QuestFields Indexer with Generic File Parser 2.2

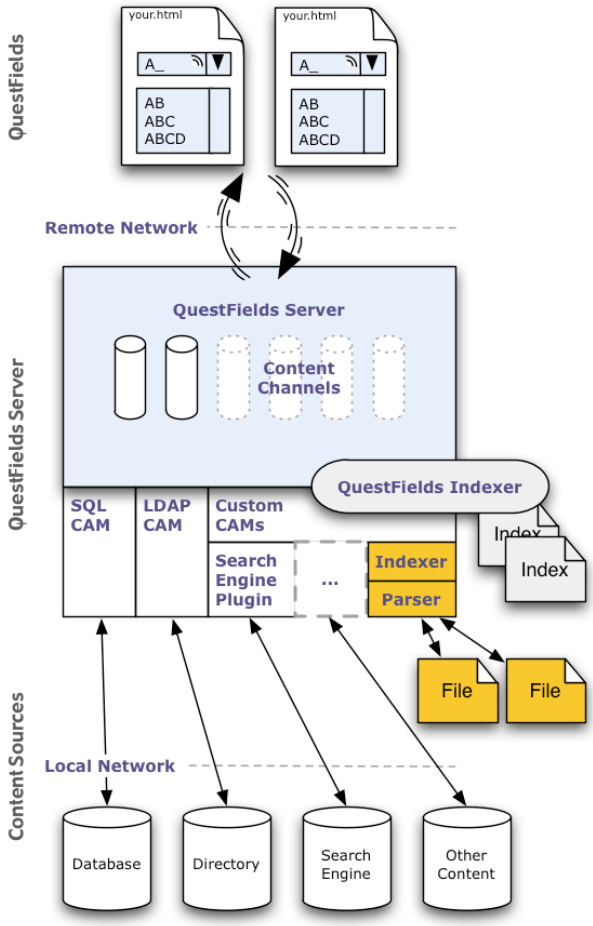
QuestFields Tech Note 015 rev 9

⇒ *This tech note describes the indexer and plugin module that allow the QuestFields Server to index, query, and retrieve information stored in flat files. This document also describes the behavior and configuration of the corresponding in-memory query engine.*

Version 2.2.1 of the QuestFields Indexer requires version 2.2.1 of the QuestFields Server.

1. About the QuestFields Indexer and the Generic File Parser

The QuestFields Indexer enables the QuestFields Server to index and query textual content. The QuestFields Indexer is bundled with the QuestFields Server, developed according to the QuestFields "Custom CAM" specifications (Custom Content Access Module). Together, the indexer and a "file parser" enable the creation of queries (used by QuestFields content channels) that allow ultra-fast retrieval of data, without the need for an external search engine or database:



The QuestFields Indexer is required for non-indexed content sources such as files. The QuestFields Indexer integrates seamlessly with the QuestFields Server and automatically indexes content that it receives from one or more file parsers. The QuestFields Indexer can also be run as a stand-alone

application, allowing the data to be indexed on a different machine without burdening the QuestFields Server.

Version 1.x of the Generic File Plugin supported tab-delimited files only. Support for other file formats can now be added as separate "File Parser" plugins. In some cases, the best approach is to write a "pre-parser" that flattens the proprietary file before import. Support for other file formats may be built into the QuestFields Indexer in the future.

Advantages

The retrieval capabilities of the QuestFields Indexer are highly optimized for AutoSuggest QuestField applications. The QuestFields Indexer offers far better search and retrieval performance at low cost. There is no need for third-party search software, and the QuestFields Server is much easier to configure and maintain than a third-party search engine.

Limitations

The QuestFields Indexer offers highly optimized yet limited query features compared to some third-party search engines. For example, it is not currently possible to include Boolean logic in the user query, because the QuestFields Server works on the start of words and has features that allow the indexer to split words (e.g., numbers in product codes). Also, to use the QuestFields Indexer, additional memory (RAM) must be allocated to the QuestFields Server. It is possible to keep the database and the word index on disk, so the additional memory footprint is very reasonable.

Whether to use a third-party database/search engine or the Generic File Plugin thus depends on your specific application needs.

Memory Use

As a rough indication, the QuestFields Indexer requires about the same amount of RAM as the raw text data to be imported, plus about 200MB for the query engine to efficiently handle user queries. This memory is required in addition to the memory that is reserved for the QuestFields Server cache and session pools. See the QuestFields Server Administration guide.

