SSL_TRUST_STORE_PASSWORD=secret
set SOLR_SSL_NEED_CLIENT_AUTH=false
```

6. If you are installing Solr as a Windows service, set up the service for HTTPS.
 - a. In the *SOLR_HOME\server\etc\jetty-ssl.xml* file, find the following lines and comment them:

```
<!--
  <Call class="org.apache.solr.util.configuration.SSLConfigurationsFactory"
    name="current">
    <Get name="keyStorePassword" id="keyStorePassword"/>
    <Get name="trustStorePassword" id="trustStorePassword"/>
  </Call>
-->
```

- b. Find the KeyStore Password lines and replace them with:

```
<Set name="KeyStorePassword">
<Env name="SOLR_SSL_KEY_STORE_PASSWORD" default="secret"/>
</Set>
```

- c. Find the TrustStore Password lines and replace them with:

```
<Set name="TrustStorePassword">
<Env name="SOLR_SSL_TRUST_STORE_PASSWORD" default="secret"/>
</Set>
```

- d. In *SOLR_HOME\server\etc\jetty-https.xml*, replace the array for the Eclipse Jetty Server connection with the following:

```
<Array type="org.eclipse.jetty.server.ConnectionFactory">
  <Item>
    <New class="org.eclipse.jetty.server.SslConnectionFactory">
      <Arg name="next">http/1.1</Arg>
      <Arg name="sslContextFactory"><Ref refid="sslContextFactory"/></Arg>
    </New>
  </Item>
  <Item>
    <New class="org.eclipse.jetty.server.HttpConnectionFactory">
      <Arg name="config"><Ref refid="sslHttpConfig"/></Arg>
    </New>
  </Item>
</Array>
```

e. Install the Windows service.

- A. Open *SOLR_HOME\solrWinService.bat* in a text editor. Find **--module=http** and change it to **--module=https**. Save the file.
- B. As an administrator, open a command window. Install the service by running *solrWinService.bat -i*.
- C. After installing the service, open Windows **Services** and find **Active Workspace Indexing Service**. Check the user account required to start it under **Log On As**.

If **Log On As** is set to **Local System**, change it to the account that logs on and starts the service. Do not use the Solr user name and password as the user account.

Open the service **Properties**, click the **Log On** tab, select **This account**, and specify the name and password.

7. Restart Solr.
8. Test the implementation. Log on to the Solr administrator page using HTTPS.

Test that HTTP no longer connects.

9. Update the **AWS_FullTextSearch_Solr_URL** preference to use the HTTPS URL, for example, **https://mysolrserver:8984/solr**.
10. Clear *TC_ROOT\TcFTSIndexer\cache\TcFtsIndexerSession.cache*.

Solr cloud configuration

SolrCloud is a configuration that supports high availability using replicas and distributed indexes using sharding. These configurations may be used independently or in combination. These terms help explain Solr's cloud configuration options:

Node	A single instance of Solr.
Collection	A single search index.
Shard	A logical section of a single collection.
Replica	A physical instance of a shard.
Leader	The replica that is designated to coordinate requests.
Cluster	A group of nodes.
ZooKeeper	Handles Solr configurations, load balancing, and fail-over behavior.

Solr single node configuration

By default, Solr is configured in a single node, as a single shard with a single replica. This is not a SolrCloud configuration, so ZooKeeper is not part of this configuration.

Understanding the cloud configuration choices

SolrCloud provides a distributed environment to manage your indexed content across servers. All Solr cloud configurations use ZooKeeper. You can choose the approach that best fits your indexing strategy and search objectives:

SolrCloud configuration	Underlying technology	Description	Benefit
High availability	A shard can have multiple replicas.	Queries are managed by the lead replica.	Maintains availability and provides fail-over redundancy.
Distributed index	Multiple shards, each shard has its own replica.	Queries are sent to all replicas simultaneously.	Improves performance, especially for large indexes.
Combination of both	Multiple shards, each shard has its own set of replicas.	A replica from each shard is grouped with replicas from other shards to form a group. Queries are sent to the lead group of replicas.	Provides availability, performance, and fail-over redundancy.

High availability configuration

This configuration is designed to manage failover if a node loses connection due to an issue with the physical server or the network. You can extend the number of replicas per shard. As a best practice, set up the replicas on separate machines. The leader position is assigned to the first replica started.

Example:

Replica1 (the leader) is on server1, replica2 is on server2. The index is duplicated across replica1 and replica2.

If server1 has a malfunction and loses its connection, search requests are routed to replica2 automatically by ZooKeeper. Replica2 on server2 takes over as leader. The new replica leader remains leader unless it's stopped or loses its connection.

Distributed index configuration

This configuration is designed to improve performance by distributing the index among shards. Though similar to high availability, each shard has its own replica. Search requests are sent to each replica simultaneously to query all sections of the index. The results are combined into a single response.

Example:

Replica1 for shard1 is on server1, replica2 for shard2 is on server2. The index is split between shard1 and shard2. Requests are sent to both replica1 and replica2.

If server1 has a malfunction and loses its connection, search requests are routed automatically to replica2 on server2. Requests are only processed by replica2. The portion of the index on shard1 is unavailable.

Combination configuration

This configuration combines the high availability and distributed index approaches in a single configuration. The index is distributed among multiple shards, and then each shard is duplicated to multiple replicas. You can choose to put each replica on a separate server, or you can combine the replicas into groups, where each group on a server contains one replica for each shard.

There is no limit to the number of replicas for each shard, which allows you to configure multiple replica groups.

Example:

Replica1 for shard1 and replica1 for shard2 are grouped together on server1. Replica2 for shard1 and replica2 for shard2 are grouped together on server2. If server1 has a malfunction and loses its connection, all search requests for both shards are fulfilled by the replicas on server2. The group of replicas on server2 become the leaders.

ZooKeeper setup

SolrCloud requires ZooKeeper for load balancing and maintaining Solr configurations. For SolrCloud environments, set up an external ZooKeeper ensemble. To provide the best support for failover scenarios, install ZooKeeper on most or all of the replica servers. A minimum of three ZooKeepers is required for the ensemble to have proper failover; you must always use an odd number of nodes when

scaling further. Each ZooKeeper runs as an independent application; however, the Solr start up script uses all the available ZooKeepers.

Download and unzip the supported version of ZooKeeper.

1. Create a *data* directory under *ZOOKEEPER_HOME*.
2. In *ZOOKEEPER_HOME\conf*, rename *zoo_sample.cfg* to *zoo.cfg*.
3. Open *zoo.cfg* and find the **dataDir** property. Modify the **dataDir** path to point to the *data* directory. The *data* directory stores all the configuration files for SolrCloud

Use double backslashes on a Windows system.

