TEAMCENTER

Structure Indexing Using Cacheless Search

Teamcenter 2412

SIEMENS

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

1. Getting Started

What is cacheless search?

Cacheless search allows you to perform spatial search of the live (latest) configured product data that is persistent in the Teamcenter database. It is not necessary to maintain secondary structure data caches or to create a separate database.

The cacheless search engine must be used instead of the QPL and appearance-based search engines that were available in earlier versions of Teamcenter. All the functionality provided by the earlier search engines is also available with cacheless search. Unlike appearance and QPL searches, cacheless searches are performed on recent data, and not on data that was saved the previous day.

The audience for this information is administrators and Teamcenter services personnel who are tasked with deploying cacheless search.

You can configure cacheless search for one of three modes:

- NX based

  Configure this mode if you exclusively use NX as your CAD solution.

- JT based

  Configure this mode if you exclusively use CAD solutions other than NX.

- Mixed

  If you work in a multi-CAD environment, you must configure both NX and JT modes.

To use cacheless search, populate the database with the following objects:

- Teamcenter bounding boxes (bbox data)

  Bounding boxes must be created for all structures you want to search. Bounding boxes are attached to their owning item revisions and to their geometry source, either a CAD or JT dataset.

  To create bounding boxes, run the **create_or_update_bbox_and_tso** utility.

- (Optional) TruShape (TSO) voxel files. TruShape files provide more accurate results but occupy additional space in the database.

  To create TSO files, run the **create_or_update_bbox_and_tso** utility.

- Spatial search indexes

- Index queue

  Teamcenter creates the index queue to hold objects that require their indexes to be created or updated. Depending on your deployment strategy, the items may be restricted to one or more selected products in the database.

  > Note:
  >
  > This queue is transient and gets used by the index update process as the objects in the queue get processed and incorporated in the spatial index.

Once you have generated the initial data, you must keep it updated to reflect changes to the CAD designs. There are two methods of doing this:

- Using Dispatcher
- Using a **cron** job or service

You can choose one of two deployment strategies when creating bounding boxes and search indexes:

- Full database

  Use this strategy if you have relatively small products or designs for only a few products.

- Product-scoped cacheless search

  Use this strategy if you want to exclude data from products that are complete or inactive from the search results.

How cacheless search works

Cacheless searches allow you to query the configured product structure in Structure Manager, Design Context, and several other applications to retrieve matching items and item revisions. You can specify any combination of the following search parameters:

- Spatial parameters including:
  - Box volumes, either inside, outside, or intersecting the box
  - Above, below, or intersecting defined planes
  - Proximity to another object
  - Approximate size
- Attributes including item or item revision ID, name, occurrence notes, and form attributes

- Classification attributes
- Saved queries

Teamcenter uses bounding boxes and TruShape voxel maps that are created during the processing of CAD designs to implement cacheless searches. Internally, it uses part-level bounding boxes to construct and maintain the necessary spatial indexes.

If you use NX and the Teamcenter Integration for NX, you can configure NX to generate bounding box data automatically. If you search JT files generated by another CAD tool, Teamcenter generates the bounding box data when spatial searches are first enabled. The generation process may take a significant time if you have large quantities of data.

> Note:
>
> If you work in NX, you should not manipulate the JT files that Teamcenter creates from the NX files.

The search initially identifies bounding boxes that intersect the bounding boxes of the selected objects, limiting the search results to objects with geometry in close proximity. This allows Teamcenter to display the initial results relatively quickly.

If you choose TruShape filtering, the search then tests the objects with intersecting bounding boxes to determine if they have intersecting TruShape parameters. Depending on your TruShape configuration, Teamcenter may simplify the geometry of each object into regular cubes (voxels) that provide a simplified representation of the actual shapes. The TSO files contain voxel representations of each part that has geometry.

The accuracy of the search results depend on the cell size you define in NX or the voxel size you configure in Teamcenter if you are using JT files. The smaller the cell or voxel size, the more accurate the results. However, smaller sizes result in larger TSO files and slower response times.

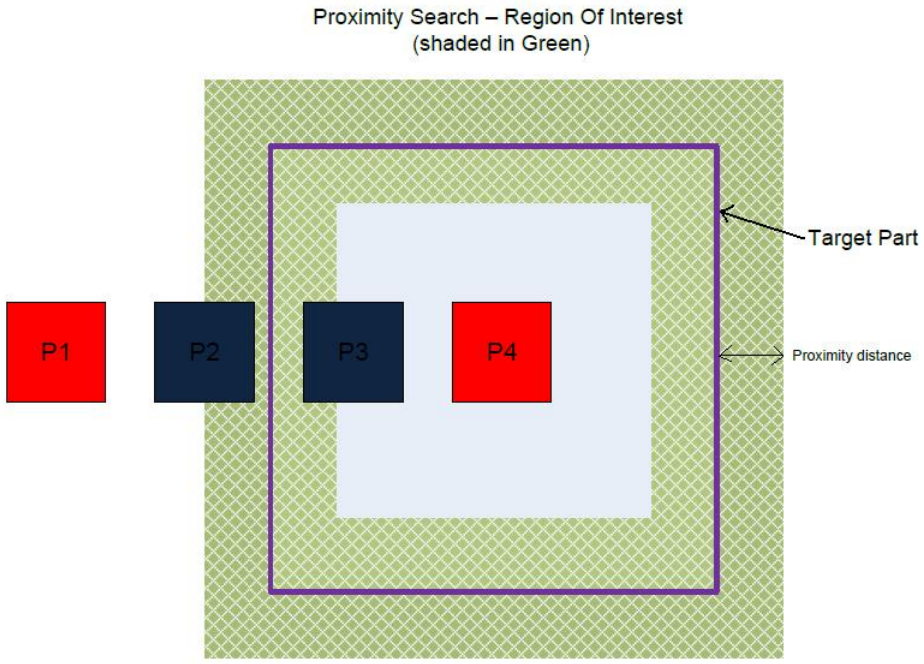
The following actions trigger an update of the bounding box data, search index, or both:

- The creation of a new Teamcenter item causes the creation of a structure index. If your system automatically creates a dataset under the item revision when you create items, this action triggers the creation of a bounding box, even if the bounding box is empty. This occurs whether you create the item directly in Teamcenter or in any of the Teamcenter integrations.
- Any structure edits, the addition or subtraction of occurrences, and absolute occurrences modifications cause the creation of the structure index.
- The creation of an NX or JT dataset triggers the creation of the bounding box. This action also triggers a structure update to roll the bounding box information up to its parent structures.
- Any modification of an NX or JT dataset triggers an update of the bounding box. This action also triggers a structure update to roll the updated bounding box information up to its parent structures.

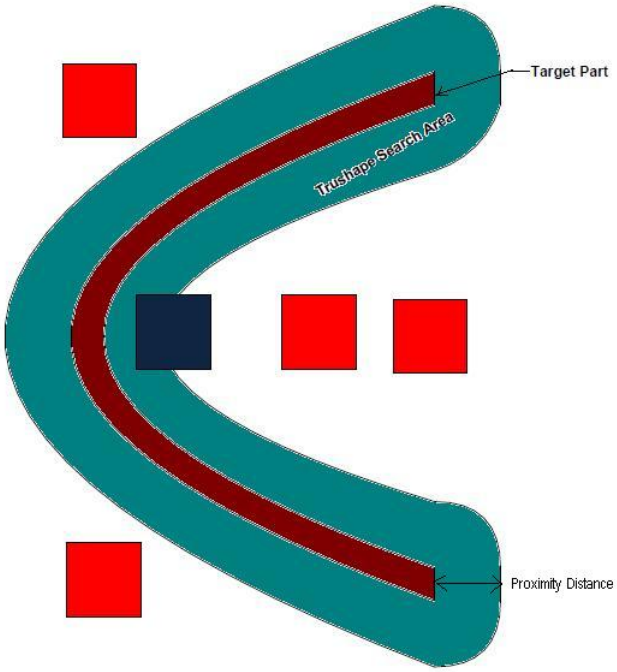
Teamcenter also makes these updates if modified data is imported from another system.

Proximity Search and TruShape

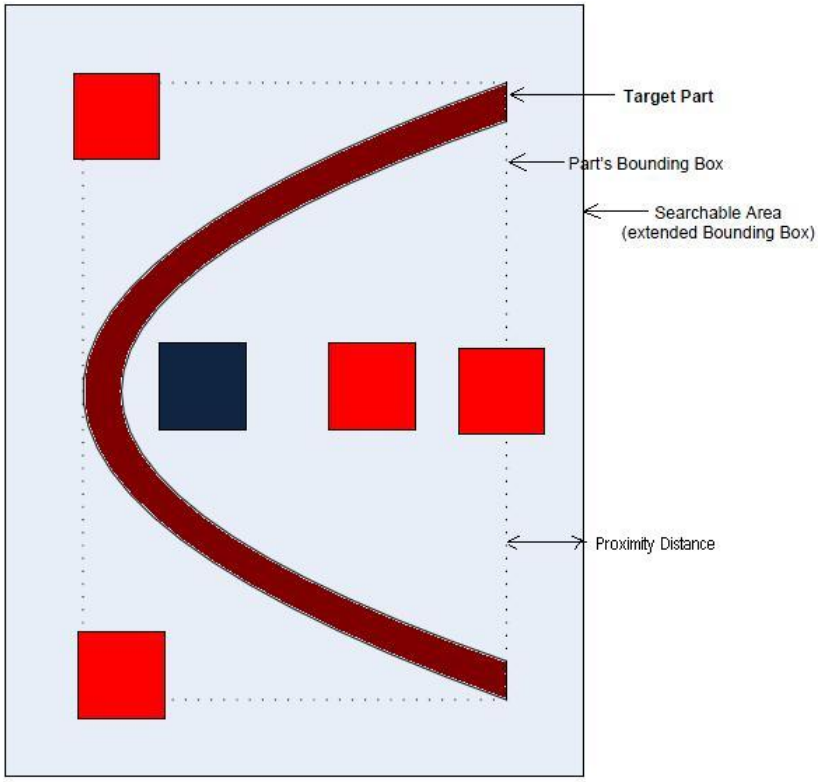
Proximity search returns parts within the proximity range from the external surface of the target part as shown in the following figure:



Proximity search with TruShape returns only the blue part shown in the following figure:



Proximity search without TruShape (only the bounding box filter) returns all the parts, including the red parts, as shown in the following figure:



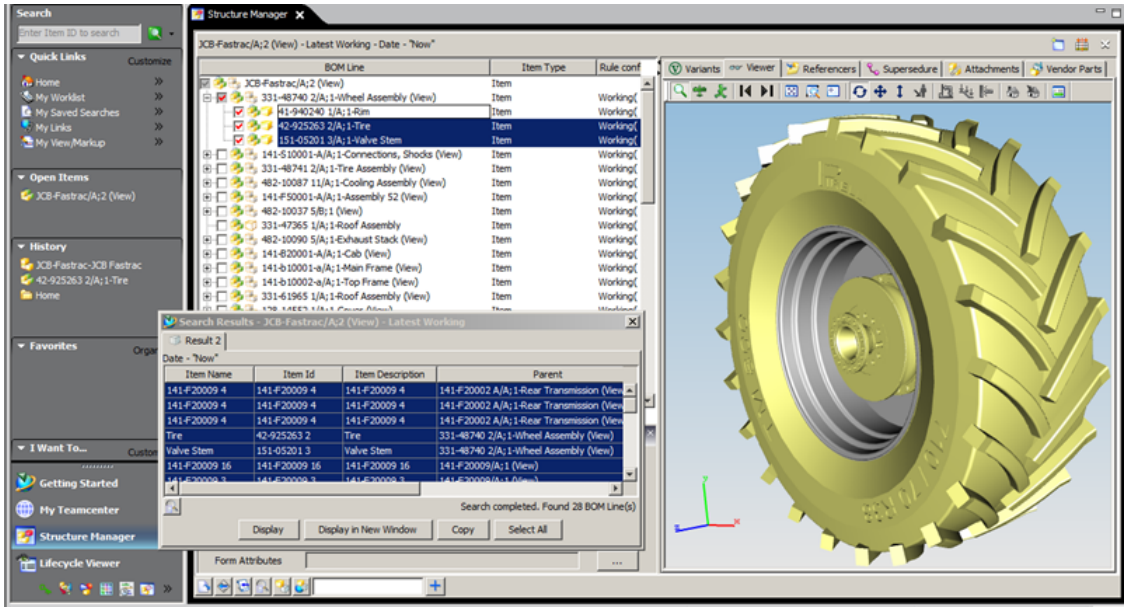
Using cacheless search in the rich client and web client

Cacheless search allows you to locate sufficient information in the context of the product you are designing or building to validate fit, form, and function for current data.

