



TEAMCENTER

Indexing Data and Configuring Search

Active Workspace 6.3

Unpublished work. © 2023 Siemens

This Documentation contains trade secrets or otherwise confidential information owned by Siemens Industry Software Inc. or its affiliates (collectively, "Siemens"), or its licensors. Access to and use of this Documentation is strictly limited as set forth in Customer's applicable agreement(s) with Siemens. This Documentation may not be copied, distributed, or otherwise disclosed by Customer without the express written permission of Siemens, and may not be used in any way not expressly authorized by Siemens.

This Documentation is for information and instruction purposes. Siemens reserves the right to make changes in specifications and other information contained in this Documentation without prior notice, and the reader should, in all cases, consult Siemens to determine whether any changes have been made.

No representation or other affirmation of fact contained in this Documentation shall be deemed to be a warranty or give rise to any liability of Siemens whatsoever.

If you have a signed license agreement with Siemens for the product with which this Documentation will be used, your use of this Documentation is subject to the scope of license and the software protection and security provisions of that agreement. If you do not have such a signed license agreement, your use is subject to the Siemens Universal Customer Agreement, which may be viewed at <https://www.sw.siemens.com/en-US/sw-terms/base/uca/>, as supplemented by the product specific terms which may be viewed at <https://www.sw.siemens.com/en-US/sw-terms/supplements/>.

SIEMENS MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS DOCUMENTATION INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT OF INTELLECTUAL PROPERTY. SIEMENS SHALL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, CONSEQUENTIAL OR PUNITIVE DAMAGES, LOST DATA OR PROFITS, EVEN IF SUCH DAMAGES WERE FORESEEABLE, ARISING OUT OF OR RELATED TO THIS DOCUMENTATION OR THE INFORMATION CONTAINED IN IT, EVEN IF SIEMENS HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

TRADEMARKS: The trademarks, logos, and service marks (collectively, "Marks") used herein are the property of Siemens or other parties. No one is permitted to use these Marks without the prior written consent of Siemens or the owner of the Marks, as applicable. The use herein of third party Marks is not an attempt to indicate Siemens as a source of a product, but is intended to indicate a product from, or associated with, a particular third party. A list of Siemens' Marks may be viewed at: www.plm.automation.siemens.com/global/en/legal/trademarks.html. The registered trademark Linux® is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a world-wide basis.

About Siemens Digital Industries Software

Siemens Digital Industries Software is a global leader in the growing field of product lifecycle management (PLM), manufacturing operations management (MOM), and electronic design automation (EDA) software, hardware, and services. Siemens works with more than 100,000 customers, leading the digitalization of their planning and manufacturing processes. At Siemens Digital Industries Software, we blur the boundaries between industry domains by integrating the virtual and physical, hardware and software, design and manufacturing worlds. With the rapid pace of innovation, digitalization is no longer tomorrow's idea. We take what the future promises tomorrow and make it real for our customers today. Where today meets tomorrow. Our culture encourages creativity, welcomes fresh thinking and focuses on growth, so our people, our business, and our customers can achieve their full potential.

Support Center: support.sw.siemens.com

Send Feedback on Documentation: support.sw.siemens.com/doc_feedback_form

Contents

Indexing data

Strategizing indexing	1-1
Understanding the indexing process and determining your indexing strategy	1-1
Considering hardware for TcFTSIndexer	1-3
Selecting the synchronous or asynchronous indexing process	1-5
Comparing synchronous and asynchronous indexing processes	1-7
Identifying the impact of data model changes	1-11
Expediting indexing using a cloned environment	1-12
Indexing data from other sites using Multi-Site Collaboration	1-12
Configuring indexing	1-13
Configuring indexing tasks	1-13
Installing indexing components overview	1-14
Start Solr	1-15
Test TcFTSIndexer connectivity	1-16
Merge the Teamcenter and Solr schemas	1-18
Optimize instances of TcFTSIndexer	1-21
Define index data and filters	1-23
Configuring search index properties	1-26
Change the user running the TcFTSIndexer	1-27
Display snippets from file content	1-27
Configure localized display values	1-28
Index right-to-left languages	1-29
Configuring the reporting microservice	1-30
Configuring asynchronous file content indexing	1-31
Indexing your data	1-39
Perform a full index of object data	1-39
Minimize downtime by updating and indexing on a cloned environment	1-41
Evaluate data model changes for indexing	1-43
Set up TcFTSIndexer synchronization flow	1-44
Run the indexer after updating a custom template	1-47
Index multi-site published objects	1-47
Referencing indexing utilities	1-48
Indexing utilities	1-48
Reviewing indexing results	1-64
Find logs related to indexing	1-64
Reviewing file content indexing status	1-68
Investigate and resolve failures	1-68
Resolving file content indexing errors	1-69
Reviewing indexed file content that search does not display	1-74
Reviewing file content facets that are not displayed	1-75
Enhancing indexing performance	1-76
Customizing indexing	1-78
Customizing the indexer overview	1-78
Customizing the Indexer prerequisites	1-78

TcFTSIndexer installation layout	1-79
TcFTSIndexer extensibility framework	1-80
Indexing object data steps	1-81
Run a sample search indexing flow	1-83
Adding search indexing steps	1-85
Adding new search indexing types	1-87
Deploy a new search indexing type	1-88
Modify an existing search indexing type	1-88
Adding search indexing flows	1-89
ObjData indexing support for saved queries	1-90
Resolving TcFTSIndexer errors	1-90
Update Indexer settings	1-92
Indexing non-Teamcenter data	1-93
Customizing asynchronous file content indexing	1-99
Customizing Solr	1-102
Configure Solr URL support for fail-over	1-102
Configure Solr for HTTPS	1-103
Upgrade Solr	1-105
Update Solr credentials	1-106
Resolving Solr errors	1-107
Configuring SolrCloud	1-110

Configuring search

Configuring search	2-1
Configuring search tasks	2-1
Configure search suggestions	2-2
Setting the default search operator	2-4
Adding wildcards to searches automatically	2-4
Configure searching for common words	2-5
Add synonyms to searches	2-6
Boosting search results	2-6
Configuring results threshold	2-8
Configuring revision rules	2-9
Define search prefilters	2-12
Configuring search prefilters for object sets	2-16
Configuring filter panel behavior	2-17
Configuring saved search	2-24
Configuring column sorting	2-25
Configuring export	2-25
Returning the parent business object for a dataset	2-25
Configuring types to return for global search	2-27
Indexing and searching alternate IDs	2-27
Configure search for multiple sites	2-29
Filter object data with a custom user exit method	2-29
Configure Solr for a two-tier environment	2-31
Configuring Advanced Search	2-32
Return external business objects with a custom user exit method	2-34
Configuring shape search	2-35

Troubleshooting search	2-37
Finding logs related to search	2-37
Troubleshooting search results	2-38
Troubleshoot search performance	2-39



1. Indexing data

Strategizing indexing

Understanding the indexing process and determining your indexing strategy

Indexing is the process of cataloging your objects, metadata, and file content so that your users can search the applicable information efficiently in Active Workspace. Review and **consider your hardware selections** as they relate to indexing. Choose the appropriate indexing method depending on the type of information.

Select a synchronous indexing process or an asynchronous file content **indexing process** based on your file content indexing requirements. Objects without associated file content are always indexed using a synchronous process. For any associated datasets, you can choose a **synchronous or asynchronous indexing process**.

Prerequisites for indexing data

- Install all indexing components.

You can install the Indexing Engine (Solr) and the Indexer using Deployment Center or Teamcenter Environment Manager (TEM).

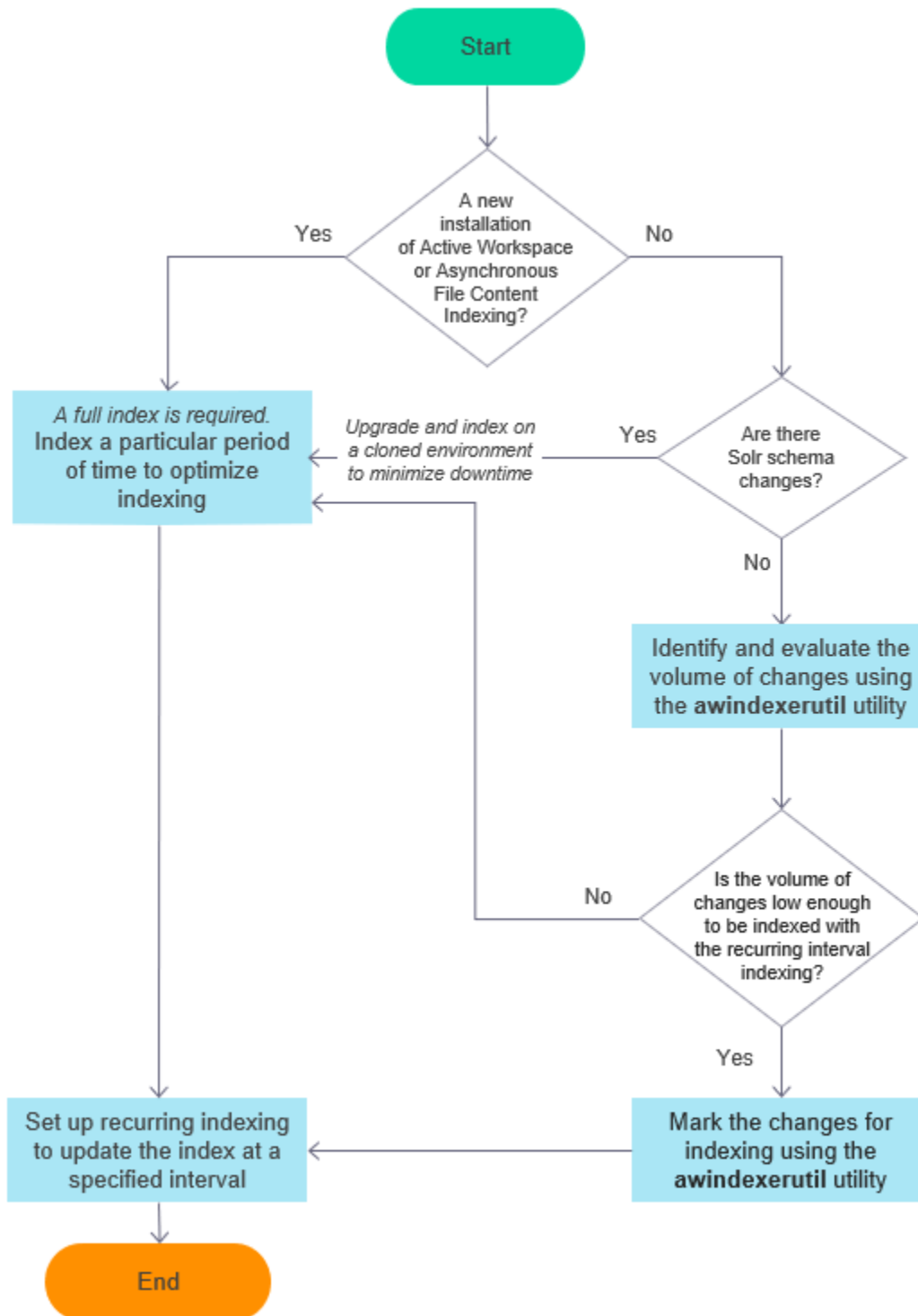
You can install Asynchronous File Content Indexing using Deployment Center.

For more information about installing the indexing components, see the applicable Teamcenter installation guide:

- Installing Teamcenter Using Deployment Center
- Installing Teamcenter Using TEM (Windows)
- Installing Teamcenter Using TEM (Linux)
- **Configure indexing.**
 - For asynchronous file content indexing, additional **configuration** steps are required for indexing CAD files originating outside Teamcenter.

Determining your indexing strategy

The following chart illustrates the process for determining your indexing strategy.



Evaluating indexing methods

Your indexing needs may vary based on hardware, available time, and data considerations.

