Document Indexing Services

A solution for searching within the contents of documents. Useful for application that do not require the power of a full featured search product.

Limitations

- Does not compete with the functionality and power of dedicated products like Apache Solr, Apache Lucene and Elastic Search.
- The relevance of search results and ability to improve them are limited by the given databases functionality.
- Performance and scalability are limited by the amount of text content stored, complexity of search query, the number of concurrent search requests and available database resources.
- Supports fewer file types than a dedicated product.
- MySQL takes approximately 1 second for each 10k documents indexed (150k documents = 15s query time). Refer to each DB vendor or your DBA for how to improve performance.

Index Document Contents Smart Service

Extracts the text content of a document and stores in a database table.

Supported file types

- DOCX
- DOC
- MSG
- PDF
- TXT
- XLSX

Node Inputs

Input	Data Type	Required	Description
JNDI Name	Text	Yes	The datasource JNDI name
Table Name	Text	Yes	The database table to store the documents contents into

Input	Data Type	Required	Description
Document	Document	Yes	The document to index
Max Characters To Index	Integer	Yes	The maximum number of characters to read from the document and save into the database. A typical page contains around 3000 characters. Be aware that most database column limits are in bytes not characters. As 1 character can require multiple bytes to represent, the database column must be larger than this value. This value must be between 1 - 64,000
Pause On Error	Boolean	Yes	Whether the node should fail on error or continue

Node Outputs

Output	Data Type	Description
Error Occurred	Boolean	Returns true if an error has occurred while indexing the document
Error Message	Text	The error that occurred

Table Definition

The plug-in stores 2 attributes in the database; the Appian document id and its text content.

Use cases for search typically require additional metadata to filter against, such as the document name, last modified date and document type. These may be stored in the same table or a separate lookup table. It is the designers responsibility to populate the metadata fields manually as this plug-in will not do it.

If a document has already been indexed, the content field will be updated.

```
CREATE TABLE `document` (
  `id` int NOT NULL,
  `content` text,
  PRIMARY KEY (`id`),
  FULLTEXT KEY `FT_CONTENT` (`content`)
);
```

Example Search Query

As Appian does not support the `MATCH` clause, wrap the search query in a stored procedure.

```
SELECT
  id,
  LEFT(content, 50) AS snippet
FROM
  new_table
WHERE
  MATCH (content) AGAINST ('ridiculous penguin shaped horse' IN BOOLEAN MODE)
LIMIT
  10;
```

Testing and Release

Adding Document Types

Certain documents are very inefficient to process with Apache Tika. This project requires each document type contains an explicit test case. I recommend validating against a large volume of real documents before releasing support for a new document type. **The risk of causing stability issues in Appian from poorly handled document types is high so test thoroughly.**

1. Uncomment the desired document type in `tika-config.xml`
2. Add a test document with 1 million bytes of text content to `src/test/resources/docs/large`
3. Run unit tests (see development build steps)

Development Build

To create a plug-in jar for testing, run the following Maven command:

- `mvn clean package`
- The jar can be found in `/target`

Release Build

To create a new public release, run the following Maven commands:

- `mvn release:clean`
- `mvn release:prepare -DautoVersionSubmodules=true -DpushChanges=false`
- `git push origin master --tags`
- The release jar can be found in `/target`