The following applications allow you to make cacheless searches.

- Structure Manager
- Design Context.
- Manufacturing Process Planner
- 4G Designer
- Teamcenter web client

You enter spatial criteria, attributes values, or a combination of both, and then click **Search**. Teamcenter displays matching parts as a list of BOM lines. You can send selected lines to the viewer for inspection.



Using JT based cacheless search

If you use CAD tools other than NX, you must install the Dispatcher Server and **JtToBboxAndTso** translator to create the necessary bounding box and TruShape data. You must also install the **QSEARCH_process_queue** service to create an index for spatial searches. Teamcenter reads the bounding box information from this index. Teamcenter stores the calculated TruShape data as a **Trushape-Data** named reference in a **DirectModel** dataset.

The accuracy of the search depends on the voxel size you configure in Teamcenter. A small voxel size provides more accurate search results, but the TruShape files are larger and the response times, slower.

> Note:
>
> The bounding box and TruShape data generated for a JT dataset are ignored if an NX dataset exists for the same item revision.

On high performance hardware with a well-tuned database, Teamcenter can create bounding boxes for 1,000 parts to 5,000 parts per hour from JT data.

Using NX based cacheless search

If you use NX, it calculates the bounding box for a part and stores it in Teamcenter as a **UGPART-BBOX** named reference in a **UGMASTER** dataset. Teamcenter reads the bounding box information from the **UGPART-BBOX** form. NX can also be configured to create TruShape data for the part and store it as a **Trushape-Data** named reference in the **UGMASTER** dataset.

The accuracy of the search depends on the cell size you configure in NX. A small cell size provides more accurate search results, but the TruShape files are larger and the response times, slower.

If you import existing NX assemblies, set the **QS_NX_RELATION_POST_ACTION_ENABLED** preference to **True** and Teamcenter creates the necessary bounding box data automatically.

In many deployments, it might not be necessary to use Dispatcher because you can use some other mechanism to create and update search indexes. However, you must update search indexes periodically. The indexer must always be up and running. Spatial search indexes are generated from the bounding boxes. That is, bounding boxes are an input to the search index. Teamcenter Integration for NX creates bounding boxes. It cannot create search indexes. Advanced NX features such as datum points, weld points, and deformable components are available on both JT and NX datasets.

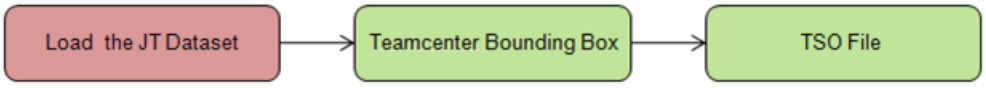
Apart from NX, for CAD integrations, such as CATIA, Solid Edge, and Pro/ENGINEER (Pro E), the JT-based integration is used to generate the bounding boxes from JT parts. For these integrations, you must install a dispatcher-based translator (**JTTOBBOXTSO**).

Use the **create_or_update_bbox_and_tso** utility with the **JTTOBBOX+JTTOTSO** option selected to generate bounding box data that includes advanced NX features. You can then run the **qsearch_process_queue** utility Dispatcher service to create the corresponding search index.

On high performance hardware with a well-tuned database, Teamcenter can create bounding boxes for 10,000 parts to 20,000 parts per hour from NX data.

Understanding bounding boxes and TSO files

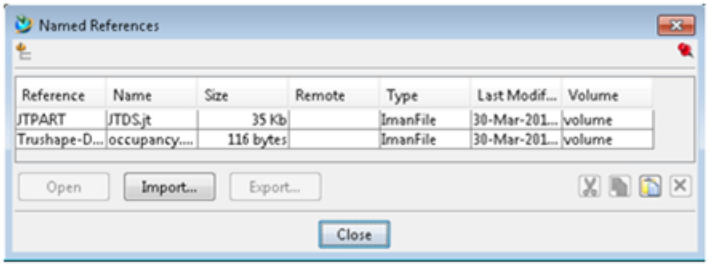
To populate the bounding boxes and TSO files, Teamcenter loads the JT file and then extracts the bounding box information. It adds that information to the Teamcenter bounding box table (**PBOUNDINGBOX**) and then creates the TSO file as a named reference in the JT dataset, as follows:



Teamcenter Bounding Box Information

PUID	PX_MIN	PY_MIN	PZ_MIN	PX_MAX	PY_MAX	PZ_MAX	RPARENTU	RPARENTC
gYN15UvXoUI4cC	-0.0059983	-0.0059983	-0.001	0.0059983	0.0059983	0.001	gUJ15UvXoUI4cC	58

The TSO File

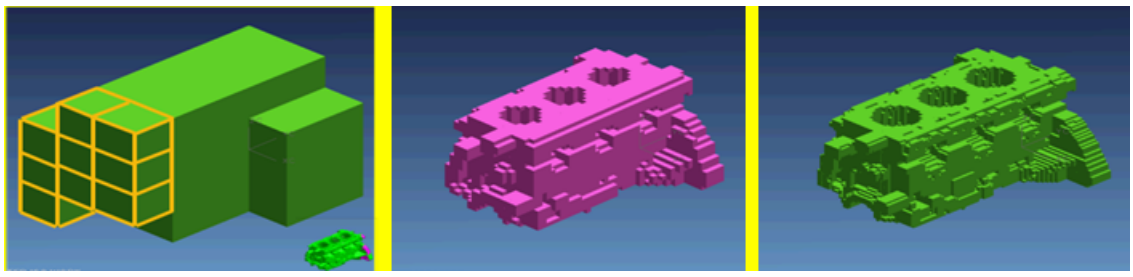


You can inspect the bounding box data for a selected part by adding **bl_itemrev_bounding_box** to the displayed properties in Structure Manager. This property shows the following values of the bounding box in the local coordinate space of the node:

- x minimum
- y minimum
- z minimum
- x maximum
- y maximum
- z maximum

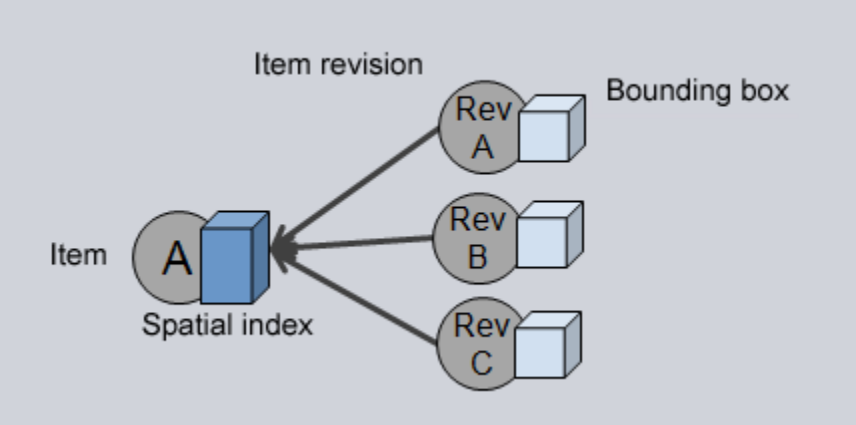
Optionally, you can refine the accuracy of the proximity search when using TSO files. To configure the cell size of the TSO file, change the setting of the **TC_JT_voxel_size** preference from its default value of 12.75 millimeters. Some testing with sample geometry may be necessary to find the optimum cell size.

The following example shows how TSO data refines the geometry:



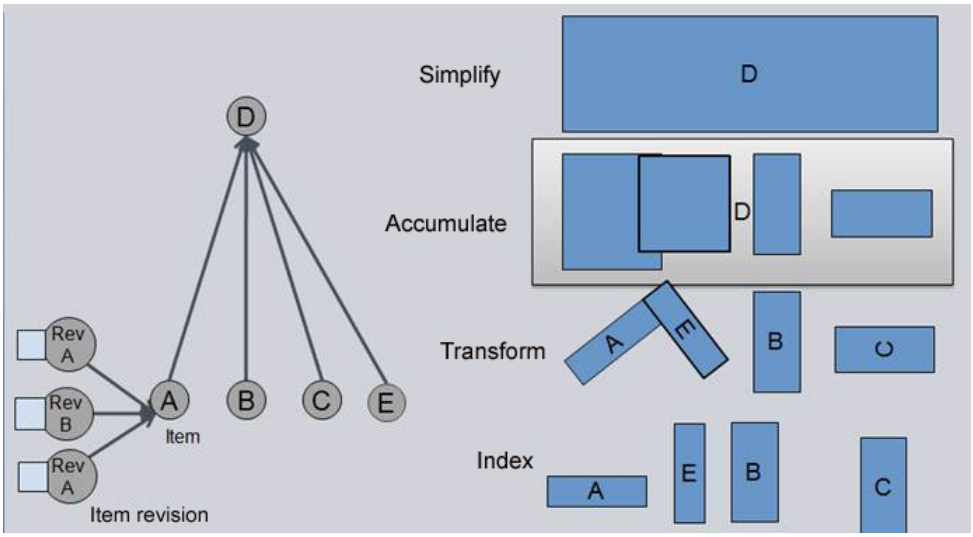
Understanding spatial search indexes

In its simplest form, a spatial index is the bounding box that encloses the bounding boxes of all item revisions of an item.



A spatial index is also the bounding box that encloses all the extents of the constituent substructure (that is, subassemblies).

When Teamcenter calculates spatial indexes, it rolls up the boxes up wherever they are used. Consequently, the boxes of Rev A, Rev B, and Rev C are rolled into the index of item A. This index is saved separately to represent item A's volume and is then propagated upward. It is transformed into the coordinate space of its parent in the structure. The parent's own index is extended to accommodate the transformed indexes of all its children.



The spatial indexes are updated when Teamcenter processes requests on the index update queue. The update queue contains items, occurrence threads, and appearance path nodes (APNs) that are processed in batches. The queue is processed in batches on a first in, first out (FIFO) basis. Teamcenter does not wait until the queue is full, but is configured with a maximum batch size for optimum scalability. Only certain types of objects are placed in the queue, for example, changes to a bounding box attached to an item revision causes Teamcenter to place the corresponding item in the queue. Likewise, a change to a transform causes Teamcenter to place the corresponding occurrence thread in the queue when the occurrence is saved in the database.

You create spatial indexes when you run the **qsearch_process_queue** utility to process the queue and can optionally validate the indexes by running the same utility with the **check_structure_indexes** option specified.

> Note:
>
> There is a single indexer (**qsearch_process_queue** process) for each database.