Perform a full index	<p>Complete a full index when:</p> <ul style="list-style-type: none"> You are completing a new installation. <p>A full index is required when completing a new installation, and for the first indexing after asynchronous file content indexing is installed.</p> <ul style="list-style-type: none"> You have Solr schema changes, for example, if the fieldtype definition is changed in the Solr schema file. Schema changes can occur in a newer version of Active Workspace, causing the need for a full index. You have a high volume or wide variety of changes. <p>If an upgrade, a patch, or data model changes result in a high volume or wide variety of updates, consider completing a full index to encompass all changes.</p> <div style="border: 1px solid black; padding: 10px; margin-top: 10px;"> <p>Note:</p> <p>If a full index is required for a situation other than a new installation, you may choose to upgrade and index in a cloned environment 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. The synchronization flow indexes any intervening changes that were made after you cloned.</p> </div>
Index a time slice for optimal performance	<p>Optimize your process by indexing only a particular period of time, or time slice, at once. 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 the initial index.</p> <p>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.</p>
Set up recurring flow to update the index	<p>Set up TcFTSIndexer scheduling to synchronize indexing at specified intervals. This method updates all indexed information that has been changed since the previous index.</p>
Identify modified data for indexing	<p>Identify and evaluate the volume of changes from a data model update to determine if a full index is required or if the synchronization flow can index the changes.</p>

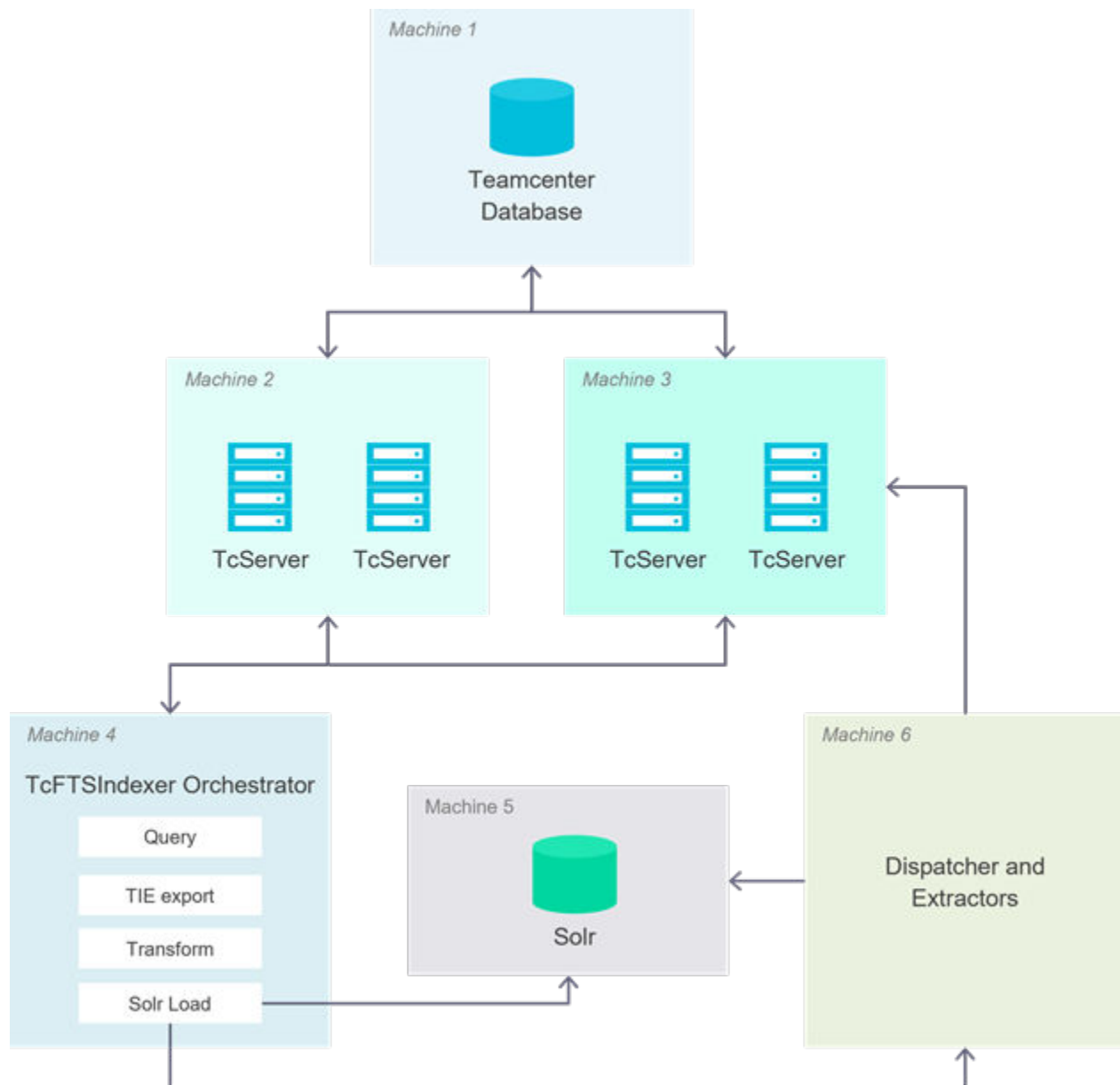
Considering hardware for TcFTSIndexer

To achieve optimal indexing and search performance, you can deploy **TcFTSIndexer** components on separate servers. Siemens Digital Industries Software strongly recommends you install the components

on servers designed for their particular use. TcServers are single threaded, so it is important to determine how many CPU cores you need on server machines. For hardware specifics and recommendations, refer to the *Deployment Reference Architecture* and the *Teamcenter Deployment Guide*.

This component	Does this	And has these indexing considerations
Teamcenter database	Handles large numbers of requests in a short time.	The database is intensive in all areas of hardware. 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.
Teamcenter server manager	Runs the indexer's query, TIE export, and Solr loader operations.	These operations run on multiple tcserver instances and connect to the database. 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 you must coordinate it 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.
TcFTSIndexer Orchestrator	Runs a single Java process that runs Query, TIE Export, Transform, and Solr Loader operations in stand-alone mode.	At high levels of parallelization, Transform is more CPU and memory intensive. The TcFTSIndexer Orchestrator connects to the Teamcenter server manager for object data indexing. TcFTSIndexer components can be set up independently on different machines, or on a single machine for a simple environment.
Solr	Stores indexed Teamcenter data for global search in Active Workspace.	Uses CPU resources intensively during query calls and can also be memory intensive during large dataset file indexing, as well as I/O intensive for reading and writing many documents. The Asynchronous File Content Indexer application requires additional storage during the indexing process. Overall index size may increase to twice the current size.
Dispatcher	Helps offload the indexer when using the Asynchronous File Content Indexer application.	Dispatcher translators can be scaled vertically or horizontally to help process large files outside the main indexer, thereby freeing up indexer resources.

Indexing system layout example



Selecting the synchronous or asynchronous indexing process

What is file content indexing?

Indexing file contents allows your users to search quickly for included information using Active Workspace. Objects without associated datasets are always indexed using a synchronous process. For any associated datasets, you can choose to use a synchronous or asynchronous indexing process during installation.

What is the difference between synchronous and asynchronous indexing?

In the *synchronous* indexing process, **TcFTSIndexer** indexes Teamcenter object metadata and file contents together in one sequential process. All content is searchable by users after indexing is complete.

In the *asynchronous* indexing process, dispatchers¹ manage file content indexing tasks in parallel with object indexing tasks. File content indexing is offloaded to a Dispatcher-based translator², which extracts and indexes the contents in separate processes. Metadata from non-dataset objects is always indexed synchronously, and users can search the metadata while file content indexing completes in the background.

If you are installing asynchronous file content indexing for the first time, you must subsequently perform a **full index of your existing data**.

Which indexing process should I use?

Use a *synchronous* indexing process when:

- **You do not need to index any files.**

The asynchronous file content indexing feature is only used for file content extraction and indexing. If you do not need to index and search for file contents, this feature is not needed.

- **You only want to index a small number of files.**

If your existing file content indexing load in a production environment is manageable, the asynchronous flow is not necessary.

- **You have limited hardware resources.**

Asynchronous file content indexing uses the Teamcenter Dispatcher framework to schedule and process file content indexing requests. Dispatcher installation is required. For best performance, install Dispatcher on a separate server from the indexer.

Choose to use an *asynchronous* indexing process when:

- **You want to index contents from CAD files associated with Teamcenter objects.**

The asynchronous file content indexing framework allows you to extract content in your users' CAD files and index that content as part of Teamcenter objects. This allows your users to search this extracted information. Configuration and installation of external CAD file extractors is required.

¹ See *Introduction to Dispatcher* in the Teamcenter documentation.

² See the section about adding custom translators in *Installing and Configuring Dispatcher* in the Teamcenter documentation.

- **You want to use metadata information to search for objects without associated datasets first.**

The synchronous indexing process ends when all indexing is completed. If you have many objects that require file content indexing, the user must wait until it is completed before searching for object metadata. If asynchronous indexing is installed, file content indexing is separated from metadata indexing. Once metadata indexing is completed, your users can search for all indexable objects without waiting for file content indexing.

However, with asynchronous file content indexing, files in datasets are indexed in parallel with metadata indexing and can be completed after metadata indexing is finished. Extracted file contents are appended to the originally indexed Teamcenter objects.

- **You want to spread out the indexing load incurred by file content indexing.**

When you have a large number of Teamcenter objects with datasets, file content indexing can be a resource-intensive process requiring additional CPU and memory resources. This can result in increased load on the overall indexing process. In contrast, asynchronous file content indexing uses separate and independent Dispatcher processes to track requests. Dispatcher translators extract file contents and can be scaled vertically or horizontally to help offload the main **TcFTSIndexer** environment. **TcFTSIndexer** is a four-tier SOA client that processes and imports Teamcenter data into the Solr indexing engine.

- **You do not want to wait for large files to finish indexing.**

If you need to extract contents from large files, the entire indexing pipeline in a synchronous flow can be blocked while waiting for the results of the extraction. Asynchronous file content indexing can help by processing large files outside the main indexer, in a separate independently scaled server. Users can search for indexed object metadata first before indexing of larger files is completed.

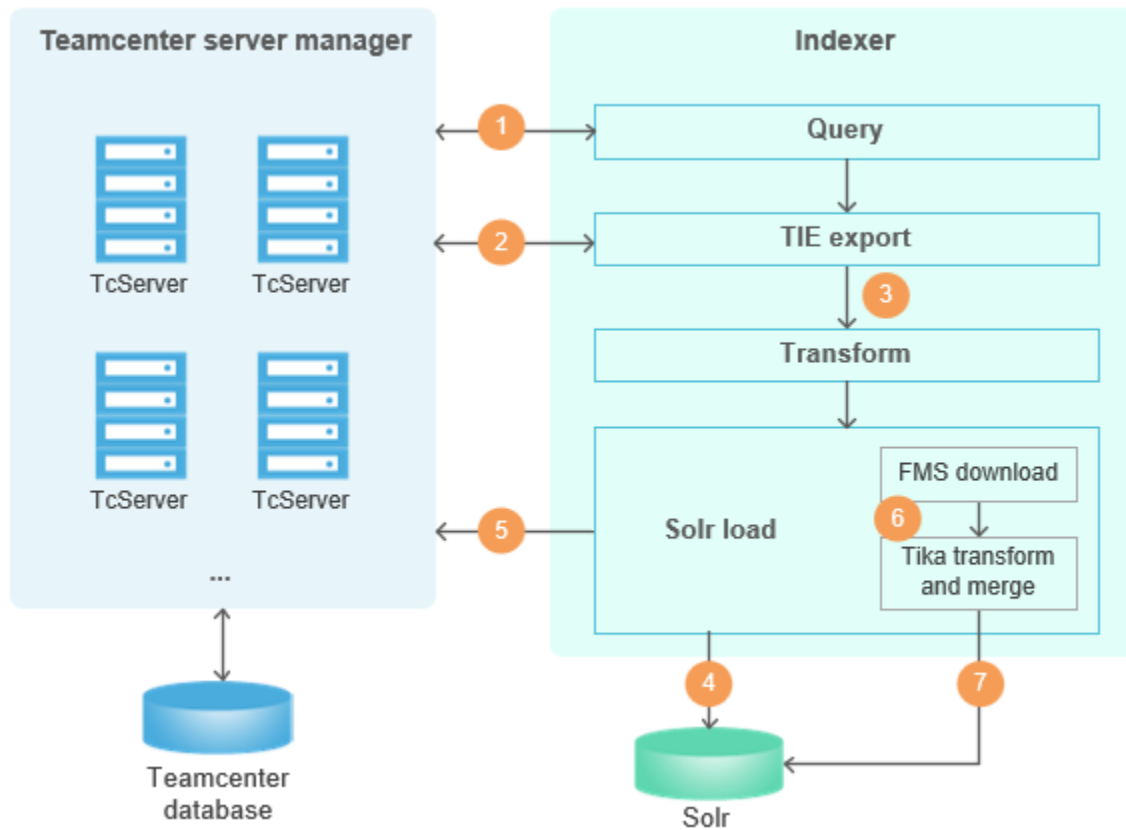
Comparing synchronous and asynchronous indexing processes

Depending on your system's requirements, determine whether you want to use the **synchronous** process or the **asynchronous** process for file contents.

Both indexing processes use **TcFTSIndexer**, a four-tier services-oriented architecture (SOA) client that processes and imports Teamcenter data into the Solr indexing engine. **TcFTSIndexer** comprises the *Query*, *Export*, *Transform*, and *Load* steps that run in sequential order.