Differences between QuestFields Server 1.x and 2.x

In QuestFields Server 1.x, each content channel that uses the Generic File Plugin retrieves results from a single text file (which, of course, may contain multiple columns or words that could be derived from multiple attributes in the source database). So, a single content channel retrieves data from specified columns in one file.

In QuestFields Server 2.x, the QuestFields Indexer can be used to define any number of content queries, regardless of whether the content is in one or more files. Each query can even use a different file parser (text file format). Content queries can be combined with other content queries (even those accessing other content source types such as SQL databases or search engines).

2. Content Access Module Configuration

Before creating one or more content channel(s) for one or more data files, a Content Access Module (CAM) instance must be configured. This is done by adding a file to the `{QO_HOME}/conf/cams/` directory and configuring it to use the QuestFields Indexer.

The CAM configuration file uses the XML schema of the Custom Content Access Module (in QuestFields Server 1.x, this module was called the Java Content Access Module). A template/example of the CAM configuration file is provided by MasterObjects. In QuestFields Server versions 2.1 and higher, the QuestFields Indexer is installed automatically with the server software. The Indexer CAM is configured as follows:

```
<customCam id="indexer">
 <factoryClassName>com.masterobjects.qo.service.cam.search.SearchFactory</factoryClassName>
  <factoryProperties>
   <entry>
    <key>reloadTime</key>
    <value>3600</value>
   </entry>
 </factoryProperties>
</customCam>
```

In prior versions of the QuestFields Server, the indexer was bundled as a plug-in in the QuestObjects Home directory. It was selected by specifying a different `factoryClassName`:

```
 <factoryClassName>com.masterobjects.qo.external.search.GenericSearchFactory</factoryClassName>
```

The QuestFields Indexer CAM uses the following configuration attributes:

`reloadTime` (mandatory) At regular intervals, the QuestFields Server will check whether any data files or indexes have been updated. This attribute specifies the update interval in seconds. Whether the indexes or the data files are checked is determined by the value of the `indexByServer` attribute in the content channel configuration of that channel. (see `indexByServer` and `checkCount` below). E.g., to have the server check for updates every hour, use: `3600`

`maximumSearchClauses` (optional) This is an expert setting. It should only be used after consultation with MasterObjects. If certain queries take too much temporary memory during querying, this can lower the maximum number of search clauses to stop those queries. Generally, this is only an issue if the index is large (200+ MB). For optimal use, use this attribute in combination with the `queryType` attribute. Our tests suggest that `30000` is a good setting for large databases (containing millions of records). See `queryType` in the next chapter for more information.

The default value for this setting is unlimited (which causes the search to take an unlimited amount of memory).

3. Content Channel or Content Query Configuration

The Generic File Plugin uses a number of custom properties that are added to the content channel configuration file. In server 2.x this is done in the content query configuration file; in server 1.x this was done using the content channel.

⇒ *Note that the content query configuration is an xml file and that `&`, `'`, `"`, `<` and `>` need to be escaped (to `&`, `&apos;`, `&quot;`, `<` and `>` respectively).*

In the content channel configuration (server 1.x) the backslash character must be escaped (so, instead of "`\`", use "`\\`"). This is not necessary in the content query configuration (server 2.0 and higher).

The following properties define the location and type of file to index and the place that will hold that index.

Generic File Parser Configuration Attributes

`fileName` (optional since version 2.2.1) This is the file name or the full path to the import data file (excluding trailing slash or backslash). On Unix platforms, the value might be as follows:
`/usr/local/data-files-for-qo/tab-delimited-files/data.txt`
On Windows, the value might be something like:

C:\\data-files-for-qo\\tab-delimited-files\\data.txt

By default, the Indexer assumes a subdirectory in {QO_HOME}/files whose name matches the content query id and file name data*.txt.

directory — (optional since version 2.2.1) The full path to the index directory for this content query (in QuestFields Server 1.0: the content channel). This directory will contain the indexes created by the QuestFields Indexer. A dedicated directory must be used for each content query definition. On Unix platforms, the value might be as follows:
/usr/local/data-files-for-qo/indexes-for-query-x
On Windows, the value might be something like:
C:\\data-files-for-qo\\indexes-for-query-x

By default, the Indexer creates a subdirectory whose name matches the content query id in directory {QO_HOME}/indexes.

encoding — (optional) This attribute indicates the character encoding of the data files. If omitted, the attribute defaults to UTF-8. Other supported values include: US-ASCII, ISO-8859-1, ISO-8859-9, and UTF-16.