Internally, a spatial search with spatial indexes is a BOM expansion guided by the index. Teamcenter traverses down one level and finds any indexes that meet the specific spatial criteria. It prunes out nonmatching indexes and continues the traversal down to the next level. Pruning removes subtrees and discards unnecessary parts of the structure, contributing to the speed of the search. It also processes overrides similarly. After configuration of the necessary lines, Teamcenter validates the results against actual bounding boxes and TruShape data.

To correctly configure the data, Teamcenter generates a partial BOM structure. The time it spends performing a search typically includes:

- Creating the partial BOM structure (50 to 70% of the total time).
- Spatial filtering by bounding boxes (10 to 20% of the total time).
- Configuring variants and processing properties (10 to 20% of the total time).

The lower the variation in geometry between different revisions of the same item, the better the search engine performs. The greater the correlation between the product structure and the geometry breakdown, the better the search engine performs.

Similarly, attribute searches require a partial BOM structure configuration, using objects that satisfy the given attribute search criteria. Teamcenter first searches for the objects in the database and then traverses up all BOM configurations. Some of the traversals may be outside of the structure you are searching. However, implementation of product scoping avoids unnecessary traversals. Teamcenter traverses up until it reaches the product in which the required results are found and the final result is the same.

Tuning the indexer

On high performance hardware with a well-tuned database, Teamcenter can index 50,000 objects to 150,000 objects per hour. Actual performance on a production system depends on the rate of change of structure and geometry, for example, occurrences, occurrence transforms, item revisions, bounding boxes (JT and CAD), and absolute occurrence transform overrides. You can use Oracle auditing and SQL scripts to measure the hourly or daily changes based on the last modified date of each object. From those measurements, you can estimate the number of changed items, occurrence threads, and APNs per hour you must index (the indexer queue only contains those objects). You can also identify the optimum mixture of queued objects for the best performance.

You should then measure the actual maximum indexing throughput, which comprises two considerations:

- Throughput of operations that do not alter indexes, which you can identify by running the **qsearch_process_queue** utility with the **–queue_update** option.

- Throughput of requests that cause index changes, which you can identify with the CAD tool or a custom script or utility.

> Tip:
>
> Avoid accumulating a large number of entries in the indexing queue. The faster updates are processed, the sooner live updates are available to users. During the deployment process, you should monitor the size of the queue by periodically running the **qsearch_process_queue** utility with the **-list_queue** option selected. If the queue size is increasing steadily, you may have to reconsider your deployment strategy. In a test environment on high performance hardware, Teamcenter can process between 60,000 and 120,000 requests per hour without the queue building up.

If you have a relatively small product structure (a thousand or less occurrence threads), you may improve performance by switching to a top-down search. This configuration limits the scope of the search to the product only. However, Teamcenter must first traverse the unconfigured structure in the database to establish the search scope. This approach may be more time-consuming with large product structures; if so, it may be better to tune the default bottom-up search for best performance in such cases.

By default, Teamcenter does not include all possible indexes that you might need for attribute queries. Additional database indexes have an associated performance cost, and you should consider carefully whether you need them. Use the Oracle database tools in a live production environment to assist with these decisions.

Understanding cacheless search and Multi-Site Collaboration

If you work in a Multi-Site Collaboration environment, you can make cacheless spatial and attribute searches of products or programs that are owned or replicated at remote sites. Where appropriate, local and remote objects are incorporated in the search results.

If you enable this feature:

- Teamcenter checks access permissions to the remote BOM lines. If you do not have the necessary permissions, you cannot import the affected remote lines. If appropriate, you can review the search results and import only the lines to which you have access permissions.

- Identical revision and variant rules must be defined at all sites or searches may return inconsistent results. Also, the content of these revision rules and variant rules must always be consistent across all the sites.

- The content of saved queries must be consistent across all sites, or searches may return inconsistent results.

- You can only initiate the search in Design Context and not in Structure Manager or Multi-Structure Manager.

You do not need to manually configure IDSM to use remote cacheless searches. When you install or upgrade the system using Teamcenter Environment Manager (TEM), it installs the callbacks necessary for remote cacheless search. Each site participating in remote cacheless search must install or upgrade the database using the same version of TEM.

Deploying cacheless search

When you deploy cacheless search, you must choose between a product-scoped deployment strategy and a full database deployment strategy.

- Full database deployment

  Teamcenter generates bounding boxes for all geometric entities in batches, ignoring items without geometry. It places all update candidates in the indexer queue. Update candidates are any item revisions with bounding boxes.

- Product-scoped deployment

  You mark one or more products as indexable, and Teamcenter processes only the relevant items. No indexes are created for old and inactive programs, saving system resources.

The feasibility of processing the complete database depends on the size and number of bounding boxes to process. As a guideline, product-scoped deployment is probably the best strategy if less than 30% of your data is in active use.

To obtain a baseline time, you can test a representative sample of data and then estimate the time needed to process all the bounding boxes and indexes. For example, if the entire database cannot be processed in one weekend, you may prefer the product-scoped deployment strategy. Even with this strategy, the initial preparation and processing of a single product may take hours or even days, depending on the number of components in the product.

An alternative strategy is to use a product-scoped deployment when you first populate the database, that is, to index your data one product at a time. You can then switch to a full database deployment when the database is populated with all products.

Setting up product-scoped cacheless search

At some customer sites, a significant proportion of the product structure data in the databases is no longer in use, for example, it belongs to programs that are complete and no longer active. If you index this data for spatial searches, it has an adverse impact on the speed at which indexes are created and

also on search performance. This enhancement allows you to limit spatial and attribute searches to active programs, which may correspond to end items, top-level items, or top-level product items. When product-scoped search is deployed, users can search on all levels of the product.

By default, all data is unsearchable and you must run the **qsearch_process_queue** utility on the chosen product structures to mark all items and occurrences that are part of those structures as indexable. The **create_or_update_bbox_and_tso** utility then processes only bounding boxes attached to indexable product structures and their components. This utility also identifies missing bounding box and TSO data for indexable products.

Once an active structure is indexed, you can make incremental updates to its spatial search index, rather than generating a complete new index each time. Incremental index updates are restricted to product structures previously identified as active. To process the queue, the administrator runs the **qsearch_process_queue** utility with the **-process_queue** option.

To deploy cacheless search for a product-scoped strategy:

1. Ensure you have enabled or upgraded cacheless search with Teamcenter Environment Manager (TEM) before you begin the deployment procedure.

2. Ensure that live background updates are turned off while you manually process the product data.

3. **Prepare your data for processing** by updating legacy transforms and removing cycles from the database. This step must be completed carefully or the results of searches may be incorrect.

4. **Enable product-scoped cacheless search**.

5. **Make the product indexable**. This step makes all components of the chosen product available for cacheless searches.

6. **Create the bounding boxes and TSO files** for the chosen product.

7. After the bounding boxes and TSO files are available, **populate the queue** with the entries for the chosen product. Do *not* run the **qsearch_process_queue** utility until the **create_or_update_bbox_and_tso** has finished.

8. **Process the queue** for the chosen product.

9. If you want to make other products in the database available for cacheless searches, repeat steps 3 through 6 for each product in turn.

10. **Query for and process any bounding boxes** remaining in the database that do not belong to the products you have already processed.

11. Configure live background updates with Dispatcher or **cron** jobs to process changes to the indexed data.

Note:

In a typical system, you may run a **cron** job every 15 minutes to process updates. The value could be adjusted, depending on how frequently data changes and the extent of those changes.

Always turn off live background updates if you run the **qsearch_process_queue** utility to manually process the queue.

Setting up full database cacheless search

Siemens Digital Industries Software recommends that you process both the bounding boxes and spatial indexes in batches if the database has a large number of datasets. The generation of bounding boxes and spatial indexes in batches helps prevent the utilities from running out of memory or causing other scalability issues. If you have a small database of less than 50,000 UIDs, you may process all the datasets without any batching.

To deploy cacheless search for a full database strategy:

1. Ensure that you have enabled or upgraded cacheless search with Teamcenter Environment Manager (TEM) before you begin the deployment procedure.

2. Ensure that live background updates are turned off while you manually process the product data.

3. **Prepare your data for processing**, by updating legacy transforms and removing cycles from the database. This step must be completed carefully or the results of searches may be incorrect.

4. **Query for a list of dataset UIDs** to process for the entire database.

5. **Create the bounding boxes and TSO files** for the database.

6. After creating the bounding boxes, regenerate the database statistics to improve search performance.

7. **Populate the queue** with the entries for the entire database.

8. **Process the queue** you populated in the previous step. Do *not* run the **qsearch_process_queue** utility until the **create_or_update_bbox_and_tso** has finished.

9. After generating the indexes for a product, **regenerate the database statistics** to improve search performance

10. **Configure live background updates** with Dispatcher or **cron** jobs to process changes to the indexed data.

Note:

In a typical system, you may run a **cron** job every 15 minutes to process updates. The value could be adjusted, depending on how frequently data changes and the extent of those changes.

Always turn off live background updates if you run the **qsearch_process_queue** utility to manually process the queue.

2. Working with Product Scoped Cacheless Search

Preparing your data

Before you use cacheless search on an existing database, you must run a few checks and clean the data.

To begin with, you must upgrade all existing legacy transforms, since the cacheless search functionality uses only the new PLM XML transform. Next, you must run the **qsearch_process_queue** utility to find and clean up all cycles (circular references). Also, check that all NX datasets have the **UGPART-BBOX** form.

> Note:
>
> Ensure that you have the typical Teamcenter environment variables set for all command line utilities.

Upgrade legacy transforms

1. Find all legacy transforms by running the **-list_legacy_transforms** option in the **qsearch_process_queue** utility in the following format:

   qsearch_process_queue -u={USER_ID} -p={PASSWORD} -g={GROUP} -list_legacy_transforms -outfile=c:\temp\legacytransforms.txt.

2. Run the **nxmgr_update_transforms** utility to upgrade the transforms in the following format:

   nxmgr_upgrade_transforms -u={USER_ID} -p={PASSWORD} -g={GROUP} -bypass=yes -i=c:\temp\legacytransforms.txt -upgrade_release=yes.

3. Run the **-list_legacy_transforms** option again to ensure that all transforms are upgraded.

Find and clean up cycles

1. Run the **find_cycles** option in the **qsearch_process_queue** utility to find all the cycles.

   To find cycles in the entire database, run **qsearch_process_queue -u={USER_ID} -p={PASSWORD} -g={GROUP} -find_cycles 2> c:\temp\list_of_cycles.txt**.

   Processing the entire database might take a long time. Instead, you can find cycles by item ID, list of item IDs, or item ID UIDs.

To find a cycle by item id, run **qsearch_process_queue -u={USER_ID} -p={PASSWORD} -g={GROUP} -find_cycles–item_id=AKT75443 2>c:\temp\AKT754432_cycles.txt**.

2. Clean up the cycles as per your organization's plan, considering that cleaning up cycles affects the content of the data.

Tip:

(Product-scoped cacheless searches only)

The **-check_structure_indexes** argument of the **qsearch_process_queue** utility flags cycles in a structure with an **index-box-check found cycle** report. This argument starts the utility at one or more specified items and works down the entire substructure. It finds apparently incorrect indexes and, at the same time, also finds cycles. Conversely, the **find_cycles** argument works up the substructure from the specified item; if you specify only top-level items, it returns no results. Also, the **find_cycles** argument takes *all* items by default.

In general, arguments with **structure** in the name interpret given item IDs as top-level items, and perform the relevant action on the entire structure below each item. Options without **structure** in the name simply perform the relevant action on each item in isolation. Therefore, the **-check_indexes** argument simply checks each item's index but does not find any cycles; the **-check_structure_indexes** argument also identifies any cycles.