Synchronous indexing process for file contents and objects

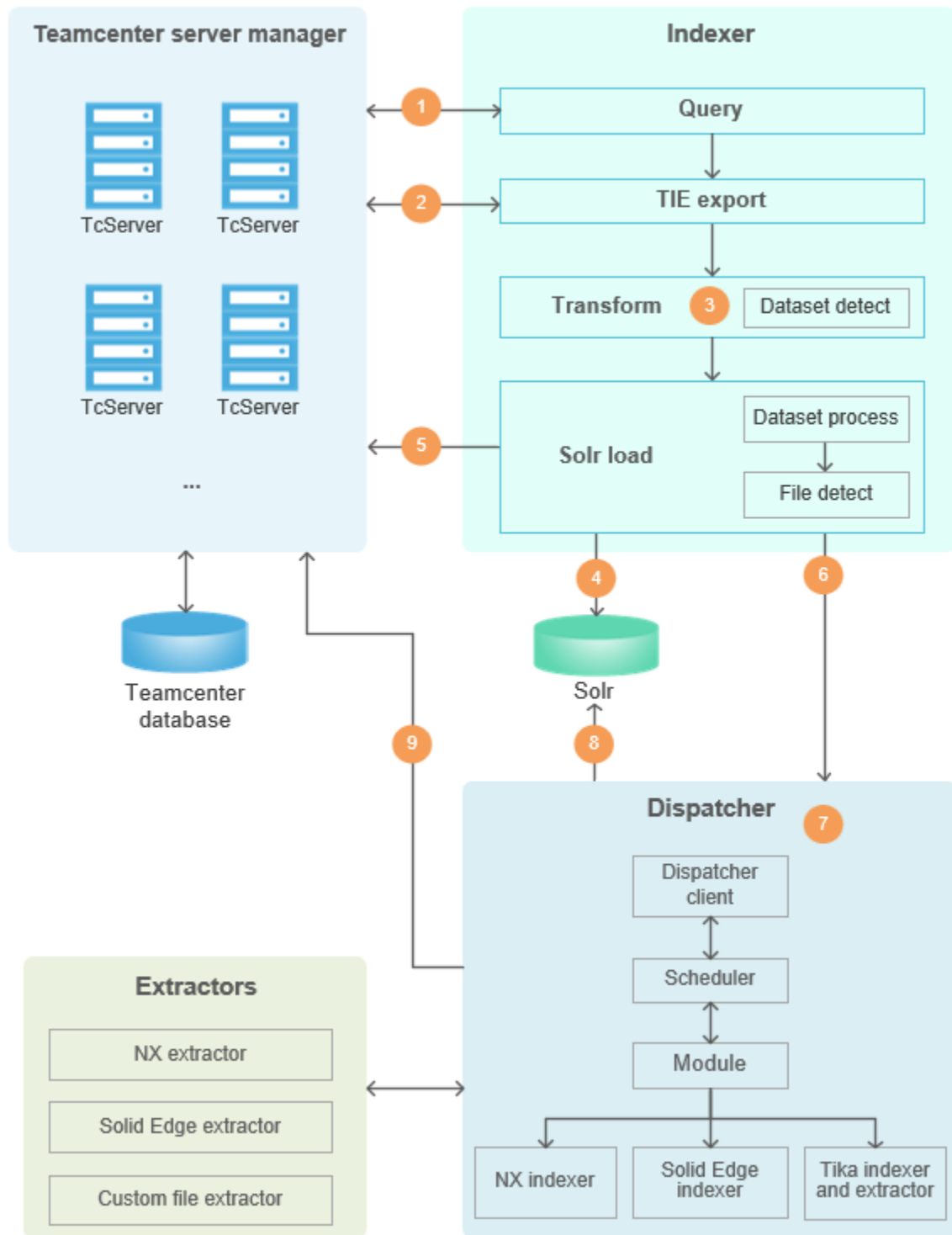
The following process diagram illustrates the steps of the synchronous indexing process for Teamcenter objects and their associated standard files, such as Microsoft® 365 files, PDF files, and text files. In this example, **TcFTSIndexer** is indexing an item revision with a PDF dataset.



Step	Action	Description
1	Query	Requests and receives a list of UIDs for all indexable objects. In this case, TcServer returns the UID for the item revision and dataset.
2	Export	Requests and receives all indexable metadata associated with the item revision and dataset as a TC XML file.
3	Transform	Transforms the exported TC XML file into a Solr XML file.
4	Load	Loads the Solr XML file into Solr. This contains the indexed metadata for both the item revision and the PDF dataset.
5	Load	Informs Teamcenter of the object indexing status.
6	Load	Performs additional steps for the indexable dataset: <ul style="list-style-type: none"> 1. Downloads files attached to the dataset using the FMS (not shown). 2. Extracts indexable contents from the downloaded files.
7	Load	Creates one single Solr XML document that merges all metadata content from the <i>transform</i> step with the extracted PDF contents and loads it into Solr.

Asynchronous indexing process for file contents and synchronous indexing process for objects

The following process diagram illustrates the steps of the asynchronous indexing process for file contents. Teamcenter object metadata is indexed using a synchronous indexing flow. Asynchronously indexing your file contents also allows you to index nonstandard file types, such as NX or Solid Edge CAD files.



Step	Action	Description
1	Query	Requests and receives a list of UIDs for all indexable objects.

Step	Action	Description
		In this case, TcServer returns the UID for an item revision and a dataset.
2	Export	Requests and receives all indexable metadata associated with the item revision and dataset as a TC XML file.
3	Transform	Transforms the exported TC XML file into a Solr XML file. This action also identifies dataset objects.
4	Load	Loads the Solr XML file into Solr. This contains the indexed metadata for both the item revision and the PDF dataset.
5	Load	Notifies Teamcenter of the success or failure of the metadata indexing process.
6	Load	Groups datasets identified in the <i>transform</i> step by dataset type and metadata properties if the metadata indexing process is successful. This action also submits groups of datasets to Dispatcher using DispatcherRequest objects.
7	Dispatcher ³	Extracts and indexes file contents associated with datasets: <ol style="list-style-type: none"> 1. Dispatcher client reads the DispatcherRequest⁴ objects and submits the requests to Scheduler. 2. Scheduler routes the request to file content indexers using the Dispatcher Module. 3. Each file content indexer downloads the files associated with datasets and extracts the contents using their extractor. <p>Different indexers are invoked depending on the type of dataset. For example, the Tika Indexer processes Microsoft Office files, while the NX indexer processes NX part files.</p>
8	Dispatcher	Adds the extracted contents to the metadata already indexed in Solr.
9	Dispatcher	Informs Teamcenter of the dataset indexing status.

Identifying the impact of data model changes

After a data model update, modified data must be indexed. You may be able to complete a delta index to process the changes. However, a full index and a delta index may have similar processing times, due to a high percentage of impacted type and property changes from the data model update. To calculate the impact, **merge the Teamcenter and Solr schemas**, and then evaluate the volume of data model changes by comparing number of changes to the number of objects your indexing infrastructure can support.

³ See *Introduction to Dispatcher* in the Teamcenter documentation.

⁴ See the section about Dispatcher requests in *Installing and Configuring Dispatcher* in the Teamcenter documentation.

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).

After merging the Teamcenter and Solr schemas, use **awindexerutil -delta -dryrun** to identify the changes between the last version and the current version. The utility identifies type and property changes to data model objects, including custom schema changes to objects made by the Business Modeler IDE. Use the resulting report to evaluate the volume of changes and **determine if the changes can be indexed** using the recurring interval indexing or if a **full index** is required.

Expediting indexing using a cloned environment

For an Active Workspace upgrade or patch, you can use the clone and merge approach to expedite the reindexing process and reduce downtime in the production environment. You can make a clone 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 changes that were made in the time between when you cloned and when you merged.

Consider the following items before **indexing on a cloned environment**.

- 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.

Indexing data from other sites using Multi-Site Collaboration

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

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.

Configuring indexing

Configuring indexing tasks

Be sure you determine if you need a **full index or delta reindex of your data** 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.

After installing all indexing components, you must configure indexing prior to indexing any data. Configuring indexing ensures your users can search data, objects, and file content efficiently.

Verify indexing components are installed and running:

- Ensure that the search indexing features are installed:
 - Installing Teamcenter Using Deployment Center

- Installing Teamcenter Using TEM (Windows)
- Installing Teamcenter Using TEM (Linux)
- **Ensure the Solr indexing engine is running.**
- **Test TcFTSIndexer connectivity.**
- **Merge the Teamcenter and Solr schemas.**

Configure indexing details:

- **Optimize instances of TcFTSIndexer**
- **Define index data and filters**
- **Configuring search index properties**
- **Change the user running the TcFTSIndexer**
- **Display snippets from file content**
- **Configure localized display values**
- **Index right-to-left languages**
- **Configure the reporting microservice**
- **Configuring asynchronous file content indexing**

Installing indexing components overview

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

You can install the Indexing Engine (Solr) and the Indexer using Deployment Center or Teamcenter Environment Manager (TEM).

You can install Asynchronous File Content Indexing using Deployment Center.

For more information about installing the indexing components, see the applicable Teamcenter installation guide:

- Installing Teamcenter Using Deployment Center
- Installing Teamcenter Using TEM (Windows)

- Installing Teamcenter Using TEM (Linux)

- **Indexing Engine (Solr)**

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. **TcFTSIndexer** manages the initial indexing for object data. You can then schedule synchronization to run periodically for subsequent updates to object data or structure data indexes.

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.

- **Asynchronous File Content Indexing**

Installs an **optional feature** that allows you to index file contents asynchronously from object metadata. Dispatcher is used to manage file content requests.

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/admin`

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.

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 log on 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 host name 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>

Value	Description
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> <p style="text-align: center;"><code>http://host:port/app-name/tccs</code></p>

- 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.

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\fts\solr_schema_files*) during Indexing Engine installation, you must merge the schemas manually.

Merging and updating the Solr schema is also required after making changes in the Business Modeler IDE that affect indexing. For example, you must merge and update the Solr schema after adding a new business object or a new indexable property.

The steps differ based on whether you are running Solr in **standalone mode** or in a **SolrCloud configuration**.

Merge and update in standalone mode

1. Stop the Solr service if it is running.

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

2. Locate the *TC_SOLR_SCHEMA.xml* and *TC_ACE_SOLR_SCHEMA.xml* files in the *TC_DATA\fts\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.

3. 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*.

4. 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.

If this command results in the File not found error, 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.

This command creates a backup of the Solr schema containing your customizations because these customizations are not included in the updated schema.

5. Review the backup schema for any customizations and make the same changes to the updated schema.
6. Restart Solr either by restarting the **Active Workspace Indexing Service** or by running the start up command:

SOLR_HOME\runSolr.bat

Merge and update in the SolrCloud mode

Prerequisites for SolrCloud schema updates

- An already available or running SolrCloud configuration
- An identified *Leader1_SOLR_HOME* installation

Merge and update procedure

Update a configuration file such as *schema.xml* or *solrconfig.xml* as follows:

1. Download the configset for the collection from ZooKeeper.

Configsets contain all the configuration files used for each collection. In standalone mode, they are stored within the collection in Solr. In SolrCloud, they are stored in ZooKeeper.

- a. Open a command prompt as administrator.
- b. Navigate to the *Leader1_SOLR_HOME\bin* directory.
- c. Run the following command for each collection. Doing so downloads the *conf* directory to the temporary path specified.

```
solr zk downconfig -n collection_name -d Temp_path\collection_name -z
hostname:port
```

where:

- n** Specifies the name of the collection.
- d** Specifies the path to a temporary directory.
- z** Specifies the ZooKeeper machine and port.

Example:

```
solr zk downconfig -n collection1 -d C:\SolrTest\collection1
-z server1:2181
```

2. Merge the files with the Teamcenter schema.

Navigate to the *Leader1_SOLR_HOME* directory and run the following command for each collection to merge the Teamcenter schema with the Solr schema.

```
TcSchemaToSolrSchemaTransform.bat TC_DATA\ftsi\solr_schema_files
Temp_path\collection_name\conf
```

Example:

```
TcSchemaToSolrSchemaTransform.bat
C:\apps\TC\TC_DATA\ftsi\solr_schema_files
C:\SolrTest\collection1\conf
```

This command creates a backup of the Solr schema containing your customizations, as these customizations are not included in the updated schema.

3. Review the backup schema for any customizations and make the same changes to the updated schema.
4. Reupload the configset to ZooKeeper.

Navigate to the *Leader1_SOLR_HOME\bin* directory, and run the following command for each collection. Doing so uploads the *conf* directory from the path specified.

```
solr zk upconfig -n collection_name -d Temp_path\collection_name -z hostname:port
```

Example:

```
solr zk upconfig -n collection1 -d C:\SolrTest\collection1 -z
server1:2181
```

5. Once all uploads are completed, restart all the nodes within the SolrCloud configuration to ensure that the schema changes take effect.

Optimize instances of TcFTSIndexer

Before you start optimization

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

1. Review the **hardware considerations**.
2. 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 time slice and batch size

After the number of machines and the components on each machine are defined, determine the optimal time slice 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.

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.

The following properties are required for searching and filtering. These properties are indexed by default and cannot be removed.

- **POM_application_object.last_mod_date**
- **POM_application_object.creation_date**
- **WorkspaceObject.object_type**

Persistent properties can be indexed for searching and filtering. The following property types cannot be indexed for searching and filtering:

- **Runtime** properties.
- **Compound** properties with a **Runtime** property or a property indicated as transient in the traversal path.
- **Compound** properties referencing a form data file that does not have its own storage class.
- Properties with the **LongString** attribute type.
- Properties with the **TypedReference** attribute type can only be indexed if the following property constants are set:
 - **Awp0SearchRefTypesNames**
 - **Awp0SearchPropFromRefType**

Tip:

You can set up separate filtering for Changes and Inbox.

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 asterisk (*) 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.

Dataset business objects must be defined for indexing using the same **Awp0SearchIsIndexed** business object constant. For example, to define *.jpeg* files for indexing, open the **JPEG** business object in Business Modeler IDE and set the **Awp0SearchIsIndexed** constant to **true**.