For a full list of supported values, restart the QuestFields Server in DEBUG logging mode. The generic file parser will write the complete list into the service log file. Do not forget to reset the logging mode afterwards.

checkCount — (optional) This is an optional attribute that, if set to true, causes the QuestFields Server to check whether the number of lines imported matches the count as it is included in each import data file. If set to true, each tab-delimited file is expected to have an extra (trailing) line that contains this number (with no additional tabs). The default value is false.

If the QuestFields Server detects a different number of lines, it assumes that the import has failed and will not replace the previous data that was read into memory. In this case, an error message will be written into the Content Service error log file (see chapter about logging in the QuestFields Server Administration Guide). By default, the QuestFields Server will try importing the file again after the interval specified by reloadTime.

The checkCount attribute defaults to false.

parser — This is the full qualified name of the java class that is responsible for parsing the file. It defaults to:

com.masterobjects.qo.external.search.index.plugin.TabDelimitedParser

Additional parsers can be developed to handle other file formats.

QuestFields Indexer Configuration Attributes

indexByServer — (optional) If set to true, the server will index the tab-delimited file if necessary. This is based on whether there is an index, and if so, whether the file's last modified has changed since the last indexing process. The server will rescan the file's last modified time after the interval specified by reloadTime. If an index is available the QuestFields Server will always read the latest index into memory first before checking if the tab delimited file is newer than the one that formed the bases of the last index.

Indexing takes place in the background: The content channel on the QuestFields Server remains available. The previous index is used until the new index is ready.

By default, this attribute is set to `false`. Chapter 4 explains how to create an index from the command line. Note that setting this can cause extra memory to be used by the QuestFields Server. Temporary memory required can increase by about 200 MB depending on the size of the data file.

`removeDuplicates` (optional) If set to `true`, the server will remove duplicate values based on the value of the key field from the index. The removal is done in the order of appearance in the index. Since the order is determined by index efficiency guidelines, no guarantees can be made as to which of the duplicate entries stay and which are removed.

If the database does not guarantee that (the combination of key and value of) records are unique, it is essential that `removeDuplicates` be set to true. Duplicate records in the index can unnecessarily slow down searches.

By default this attribute is set to `false`.

Using the following custom attributes, the content channel specifies the various data elements in the tab-delimited file that are to be indexed by the QuestFields Server. Each element is identified by an id. In tab-delimited files, this id corresponds to the zero-based column number (i.e., the first tab-delimited column is identified by `0`, the second column by `1`, etc.).

`valueId` (mandatory) This is the id of the data element containing the value to be indexed and retrieved. The indexing process will index every word contained in this element, and additional "parts" of words depending on the retrieval engine configuration, as defined in the Content Access Module.

The element identified by `valueId` is automatically indexed, so the element does not need to be added to `searchIdList` (see below).

Query behavior and result formatting

The following examples are based on value elements containing the literal string "`Sony PlayStation 3 (Game Console)`" and "`Microsoft Xbox 360 SKU334950`"
The in-memory query engine always performs case-insensitive queries.

- If the user types the beginning of any word in the string to be found, then the result is displayed starting with that word, followed by a comma:
  Type "`Son`". Result: `Sony PlayStation 3 (Game Console)`
- If a user types additional words, then the retrieval engine finds all results that start with the first word, and contain the second:
  Type "`sony game`". Result: `Sony PlayStation 3 (Game Console)`
- If a user starts by typing the second or consecutive word in the value, then the result is reformatted so that it starts with that word. The start of the value is then appended to the end of the result.
  Type "`xb`". Result: `Xbox 360 SKU334950, Microsoft`
  Type "`360`". Result: `360 SKU334950, Microsoft Xbox`
- If the last word(s) of the value are enclosed by parentheses, then those words are always displayed at the end of result string:
  Type "`play`". Result: `PlayStation 3, Sony (Game Console)`
- If the word in parentheses is typed, then the parentheses are removed:
  Type "`game`". Result: `Game Console, Sony PlayStation 3`
- Optionally, the retrieval engine can be configured to recognize parts of words, such as "`station`" in "`PlayStation`" or "`334950`" in "`SKU334950`". If the user types the

first letters of such a "mid word", then the result will be displayed starting with the full word in which it belongs:
Type "station". Result: `PlayStation 3, Sony (Game Console)`
Type "334". Result: `SKU334950, Microsoft Xbox 360`