To check for cycles under a top-level item, enter a command similar to the following example:

qsearch_process_queue -dump_substructure -item_id=*top-level-item-id(s)* **2>&1) | grep " item " | sed -e "s/ *item *//" | sort -u >***all_substructure_item_uids_file* **qsearch_process_queue -find_cycles -uid=@***all_substructure_item_uids_file*

The **-dump_substructure** argument produces an output similar to the following example, followed by a list of the item UIDs of all items in the structure. The **-find_cycles** then reads and processes the list of items.

```
# descend_structure: at level 1
item gsWFWitwAAgcRA
occThread gBUFWitwAAgcRA
occThread gBfFWitwAAgcRA
# descend_structure: at level 2
item g4SFWitwAAgcRA
item wESFWitwAAgcRA
occThread ARXFWitZAAgcRA
occThread ABRFWi9dAAgcRA
occThread gJZFWitwAAgcRA
occThread gJdFWitwAAgcRA
apn wYXFWivRAAgcRA
# descend_structure: at level 3
item wIaFWitwAAgcRA
item w4aFWitwAAgcRA
```

Check and generate missing UGPART-BBOX creation form (NX)

1. Run the **create_or_update_bbox_and_tso** utility in the following format to find all NX datasets that do not have the **UGPART-BBOX** form:

   create_or_update_bbox_and_tso -u={USER_ID} -p={PASSWORD} -g={GROUP} -translation_mode=NXBBOXFORM -mode=query -output_dir=c:\temp\ugpart_bbox.

2. Set the **QSEARCH_update_enabled** preference to **False** and then run the **run_tc_publishing_utility.bat** utility in the following format to generate the missing forms:

   run_tc_publishing_utility.bat -u=%tc_admin_user% -p=%tc_admin_pwd% -g=dba -i="!file!" -publish_tr=yes -record_pa=yes -log="!file!_refile.txt"

Generate TruShape (TSO) files (NX)

1. In NX, click **Assemblies**->**Site Standards**->**TruShape** tab.

   > Note:
   >
   > The use of refined search based on TSO files is optional for spatial searches.

2. Set the cell size and cell units, and click **Generate Component Shape Representations on Save**.

   TSO files are generated upon every save.

   > Note:
   >
   > The default cell size in NX is 0.0 mm. A cell size of 5.0 mm is recommended for initial testing. You can define your own cell size that is best for your design practice. Bear in mind that smaller the cell size, the longer the proximity search takes.

3. Run the **run_tc_publishing_utility.bat** utility in the following format to generate TSO files for the existing NX dataset:

   run_tc_publishing_utility.bat -u=%tc_admin_user% -p=%tc_admin_pwd% -g=dba -i="!file!" -publish_tr=yes -record_pa=yes -log="!file!_refile.txt"

> Note:
>
> The **run_tc_publishing_utility.bat** utility is supplied with NX and more information is provided in the Teamcenter Integration for NX documentation in the NX Help.

Enabling product-scoped cacheless search

Prior to running the product-scoped cacheless search, you must enable it by running the **qsearch_process_queue** command once for every database in the following format:

qsearch_process_queue -u={USER_ID} -p={PASSWORD} -g={GROUP} -enable_product_scoping.

This generates the **QSEARCH_filter_queries_by_product_scoping** preference in Teamcenter with a value of 1 (true). It also creates the following tables in the Teamcenter database:

- **PQSEARCHINDEXABLEITEM**
- **PQSEARCHINDEXABLEOCCTHREAD**
- **PQSEARCHINDEXABLEAPN**

> Note:
>
> If you want to disable the product scoped-cacheless search, run the **qsearch_process_queue -u={USER_ID} -p={PASSWORD} -g={GROUP} -disable_product_scoping -drop_table** command once for every database.

Make the product indexable

When you make a product indexable, all the components of the product are placed into a database table and tracked as updates are made. This improves performance by only searching the required data and not the complete database.

1. Run the **qsearch_process_queue -u={USER_ID} -p={PASSWORD} -g={GROUP} -clear_all_indexes** and **qsearch_process_queue -u={USER_ID} -p={PASSWORD} -g={GROUP} -clear_queue** commands to clear any existing index or queues.

2. Run the **qsearch_process_queue -u={USER_ID} -p={PASSWORD} -g={GROUP} -make_indexable -product={PRODUCT_ITEM_ID} 2> make_indexable.out** command to make the product indexable.

   If there are any missing product components, the **check_indexable** option lists it. You must index it again for it to be picked up.

3. Run the **qsearch_process_queue -u={USER_ID} -p={PASSWORD} -g={GROUP} -check_indexable -product={PRODUCT_ITEM_ID} 2> check_indexable.out** command to validate that the product is indexed.

4. Run the **qsearch_process_queue -u={USER_ID} -p={PASSWORD} -g={GROUP} -list_indexable -product={PRODUCT_ITEM_ID} 2> list_indexable.out** command to list all the contents of the indexed product.

Note:

- Run the **qsearch_process_queue -u={USER_ID} -p={PASSWORD} -g={GROUP} -make_non_indexable -product={PRODUCT_ITEM_ID}** command if you want to make the product non indexable.
- After you index a product, it is recommended that you regenerate the database statistics to improve the search performance, as described in the database documentation.

Creating bounding boxes

The Teamcenter bounding boxes are generated by populating the bounding box table in Teamcenter with the information from the **UGPART-BBOX**. The **create_or_update_bbox_and_tso** utility is used to generate the Teamcenter bounding boxes. This is a task is performed once, when you initially populate the database.

Creating bounding boxes for NX

The figure below illustrates the process of copying the information from the **UGPART-BBOX** form to the Teamcenter **PBOUNDINGBOX** table.

UGPART-BBOX Information

PUID	PXMIN	PXMAX	PYMIN	PYMAX	PZMIN	PZMAX
UID	2.183406	2.483865	-0.76347	-0.674	0.373581	0.528226

Teamcenter Bounding Box Information

PUID	PXMIN	PXMAX	PYMIN	PYMAX	PZMIN	PZMAX	RPARENTU	RPARENTC
UID	2.183406	2.483865	-0.76347	-0.674	0.373581	0.528226	UID	776

Before you start creating the bounding boxes, it is recommended that you do the following:

- Set up the environment variable to manage the syslogs, **TC_KEEP_SYSTEM_LOG=Y**.
- Set up a temporary directory, for example *TC_TMP_DIR=C:\temp*.

Query dataset UIDs for creating bounding boxes

1. Run the **create_or_update_bbox_and_tso** utility to query the database and find out how many bounding boxes need to be processed for the entire database.

Run the utility in this format: **create_or_update_bbox_and_tso -u={USER_ID} -p={PASSWORD} -g={GROUP} -translation_mode=NXBBOXTOBBOX -mode=query -product=%ITEM_ID% -output_dir=%TC_TMP_DIR%**.

> Note:
>
> This can be done only if the product has been made indexable.

2. Check the output directory for the report log file.

   Each report has 50,000 UID entries. Till the query is completed, reports are generated, and the report names are incremented by 1.

Create bounding boxes

1. Set the **QSEARCH_update_enabled** preference to **False**.

2. Run the **create_or_update_bbox_and_tso** utility in the following format:

   create_or_update_bbox_and_tso -u={USER_ID} -p={PASSWORD} -g={GROUP} -translation_mode=NXBBOXTOBBOX -mode=process -product={PRODUCT_ITEM_ID} .

   Using the **-product** option, the bounding boxes are generated by the specific product.

Delete bounding boxes

If you delete bounding boxes, the operation removes the bounding boxes, multi-bounding boxes, and the relations that link them together.

Depending on your requirement, you may run any of these commands:

- Run the **create_or_update_bbox_and_tso** command in the following format to delete all the bounding boxes in the database:

  create_or_update_bbox_and_tso -u={USER_ID} -p={PASSWORD} -g={GROUP} -delete_all_bboxes.

- Run the **create_or_update_bbox_and_tso** command in the following format to delete the bounding boxes for a specific dataset:

  create_or_update_bbox_and_tso -u={USER_ID} -p={PASSWORD} -g={GROUP} -mode=delete -dataset=hBH1IcSSoQUWxA.

- Run the **create_or_update_bbox_and_tso** command in the following format to delete bounding boxes for a specific product:

create_or_update_bbox_and_tso -u={USER_ID} -p={PASSWORD} -g={GROUP} -mode=delete -product={PRODUCT_ITEM_ID}.

Creating bounding boxes and TSO files for JT

To create bounding boxes and TSO files, you must first load the JT file, extract the bounding box information, add that to the Teamcenter bounding box table, and then create the TSO as a named reference to the JT dataset.

Do the following:

- Set the **TC_KEEP_SYSTEM_LOG** environment variable to **Y** to manage the syslog.
- Create a temporary directory similar to *TC_TMP_DIR=C:\temp*.

Query dataset UIDs for creating bounding boxes

1. Run the **create_or_update_bbox_and_tso** utility in the following format to query the database and find out how many bounding boxes need to be processed for the entire database:

   create_or_update_bbox_and_tso -u={USER_ID} -p={PASSWORD} -g={GROUP} -translation_mode= JTTOTSO -mode=query -product=%ITEM_ID% -output_dir=%TC_TMP_DIR%.

   > Note:
   >
   > This can be done only if the product specified in the command line in the **-product** option has been made indexable.

2. Check the output directory for the report log file.

Create bounding boxes

1. Set the **QSEARCH_update_enabled** preference to **False**.
2. Run the **create_or_update_bbox_and_tso** utility in this format:

   create_or_update_bbox_and_tso -u={USER_ID} -p={PASSWORD} -g={GROUP} -translation_mode=JTTOBBOX -mode=process -product={PRODUCT_ITEM_ID}.

   Using the **-product** option, the bounding boxes are generated by the specific product.

Delete bounding boxes

If you delete bounding boxes, the operation removes the bounding boxes, multi-bounding boxes, and the relations that link them together.

Depending on your requirement, you may run any of these commands:

- Run the **create_or_update_bbox_and_tso** command in the following format to delete all the bounding boxes in the database:

  create_or_update_bbox_and_tso -u=*Tc-admin-user* -p=*password* -g=*group* **-delete_all_bboxes**.

- Run the **create_or_update_bbox_and_tso** command in the following format to delete the bounding boxes for a specific dataset:

  create_or_update_bbox_and_tso -u=*Tc-admin-user* -p=*password* -g=*group* **-mode=delete -dataset=hBH1IcSSoQUWxA** .

- Run the **create_or_update_bbox_and_tso** command in the following format to delete bounding boxes for a specific product:

  create_or_update_bbox_and_tso -u=*Tc-admin-user* -p=*password* -g=*group* **-mode=delete -product={PRODUCT_ITEM_ID}**.

Creating a spatial index

After generating the bounding boxes from NX or JT, you can generate the index using the **qsearch_process_queue** utility. This provides the option to create, update, or delete the index boxes.

Populating the queue

After you create the bounding boxes you populate the queue. The queue entries are Very Large Arrays (VLA) and contain the item, the occurrence thread, or the appearance path node that is related to the particular CAD geometry that the bounding box represents.

Before you process any updates, ensure the following:

- **TC_KEEP_SYSTEM_LOG** environment variable is **Yes**.
- **QSEARCH_update_enabled** preference is **True**.
- **QSEARCH_foreground_processing_halted** preference is **True**.
- **QSEARCH_force_simplify_to_one_box** preference is **True**.
- Set a temporary directory, for example *TC_TMP_DIR=C:\temp\create_bbox_uids*.

Populate the queue by running the **qsearch_process_queue** command in the following format:

qsearch_process_queue -u={USER_ID} -p={PASSWORD} -g={GROUP} -force_queue_product_substructure_update -product={PRODUCT_ITEM_ID}.