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.

5. (Optional) Enter a value for the **AW_Indexable_File_Extensions** preference.

This 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*, *.zip*, *.jpeg*, *.png*, and so on).

If no value is specified for this preference, the file extensions listed under the **Awp0IndexableFileTypes** global constant are used.

In animated *.gif* files, only the first image in the sequence is read for text content extraction.

The *.heic* and *.avif* file formats are not supported for text content extraction.

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 to 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.

- 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>
```

- 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`.

If you are using the asynchronous file content indexing flow, configure properties in both the `TcFTSIndexer_objdata.properties` file and the `TcFTSIndexer_filecontent.properties` file located in `Dispatcher_Root\Module\Translators\TcFTSIndexer\conf`.

Most search properties are defined during installation of the **TcFTSIndexer** component. You can edit the predefined properties or define additional properties listed in the file.

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**.

To index image files for text content extraction, set the **extractInlineImages** parameter value to **true** in the `TcFtsTikaConfig.xml` file located in `TC_ROOT\TcFTSIndexer\conf`.

The maximum image dimensions supported for text content extraction are 32,767 pixels by 32,767 pixels.

Note:

Poor image resolution can cause inaccurate text content extraction. Testing is recommended before indexing a many images for text content extraction at once.

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=
```

Display snippets from file content

You can display the location of search terms within file contents attached to items, including images. 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 find matches for a search term in the results, especially if the result does not match the name or description exactly. Snippets obey the stemming rule 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**.

- From a Teamcenter command prompt, run:

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

- Merge the Teamcenter and Solr Schemas.**

Restart Solr and reindex your data, including the datasets.

- The **AWC_Search_Enable_Snippets** preference controls whether the highlighted phrases are displayed for global search, and it also checks whether the datasets are indexed. 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.
- 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 not enabled, the preferences controlling snippets are ignored.

Configure localized display values

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

- You can search for these reference properties using their display names:
 - Use **Type** for **object_type**.
 - Use **Release Status** for **release_status**.
 - Use **Projects** for **project_list**.

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

- In Business Modeler IDE, change the **Awp0IndexLocalizedValues** global constant to **true**.
- 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.

Configuring 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 on Windows or Linux 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.

Configuring asynchronous file content indexing

Configuring asynchronous file content indexing overview

Prerequisites

Before you can run asynchronous file content indexing for Teamcenter datasets or configure asynchronous file content indexing for CAD files, you must ensure that:

- The **Asynchronous File Content Indexer** application is installed.

Ensure the location of the desired extractors is specified in the **Dispatcher Module Translators Settings** section.

- Object data indexing is properly **configured**.
- Dispatcher Scheduler, Dispatcher Module, and Dispatcher Client applications are running.

Configuring Asynchronous File Content Indexing

Installing the **Asynchronous File Content Indexing** application allows you to index datasets separately from object metadata. Configuring this application allows you to:

- Index and search for NX or Solid Edge CAD file content.
- **Configure indexing and search filters for NX part attributes.**
- Restrict when CAD files are indexed based on a condition.
- Process ITAR controlled datasets in designated locations.
- **Disable faceting on indexed file contents**
- **Test Asynchronous File Content Indexing set up**

Disable Asynchronous File Content Indexing

Update the object data configuration in the main indexer by setting the value of the **enableAsyncFileContentIndexing** property to **false** in the *TC_ROOT/TcFTSIndexer/conf/TcFTSIndexer_objdata.properties* file.

Configure asynchronous file content indexing for NX or Solid Edge CAD files

Configuration is required to index NX and Solid Edge CAD files. If you are only indexing your proprietary file type, this configuration is not necessary.

1. To index NX and Solid Edge files, mark them as an indexable type.

If you have this file type...	Do this...
NX part files	<ul style="list-style-type: none"> • Mark the UGMASTER dataset as an indexable type by enabling the Awp0SearchIsIndexed constant. • Add extension .prt to either the AW_Indexable_File_Extensions preference or the Awp0IndexableFileTypes global constant.
Solid Edge draft files	<ul style="list-style-type: none"> • Mark the SE Draft dataset as an indexable type by enabling the Awp0SearchIsIndexed constant. • Add extension .dft to either the AW_Indexable_File_Extensions preference or the Awp0IndexableFileTypes global constant.

2. Index your NX **UGMASTER** datasets and Solid Edge **SE Draft** datasets using initial indexing, sync, or any available object data indexing methods:
 - If you are indexing for the first time after installing asynchronous file content indexing, you must perform a full index.
 - If you have already installed asynchronous file content indexing and you are enabling indexing for NX and Solid Edge files, you can perform a full index or a delta reindex.

File content indexing is now submitted to the dispatcher.

3. Your CAD file contents from NX and Solid Edge are now available for search.

Configure indexing and search filters for NX part attributes

For NX CAD files, you can index part attributes and product manufacturing information (PMI) so that your users can search for information in drawing notes, tables, and labels in Active Workspace. You can also configure NX part attributes to display as search filters in Active Workspace.

Prerequisites

A minimum version of NX 2212 is required.

Procedure

1. Open the `\Dispatcher_root\Module\Translators\TcFTSIndexer\conf\filecontent\extractors\NXExtractorConfig.json` file and copy the following configuration for part attributes and PMI from the `\UGII\extractors\indexer\schema\SampleConfiguration.json` file.

```
"attribute" : {
  "object" : ["part", "solid body"]
},
"pmi" : {
  "object" : ["notes", "tables", "labels"]
}
```

2. Export the NX mapping configuration file using the **aw_search_config_manager** utility to ensure that you have the most recent version.

```
aw_search_config_manager -export -config_id=nxconfig -output=c:\nxconfig.json
```

3. Open the `\Dispatcher_root\Module\Translators\TcFTSIndexer\conf\filecontent\extractors\nxconfig.json` file and add configurations for part attributes that you want to display as filters.

You can review the definitions of the mapping configurations in the `mappingConfig.schema.json` file.

For example, if you want the NX volume values to display as filters, add the following configuration:

```
"path": "attribute.part.*"
"id": "part attributes",
"ruleSet": [
  {
    "fieldName": "title",
    "fieldValue": "value",
    "fieldUnit": "unit",
    "type": "plm.doubleList",
    "measure": "Volume",
    "conditions": [
      {
        "fieldName": "NX Volume"
      }
    ]
  }
]
```

4. Import the changes for the NX mapping configuration file to the database using the **aw_search_config_manager** utility.

```
aw_search_config_manager -import -dir=mapping_configs
```

5. Verify the file content indexing is running using the **runTcFTSIndexer** utility.

```
runTcFTSIndexer -task=filecontent:test -standalone
```

6. Verify the `aw_search_consolidated_mapping.json` file is updated using the **runTcFTSIndexer** utility.

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

Restrict when CAD files are indexed based on a condition

Indexing CAD files for every drawing modification can be resource intensive. You have the option of reducing the number of times a drawing is indexed by creating a Teamcenter property-driven condition.

Prerequisites

Configure the indexer to index NX or Solid Edge CAD files or follow the customization procedure for other CAD files.

Create a property-driven Teamcenter condition

This condition is added to the grouping configuration file. This file specifies which datasets the main indexer groups and submits to the dispatcher for indexing.

1. Open the grouping configuration file from the main indexer installation, `TC_ROOT/TcFTSIndexer/conf/DatasetGroupingConfig.json`.

For **UGMASTER**, notice the following provided default grouping configuration:

```
UGMASTER": {
  "UGPART": [
    {
      "name": "tcftsindexerfilecontentnxindex",
      "limit": 10
    }
  ]
}
```

In this configuration, each time the indexer encounters a named reference **UGPART** associated with dataset **UGMASTER**, it will send it to the dispatcher service **tcftsindexerfilecontentindex** for indexing.

2. Add conditions using JSON objects with syntax: *Teamcenter-internal-property-name:internal-value*.

If a condition is present, the indexer will only index the file if the condition is met. This reduces the number of times a file is indexed.

Tip:

- Property names and values in grouping configurations must match internal Teamcenter names and values exactly, including casing.
- Property values must be strings. In a condition, exact match comparisons are performed between the desired property values and the current property values.
- Conditions using date and time properties are not supported.
- When using reference properties in a condition, use the first source property pointed to by the constant **Awp0SearchPropFromRefType**.

Add a single condition

For example, if you want to index the contents of **UGMASTER** datasets only when the **release status** property is **Released**, add the below condition.

```
"UGMASTER": {
  "UGPART": [
    {
      "name": "tcftsindexerfilecontentnxindex",
      "limit": 10,
      "conditions": [
        {
          "release_status_list": [ "Released" ]
        }
      ]
    }
  ]
}
```

Add a single condition with multiple values in an AND relationship

For example, if you want to index datasets assigned to projects **ProjectA** and **ProjectB**, add the below condition.

```
"conditions": [
  {
    "project_list": [ "ProjectA", "ProjectB" ],
    "operation": "AND"
  }
]
```

If the operation field is left empty, **OR** is the default operator.

Add a single condition with multiple values in an OR relationship

For example, if you want to index datasets belonging to either **ProjectA** or **ProjectB**, add the below condition.

```
"conditions": [
  {
    "project_list": [ "ProjectA", "ProjectB" ]
  }
]
```

```
    }
  ]
```

Since the operation field is not specified, **OR** is the default operator.

Add multiple conditions with multiple values

For example, if you want to index datasets only when the **release_status_list** contains **Released** and **TCM Released** and when **owning_group** contains **design** or **engineering**, add the below condition.

```
{
  "conditions": [
    {
      "release_status_list": [ "Released", "TCM Released" ],
      "operation": "AND"
    },
    {
      "owning_group": [ "design", "engineering" ]
    }
  ]
}
```

Multiple conditions have an **AND** relationship between them.

Process ITAR controlled datasets in designated locations

In order to maintain compliance, access to ITAR data must be controlled. Processing in a specific location may be required. Using the following procedure, you can configure asynchronous file content indexing to process specific datasets in servers based in a specific region.

Prerequisites

Install and configure an ITAR compliant dispatcher⁵. For example, configure a US-based dispatcher that accesses a US-based volume.

Configure dispatcher to send specific datasets to a specific server

In this example, we create a US-based dispatcher service that is assumed to be running on a US-based server. Then, we specify which files are sent to this service to be processed. This example uses the **tcftsindexerfilecontenttikaindex** service that supports standard files, such as Microsoft Office files, PDFs, and text files. Choose the appropriate service for your system configuration:

If you are indexing this type of file...	Search for this service...
Standard files such as Microsoft Office files, PDFs, and text files	tcftsindexerfilecontenttikaindex
NX	tcftsindexerfilecontentnxindex

⁵ See *Introduction to Dispatcher* in the Teamcenter documentation.

If you are indexing this type of file...	Search for this service...
Solid Edge	tcftsindexerfilecontentsolidedgeindex
Other proprietary files	Your custom service

1. Create a new dispatcher translator service using the OOTB **tcftsindexerfilecontentttikaindex** service in the *Dispatcher_Root\Module\conf\translator.xml* configuration file as a template.

Copy the existing **tcftsindexerfilecontentttikaindex** service definition and rename it to **tcftsindexerfilecontentttikaindex_us**.

Rename the XML element name to conform to the standard camel case syntax as shown in the original service definition. **TcFtsIndexerFileContentTikaIndex_US** follows the standard camel case syntax.

```
<TcFtsIndexerFileContentTikaIndex_US provider="SIEMENS"
  service="tcftsindexerfilecontentttikaindex_us"
  maxlimit="3" isactive="true">
  <TransExecutable dir="%MODULEBASE%\Translators\TcFtsIndexer\bin"
name="runTcFtsIndexer.bat" />
  <Options>
    <Option name="flow" string="-task=filecontent:tika"
description="TcFtsIndexer flow to extract and index file contents using Apache
Tika." />
    <Option name="mode" string="-standalone" description="Switch to indicate
that TcFtsIndexer needs to run in standalone mode." />
    <Option name="inputpath" string="-i=" description="Full path to the input
file or directory." />
  </Options>
  <TransErrorExclStrings>
    <TransInputStreamExcl string="Error: 0" />
  </TransErrorExclStrings>
</TcFtsIndexerFileContentTikaIndex_US>
```

The remainder of the service definition remains the same as the default **tcftsindexerfilecontentttikaindex** service. This service calls the **runTcFtsIndexer** utility with the flow action input set to **-task=filecontent:tika**.

Ensure **isactive** is set to **true**.

2. In the main Indexer, create a grouping configuration that routes specified datasets to the new service.

Modify *TC_ROOT\TcFtsIndexer\conf\DatasetGroupingConfig.json* and add the following rule:

```
{
  "MSWordX": {
    "word": [
      {
        "name": " tcftsindexerfilecontentttikaindex_us",
        "limit": 1000,
        "conditions": [
```

```

        {
            "owning_group": [ "engineering" ]
        }
    ]
},
"default": {
    "name": "tcftsindexerfilecontenttikaindex",
    "limit": 1000
}
}

```

This configuration will route all MSWordX datasets owned by the group **engineering** to the service **tcftsindexerfilecontenttikaindex_us**. The service is assumed to be running on a US-based server. All other indexable datasets will continue to be processed by the default **tcftsindexerfilecontenttikaindex** service.