`valueBoost` (optional, available in version 2.1 or later) This is a number (with floating decimal point) that is used to "boost" results that have a value matching the search query, versus results that were found because of matches in the metadata. A setting of `1` means that the value is considered equally important as the metadata, assuming that both have the same length. If the value is shorter than the metadata, then it is boosted slightly, even if `valueBoost` is set to `1`. Default: `2.0`.

`keyId` (mandatory) This is the "id" of the key element. When retrieving data, the QuestFields Server automatically filters consecutive results out of the result list if they have the same key. If `removeDuplicates` is defined it is used to identify duplicate entries in the index.

By default, the element identified by `keyId` is not indexed for user queries, so the same element may need to be added to `searchIdList` (see below) if you want it to be searchable.

`metaIdList` (optional) This is a comma-separated list containing the ids of the elements to be retrieved as metadata in the result list. Do not include spaces in the list. Note that, in addition to this list, you can optionally have the indexer add the search score as a final metadata value. See `addScoreToMeta` under *search configuration* below.

`searchIdList` (optional) This is a comma-separated list containing the ids of the elements to be indexed beside the value. Words will be handled in the same way as described previously with the `valueId` except with a lower importance. Do not include spaces in the list.

Note that the elements defined by `valueId`, `keyId`, and `qualifierId` are always indexed and thus do not need to be included in this list. Key and qualifier use "non tokenized" indexes, so you cannot search for words inside of them unless you also include them in `searchIdList`.

`promoteId` (optional) This identifies an element that is used to "promote" results in the results list. The value of the element should be an integer number. The QuestFields Server places results containing *higher* numeric values in this element at the *top* of the results list. If multiple results have the same "`promote`" number, their ordering in the result set remains untouched (i.e., they are sorted by search score).

The value of the element identified by `promoteId` must be an integer between -10,000,000 and 10,000,000 (smaller or larger values than negative or positive ten million are reserved for sorting routines of the indexer).

`promoteDivisor` (optional, only in combination with `promoteId`, available in version 2.1 or later) This is an integer number by which the `promoteId` field value is divided to determine the relative priority of search results. Without this value, search results are sorted solely by `promoteId`, regardless of the "score" that is calculated by the search engine and that indicates how well a result matches the user's query.

If `promoteDivisor` is used, then results are sorted by search score, which is then multiplied by `promoteId` value and divided by `promoteDivisor`.

E.g., if a record has a `promoteId` value of `1,000` and the `promoteDivisor` is 500, then the result's score (search rank) is boosted by `1,000 ÷ 500 = 2.0`.

The promoteId field value is divided by promoteDivisor, if and only if the promoteDivisor and the promoteId field value are larger than 0. PromoteId values of zero and negative values are not influenced.

The result is used to "boost" the result relative to other results. I.e.,

Default: 0 (unused)

qualifierId — (optional) This identifies an element that is used as a filter. If the QuestField configuration specifies a qualifier, the QuestFields Server will only return results that have a matching value in this element. I.e., the QuestFields Server uses the QuestField's qualifier to filter results based on the element identified by qualifierId. See the QuestField Administration Guide for more information about the optional qualifier.

Data elements (currently, tab-delimited columns) that are not included in the content channel configuration are ignored, are not read into memory, and are thus not transmitted to the QuestField as part of the results list. Note that a single element may serve multiple purposes. E.g., the qualifierId or promoteId may also be included in the metaIdList.

Index Options

The Generic File CAM reads all rows from the data file and then creates a highly optimized index based on one or more text columns contained in each file (as configured in the content channel -- see below). QuestField users can find results by typing the start of any word contained in the text column followed by a space and the start of any other word. The following attributes allow the administrator to configure the indexing. Some of these attributes will increase the size of the index, which can have a small impact on the speed of the search. In general the index is expected to be close to the size of the file that has been indexed, with the attributes defined:

camelWords — (optional) When set to true, this attribute causes the QuestFields Server to recognize parts of words that start with an uppercase character after a lowercase character ("Camel Case"). E.g., the server will allow users to type "mac" in order to find "iMac" or to type "objects" to find "MasterObjects". The default value of this attribute is true.