This updates the queue with all the necessary objects related to the product.

> Note:
>
> Populating the queue generates entries into the **PQSEARCHINDEXUPDATEENTRY** table for each bounding box.

Listing the queue

After you populate a queue, list the queue by running the **qsearch_process_queue** utility in the following format:

qsearch_process_queue -u={USER_ID} -p={PASSWORD} -g={GROUP} -list_queue –verbose 2> %TC_TMP_DIR%\list_queue.txt

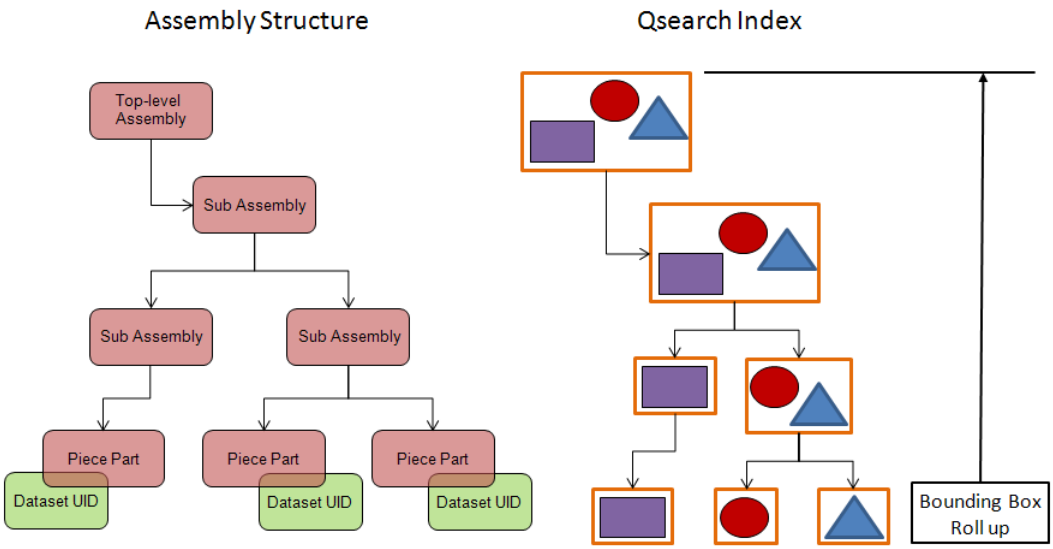
The output is similar to this:

```
Queue contains 7 objects in 1 entry entry = 00007445 = gQM14MFEoQUWxA
object = 00007446 = BpF1IcSSoQUWxA, type = 1 (itemType), date = 18-Mar-2011
10:33:40, box-delta = 0, state = 0 (unprocessed), session = 00007444 =
AII14MFEoQUWxA, is-run-time = FALSE Item "15259388-BELT ASM-R/SEAT CTR BKL"
object = 00007447 = BpH1IcSSoQUWxA, type = 1 (itemType), date = 18-Mar-2011
10:33:40, box-delta = 0, state = 0 (unprocessed), session = 00007444 =
AII14MFEoQUWxA, is-run-time = FALSE Item "11561331-BOLT ASM"
```

Processing the queue

When the queue is populated, the next step is to generate the index. This populates the **PQSEARCHINDEX** table with the spatial index information for items, occurrence threads, and appearance path nodes that are impacted by the bounding boxes. The spatial index is the rolled up bounding box of all children and all their revisions as described in *Understanding spatial search indexes*.

In the example below if we start from the bottom and take the bounding box of the dataset (piece part), its transform then rolls up to its subassembly. We then continue rolling up the assembly structure and populating the information for each item. The following example uses the **QSEARCH_force_simplify_to_one_box** method.



The entries of the **PBOUNDINGBOX** table are as follows:

PUID	PX_MIN	PY_MIN	PZ_MIN	PX_MAX	PY_MAX	PZ_MAX	RPARENTU	RPARENTC
AJJ15AZgoU2mFA	0	0	0	0.1	0.1	0.1	g4P15AZgoU2mFA	57
ANC15AZgoU2mFA	-0.025	-0.025	0	0.025	0.025	0.025	AJL15AZgoU2mFA	57
ANL15AZgoU2mFA	-0.025	-0.025	0	0.025	0.025	0.1	ANE15AZgoU2mFA	57

The entries of the **PQSEARCHINDEX** table are as follows:

PUID	PXMIN	PYMIN	PZMIN	PXMAX	PYMAX	PZMAX	PKEY	PTYPE
g8O15As9oU2mFA	-0.025	-0.025	0	0.025	0.025	0.025	xJ1Z8r6oU2mFAAAAAAAAAAAAAA	1
wAA15As9oU2mFA	-0.025	-0.025	0	0.1	0.1	0.1	RBJ1Z8r6oU2mFAAAAAAAAAAAAAA	1
wAG15As9oU2mFA	-0.025	-0.025	0	0.1	0.1	0.1	RbF1Z8r6oU2mFAAAAAAAAAAAAAA	0
wAI15As9oU2mFA	-0.025	-0.025	0	0.025	0.025	0.025	CgA1Z8r6oU2mFAAAAAAAAAAAAAA	0
wEA15As9oU2mFA	-0.025	-0.025	0	0.1	0.1	0.1	BnN1Z8r6oU2mFAAAAAAAAAAAAAA	1
wEG15As9oU2mFA	-0.025	-0.025	0	0.1	0.1	0.1	RjP1Z8r6oU2mFAAAAAAAAAAAAAA	0
g4E15As9oU2mFA	-0.025	-0.025	0	0.025	0.025	0.025	h_G1Z8r6oU2mFAAAAAAAAAAAAAA	1
wEL15As9oU2mFA	-0.025	-0.025	0	0.1	0.1	0.1	BbP1Z8r6oU2mFAAAAAAAAAAAAAA	1
g4G15As9oU2mFA	0	0	0	0.1	0.1	0.1	RNN1Z8r6oU2mFAAAAAAAAAAAAAA	1
g4I15As9oU2mFA	-0.025	-0.025	0	0.025	0.025	0.1	RiH1Z8r6oU2mFAAAAAAAAAAAAAA	1
g8D15As9oU2mFA	-0.025	-0.025	0	0.025	0.025	0.1	hSO1Z8r6oU2mFAAAAAAAAAAAAAA	0
g8F15As9oU2mFA	0	0	0	0.1	0.1	0.1	x5D1Z8r6oU2mFAAAAAAAAAAAAAA	0
g8H15As9oU2mFA	-0.025	-0.025	0	0.025	0.025	0.025	x$C1Z8r6oU2mFAAAAAAAAAAAAAA	0

After processing the queue, it is recommended that you generate the database again to improve search performance.

Process all updates

- Run the **qsearch_process_queue** utility in the following format to process all the entries in the queue:

  qsearch_process_queue -u={USER_ID} -p={PASSWORD} -g={GROUP} -process_queue.

Process by UID

- Run the **qsearch_process_queue** utility in the following format to process all multiple UIDs using a comma-separated list:

qsearch_process_queue -u={USER_ID} -p={PASSWORD} -g={GROUP} -process_queue -uid=0$0F56myQyYmUC.

Process by item id list

- Run the **qsearch_process_queue** utility in the following format to process multiple Item ID's using a comma-separated list:

  qsearch_process_queue -u={USER_ID} -p={PASSWORD} -g={GROUP} -process_queue -item_id=@list_of_items.txt.

Deleting an index

If you want to delete an index in your database, use the **qsearch_process_queue** command in the following format:

qsearch_process_queue -u={USER_ID} -p={PASSWORD} -g={GROUP} -clear_product_indexes -product=%ITEM_ID%.

Setting up live background updates

The live background updates can be done by using Dispatcher, **cron** jobs, or using a Windows service.

If you use Dispatcher, you must set up to manage the daily updates of the spatial index. Set its timeout period to a duration long enough for the application to process **QSearchProcessQueue** requests.

Setting up Dispatcher to manage updates (NX)

To set up Dispatcher for managing updates, ensure that you have:

- Installed the **QSearchProcessQueue** Dispatcher translator, using TEM as shown in the following figure.

- Set the **QSEARCH_update_enabled** preference to **True**.
- Set the **QS_BBOX_GENERATION_FROM_NX_ENABLED** preference to **True**.

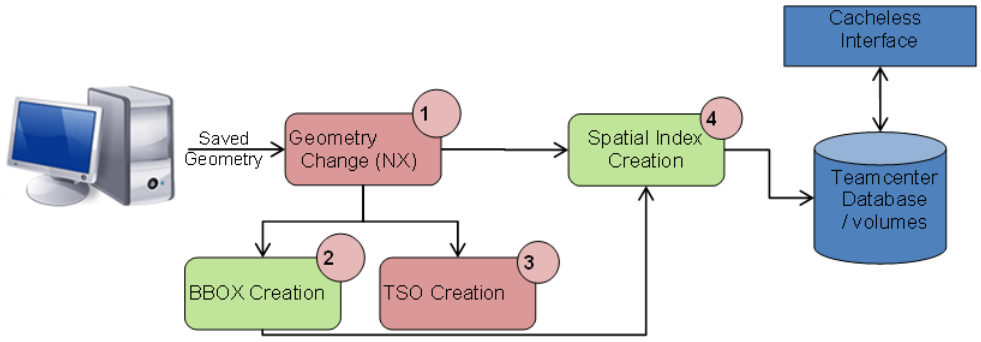
The following figure illustrates the process of how an NX dataset is created, saved, and the spatial index, updated with Dispatcher setup.

1. When a geometry is saved, triggers are automatically set for creating bounding boxes and TSO within the NX application. It also triggers a structure update to roll up the bounding box information to its parent structure.

2. BBOX is created or updated.

3. TSO is created or updated.

4. The creation of spatial indexes is triggered in the form of a Dispatcher request.



Note:

If you would like to configure the **QsearchProcessQueue** Dispatcher translator as a recurring request, set the following preferences as required:

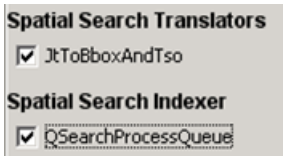
- **QSEARCH_queue_recurring_background_updates**
- **QSEARCH_recurring_background_update_survival_time**
- **QSEARCH_recurring_background_update_interval**

If you want to reduce the objects getting added to the queue and also reduce the number of Dispatcher requests, specify the item types you want to avoid in the **QSEARCH_types_to_avoid_processing** preference.

Setting up Dispatcher to manage updates (JT)

To set up Dispatcher to manage updates, ensure that you have:

- Installed the following Dispatcher translators, using TEM, as shown in the following figure:
  - **JtToBboxAndTso**
  - **QSearchProcessQueue**

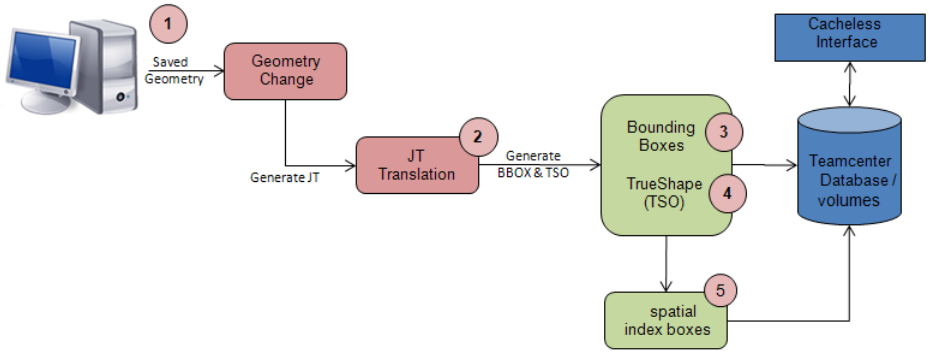


> Note:
>
> It is recommended that you use only one translator for the spatial search indexer.