Disable faceting for indexed file contents

You can disable faceting to ensure indexed file contents do not add facets to your search filters.

Procedure

1. Update the mapping configuration JSON file by setting the **filter** property to **false**. Refer to the mapping specification document *TC_ROOT/TcFTSIndexer/conf/specification/Mapping_Specification.docx*.
2. Run the **aw_search_config_manager** utility to update the configuration store with the new mapping rules.

Example:

```
aw_search_config_manager -u=admin -p=pwd -g=dba -import -
dir=c:\mappingConfigs
```

3. Start a new Teamcenter session to display any faceting changes.

Verify asynchronous file content indexing connectivity

You need to ensure the Asynchronous File Content Indexer application is installed and running.

1. Verify that the Asynchronous File Content Indexer application is installed:
 - Ensure the **AWS_asynchronous_file_content_indexing_enabled** preference is set to **true**.
 - (Optional) If indexing is enabled for NX, Solid Edge, or other proprietary files through a custom configuration, run the **aw_search_config_manager** utility to confirm your mapping configurations are installed:

```
aw_search_config_manager -u=admin -p=pwd -g=dba -list
```

2. The asynchronous file content indexer application installs the **TcFTSIndexer** client as part of the Dispatcher Module. Verify that all necessary prerequisites have been satisfied by running the indexer's **filecontent:test** flow action from the *Dispatcher_Root/Module/Translators/TcFTSIndexer/bin* directory.

```
runTcFTSIndexer.bat -task=filecontent:test
```

This flow action displays a status report, which includes:

- Confirmation of connectivity to Teamcenter, FMS, and Solr.
- Confirmation that all required configuration files are present.

Indexing your data

Perform a full index of object data

If you are running an initial index, be sure you **optimize your instances of TcFTSIndexer**. You can choose to index a particular time slice or a batch size for better performance.

Note:

It is recommended 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. 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
```

- (Optional) To clear the existing search data and perform a new index, open a Teamcenter command prompt, navigate to `TC_ROOT\TcFTSIndexer\bin`, and enter the following commands:

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

- On the machine on which the **Indexer** feature is installed, open a command prompt.
- Navigate to the **bin** directory of the TcFTSIndexer, for example, `TC_ROOT\TcFTSIndexer\bin`.
- 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.

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.

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`.

- Determine whether index failures occurred. Check the report generated on the console and stored in `TC_ROOT\TcFTSIndexer\logs\TcFTSIndexer_objdata.log`.
- 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.

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

- Active Workspace Gateway

10. 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.

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

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

Minimize downtime by updating and indexing on a cloned environment

For an Active Workspace upgrade or patch that requires a full index, 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.

Prerequisites

Before upgrading the cloned or production environment, test the upgrade or patch in a testing environment.

Procedure

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. Upgrade 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. Upgrade the production environment with 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
```

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

13. Resume the synchronization flow in the production environment.

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

Evaluate data model changes for indexing

After a data model update, **modified data** must be indexed. You may be able to complete a delta index to encompass the changes. However, a full index and a delta index may have similar processing times, due to a high percentage of impacted type and property changes from the data model update. To calculate the impact, identify the changes, evaluate the volume of changes by comparing the number of changes to the number of objects your indexing infrastructure can support, and determine if a delta index is adequate or if a full index is required.

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

1. **Merge the Teamcenter and Solr schemas** if Teamcenter and Active Workspace are not patched.
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 indexing.
 - If there is a high volume or wide variety of changes, complete a **full index**.
 - If you want to use the delta of changes for indexing, run the **awindexerutil** utility to set the changes from the report for indexing.

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

The identified updates are indexed with the synchronization flow. It may take several instances of the synchronization flow to index all changes.

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

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

6. Restart the synchronization flow using the **runTcFTSIndexer** utility:

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

Set up TcFTSIndexer synchronization flow

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

Tip:

If you are migrating a large volume of data, you can index the migrated data separately from the synchronization flow to improve performance.

Disable **fast_sync_add_trigger** in the database to prevent the migrated data from being indexed in the synchronization flow and stop the synchronization flow during the migration. When the migration is complete, enable **fast_sync_add_trigger** and resume the synchronization flow. Index the migrated data on a separate instance of **TcFTSIndexer** using **a time slice** for the duration of the migration.

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 actions:

- 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.

- 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.

- Query for deleted objects, connect to Solr, and delete the objects.
- Query for new objects, and run the list of objects through TIE export and transform. Then load into Solr.
- Query for objects affected by an Access Manager rule change, run the list of objects through TIE export and transform, and load to Solr.
- Query for modified objects, run the list of objects through transform, and load to Solr.
- 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.

- 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.

- 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.

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. **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.

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.

Referencing 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.

Use the **aw_search_config_manager** utility to manage mapping configurations used for indexing extracted data from non-standard datasets such as CAD files.

runTcFTSIndexer

Indexes data into the Solr indexing engine. Run this command from the *FTS_INDEXER_HOME* directory, for example, *TC_ROOT\TcFTSIndexer\bin*. This command is not required to be run in a specific environment.

SYNTAX

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

Note:

Operating system considerations:

Linux	<p>A <i>.sh</i> extension is required, for example, <i>./runTcFTSIndexer.sh</i>.</p> <p>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 _.</p>
Windows	<p>The <i>.bat</i> extension is not required for runTcFTSIndexer.</p> <p>For UIDs on Windows, do not enclose in straight, single quotation marks.</p>

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:build_suggester**

Rebuilds the suggestions dictionary for Solr. This flow is separate from the sync flow.

Run **objdata:build_suggester** if the **objdatasyncstep.build_suggester_inline** property in the **TcFtsIndexer_objdata.properties** file is set to **false**.

- This action can use the **-interval=seconds** argument, for example, to rebuild the suggestions dictionary every five minutes (300 seconds):

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

The interval value must be 300 or greater. Running this flow action with an interval is recommended.

- You can rebuild the suggestions dictionary once, for example:

```
runTcFTSIndexer -task=objdata:build_suggester
```

- **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 UUIDs of the objects that match the specified state. Each UUID 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 UUIDs in the output format *UUID | 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 UUIDs 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 goZRkWxoqdl2DyB
```

```
./runTcFTSIndexer.sh -task=structure:syncone 'Abed$b8dBhGXHA'
```


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.

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 .

-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
```

aw_search_config_manager

Manages mapping configurations used when **customizing asynchronous file content indexing**. Mapping configurations specify how data extracted from non-standard files, such as CAD files, are indexed into Solr.

Run the utility from the `TC_ROOT\bin` directory. (You can also use the `TC_BIN` directory if it is configured.)

SYNTAX

```
aw_search_config_manager -u=user-ID {-p=password | -pf=password-file} -g=group
  [-import -dir= config-directory]
  [-export -config_id= config-id -output= output-file-path]
  [-remove -config_id= config-id]
  [-list]
  [-h]
```

ARGUMENTS

-u

Specifies the user ID. The user must have 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 for a user ID and password.

-p

Specifies the user's password. This argument is mutually exclusive with the **-pf** argument.

-pf

Specifies the password file. This argument is mutually exclusive with the **-p** argument.

-g

Specifies the group associated with the user.

-import

Imports the configuration files from the given directory.

-dir

Used with **-import** to specify the directory from which configuration files are imported.

-export

Imports the configuration files from the given directory.

-output

Used with **-export** to specify the full path, including file name, where you want the JSON configuration saved.

-remove

Removes a mapping configuration.

-config_id

- When used with **-export**, specifies the ID of the configuration to export.
- When used with **-remove**, specifies the ID of the configuration to remove.

-list

Displays a list of the stored configuration IDs.

-h

Displays help for this utility.

EXAMPLES

For example, a mapping configuration file, *C:\mappingConfigs\extConfig.json*, contains a mapping configuration with the ID *extConfig*. This ID (*extConfig*) contains rules that allow the file content transform process to transform data extracted from external sources to a format that can be indexed and searched.

Run the following command to upload all configurations present in the *C:\mappingConfigs* directory:

```
aw_search_config_manager -u=admin -p=pwd -g=dba -import -dir=c:\mappingConfigs
```

Configurations present within *extConfig.json* are now available for use in the transform process.

- List all available configurations:

```
aw_search_config_manager -u=admin -p=pwd -g=dba -list
```

- Export a configuration to a JSON file:

```
aw_search_config_manager -u=admin -p=pwd -g=dba -export -config_id=extConfig  
-output=c:\exportedConfigs\extConfig.json
```

- Remove an existing configuration:

```
aw_search_config_manager -u=admin -p=pwd -g=dba -remove -config_id=extConfig
```

Reviewing indexing results

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 default settings after you complete your debugging. After you finish troubleshooting, be sure to also restart the system.

TcFTSIndexer logs

These logs are located in the `TC_ROOT\TcFTSIndexer\logs\` directory. The logs contain messages for object data indexing and structure data indexing. These logs also provide basic information about the indexer and the indexer framework, which includes logs from all supported **TcFTSIndexer** types.

- *TcFTSIndexer.log* contains all the messages related to framework and all the TcFTSIndexer flows.
- *TcFTSIndexer_objdata.log* contains object data messages related to object indexing flows. This also includes messages related to files indexed when using the synchronous file content indexing flow.
- *TcFTSIndexer_filecontent.log* contains messages related to files indexed when using the asynchronous file content indexing flow.
- *TcFTSIndexer_structure.log* contains messages related to structure data indexing 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 a 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 tcGSMessageId="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**.

- Restart the Pool Manager.
- 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.

Reviewing file content indexing status

You can query the indexing status of all dataset objects. In addition, you can query for the status of parent business objects when file contents are indexed against them instead of datasets. To display status, query the **ACCT_TABLE** in the Teamcenter database for the following attributes:

To list objects in this state...	Run this SQL Query...
Pending datasets	<code>SELECT island_anchor_uid FROM ACCT_TABLE WHERE app_id=':FTS_FILECONTENT' AND state=1</code>
Successfully indexed	<code>SELECT island_anchor_uid FROM ACCT_TABLE WHERE app_id=':FTS_FILECONTENT' AND state=2</code>
Failed datasets	<code>SELECT island_anchor_uid FROM ACCT_TABLE WHERE app_id=':FTS_FILECONTENT' AND state=4</code>

Investigate and resolve failures

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

- Copy files in *Staging Directory\TcFTSIndexer_objdata* and *TC_ROOT\TcFTSIndexer\logs* to a separate directory.
- 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 for object data, run **runTcFTSIndexer -task=objdata:recover**.
- If you are using the **recover** flow for pending datasets, run **runTcFTSIndexer -task=objdata:filecontentrecover**.
- 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.
5. 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 UUIDs 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 UUIDs in the output format *UUID|object_string|object_type*. For example:

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

Resolving file content indexing errors

Resolving file content indexing errors

Occasionally during the indexing process, you may encounter error messages when indexing a file. Use the following suggested recovery procedures on the affected file to allow reindexing of the file. The amount of data that can be recovered is dependent on the file and the particular error.

Error messages can be seen in the log file or in the command window where indexing is executed. Values in messages might vary depending on your system.

Errors affecting text files

For the following error, perform the **text recovery procedure**.

```
Caused by: java.lang.IllegalStateException: Potential loop detected -
Block 0 was already claimed but was just requested again
```

Errors affecting PDF files

For the following errors, there is no recovery procedure available due to the file being corrupted or password protected.

- `org.apache.pdfbox.pdmodel.encryption.InvalidPasswordException: Cannot decrypt PDF, the password is incorrect`
- `java.io.IOException: Missing root object specification in trailer`
- `java.io.IOException: Page tree root must be a dictionary`
- `java.lang.IllegalArgumentException: The end (65642) must not be before the start (65648)`

Resave recovery procedure

Perform the **PDF file resave procedure** when you encounter the following errors:

- `java.lang.IllegalStateException: Expected 'Page' but found COSName{Font}`
- `Unknown dir object c='>' cInt=62 peek='>' peekInt=62 at offset 195699`
- `java.lang.IllegalStateException: Expected 'Page' but found COSName{Font}`
- `Caused by: java.io.IOException: Invalid font type: COSName{Font}`

After performing this procedure, you may receive the error **WARN No Unicode mapping for error-text in font font-name**. This indicates that the *error-text* is not indexable.

Errors affecting Microsoft PowerPoint, Excel, and Word files

For the following error, there is no recovery procedure available because the file is blocked by Microsoft Trust Center.

```
org.apache.poi.poifs.filesystem.NotOLE2FileException: The supplied data
appears to be an old Word version 2 file. Apache POI doesn't currently
support this format
```

Resave recovery procedure

Perform the **Microsoft Office file resave procedure** when you encounter the following errors:

- Table Stream '0Table' wasn't found - Either the document is corrupt, or is Word95 (or earlier)
- Initialization of record 0x809(BOFRecord) left 4082 bytes remaining still to be read.
- This paragraph is not the first one in the table
- java.lang.ArrayIndexOutOfBoundsException
- java.lang.IllegalArgumentException: The end (65642) must not be before the start (65648)

Resave OOXML file procedure

Perform the **OOXML file resave procedure** when you encounter the following error:

```
Strict OOXML isn't currently supported, please see bug #57699
```

Resolve OOXML relationship

Perform the **OOXML file resolve relationship procedure** when you encounter the following error:

```
No part found for relationship id=rId3
- container=org.apache.poi.openxml4j.opc.ZipPackage@6a539bca
- relationshipType=http://schemas.openxmlformats.org/officeDocument/
2006/relationships/oleObject - source=/ppt/slides/slide64.xml -
target=/ppt/slides/NULL,targetMode=INTERNAL
```

Recover text

Perform the **recover text procedure** when you encounter the following errors:

- Caused by: java.lang.IllegalStateException: Potential loop detected - Block 0 was already claimed but was just requested again
- Caused by: java.io.IOException: The text piece table is corrupted, expected byte value 2 but had 0

Recover text files

Perform the following procedure to recover text files and resolve errors.