Implications: The index will take more memory.

splitNumbers — (optional) This attribute, when set to true, causes the QuestFields Server to recognize numbers and alphanumeric sequences that are part of a longer string, such as a product type. For example, users would be able to type "400" in order to find a product containing the text "LG LW-T400A" in its text column. The default value of this attribute is true.

Implications: Memory used by the index increases by the number of newly detected words.

minimumSplit — (optional) This attribute contains the minimum number of digits or alphanumeric characters that will cause the QuestFields Server to add a word to its index when the string is split using the splitNumbers attribute. If splitting the string results in "words" that are less than minimumSplit, then those words are not included in the retrieval index. So if minimumSplit is set to 2 (the default), "LG LW-T400A4GT" would not be retrieved by typing just "a", but it would be retrieved by typing "gt". The default value of this attribute is 2.

ignoreList — (optional*) This is a comma-separated list of words that are excluded from the retrieval index, i.e., they can not be the first word typed by the user. These words are commonly referred to as "stop words". Do not include spaces in the list. Example: for,to,the,at,and,of,in,y,&

`separatorList` (optional*) This is a non-separated list of characters that are recognized as word separators, that is, they index the words that follow them: If a dash (`-`) is included in this list, the QuestFields Server will allow users to type "`T40`" in order to find "`LG LW-T400A`", even if a dash is included in the `stripList` attribute (described below). Do not include spaces in the list. Example:
`.\\,\-/+_"'`&`

Implications: Memory used by the index increases by the number of newly detected words.

`stripList` (optional*) This is a non-separated list of characters that are automatically stripped from user queries, and from text in the word index. For example, after including a dash (`-`) and a slash (`/`) in the comma-separated list, the QuestFields Server will allow users to find "`LG LW-T400`" by typing "`LWT`", "`LW-T`", or "`LW/T`". Do not include spaces in the list. Example:
`.\\,\-/+_"'`&`

Implications: Memory used by the index increases by the number of newly detected words.

`stripZeros` (optional) When set to `true`, the QuestFields Server will create extra words in the index for numbers that have leading zeros. Users will be able to find results containing a number without having to type the leading zeros. So, a result containing "`000123`" can be found by typing "`123`". Default: `false`.

Implications: Memory used by the index increases by the number of newly detected words.

`flattenLatin` (optional) When set to `true`, the QuestFields Server will "flatten" characters that have an accent (´,¨,etc). So, a result containing "`é`" can be found by typing "`e`". Default: `false`.

*To include a dash (`-`), backslash (`\`), or square bracket (`[` or `]`) in any of the non-separated lists, use backslash escaping:`\- \\ \[ \]`.

Search Options

The following attributes influence searching, regardless of the index configuration.

`sortByValue` (optional, available in version 2.1 or later) When `true`, results will be sorted alphabetically (but this default sort is optionally overridden by `promoteValueMatches` and `promoteId`). Default value: `false`

`promoteValueMatches` (optional, available in version 2.1 or later) When `true`, results that have a value containing a word matching the first word in the user's query are displayed at the top of the result list. Default value: `true`

`addScoreToMeta` (optional, available in version 2.1 or later) When `true`, the indexer will add an additional metadata value to each result, containing the relative score of the document in the results list. Default value: `false`

`queryType` (optional) This attribute currently has three possible values. It determines the type of query that is executed against the index.

The default value (`2`) provides a Prefix query. This means that all words are found that start with, or are equal to, the words in the actual query. This can cause a significant amount of temporary memory usage during a query when using a large (200+ MB) index. If no prefix query results are found, the indexer will optionally perform mid-

word and/or fuzzy queries, as determined by the `midQueryLength` and `fuzzyResults` options explained below.

The second value (`4`) provides a Boolean query. Only words that are equal to the words in the query will be found. While the amount of temporary memory required is not problematic with this setting, it might not provide all the results that the user expects.

The third possible value (`6`) provides a hybrid approach. The QuestFields Server will first perform a Boolean query. If the number of results is lower than the maximum set for the channel, a Prefix query is done for all words that are 3 characters or longer. If the maximum is still not reached a Prefix query is done for all words of 2 characters and longer, etc. until all words are used in a Prefix query.