- Set the **QSEARCH_update_enabled** preference to **True** if you want to generate TSO files.

The following figure illustrates the process of how a JT dataset is created, saved, and the spatial index updated with Dispatcher setup.

1. When a geometry is saved, triggers are automatically set for creation of a JT file.
2. The JT file is created and it requests the Dispatcher to generate the bounding boxes and TSO files.
3. The JT is loaded and the bounding box information is populated in the bounding box table. The tables in the database are populated with the bounding box information.
4. TSO files are generated and attached as a named reference.
5. A Dispatcher request is triggered, and a spatial index is created.



Note:

If you would like to configure the **QsearchProcessQueue** Dispatcher translator as a recurring request, set the following preferences as required:

- **QSEARCH_queue_recurring_background_updates**
- **QSEARCH_recurring_background_update_survival_time**
- **QSEARCH_recurring_background_update_interval**

If you want to reduce the objects getting added to the queue and also reduce the number of dispatcher requests, specify the item types you want to avoid in the **QSEARCH_types_to_avoid_processing** preference.

Setting up cron jobs

If you want to generate the spatial indexes using a **cron** job script, set the **QSEARCH_dispatcher_not_available** preference to **True**.

Configuring cacheless search in a Multi-Site Collaboration environment

If you work in a Multi-Site Collaboration environment, you can make cacheless spatial and attribute searches of products or programs that are owned or replicated at remote sites. Where appropriate, local and remote objects are incorporated in the search results.

If you enable this feature:

- Teamcenter checks access permissions to the remote BOM lines. If you do not have the necessary permissions, you cannot import the affected remote lines. If appropriate, you can review the search results and import only the lines to which you have access permissions.

- Identical revision and variant rules must be defined at all sites or searches may return inconsistent results. Also, the content of these revision rules and variant rules must always be consistent across all the sites.

- The content of saved queries must be consistent across all sites or searches may return inconsistent results.

- You can only initiate the search in Design Context, not in Structure Manager or Multi-Structure Manager.

To configure this feature, you must set the following preferences:

- **QS_remote_master_site**

  This preference lists the remote sites and the product or program that can be searched at each of those sites. It must be defined in a format similar to the following example:

  DEFAULT:site1
  Product2:master_site2
  Product3:master_site3

  The default site is always searched.

- **QS_remote_master_site_override**

  This preference allows the user to define a preferred remote site to make cacheless searches. It must be defined in a format similar to the following example:

  Product3:master_site3

  Whenever you add a new product or assembly to remotely search in Design Context, the administrator must update this preference with the new product and its owning master site at all sites in the Multi-Site Collaboration network.

You do not need to manually configure IDSM to use remote cacheless searches. When you install or upgrade the system using Teamcenter Environment Manager (TEM), it installs the callbacks necessary for remote cacheless search. Each site participating in remote cacheless search must install or upgrade the database using the same version of TEM.

3. Working with full database cacheless search

Preparing your data

Before you use cacheless search on an existing database, you must run a few checks and clean the data.

To begin with, you must upgrade all existing legacy transforms, since the cacheless search functionality uses only the new PLM XML transform. Next, you must run the **qsearch_process_queue** utility to find and clean up all cycles (circular references). Also, check that all NX datasets have the **UGPART-BBOX** form.

> Note:
>
> Ensure that you have the typical Teamcenter environment variables set for all command line utilities.

Upgrade legacy transforms

1. Find all legacy transforms by running the **-list_legacy_transforms** option in the **qsearch_process_queue** utility in the following format:

   qsearch_process_queue -u={USER_ID} -p={PASSWORD} -g={GROUP} -list_legacy_transforms -outfile=c:\temp\legacytransforms.txt.

2. Run the **nxmgr_update_transforms** utility to upgrade the transforms in the following format:

   nxmgr_upgrade_transforms -u={USER_ID} -p={PASSWORD} -g={GROUP} -bypass=yes -i=c:\temp\legacytransforms.txt -upgrade_release=yes.

3. Run the **-list_legacy_transforms** option again to ensure that all transforms are upgraded.

Find and clean up cycles

1. Run the **find_cycles** option in the **qsearch_process_queue** utility to find all the cycles.

   To find cycles in the entire database, run **qsearch_process_queue -u={USER_ID} -p={PASSWORD} -g={GROUP} -find_cycles 2> c:\temp\list_of_cycles.txt**.

   Processing the entire database might take a long time. Instead, you can find cycles by item ID, list of item IDs, or item ID UIDs.

To find a cycle by item id, run **qsearch_process_queue -u={USER_ID} -p={PASSWORD} -g={GROUP} -find_cycles–item_id=AKT75443 2>c:\temp\AKT754432_cycles.txt**.

2. Clean up the cycles as per your organization's plan, considering that cleaning up cycles affects the content of the data.

Tip:

(Product-scoped cacheless searches only)

The **-check_structure_indexes** argument of the **qsearch_process_queue** utility flags cycles in a structure with an **index-box-check found cycle** report. This argument starts the utility at one or more specified items and works down the entire substructure. It finds apparently incorrect indexes and, at the same time, also finds cycles. Conversely, the **find_cycles** argument works up the substructure from the specified item; if you specify only top-level items, it returns no results. Also, the **find_cycles** argument takes *all* items by default.

In general, arguments with **structure** in the name interpret given item IDs as top-level items, and perform the relevant action on the entire structure below each item. Options without **structure** in the name simply perform the relevant action on each item in isolation. Therefore, the **-check_indexes** argument simply checks each item's index but does not find any cycles; the **-check_structure_indexes** argument also identifies any cycles.

To check for cycles under a top-level item, enter a command similar to the following example:

qsearch_process_queue -dump_substructure -item_id=*top-level-item-id(s)* **2>&1) | grep " item " | sed -e "s/ *item *//" | sort -u >***all_substructure_item_uids_file* **qsearch_process_queue -find_cycles -uid=@***all_substructure_item_uids_file*

The **-dump_substructure** argument produces an output similar to the following example, followed by a list of the item UIDs of all items in the structure. The **-find_cycles** then reads and processes the list of items.

```
# descend_structure: at level 1
item gsWFWitwAAgcRA
occThread gBUFWitwAAgcRA
occThread gBfFWitwAAgcRA
# descend_structure: at level 2
item g4SFWitwAAgcRA
item wESFWitwAAgcRA
occThread ARXFWitZAAgcRA
occThread ABRFWi9dAAgcRA
occThread gJZFWitwAAgcRA
occThread gJdFWitwAAgcRA
apn wYXFWivRAAgcRA
# descend_structure: at level 3
item wIaFWitwAAgcRA
item w4aFWitwAAgcRA
```

Check and generate missing UGPART-BBOX creation form (NX)

1. Run the **create_or_update_bbox_and_tso** utility in the following format to find all NX datasets that do not have the **UGPART-BBOX** form:

   create_or_update_bbox_and_tso -u={USER_ID} -p={PASSWORD} -g={GROUP} -translation_mode=NXBBOXFORM -mode=query -output_dir=c:\temp\ugpart_bbox.

2. Set the **QSEARCH_update_enabled** preference to **False** and then run the **run_tc_publishing_utility.bat** utility in the following format to generate the missing forms:

   run_tc_publishing_utility.bat -u=%tc_admin_user% -p=%tc_admin_pwd% -g=dba -i="!file!" -publish_tr=yes -record_pa=yes -log="!file!_refile.txt"

Generate TruShape (TSO) files (NX)

1. In NX, click **Assemblies**->**Site Standards**->**TruShape** tab.

   > Note:
   >
   > The use of refined search based on TSO files is optional for spatial searches.

2. Set the cell size and cell units, and click **Generate Component Shape Representations on Save**.

   TSO files are generated upon every save.

   > Note:
   >
   > The default cell size in NX is 0.0 mm. A cell size of 5.0 mm is recommended for initial testing. You can define your own cell size that is best for your design practice. Bear in mind that smaller the cell size, the longer the proximity search takes.

3. Run the **run_tc_publishing_utility.bat** utility in the following format to generate TSO files for the existing NX dataset:

   run_tc_publishing_utility.bat -u=%tc_admin_user% -p=%tc_admin_pwd% -g=dba -i="!file!" -publish_tr=yes -record_pa=yes -log="!file!_refile.txt"

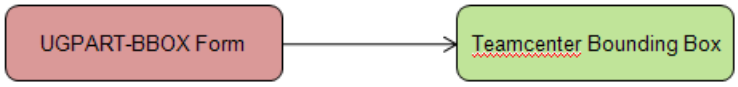
> Note:
>
> The **run_tc_publishing_utility.bat** utility is supplied with NX and more information is provided in the Teamcenter Integration for NX documentation in the NX Help.

Creating bounding boxes

The Teamcenter bounding boxes are generated by populating the bounding box table in Teamcenter with the information in the **UGPART-BBOX**. The **create_or_update_bbox_and_tso** utility is used to generate the Teamcenter bounding boxes. This task is performed once, when you initially populate the database.

Creating bounding boxes for NX

The figure below illustrates the process of copying the information from the UGPART-BBOX form to the Teamcenter bounding box table **PBOUNDINGBOX** table.



UGPART-BBOX Information

PUID	PXMIN	PXMAX	PYMIN	PYMAX	PZMIN	PZMAX
UID	2.183406	2.483865	-0.76347	-0.674	0.373581	0.528226

Teamcenter Bounding Box Information

PUID	PXMIN	PXMAX	PYMIN	PYMAX	PZMIN	PZMAX	RPARENTU	RPARENTC
UID	2.183406	2.483865	-0.76347	-0.674	0.373581	0.528226	UID	776

Before you start creating the bounding boxes, it is recommended that you do the following:

- Set up the environment variable to manage the syslogs, **TC_KEEP_SYSTEM_LOG=Y**.
- Set up a temporary directory, for example, *TC_TMP_DIR=C:\temp*.

Query dataset UIDs for creating bounding boxes

1. Run the **create_or_update_bbox_and_tso** utility in the following format to query the database and find out how many bounding boxes need to be processed for the entire database.

   create_or_update_bbox_and_tso -u={USER_ID} -p={PASSWORD} -g={GROUP} -translation_mode=NXBBOXTOBBOX -mode=query -output_dir=%TC_TMP_DIR%.

   This generates the **createOrUpdateBBoxAndtso_report{%ID%_0.log** file.

2. Check the output directory for the report log file.

   Each report has 50,000 UID entries. Till the query is completed, reports are generated and the report names are incremented by 1.

An ideal report reads like this:

```
Following NX Datasets need to update Bounding Boxes:
x5D1IcSSoQUWxA
RNI1IcSSoQUWxA
htB1IcSSoQUWxA
xpH1IcSSoQUWxA
h5M1IcSSoQUWxA
RiN1IcSSoQUWxA
hBH1IcSSoQUWxA
```

Create bounding boxes

1. Set the **QSEARCH_update_enabled** preference to **False**.

2. Run the **create_or_update_bbox_and_tso -u={USER_ID} -p={PASSWORD} -g={GROUP} -translation_mode=NXBBOXTOBBOX -mode=process -object_list=%TC_TMP_DIR%\createOrUpdateBBoxAndtso_report{%ID%}_0.log** command.

   > Note:
   >
   > You can run the command only if the product has been made indexable when using product scoped search.

3. Run the **create_or_update_bbox_and_tso -u={USER_ID} -p={PASSWORD} -g={GROUP} -translation_mode=NXBBOXTOBBOX -mode=query –scope=ALL -output_dir=%TC_TMP_DIR%** command to query all datasets of all indexable products.