Procedure

1. Open the Microsoft application for the affected file.
2. Click **Open→Browse**.
3. Navigate to the location of the affected file.
4. Select the file name.

In the **All Files** drop-down, select **Recover Text from Any File**.

5. Click **Open**.

The **Show Repairs** window opens.

6. Close the **Show Repairs** window.

This is not needed for this procedure.

7. Save the file.
8. Add renamed file to Teamcenter and use the **runTcFTSIndexer** utility to index the renamed file.

Resave PDF files

Perform the following procedure to resave PDF files and resolve errors.

Procedure

1. Open affected file.
2. If displayed, click **Fix Now**, then proceed to step 5.

Otherwise, continue with step 3.

3. Click **File→Save As**.
4. Rename the file.
5. Save the file.
6. Add renamed file to Teamcenter and use the **runTcFTSIndexer** utility to index the renamed file.

Resave Microsoft Office files

Perform the following procedure to resave Microsoft Office files and resolve errors.

Procedure

1. Open affected file using its respective Microsoft application.
2. Click **File**→**Save As**→**Browse**.
3. Rename the file.
4. In the **Save as type** drop-down list, select the latest Microsoft Office file format.

If the file extension displays, select an extension, such as *.docx*, *.pptx*, or *.xlsx*.

5. Click **Save**.

A new window displays notifying you that the file is upgraded to the newest file format.

6. Click **Okay**.

The layout of the file may be altered.

7. Add the renamed file to Teamcenter and use the **runTcFTSIndexer** utility to index this file.

Resave OOXML files

Perform the following procedure to resave OOXML files and resolve errors.

Procedure

1. Open affected file using its respective Microsoft application.
2. Click **File**→**Save As**→**Browse**.
3. Rename the file.
4. In the **Save as type** drop-down list, select the latest Microsoft Office file format.

If the file extension displays, select an extension, such as *.docx*, *.pptx*, or *.xlsx*.

5. Click **Save**.

6. Add renamed file to Teamcenter and use the **runTcFTSIndexer** utility to index the renamed file.

Resolve OOXML file relationships

Perform the following procedure to resolve OOXML file relationships and resolve errors.

Procedure

1. Open affected file using its respective Microsoft application.
2. Navigate to the issue relationship highlighted in the error message.
3. Copy the issue relationship item.
4. Paste the item as a picture.

Right-click and select **Paste→Picture**.

5. Repeat steps 2 through 4 for all relationship issue items.
6. Save the file.
7. Add renamed file to Teamcenter and use the **runTcFTSIndexer** utility to index the renamed file.

Reviewing indexed file content that search does not display

If search does not return datasets with the indexed file content, verify the following:

- The dataset type is marked as indexable.
- The extension of the file or named reference is included in the list of indexable file extensions.
- **Asynchronous file content indexing is enabled.**
- The indexer is running.
- All dispatcher⁶ components are running:
 - Dispatcher scheduler
 - Dispatcher client
 - Dispatcher module

⁶ See *Introduction to Dispatcher in the Teamcenter* documentation.

Check if the appropriate service is installed and set to active in the configuration file *Dispatcher_Root/Module/conf/translator.xml*.

If you are indexing this type of file...	Search for this service...
Standard files such as Microsoft Office files, PDFs, and text files	tcftsindexerfilecontenttikaindex
NX	tcftsindexerfilecontentnxindex
Solid Edge	tcftsindexerfilecontentsolidedgeindex
Other proprietary files	Your custom service

- (Optional) A mapping configuration for your dataset is installed by running the following:

```
aw_search_config_manager -u=admin -p=pwd -g=dba -list
```

A mapping configuration file is not required when indexing only standard files such as Microsoft Office files, PDF files, and text files.

- (Optional) The configuration file *Dispatcher_Root/Module/Translators/TcFTSIndexer/conf/ExtractorConfig.json* contains the correct path for your extractor.

An extractor is not required when indexing only standard files such as Microsoft Office files, PDF files, and text files.

- Dataset objects were indexed successfully by checking the indexer logs for errors.

Reviewing file content facets that are not displayed

If the facets are not displaying correctly, verify the following:

- The mapping configuration file is correct. Refer to the mapping specification document *TC_ROOT/TcFTSIndexer/conf/specification/Mapping_Specification.docx*.

- Faceted properties are set to **true** in the mapping entry.

This enables the filtering on faceted properties.

- Display names are set for your locale.

If the display name is not set for your locale, facet names may not appear as expected.

Correct any issues found and run the **aw_search_config_manager** utility to load the mapping configuration file into the configuration store.

- **AWS_asynchronous_file_content_indexing_enabled** preference is set to **true**.

Enhancing indexing performance

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

ACCT_TABLE-locking issue or slow synchronization on Microsoft SQL Server database

Use these suggestions to avoid **ACCT_TABLE**-locking issues during indexing, and query performance issues during synchronization:

- Reduce the number of Teamcenter connections using the **tc.maxConnections** property as suggested in the *Optimize the Indexer* section.
- Reduce the query time span for each object data query by reducing the value of the **querytimeslicestep.maxQueryTimeSpan** in the indexing step.

The default value is 20000 minutes or about 2 weeks. You can lower this value to reduce the load on individual timeslice queries.

- The following suggestions improve the performance of the synchronization query that is run to identify modified objects:
 - Set the **Index_Sync_Mssql_Hint** preference to **true**.
 - Set the Microsoft SQL Server's compatibility mode to **2012**.

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.

Customizing indexing

Customizing the indexer overview

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.

Customizing the Indexer 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.

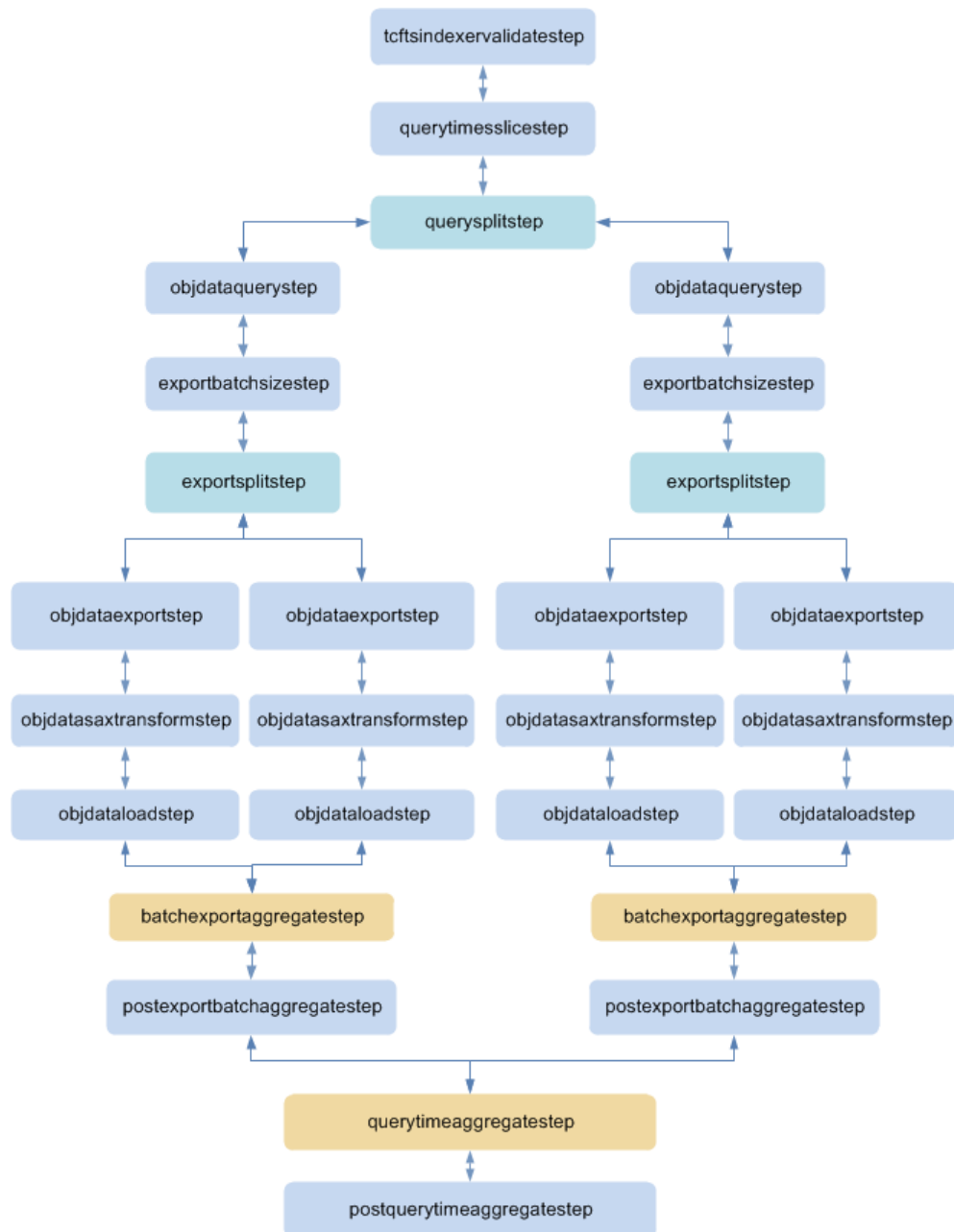
- **Split step**

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.

Indexing object data 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.

- **tcftsindexervalidatestep**

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 **querytimesslicestep** 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 **querysplitstep** 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:

```
-----
-----
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 UIDs of specific batch size. This step breaks the list of lists into multiple threads processing each batch size list of UIDs.

- 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 UIDs. 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.

Resolving TcFTSIndexer errors

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.

Update Indexer settings

You must update the Indexer settings after patching Teamcenter and Active Workspace. You can use Teamcenter Environment Manager (TEM) in maintenance mode or Deployment Center to make the changes.

Update the Indexer settings in TEM

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**.

Update the Indexer settings in Deployment Center

1. In **Components**, select the **Indexer**.
2. Configure all applicable Indexer settings.
3. Click **Save Component Settings**.
4. Generate deployment scripts and deploy software on affected machines.

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
```

3. 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)

1. 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.

2. 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 Digital Industries Software.

- **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_Awp0AWCEExternalSystemObject_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_Awp0AWCEExternalSystemObject_0Y0_awp0External
      SysObjClassName">The object class name defined in the BMIDE</field>
    <field name="TC_0Y0_Awp0AWCEExternalSystemObject_0Y0_awp0External
      SystemName">The name of the external system</field>
    <field name="TC_0Y0_Awp0AWCEExternalSystemObject_0Y0_awp0External
      SystemURI"><field name="TC_0Y0_Awp0AWCEExternalSystemObject_0Y0_
      awp0PropertyNameList">TC_0Y0_A4_address_0Y0_
      a4configuration</field>
    ...
    <field name="TC_0Y0_Awp0AWCEExternalSystemObject_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
```


Customizing asynchronous file content indexing

Customizing asynchronous file content indexing overview

Installing the Asynchronous File Content Indexing application allows you to index datasets separately from object metadata. Configuring this application allows you to index and search information found in NX and Solid Edge CAD files. Further customizing this application allows you to Index contents of other proprietary datasets using a custom extractor.

Indexing file contents of other proprietary files

By default, the Asynchronous File Content Indexing application provides support for NX and Solid Edge CAD files. This application can index proprietary file formats when using a custom extractor.

Prerequisites

- Install the **Asynchronous File Content Indexer** application.
- Mark the dataset type **SampleDataset** as indexable by enabling the **Awp0SearchIsIndexed** constant.
- Add the extension *.sample* to either the **AW_Indexable_File_Extensions** preference or the **Awp0IndexableFileTypes** global constant.
- Create or acquire a custom extractor that extracts contents from your proprietary file format.

The asynchronous file content indexing framework is compatible with any extractor as long as the extracted content is in XML or JSON format. Refer to *TC_ROOT/TcFTSIndexer/conf/specification/Mapping_Specification.docx* for example JSON and XML structures that the mapping framework can use to parse and index.

- If the extracted content is in JSON format, the key should represent a property. The value should represent the values you want to index.
- If the extracted content is in XML format, the XML element should represent a property. The element value should represent the values you want to index.

Ensure the machine with installed dispatchers can access the extractor.

Using a custom extractor to index your proprietary file type

The following example describes how to index the named reference **sampleFile** in the Teamcenter dataset **SampleDataset**. The extension of the file is *.sample*. A custom extractor that can extract indexable contents from the *.sample* files is assumed to be available.

1. Create a new dispatcher translator service using the supplied **tcftsindexerfilecontentnxindex** service in the *Dispatcher_Root\Module\conf\translator.xml* configuration file as a template.

Copy the existing **tcftsindexerfilecontentnxindex** service definition and rename it to **mysampleservice**.

Rename the XML element name to conform to the standard camel case syntax as shown in the original service definition. **MySampleService** follows the standard camel case syntax.