```
dataDir=C:\\workdir\\apps\\zookeeper-version\\zookeeper-version\\data
```

Continue if you are creating a ZooKeeper ensemble. If you are creating a single ZooKeeper, skip to **updating the zkServer.cmd**.

4. Add statements to the bottom of the file, one for each ZooKeeper instance that will be running. This example defines two servers:

```
server.ID=realdealsolr:2888:3888
server.ID=realdealdisp:2888:3888
```

Identify the ZooKeeper servers in *server.ID*. The ports 2888 and 3888 are the default communication ports for leader selections.

Save and close the file.

5. Create a file named *myid* and enter the ID of the ZooKeeper server. The ID was defined in *zoo.cfg* using *server.ID*, for example, **myserver.234** where the ID is **234**.

Each ZooKeeper server needs its own *myid* file. Save it in the *\data* folder.

6. Open the *ZOOKEEPER_HOME\bin\zkServer.cmd* script. Add **"-Djute.maxbuffer=5097152"** as follows:

```
call %JAVA% "-Djute.maxbuffer=5097152" "-Dzookeeper.log.dir=%ZOO_LOG_DIR%"
"-Dzookeeper.root.logger=%ZOO_LOG4J_PROP%" -cp "%CLASSPATH%"
%ZOOMAIN% "%ZOOCFG%" %*
```

Save and close the file.

7. To start ZooKeeper, run **zkServer.cmd**.

8. For each ZooKeeper installation, repeat these steps. All of the ZooKeeper servers must be identical except for the *myid* server ID.

Create configsets and collections

Determine the number of shards and replicas that your configuration needs. The following sample procedure combines the high availability approach and the distributed index approach and creates replacement Solr collections and two shards. The data must be reindexed.

1. Copy the current *SOLR_HOME* collections (**collection1**, **collection2**, **admin**, and **ProductScopeCollection**) to a temporary directory.
2. Make a copy of the existing Solr installation on the Solr server, referred to as *Leader1_SOLR_HOME* going forward.
3. Open *Leader1_SOLR_HOME\server\etc\webdefault.xml*. Comment the statements for **<security-constraint>** and **<login-config>**.

This action disables Basic authentication. To use Basic authentication with SolrCloud, refer to [Basic authentication setup](#).

4. Delete all the collections under *Leader1_SOLR_HOME\server\solr*.
5. Open *Leader1_SOLR_HOME\runSolr.bat* and find:

```
call %SCRIPT_DIR%/bin/solr start -f
```

and replace it with:

```
%SCRIPT_DIR%/bin/solr start -f -cloud -s ..\server\solr -p 8983 -z hostname:port
```

The **-z** parameter specifies the ZooKeeper machine and port. If you have more than one ZooKeeper, specify them in a comma-separated list, for example, **-z server1:2181,server2:2181,server3:2181**.

6. Start the *Leader1_SOLR_HOME* instance. Confirm that Solr has started in cloud mode and there are no existing collections.
7. Create your collections for *Leader1_SOLR_HOME*. Open another command prompt and run the following command for each collection you copied to the temporary directory.