4. Validate the bounding boxes by sending an assembly to Structure Manager, adding the **Bounding Boxes** column, and then expanding the assembly to a level that has geometry.

   The column will be populated with the bounding box information.

Delete bounding boxes

If you delete bounding boxes, the operation removes the bounding boxes, multi-bounding boxes, and the relations that link them together.

Depending on your requirement, you may run any of these commands:

- Run the **create_or_update_bbox_and_tso -u={USER_ID} -p={PASSWORD} -g={GROUP} -delete_all_bboxes** command to delete all the bounding boxes in the database.

- Run the **create_or_update_bbox_and_tso -u={USER_ID} -p={PASSWORD} -g={GROUP} -mode=delete -dataset=hBH1IcSSoQUWxA** command to delete the bounding boxes for a specific dataset.

- Run the **create_or_update_bbox_and_tso -u={USER_ID} -p={PASSWORD} -g={GROUP} -mode=delete -product={PRODUCT_ITEM_ID}** command to delete bounding boxes for a specific product.

Creating bounding boxes and TSO files for JT

To create bounding boxes and TSO files, you must first load the JT file, extract the bounding box information, add that to the Teamcenter bounding box table, and then create the TSO as a named reference to the JT dataset. To start with, do the following:

- Set the **TC_KEEP_SYSTEM_LOG** environment variable to **Y** to manage the syslog.

- Create a temporary directory, for example, *TC_TMP_DIR=C:\temp*.

Query dataset UIDs for creating bounding boxes and TSO

Depending on your requirement, run one or all the utilities to query the database to find out how many UIDs will need to be processed.

1. Run the **create_or_update_bbox_and_tso –u={USER_ID} -p={PASSWORD} -g={GROUP} -translation_mode=JTTOBBOX+JTTOTSO -mode=query -output_dir=%TC_TMP_DIR%** utility to query the database to find out how many UIDs will need to be processed for the complete database (both bounding boxes and TSO files).

2. Run the **create_or_update_bbox_and_tso –u={USER_ID} -p={PASSWORD} -g={GROUP} -translation_mode=JTTOBBOX -mode=query -output_dir=%TC_TMP_DIR%** utility to query the database to find out how many UIDs will need to be processed for the bounding boxes only.

3. Run the **create_or_update_bbox_and_tso -u={USER_ID} -p={PASSWORD} -g={GROUP -translation_mode=JTTOTSO -mode=query -output_dir=%TC_TMP_DIR%** utility to query the database to find out how many UIDs will need to be processed for the TSO files only.

Create bounding boxes and TSO files

1. Set the **QSEARCH_update_enabled** preference to **False**.

2. Run the **create_or_update_bbox_and_tso** -u=*Tc-admin-user* -p=*password* -g=*group* **-translation_mode=JTTOBBOX+JTTOTSO -mode=process -object_list=%TC_TMP_DIR%\createOrUpdateBBoxAndtso_report{%ID%}_.lo** utility to create both entities, the bounding boxes and the TSO files.

   In a multisite environment, the **create_or_update_bbox_and_tso** utility creates the TSO files only for datasets that belong to the owning site. It does not create the TSO files for replica datasets.

3. Run the **create_or_update_bbox_and_tso** -u=*Tc-admin-user* -p=*password* -g=*group* **-translation_mode=JTTOBBOX -mode=process -object_list=%TC_TMP_DIR% \createOrUpdateBBoxAndtso_report{%ID%}_.log** utility to create only the bounding box.

4. Run the **create_or_update_bbox_and_tso** -u=*Tc-admin-user* -p=*password* -g=*group* **-translation_mode=JTTOTSO -mode=process -object_list=%TC_TMP_DIR% \createOrUpdateBBoxAndtso_report{%ID%}_.log** utility to create only the TSO files.

5. Validate the bounding boxes by sending an assembly to Structure Manager, adding the **Bounding Boxes** column, and then expanding the assembly to a level that has geometry.

   The column will be populated with the bounding box information.

Delete bounding boxes

If you delete bounding boxes, the operation removes the bounding boxes, multi-bounding boxes, and the relations that link them together.

Depending on your requirement, you may run any of these commands:

- Run the **create_or_update_bbox_and_tso** -u=*Tc-admin-user* -p=*password* -g=*group* **-delete_all_bboxes** command to delete all the bounding boxes in the database.

- Run the **create_or_update_bbox_and_tso** -u=*Tc-admin-user* -p=*password* -g=*group* **-mode=delete -dataset=hBH1IcSSoQUWxA** command to delete the bounding boxes for a specific dataset.

- Run the **create_or_update_bbox_and_tso** -u=*Tc-admin-user* -p=*password* -g=*group* **-mode=delete -product={PRODUCT_ITEM_ID}** command to delete bounding boxes for a specific product.

Creating a spatial index

After generating the bounding boxes from NX or JT, you can generate the index using the **qsearch_process_queue** utility. This provides the option to create, update, or delete the index boxes.

Populating the queue

After you create the bounding boxes you populate the queue. The queue entries are Very Large Arrays (VLA) and contain the item, the occurrence thread, or the reappearance path node that is related to the particular CAD geometry that the bounding box represents.

Before you process any updates, ensure the following:

- **TC_KEEP_SYSTEM_LOG** environment variable is **Yes**.

- **QSEARCH_update_enabled** preference is **True**.

- **QSEARCH_foreground_processing_halted** preference is **True**.
- **QSEARCH_force_simplify_to_one_box** preference is **True**.
- Set a temporary directory, for example, *TC_TMP_DIR=C:\temp\create_bbox_uids*.

> Note:
>
> Populating the queue generates entries into the **PQSEARCHINDEXUPDATEENTRY** table for each bounding box.

Processing all updates

This processes all updates of all bounding box UIDs, including the ones that already have index boxes). Run the **qsearch_process_queue** command once in the following format:

qsearch_process_queue -u={USER_ID} -p={PASSWORD} -g={GROUP} -force_queue_all_possible_updates.

Processing necessary updates

When you want to process only the necessary updates (where index boxes are not generated yet) or only add the bounding box UIDs of those that have not been processed, run the **qsearch_process_queue** command in the following format:

qsearch_process_queue -u={USER_ID} -p={PASSWORD} -g={GROUP} -force_queue_all_necessary_updates.

Listing the queue

After you populate a queue, list the queue by running the **qsearch_process_queue** utility in the following format:

qsearch_process_queue -u={USER_ID} -p={PASSWORD} -g={GROUP} -list_queue –verbose 2> %TC_TMP_DIR%\list_queue.txt

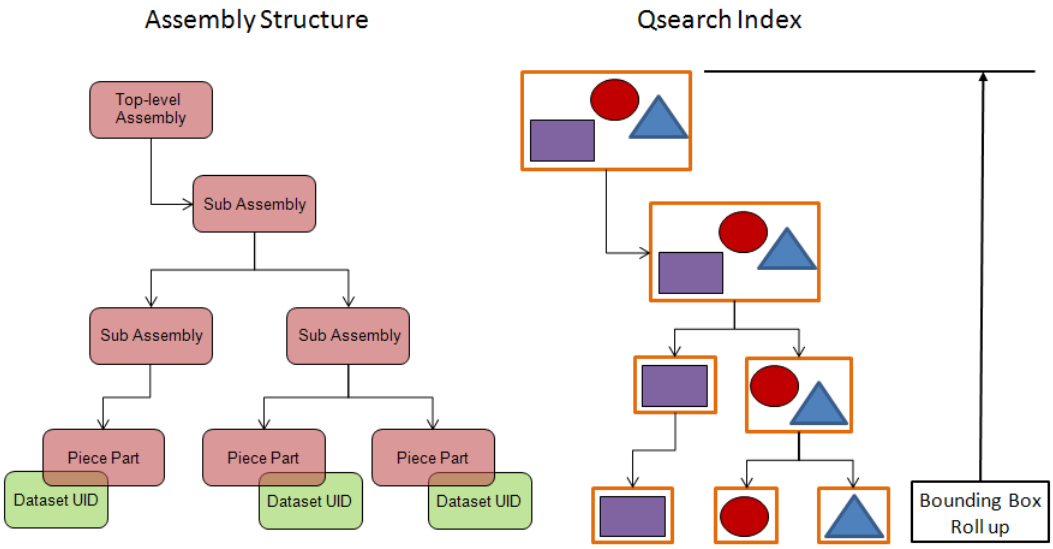
The output is similar to this:

```
Queue contains 7 objects in 1 entry entry = 00007445 = gQM14MFEoQUWxA
object = 00007446 = BpF1IcSSoQUWxA, type = 1 (itemType), date = 18-Mar-2011
10:33:40, box-delta = 0, state = 0 (unprocessed), session = 00007444 =
AII14MFEoQUWxA, is-run-time = FALSE Item "15259388-BELT ASM-R/SEAT CTR BKL"
object = 00007447 = BpH1IcSSoQUWxA, type = 1 (itemType), date = 18-Mar-2011
10:33:40, box-delta = 0, state = 0 (unprocessed), session = 00007444 =
AII14MFEoQUWxA, is-run-time = FALSE Item "11561331-BOLT ASM"
```

Processing the queue

When the queue is populated, the next step is to generate the index. This populates the **PQSEARCHINDEX** table with the spatial index information for items, occurrence threads, and appearance path nodes that are impacted by the bounding boxes. The spatial index is the rolled up bounding box of all children and all their revisions as described in *Understanding spatial search indexes*.

In the example below if we start from the bottom and take the bounding box of the dataset (piece part), its transform then rolls up to its subassembly. We then continue rolling up the assembly structure and populating the information for each item. The following example uses the **QSEARCH_force_simplify_to_one_box** method.



The entries of the **PBOUNDINGBOX** table are as follows:

PUID	PX_MIN	PY_MIN	PZ_MIN	PX_MAX	PY_MAX	PZ_MAX	RPARENTU	RPARENTC
AJJ15AZgoU2mFA	0	0	0	0.1	0.1	0.1	g4P15AZgoU2mFA	57
ANC15AZgoU2mFA	-0.025	-0.025	0	0.025	0.025	0.025	AJL15AZgoU2mFA	57
ANL15AZgoU2mFA	-0.025	-0.025	0	0.025	0.025	0.1	ANE15AZgoU2mFA	57

The entries of the **PQSEARCHINDEX** table are as follows:

PUID	PXMIN	PYMIN	PZMIN	PXMAX	PYMAX	PZMAX	PKEY	PTYPE
g8O15As9oU2mFA	-0.025	-0.025	0	0.025	0.025	0.025	xJ1Z8r6oU2mFAAAAAAAAAAAAAA	1
wAA15As9oU2mFA	-0.025	-0.025	0	0.1	0.1	0.1	RBJ1Z8r6oU2mFAAAAAAAAAAAAAA	1
wAG15As9oU2mFA	-0.025	-0.025	0	0.1	0.1	0.1	RbF1Z8r6oU2mFAAAAAAAAAAAAAA	0
wAI15As9oU2mFA	-0.025	-0.025	0	0.025	0.025	0.025	CgA1Z8r6oU2mFAAAAAAAAAAAAAA	0
wEA15As9oU2mFA	-0.025	-0.025	0	0.1	0.1	0.1	BnN1Z8r6oU2mFAAAAAAAAAAAAAA	1
wEG15As9oU2mFA	-0.025	-0.025	0	0.1	0.1	0.1	RjP1Z8r6oU2mFAAAAAAAAAAAAAA	0
g4E15As9oU2mFA	-0.025	-0.025	0	0.025	0.025	0.025	h_G1Z8r6oU2mFAAAAAAAAAAAAAA	1
wEL15As9oU2mFA	-0.025	-0.025	0	0.1	0.1	0.1	BbP1Z8r6oU2mFAAAAAAAAAAAAAA	1
g4G15As9oU2mFA	0	0	0	0.1	0.1	0.1	RNN1Z8r6oU2mFAAAAAAAAAAAAAA	1
g4I15As9oU2mFA	-0.025	-0.025	0	0.025	0.025	0.1	RiH1Z8r6oU2mFAAAAAAAAAAAAAA	1
g8D15As9oU2mFA	-0.025	-0.025	0	0.025	0.025	0.1	hSO1Z8r6oU2mFAAAAAAAAAAAAAA	0
g8F15As9oU2mFA	0	0	0	0.1	0.1	0.1	x5D1Z8r6oU2mFAAAAAAAAAAAAAA	0
g8H15As9oU2mFA	-0.025	-0.025	0	0.025	0.025	0.025	x$C1Z8r6oU2mFAAAAAAAAAAAAAA	0

After processing the queue, it is recommended that you generate the database again to improve search performance.