```
<MySampleService provider="SIEMENS"
  service="mysampleservice"
  maxlimit="3" isactive="true">
  <TransExecutable dir="&MODULEBASE;/Translators/TcFTSIndexer/
bin" name="runTcFTSIndexer.bat" />
  <Options>
    <Option name="flow" string="-task=filecontent:custom"
description="TcFtsIndexer flow to extract and index file contents
using a custom extractor."/>
    <Option name="mode" string="-standalone"
description="Switch to indicate that TcFtsIndexer needs to run in
standalone mode."/>
    <Option name="inputpath" string="-i=" description="Full
path to the input file or directory."/>
  </Options>
  <TransErrorExclStrings>
    <TransInputStreamExcl string="Error: 0"/>
  </TransErrorExclStrings>
</MySampleService>
```

The remainder of the service definition remains the same as the default **tcftsindexerfilecontentnxindex** service. This service calls the **runTcFTSIndexer** utility with the flow action input set to **-task=filecontent:custom**.

Ensure **isactive** is set to **true**.

2. In `TC_ROOT\TcFTSIndexer\conf\DatasetGroupingConfig.json`, add a new grouping entry for your dataset and named reference combination. Ensure the service name matches the service you created in the previous step.

```
{
  "SampleDataset": {
    "sampleFile": [
      {
        "name": "mysampleservice",
        "limit": 10
      }
    ]
  },
  ...
}
```

This entry creates a grouping rule for your dataset. Each time the indexer encounters the Teamcenter dataset **SampleDataset** with named reference **sampleFile**, it will route the file content indexing request to **mysampleservice**.

3. Create a mapping configuration file that contains mapping rules for the indexer. The indexer relies on these mapping rules to transform extracted file contents into an indexable format.

In the following example, the name **sampleMappingConfig** is assigned to the mapping rules and saved as a JSON file.

Refer to *TC_ROOT/TcFTSIndexer/conf/specification/Mapping_Specification.docx* for detailed instructions on how to create your own mapping file.

```
{
  "schemaVersion": "1.0.0",
  "configurations": {
    " sampleMappingConfig ": {
      "configVersion": "1.0.0",
      "description": "Mapping file for SampleDatasets.",
      "sourceFormat": "json",
      "rootPath": "$.",
      "dateTimeFormat": "yyyy-MM-dd",
      "mappings": [
        ...
      ]
    }
  }
}
```

4. Upload the mapping configuration file to the mapping configuration store by using the **aw_search_config_manager** utility. Run the following command from the Teamcenter command prompt:

```
aw_search_config_manager -import -dir=mapping-file-directory
```

5. In the *Dispatcher_Root\Module\Translators\TcFTSIndexer\conf\ExtractorConfig.json* file, add a new entry for your dataset and reference name combination.

Configure the **extractor** section for your extractor executable. In the following example, this extractor is *c:\extractors\custom_extractor\sample_extractor.exe*.

Configure the **transformer** section so the **configID** matches the identifier of the mapping configuration created previously.

```
"SampleDataset/File": {
  "extractor": {
    "type": "system-executable",
    "path": "
c:\extractors\custom_extractor\sample_extractor.exe ",
```

```

        "inputFileArg": "",
        "outputFileArg": "",
        "optionalArg": "",
        "argValueSeparator": " "
    },
    "transformer": {
        "configID": "sampleMappingConfig"
    }
}

```

6. Index your **SampleDataset** using the **runTcFTSIndexer** standard **objdata** flows, such as **index** or **sync**.

The main indexer routes file content indexing requests for your datasets to the target service. Then, the target service will invoke your extractor. Using the mapping rules you have created, contents are indexed into Solr.

Customizing Solr

Configure Solr URL support for fail-over

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. **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 `KeyStorePassword` lines and replace them with:

```
<Set name="KeyStorePassword">
<Env name="SOLR_SSL_KEY_STORE_PASSWORD" default="secret"/>
</Set>
```

- c. Find the `TrustStorePassword` 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**.

Upgrade Solr

New version of Solr

In Active Workspace 6.3, 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.

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 or Deployment Center 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.

Update Solr credentials in TEM

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 where the Active Workspace components are installed.
4. 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.

Update Solr credentials in Deployment Center

1. In **Components**, select the **Indexing Engine**.
2. Configure all applicable Indexing Engine settings, including the **User** and **Password**.
3. Click **Save Component Settings**.
4. Generate deployment scripts and deploy software on affected machines.

Resolving Solr errors

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* file 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:

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

Solr memory issues

Be sure you understand the **indexing process** and **consider your hardware for indexing**.

If you encounter Solr memory issues, you can increase the Java heap size. Refer to the **JVM Settings** section of the Solr documentation for information about analyzing memory usage:

<https://solr.apache.org>

As an initial strategy, for an index under 10 million documents, you can set the heap size to 10-16 GB. For an index over 10 million documents, you can increase the heap size to 50% of the machine memory. For example, if the machine has 80 GB of memory, set the heap size to 40 GB.

Regularly monitor your memory usage and adjust the heap size as necessary. For example, if you set the heap size to 40 GB but the memory usage is 20 GB, set the heap size to 25 GB to account for the actual usage plus 25%.

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.

Configuring SolrCloud

Configuring SolrCloud overview

SolrCloud configuration supports high availability of data using replicas and distributed indexes using sharding. Use these configurations either independently or in combination. These terms help explain Solr's cloud configuration options:

Node	A single instance of Solr.
Cluster	A group of nodes.
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.
ZooKeeper	Handles Solr configurations, load balancing, and fail-over behavior.

Standalone Solr configuration

By default, standalone 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 SolrCloud configurations use ZooKeeper. You can choose the approach that best fits your indexing strategy and search objectives:

Note:

Smart Discovery only supports SolrCloud configurations with a single shard and one or more replicas.

SolrCloud configuration	Underlying technology	Description	Benefit
High availability	A single shard with one or more replicas.	Queries are managed by the lead replica.	Maintains availability and provides fail-over redundancy.
Distributed index	Multiple shards, each shard having its own replica.	Queries are sent to all replicas simultaneously.	Improves performance, especially for large indexes.
Combination	Multiple shards, each shard having 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 is 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 configuration, 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** becomes the leaders.

Set up ZooKeeper

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.

Procedure

1. Download and unzip the supported version of ZooKeeper.
2. Create a *data* directory under *ZOOKEEPER_HOME*.
3. In *ZOOKEEPER_HOME\conf*, rename *zoo_sample.cfg* to **zoo.cfg**.
4. 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](#).

5. 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.

6. 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.

7. 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.

8. To start ZooKeeper, run **zkServer.cmd**.
9. 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.

Procedure

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.

- 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.

Ensure that each server instance has a unique directory, and there are no duplicates across all the others. Duplicates are directories already assigned to other servers. Then, delete the duplicate directories on each server.

- Start *Leader1_SOLR_HOME* instance first, and then start each **Solr_Cloud_Home** instance in the order that you would like the leaders assigned.

Set up basic authentication

You must set up basic authentication for SolrCloud configuration.

Procedure

- 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.
- 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":"..."}
  }
}
```

The example uses the encrypted password for the standard Solr password **SolrRocks**.

- 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
```

- Start SolrCloud.

- 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**.

6. Go to the Solr admin page (<http://hostname:port/solr/>) and log on using the username and password you just set up.

2. Configuring search

Configuring search

Configuring search 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.**
- **Set the default search operator**
- **Add wildcards to searches automatically**
- **Configure searching for common words**
- **Add synonyms to searches**
- **Boosting search results**
- **Configure results threshold**
- **Configure revision rules**
- **Define search prefilters.**
- **Configure search prefilters for object sets**
- **Configure filter panel behavior.**
- **Configure saved search.**
- **Configure column sorting**
- **Configure export to Microsoft Excel**
- **Return the parent business object for a dataset**

- **Configure types to return for global search**
- **Index and search alternate IDs**
- **Configure search for multiple sites**
- **Filter object data with a custom user exit method**
- **Configure Solr for a two-tier environment**
- **Configure Advanced Search.**
- **Return external business objects with a custom user exit method**
- **Configure shape 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_Phrased_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:

```
bmidemodeltool -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.

Setting 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**.

Adding 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 included in the Solr documentation:

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

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

When you run a query with a search phrase containing stop words enclosed in quotation marks, the following happens:

- 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 must 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. (Optional) Include all stop words in searches.
 - Search for lines containing *stopwords_language.txt* and comment them out.

Example:

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

3. (Optional) Edit the stop word list.
 - a. Open the *stopwords_language.txt* files and edit the list.
 - b. Save and close the files.
4. Restart 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.SynonymFilterFactory**.

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

Search result relevance for your users is determined by calculating a score for each result. First, initial scoring is generated. Then, each score may be boosted based on specific criteria. After scoring is complete, you may see the following:

- Properties with an exact match may take precedence over partial matches.
- Properties with any match may take precedence over exact matches in file content.

- Boosting that has been configured for specific object data attributes.

You can alter the boosting of search results based on:

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

Although some matching results may be boosted higher in the list, all results matching the search criteria are returned. An initial score of one object may also exceed a boosted score of another object. You must test your configuration to determine whether your boosting strategy has the desired effect on search results.

Configuring boosting based on 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.

Configuring boosting based on 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: <i>user-name</i>
	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: <i>user-group</i>
	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: <i>user-project</i>
	WorkspaceObject.project_list:LOW:\$MY_PROJECT

Configuring boosting based on date

Configure result boosting based on date using the **AWC_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).

Configuring boosting based on proximity

Configure result 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.

Configuring 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.

Configuring 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 levels

When a user's search includes a revision rule, sometimes the user cannot access an item that matches both the search criteria and the 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 will not 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 returns 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 is applied. 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 ... name="TC_RevisionAnchor" ... docValues="true" ... type="string"/>
<field ... name="TC_PRIV_am_rule_str" ... docValues="true" ... type="string"/>
<field ... name="TC_PRIV_matchingRevisionSelectors" ... docValues="true" ...
type="string"/>
```

5. Restart Solr.
6. Start indexing:

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

If object data was indexed with the same version of Active Workspace previously, you can skip this step.

7. After indexing is complete, start synchronization.

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

Effectivity dates in revision rules

You can specify 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**.

Implement effectivity for a revision rule as follows:

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 *TC_ROOT\TcFTSIndexer\conf\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. Use the **bmide_modeltool** utility to 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.

Supported revision rule entries and clauses

Supported revision rule entries and revision rule clauses can be applied to searching. If the revision rule contains both supported and unsupported clauses, the supported clauses are applied, and the unsupported clauses are ignored.

Revision rule entry	Revision rule clause
Working	Working() Working(Owning User = Current) Working(Owning User = <user_id>) Working(Owning Group = Current)

Revision rule entry	Revision rule clause
	Working(Owning Group =<group_name>) Working(Owning User = Current, Owning Group = Current) Working(Owning User = Current, Owning Group = <group_name>) Working(Owning User = <user_id>, Owning Group = Current) Working(Owning User = <user_id>, Owning Group = <group_name>)
Status	Has Status(Any Release Status, Configured Using Released Date) Has Status(<status_name>, Configured Using Released Date) Has Status(<status_name>, Configured Using Effective Date) Ensure effectivity dates are implemented to use this revision rule.
Latest	Latest (Creation Date) Latest (Alphanumeric Rev ID) Latest (Numeric Rev ID) Latest (Alpha+Numeric Rev ID)

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.
2. 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.

- 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
```