Using this third setting only makes sense in combination with setting the `maxSearchClauses` attribute in the Content Access Module configuration. If searches exceed the search clauses limit, a result is appended to the results list to inform the user. For example: *"Results limited to exact matches for xx, yy, zz."*. Users can then add characters to the words in their query. Because of the possible multiple searches, it takes longer to get a result. However the temporary memory requirement is controllable with the `maxSearchClauses` attribute and the results are what the user expects.

Default value: `2`.

`midQueryLength` (optional) This option extends Prefix query behavior by performing Mid queries if insufficient results were found (`queryType` 2 or 6):

If set to a number greater than `0`, the QuestFields Server will do mid-searches (finding results that contain the word besides start with the word) with words that are equal or longer than `midQueryLength`. The smaller this number, the higher the number of search clauses. Default value: `0` (which means mid queries are off).

Implications: Searching will be significantly slower if the user types longer words that do not return sufficient prefix matches (as determined by `maxResults` in the content channel).

`fuzzyResults` (optional, available in version 2.1 or later) This option extends Prefix query behavior by performing Fuzzy queries if no results were found using prefix- and optional mid queries (`queryType` 2 or 6):

If set to a number greater than `0`, the QuestFields Server will perform fuzzy queries (thus providing suggested results that do not match the query literally) if the other query options returned no results. Default value: `0` (which means fuzzy queries are off).

`fuzzyQueryLength` (optional, available in version 2.1 or later) This option can be used to further influence Fuzzy query behavior and only does anything if `fuzzyResults` is larger than `0`.

The QuestFields Server will only perform fuzzy searches with words that are equal or longer than `fuzzyQueryLength`. The smaller this number, the higher the number of search clauses. Default value: `4`.

`resultsLimitedWarnings` (optional, available in version 2.1 or later) When `true`, an extra record will be added to the end of a result list that was limited because of an indexer limit. If set to `false`, then warnings `-11`, `-12`, `-20`, and `-21` (described in chapter 6) will not appear in the results list. Default value: `true`

Expert Configuration

Some attributes can be used to tweak performance and memory use of the QuestFields Indexer. Most of these attributes have sane defaults and should only be set if advised to do so by MasterObjects:

`loadIndexInMemory` (optional) When set to `true`, the QuestFields Server will load the entire index into memory, rather than performing searches using the index on disk. This will result in faster search times at the expense of higher memory usage. The memory requirement for loading the index into memory is about the size of the index on disk. For example a 50MB index on disk will use about 50MB in RAM in addition to the normal requirements of the server.

Default: `false` (in versions below 2.1, the default was `true`)

`maxRecordTerms` (optional, available in version 2.1 or later) An integer that represents the maximum number of terms that will be indexed for the fields in `searchIdList`. This limits the amount of memory required for indexing, so that collections with very large files will not crash the indexing process by running out of memory. This setting refers to the total number of terms found in the metadata, i.e. every word plus any parts thereof that are indexed. Default: `100000`.

Note: setting this too low means that the indexer silently truncates large documents, excluding from the index all terms that occur further in the field. If you know your source fields are large, be sure to set this value high enough to accommodate the expected size.

`optimizeResults` (optional) When set to `true`, the QuestFields Server will create an in-memory copy of the results and metadata. This will speed up retrieval of results at a cost of almost doubling the amount of memory needed for the index (count on memory use going up by about half the data file size).

This option can be used even if the index itself is not kept in memory.
The speed of the QuestFields Server should be fast enough in most cases, but this option will help if results contain a lot of metadata, or if you allow users to fetch many results (i.e., if you set `maxResults` of the content channel to a high number).

Default: `false`.

`intern` (optional) When used in combination with `optimizeResult` and set to `true`, the QuestFields Server will optimize the in-memory results by "interning" all text strings, thereby lowering the amount of extra memory used for optimization, potentially by about half.

Note: If this setting is used, you must add the following setting to the Java Runtime configuration: `-XX:MaxPermSize={memoryusageinMB}m`. Set this to the size of the index.

Default: `false`.