```
solr create_collection -c collection_name -d Temp_path\collection_name  
-n collection_name -shards n -p 8983 -replicationFactor n
```

-shards is the number of shards for your configuration, and **replicationFactor** is the number of replicas per shard for your configuration. For example:

```
solr create_collection -c collection1 -d c:\SolrTest\collection1 -n collection1 -shards 2 -p 8983 -
replicationFactor 2
```

8. Before proceeding, you may configure Solr to use SSL by **configuring HTTPS**. Perform this procedure on *Leader1_SOLR_HOME* so that subsequent replicas of *Leader1_SOLR_HOME* include the SSL configuration.
9. Stop Solr. Copy the entire *Leader1_SOLR_HOME* instance to all servers which will host replicas.

Going forward, *Leader1_SOLR_HOME* refers to the original instance and **Solr_Cloud_Home** refers to its copies.

10. Determine which servers will host which shards and replicas, based on your server infrastructure plan and your strategy for grouping replicas.

The directories are labeled with the shard number and the replica number they control. On *Leader1_SOLR_HOME* and each of the **Solr_Cloud_Home** servers, determine which directory under *\server\solr* you want that server to control.

Make sure that each server instance has a unique directory, and there are no duplicates across all the others. Duplicates would be the directories already assigned to other servers. Then delete the duplicate directories on each server.

11. Start *Leader1_SOLR_HOME* instance first, and then start each **Solr_Cloud_Home** instance in the order that you would like the leaders assigned.

Basic authentication setup

Download **cURL** from <https://curl.haxx.se/download.html> and unzip to a directory. Update the **Path** environment variable with the path to the *Curl\bin* directory.

1. Create a *security.json* file in *Leader1_SOLR_HOME\server\scripts\cloud-scripts*. Provide the **username** for the encrypted password set up for the Solr administrator.

```
{
  "authentication":{
    "blockUnknown": true,
    "class":"solr.BasicAuthPlugin",
    "credentials":{"username":"IV0EHq1OnNrj6gvRCwvFwTrZ1+z1oBbnQdiVC3otuq0=
      Ndd7LKvVBAaZIFOQAVi1ekCfAJXr1GGfLtRUXhgrF8c="}
  }
}
```

The example uses the encrypted password for the standard Solr password SolrRocks.

2. Run the following command from *Leader1_SOLR_HOME\bin*:

```
solr zk cp file:Leader1_SOLR_HOME\server\scripts\cloud-scripts\security.json  
zk:/security.json -z localhost:2181
```

3. Start SolrCloud.

4. Run **cURL**:

```
curl --user solr_admin:SolrRocks http://localhost:8983/solr/admin/authentication  
-H 'Content-type:application/json' -d '{"set-user":{"solr_admin":{"NewPassword"}}}'
```

In this case, specify **SolrRocks** as the original password, which is required by *security.json*. Then change the password in **NewPassword**.

5. Go to the Solr admin page (<http://hostname:port/solr/>) and log on using the username and password you just set up.

Siemens Digital Industries Software

Headquarters

Granite Park One
5800 Granite Parkway
Suite 600
Plano, TX 75024
USA
+1 972 987 3000

Americas

Granite Park One
5800 Granite Parkway
Suite 600
Plano, TX 75024
USA
+1 314 264 8499

Europe

Stephenson House
Sir William Siemens Square
Frimley, Camberley
Surrey, GU16 8QD
+44 (0) 1276 413200

Asia-Pacific

Suites 4301-4302, 43/F
AIA Kowloon Tower, Landmark East
100 How Ming Street
Kwun Tong, Kowloon
Hong Kong
+852 2230 3308

About Siemens Digital Industries Software

Siemens Digital Industries Software is a leading global provider of product life cycle management (PLM) software and services with 7 million licensed seats and 71,000 customers worldwide. Headquartered in Plano, Texas, Siemens Digital Industries Software works collaboratively with companies to deliver open solutions that help them turn more ideas into successful products. For more information on Siemens Digital Industries Software products and services, visit www.siemens.com/plm.

This software and related documentation are proprietary and confidential to Siemens.

© 2021 Siemens. A list of relevant **Siemens trademarks** is available. Other trademarks belong to their respective owners.