Configuring 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
```

Configuring 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 configure the behavior of filters to make it easier for users to find and apply filters. Users can change some of these preferences themselves using the **Search Settings** panel.

Configure type-ahead behavior

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_TypeAheadFacetSearch_Delay** 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 value is **500**.

Configure wildcard behavior for matching filter values

Use the **AWC_search_filter_wildcard** preference to configure the use of wildcards in searching behavior.

- | | |
|----------|--|
| 0 | No wildcard character is applied. |
| 1 | A wildcard is applied as a suffix to the search criteria. |
| 2 | A wildcard is applied as a prefix to the search criteria. |
| 3 | A wildcard is applied both as a prefix and as a suffix to the search criteria. |

If the preference is not set or has a value other than 0, 1, 2, 3, the **AWC_search_automatic_wildcard preference** setting applies.

Configure the bulk filter option

You can use the **AWC_Search_Show_Auto_Update_Filters** preference to configure the **Show Auto-update Filters Option** in **Search Settings**. When configured, the user can select the **Show Auto-update Filters Option** in **Search Settings** to display **Auto-update Filters** on the **Filters** panel.

When **Auto-update Filters** is selected on the **Filters** panel, the search results are refreshed when a filter is selected. When **Auto-update Filters** is not selected, the user can select multiple filters and click **Apply All** to refresh the search results.

Filters

Close

Auto-update Filters:

Filter by Property

▼ CATEGORY

☐ Category Two (1)
 ☐ Files (1)
 ☐ TableProperty (1)

▼ TYPE

☒ Item Revision (99)
 ☐ CFx_Table_Property_ItemRevision (1)
 ☐ Logical Connection Revision (1)
 ☐ MS WordX (1)
 ☐ Requirement Revision (1)

▼ OWNER

☒ Oscar OEM (oscar) (48)
 ☐ ed (ed) (48)
 ☐ ACE_RevBOM (ace_revbom) (1)
 ☐ Umarkar,Pankaj (umarkarp) (1)
 ☐ sptn_1 (sptn_1) (1)

[Show All](#)

Apply All

Configure the display of unassigned values

Use the **AWC_dynamicPriority_hideUnassignedValueFacetCategories** preference to toggle the display of filter categories with unassigned values from the **Filters** panel. Set the preference to **true** to hide filters with only unassigned values. Set the preference to **false** to display unassigned filter categories.

Limiting expanded filters takes precedence over hiding unassigned filters. If you hide unassigned values and enable the **AWC_Limited_Filter_Categories_Enabled** preference, some filters might still display unassigned values.

Configure filter groups

Filter groups are configured by default. Each user can select whether filters display in a list or in groups through the **Search Settings** panel. You can use the **AWC_Search_Filter_Categories_Display_Mode** preference to turn off filter groups.

The default groups are **Basic**, **Date**, and **Other**. Any filters not categorized as basic or date are included in the **Other** filter group.

You can customize your own filter groups. In the Business Modeler IDE, use the **Awp0SearchFilterPanelGrouping** list of values (LOV) template to create a new LOV with your unique groups.

Configure the **AWC_Search_Filter_Panel_Groups_Lov_Name** preference with the name of your new LOV.

Details

Project: aws2

Name: Awp0SearchFilterPanelGrouping

Description: Search Filter Panel Grouping LOV.

Type: ListOfValuesString

Usage: ☒ Exhaustive ☐ Suggestive ☐ Range

LOV Value Management: ☒ Enter values using BMIDE and store values in my template ☐ Supply values directly to Teamcenter database using "bmide_manage_batch_lovs" command line utility

Reference:

☒ Show Cascading View

Value	Description	Condition	COTS	Template
Basic		isTrue		aws2
Awp0SearchFilterPanelBasicGrouping	Basic Group of search filter categories.	isTrue		aws2
Categorization.category		isTrue		aws2
WorkspaceObject.object_type		isTrue		aws2
POM_application_object.owning_group		isTrue		aws2
WorkspaceObject.project_list		isTrue		aws2
WorkspaceObject.release_status_list		isTrue		aws2
Date		isTrue		aws2
Awp0SearchFilterPanelDateCategoriesGrouping	Group of Date Categories for Search Filter Panel	isTrue		aws2
POM_application_object.creation_date		isTrue		aws2
POM_application_object.last_mod_date		isTrue		aws2
WorkspaceObject.date_released		isTrue		aws2

☐ COTS?

Template: aws2

Buttons: Add Sub LOV, Remove Sub LOV, Edit..., Expand All, Collapse All

Specify the set of expanded filter groups

You can configure which filter groups expand automatically when search results are returned. Reducing the number of expanded groups makes it easier to view a long list or view filter groups that are more commonly used.

The **AWC_Search_Filter_Panel_Expand_Selected_Groups** preference is enabled by default. When enabled, it limits the groups that are expanded automatically.

Add the names of one or more groups that you want to expand automatically to the **AWC_Search_Filter_Panel_Groups_Expanded** preference.

Each user can also select additional groups to expand through the **Search Settings** panel.

Specify the maximum number of displayed filter groups

You can limit the number of filter groups that are displayed at one time, making it easier to view a long list of filter groups. If more filter groups are available, users can click **Show All** at the bottom of the list to expand it.

Use the **AWC_Search_Filter_Panel_Number_Of_Groups_Shown** preference to specify how to display filter groups.

You can limit the number of filters that are displayed in each group. Having fewer filter categories makes it easier to view a long list of filter values. If more filters are available, users can click **Show All** at the bottom of the list to expand it.

Use the **AWC_Search_Filter_Panel_Number_Of_Categories_Shown_Inside_Each_Group** preference to specify how to display filter groups.

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.

The **AWC_Limited_Filter_Categories_Enabled** preference is enabled by default. When enabled, it limits the categories that are expanded automatically. The **Category** and **Type** filter categories are always expanded for your users with no additional configuration.

Each user can also select additional categories to expand through the **Search Settings** panel. If the user wants to expand a category that is not available in the **Filters to Expand** drop-down list, add the category name to the **AWC_Master_List_Limited_Filter_Categories_Expanded** preference.

If you want more than the **Category** and **Type** filter categories to automatically expand for your users, add the additional category names to the **AWC_Limited_Filter_Categories_Expanded** preference. Specify one or more properties to expand automatically, with one value in each line.

Example:

```
AWC_Limited_Filter_Categories_Expanded=
POM_application_object.owning_user
POM_application_object.last_mod_user
```

Category and **Type** filter categories are always expanded, even though they might not be listed in the **AWC_Limited_Filter_Categories_Expanded** preference.

Site preference settings also configure the user's **Search Settings** panel.

In the following example, the **Type**, **Category**, and **Last Modifying User** categories are expanded automatically for your users.

Preference	Configuration
AWC_Limited_Filter_Categories_Enabled	Enabled (default). Users see the Selected Filters and All Filters options under Expand in the Search Settings panel selected automatically.
AWC_Limited_Filter_Categories_Expanded	Add POM_application_object.last_mod_user to the value list. Users see Last Modifying User selected for expansion automatically in the Selected Filters drop-down list.
AWC_Master_List_Limited_Filter_Categories_Expanded	If POM_application_object.last_mod_user is not present in the value list, add it. Users see all categories listed in the Selected Filters drop-down list. Users can choose to expand any additional category.

Specify the maximum number of displayed filter values

You can limit the number of filter values that are displayed for each specified category. Few 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. The remaining filters can be viewed by clicking 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 are displayed 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 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%, 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), 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 this 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 triggers 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.

Specify first day of the week

When a property displays date boxes, your users can enter the start and end dates. When the filter displays the dates, they are grouped in increments depending on the date range. Specify which day of the week is the first day in the **Start Of Week** site preference.

Configuring 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.

Configuring 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.

Configuring 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**.

Returning 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 dataset types 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.

For example, to 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
```

The **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.

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 file content, the **AWS_FullTextSearch_Index_Dataset_File_Content** preference must be set to **on**.
- Configure the **objdataloadstep.indexStandaloneDatasets** property in your properties file to return datasets in your search results in addition to the parent business object.
- **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

These examples show how to configure the visibility of datasets in your user's search results.

First, ensure indexing is configured for inline indexing:

- Set the **Awp0SearchDatasetIndexingBehavior** constant to **Inline**.
- Configure the **Awp0DatasetTypeToBeIndexedInline** constant for the file types you want to index.
- Set the **AWS_FullTextSearch_Index_Dataset_File_Content** preference to **true**.

To return both the parent business object and the attached dataset in your user's search results, set the **objdataloadstep.indexStandaloneDatasets** property to **true**. Datasets are indexed in standalone mode.

To return only the parent business object in your user's search results, set the **objdataloadstep.indexStandaloneDatasets** property to **false**. Datasets are indexed inline. Standalone dataset content indexing can also be disabled by changing the **Awp0SearchIsIndexed** constant for the desired dataset types. If the **Awp0SearchIsIndexed** constant is not changed and the **objdataloadstep.indexStandaloneDatasets** property is set to **false**, standalone dataset content indexing is disabled.

Configuring 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.

Indexing and searching 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

You 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 **AWP0CUSTOM_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
    AWP0CUSTOM_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**.

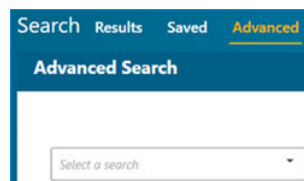
Configuring Advanced Search

Advanced Search is a built-in Active Workspace capability that allows you to provide users with predefined queries that search specific object properties. If users regularly search for specific object properties not included in the predefined queries, you can create additional custom queries. Users can view search results either as a list or in a table. For tables, users can set the column sorting criteria based on your organization's requirements.

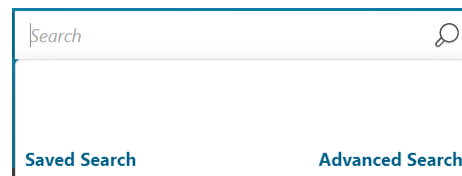
Control visibility

Advanced Search can be shown or hidden for Active Workspace users. Set the **AW_Advanced_Search_Visibility** preference to show or hide the following items. The default value for this preference is **true**, which makes Advanced Search accessible to users.

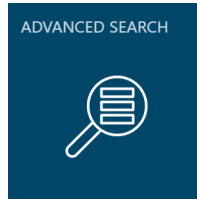
- **Advanced** page in the **Search** page header area.



- **Advanced Search** link in the global **Search** box.



- **ADVANCED SEARCH** tile on the home page.



Create queries

You can create **Advanced Search** queries using Query Builder in the rich client.

Limit available queries

You can limit the list of available queries in **Advanced Search** to a defined list of saved queries by configuring the **AWC_Search_Saved_Queries_List** preference.

You can set a list of preferred queries in **Advanced Search** by configuring the **QRYColumnsShownPref** preference.

Display count for advanced search results

You can set the display count of advanced search results by configuring the **WSOM_find_set_search_limit** preference. The default value for this preference is 1000. For example, if the **WSOM_find_set_search_limit** is set to 1000, the exact number of search results appears when the total is 1000 or fewer results. If more than 1000 results are returned, the display count is **1000+**.

You can set the display count to provide information on results that the user does not have access to, also known as unreadable content, by configuring the **AW_Display_Not_Readable_Count** preference. The default value for this preference is **false**. If this preference is configured, a lock icon displays when unreadable results are returned. Hovering over this icon displays the number of unavailable results.



Use delimiters to combine search criteria

Delimiters can help your users combine multiple search criteria so that results are displayed in the same table. However, enabling certain delimiters with the **AWC_WSOM_find_list_separator** preference can affect search results if the delimiter is also present in the system data. The default values for this preference are **SEMICOLON**, **NEWLINE**, and **TAB**.

Set column sort criteria for results

Tables returned by **Advanced Search** contain objects that match the **Search Type** specified in the query definition. The columns provided by default display certain properties from returned objects as well as certain properties from related objects. Sorting on the columns provided does not require configuration.

Configure the **AW_Advanced_Search_Reference_Sort_property** preference in the following situations.

- To change the sorting criteria for columns on a related object:

Configure the existing values in the preference.

For example, if you want the **Owner** column to be sorted by the **user_id** instead of the **user_name** property, change the existing **owning_user.user_name** value to **owning_user.user_id**.

- To add a column that displays a property from a related object:

Add the new property to the preference values using the form **typedReferenceProperty.attribute**.

For example, if you add a column that displays the owning group name, add **owning_group.name** to the value list.

AW_Advanced_Search_Reference_Sort_property preference value requirements:

Supported	<p>For local queries, valid preference values must be single (non-array) TypedReference properties.</p> <p>For example, you can sort on owning_user.user_id but not Item.owning_user.user_id because the latter is a compound reference.</p>
Not supported	<p>Sorting on remote queries or queries using custom code is not supported.</p> <p>Users cannot sort on:</p> <ul style="list-style-type: none"> • Arrays • Runtime properties • Compound properties • UntypedReference properties • Table properties, including Name-Value properties

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.

Configuring shape search

If you install the **Shape Search** feature on Windows or Linux, you must create the following preference and configure it:

GeolusServer

Defines the URL used for communication between shape search and Geolus.

After the **Shape Search** feature is installed successfully, you can configure it using the following preferences. These are added automatically to the preference **AWC_StartupPreferences** during the installation process:

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.

Tip:

Use the *Administering Search Guidemap* in the Teamcenter help to learn about different types of search available for your users.

Set the GeolusServer preference

The **Shape Search** feature in Active Workspace uses the Geolus shape search engine. To enable communication between Active Workspace and Geolus, you must create the **GeolusServer** preference.

Using **Preference Management** in Active Workspace or **Organization** in the rich client, create the **GeolusServer** preference with the following properties:

For this property	Do this
Name	Type GeolusServer .
Category	Type DecisionApps.ShapeSearch.Preferences .
Description	Type a useful description for the preference.
Value	Type the Geolus server URL in the following format: <i>protocol://gServer:gPort/gContext</i>

For this property	Do this
	<p>where:</p> <ul style="list-style-type: none"> • <i>protocol</i> can be http or https. • <i>gServer</i> is the machine name or IP address of the machine running the Geolus server. It must be accessible to all Teamcenter clients that need to connect to it. • <i>gPort</i> is the port number that the server uses to handle HTTP or HTTPS requests. • <i>gContext</i> is the context root of the Geolus server.

Troubleshooting search

Finding 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.

Troubleshooting 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 on Windows or Linux.

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.