4. Indexing

At startup of the channel, the QuestFields Server will look in the defined `directory` for an available index. It will take the newest valid index. Whether an index is valid is determined by a CRC checksum on the index. If no valid index is found, two things can happen: If `indexByServer` is set to `true`, the server will start indexing the tab-delimited file defined by attribute `filename`. If `indexByServer` is `false`, the server will log an error and the users will see a result that says "`Indexing…`".

How the QuestFields Indexer works

The index stores information about the tab-delimited file to determine whether the file needs to be re-indexed. The indexing process will check this information before starting the indexing process. The second step is creating a new index beside the current index in the defined `directory`. This is to make sure that any problem that occurs during the indexing step will not cause the already existing index to become invalid. The indexing step is followed by an optimization step to optimize the index. The index is then check summed, and a verification file is written. After indexing is finished, the directory is cleaned up. This cleanup removes all but the last 2 valid indexes and a possible new index being created. All other partial or valid indexes will be removed, as will their verification file. At the end there will be at least one valid index available to the QuestFields Server.

Command line indexing

The QuestFields Indexer supports offline file indexing. It uses the `java` command on the command line to perform the indexing process.

Two examples of the command are shown below, one for windows and one for Unix variants. They both assume that the `java` command is available in the path. This can be verified by typing:

```
java –version
```

The following commands must be typed as a single line (using a space after each parameter).

Unix

```
java
 –Xmx512m
 –jar qo-indexing-utility-2.1.1.jar
{/path-to-QO_HOME}
{content-query-id}
<optional_path_to_plugins>
```

Windows

```
java
 –Xmx512m
 –jar qo-indexing-utility-2.1.1.jar
{C:\path-to-QO_HOME}
{content-query-id}
<optional_path_to_plugins>
```

Note: The "-Xmx512m" option ensures that 512 megabytes of RAM is available to the indexing process.

5. Channel Variable Mappings

When the QuestFields search engine performs a QuestFields Indexer content query, it can map QuestField values (INPUT_BUFFER, QUALIFIER) to the following variables using `groupToVariableMapping` in the content channel:

`query` This is the string of the full-text search to be performed. The search engine performs an "and" search using the words of this *query string* (possibly using fragments of words as determined by the search options defined in chapter 3) on the elements defined by `searchIdList`.

`qualifier` This is the string to be used as a filter in the search. If specified (non-empty), the search engine will perform a prefix search on the element defined by `qualifierId` using this *qualifier string*.

`and` This is an optional search string containing words that must exist. If specified (non-empty), the search engine will add this *and string* to the *query string* when performing the search.

6. Special Results

The QuestFields Indexer returns specially formatted results when it is indexing or when errors or warnings occur. These results can be recognized in the client by id string "QO_INDEXER" as the first metadata value (in QuestFields Server versions 2.1 and up, the result also has its type set to 1). Each error has a unique number, enabling translation into different languages. QuestFields Server 2.1 will automatically translate messages depending on the users' language (as configured in the QuestFields Client).

Key	Value	meta0, meta1, meta2	Other metadata
-1	Indexing "*{contentQueryId}*"…	QO_INDEXER, INFO, contentQueryId	meta3: The Id of the content query.
-2	The search engine was restarted. Verifying index "*{contentQueryId}*"…	QO_INDEXER, INFO, contentQueryId	meta3: The Id of the content query.
-3	Index "*{contentQueryId}*" is temporarily unavailable.	QO_INDEXER, INFO, contentQueryId	meta3: The Id of the content query.
-10	The query consists of too many words.	QO_INDEXER, INFO, contentQueryId	none
-11	Results limited to exact matches for meta3. To find words that start with meta4, please add at least one more character.	QO_INDEXER, INFO, contentQueryId	meta3: Words used for exact matches. meta4: Words used for prefix matches.
-12	Skipped results because more than meta3 words were found based on "meta4". Please add additional characters.	QO_INDEXER, INFO, contentQueryId	meta3: Maximum number of search clauses as defined by maximumSearchClauses (see chapter 2)
-13	Too many matches were found for "meta3". Please narrow down your search.	QO_INDEXER, INFO, contentQueryId	meta3: The query value
-20	Literal matches for "meta3" were treated as stop words and may have been skipped.	QO_INDEXER, INFO, contentQueryId	meta3: Words that were ignored
-21	Showing suggestions because no literal matches were found. Please verify your query.	QO_INDEXER, INFO, contentQueryId	