Process all updates

- Run the **qsearch_process_queue** utility in the following format to process all the entries in the queue:

  qsearch_process_queue -u={USER_ID} -p={PASSWORD} -g={GROUP} -process_queue.

Process by UID

- Run the **qsearch_process_queue** utility in the following format to process all multiple UIDs using a comma-separated list:

  qsearch_process_queue -u={USER_ID} -p={PASSWORD} -g={GROUP} -process_queue -uid=0$0F56myQyYmUC.

Process by item id list

- Run the **qsearch_process_queue** utility in the following format to process multiple Item ID's using a comma-separated list:

  qsearch_process_queue -u={USER_ID} -p={PASSWORD} -g={GROUP} -process_queue -item_id=@list_of_items.txt.

Deleting all indexes

If you want to delete all indexes from your database, use the **qsearch_process_queue** utility in the following format:

qsearch_process_queue -u={USER_ID} -p={PASSWORD} -g={GROUP} -clear_all_indexes.

Setting up live background updates

The live background updates can be done by using Dispatcher, **cron** jobs, or using a Windows service.

If you use Dispatcher, you must set up to manage the daily updates of the spatial index. Set its timeout period to a duration long enough for the application to process **QSearchProcessQueue** requests.

Setting up Dispatcher to manage updates (NX)

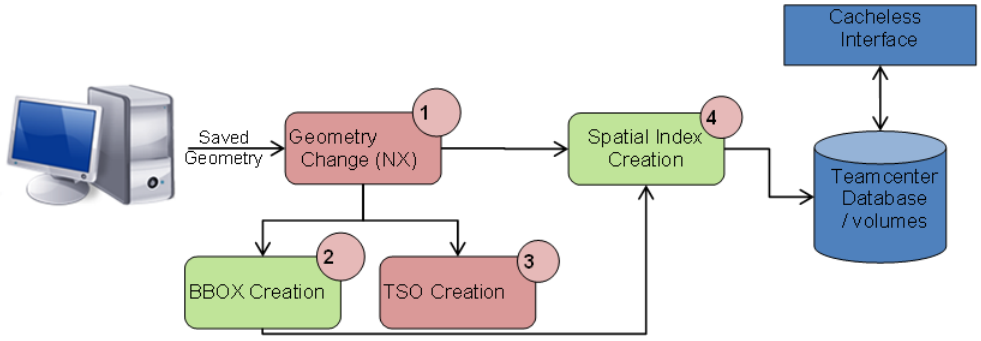
To set up Dispatcher for managing updates, ensure that you have:

- Installed the **QSearchProcessQueue** Dispatcher translator, using TEM as shown in the following figure.

- Set the **QSEARCH_update_enabled** preference to **True**.
- Set the **QS_BBOX_GENERATION_FROM_NX_ENABLED** preference to **True**.

The following figure illustrates the process of how an NX dataset is created, saved, and the spatial index, updated with Dispatcher setup.

1. When a geometry is saved, triggers are automatically set for creating bounding boxes and TSO within the NX application. It also triggers a structure update to roll up the bounding box information to its parent structure.
2. BBOX is created or updated.
3. TSO is created or updated.
4. The creation of spatial indexes is triggered in the form of a Dispatcher request.



Note:

If you would like to configure the **QsearchProcessQueue** Dispatcher translator as a recurring request, set the following preferences as required:

- **QSEARCH_queue_recurring_background_updates**

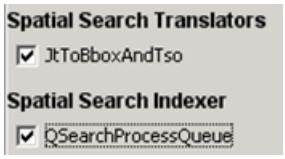
- **QSEARCH_recurring_background_update_survival_time**
- **QSEARCH_recurring_background_update_interval**

If you want to reduce the objects getting added to the queue and also reduce the number of Dispatcher requests, specify the item types you want to avoid in the **QSEARCH_types_to_avoid_processing** preference.

Setting up Dispatcher to manage updates (JT)

To set up Dispatcher to manage updates, ensure that you have:

- Installed the following Dispatcher translators, using TEM, as shown in the following figure:
  - **JtToBboxAndTso**
  - **QSearchProcessQueue**

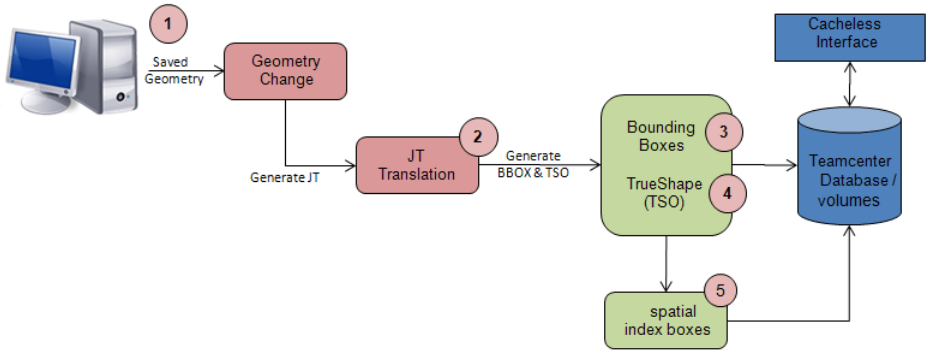


> Note:
>
> It is recommended that you use only one translator for the spatial search indexer.

- Set the **QSEARCH_update_enabled** preference to **True** if you want to generate TSO files.

The following figure illustrates the process of how a JT dataset is created, saved, and the spatial index updated with Dispatcher setup.

1. When a geometry is saved, triggers are automatically set for creation of a JT file.
2. The JT file is created and it requests the Dispatcher to generate the bounding boxes and TSO files.
3. The JT is loaded and the bounding box information is populated in the bounding box table. The tables in the database are populated with the bounding box information.
4. TSO files are generated and attached as a named reference.
5. A Dispatcher request is triggered, and a spatial index is created.



Note:

If you would like to configure the **QsearchProcessQueue** Dispatcher translator as a recurring request, set the following preferences as required:

- **QSEARCH_queue_recurring_background_updates**
- **QSEARCH_recurring_background_update_survival_time**
- **QSEARCH_recurring_background_update_interval**

If you want to reduce the objects getting added to the queue and also reduce the number of dispatcher requests, specify the item types you want to avoid in the **QSEARCH_types_to_avoid_processing** preference.

Setting up cron jobs

If you want to generate the spatial indexes using a **cron** job script, set the **QSEARCH_dispatcher_not_available** preference to **True**.

Configuring cacheless search in a Multi-Site Collaboration environment

If you work in a Multi-Site Collaboration environment, you can make cacheless spatial and attribute searches of products or programs that are owned or replicated at remote sites. Where appropriate, local and remote objects are incorporated in the search results.

If you enable this feature:

- Teamcenter checks access permissions to the remote BOM lines. If you do not have the necessary permissions, you cannot import the affected remote lines. If appropriate, you can review the search results and import only the lines to which you have access permissions.

- Identical revision and variant rules must be defined at all sites or searches may return inconsistent results. Also, the content of these revision rules and variant rules must always be consistent across all the sites.

- The content of saved queries must be consistent across all sites or searches may return inconsistent results.

- You can only initiate the search in Design Context, not in Structure Manager or Multi-Structure Manager.

To configure this feature, you must set the following preferences:

- **QS_remote_master_site**

  This preference lists the remote sites and the product or program that can be searched at each of those sites. It must be defined in a format similar to the following example:

  DEFAULT:site1
  Product2:master_site2
  Product3:master_site3

  The default site is always searched.

- **QS_remote_master_site_override**

  This preference allows the user to define a preferred remote site to make cacheless searches. It must be defined in a format similar to the following example:

  Product3:master_site3

  Whenever you add a new product or assembly to remotely search in Design Context, the administrator must update this preference with the new product and its owning master site at all sites in the Multi-Site Collaboration network.

You do not need to manually configure IDSM to use remote cacheless searches. When you install or upgrade the system using Teamcenter Environment Manager (TEM), it installs the callbacks necessary for remote cacheless search. Each site participating in remote cacheless search must install or upgrade the database using the same version of TEM.

A. Oracle Indexes

Ensure that the following Oracle indexes are created for the best performance of cacheless search.

- **PVARIANTEXPRESSION(POPERATOR,PFORMULA_STATE)**
- **PSOccurrenceNotes PPNOTE_TEXTS(UPPER(PVALU_0))**
- **PPSOCCURRENCE(RCHILD_ITEMU,ROCC_THREADU)**
- **PPSOCCURRENCE(RALTERNATE_ETC_REFU,ROCC_THREADU)**
- **PITEMREVISION(PUID,RITEMS_TAGU)**
- **PPOM_APPLICATION_OBJECT(ROWNING_GROUPU,PUID)**
- **PEXPRESSIONS(PUID)**
- **PVARIANTEXPRESSION(POPERATOR, PUID)**

B. Troubleshooting

Some common issues and how to troubleshoot them are included here for your reference.

How do I validate a spatial index?

Validate the index by running the **qsearch_process_queue** utility with the **check_structure_index** option.

Why is qsearch_process_queue taking so long?

A recurring request to the **qsearch_process_queue**, or any request to **qsearch_process_queue** may take longer than the dispatcher timeout period as it might have a lot of data to process.

The dispatcher assumes that the old request was finished, and launches a new **qsearch_process_queue** request for any new incoming request. This causes more than one **qsearch_process_queue** request to be processed at the same time, leading to locking contention, and subsequent problems with spatial index updates.

To avoid this, it is advised to set the timeout period for dispatcher to be high value.

How do I improve search engine performance?

If you find that the search engine performance is not the best, consider the following:

- If you have a small product structure (one thousand or less occurrence threads), you may improve performance by switching to a top down search.
- The lower the variation in geometry between different revisions of the same item, the better the search engine performs.
- The greater the correlation between the product structure and the geometry breakdown, the better the search engine performs.
- The accuracy of the search depends on the cell size you configure in NX or the voxel size in JT. A small cell or voxel size provides more accurate search results, but smaller cell sizes result in larger JT or TruShape files and response time takes more time.
- Avoid accumulating large number of entries in the indexing queue. The faster the updates are processes, the sooner live updates are available.
- During the deployment process, monitor the size of the queue by periodically running the **qsearch_process_queue** utility with the **-list_queue** option. If the queue search is increasing steadily, you might have to reconsider your deployment strategy.

Why is my search slow?

If you have a consistently slow search, set the following environment variables and execute a search.

- **TC_JOURNAL=SUMMARY**
- **TC_KEEP_SYSTEM_LOG=1**

You can then examine the **tcserver** system logs, which show a system status summary, a list of low-level call stacks, and the time consumed by each call. You can identify operations that consume considerable time and report them to get help from us. If the search is slow because of database issues, then get help from the database administrator.