TMEngine

An Open Source Translation Memory Manager

Table of Contents

Overview

TMEngine

TMEngine is an open source Translation Memory (TM) manager written in Java.

TMEngine is based on the translation memory library used by Swordfish III, Fluenta and RemoteTM.

TMEngine can be used in two ways:

- As a standalone TM server via its REST API.
- As an embedded library that manages translation memories in a Java application.

The standalone server runs on these platforms:

- Microsoft Windows 8, 1.1 and 10
- macOS 10.13, 10.14 and 10.15
- Linux (any version capable of running Java 11)

A TMEngine server allows sharing Translation Memory data in a local network or over the Internet.

Note

The .jar files included in TMEngine distributions are compiled with Java 11.

Running as a Standalone Server

Starting the Server

Running `.\tmserver.bat` or `./tmserver.sh` without parameters displays help for starting TMEngine as a standalone server.

```
Usage:

      tmserver.sh [-help] [-version] [-port portNumber]

      Where:

      -help:      (optional) Display this help information and exit
      -version:   (optional) Display version & build information and exit
      -port:      (optional) Port for running HTTP server. Default is 8000
```

Note

You can verify that the server is running by visiting its default web page: http://localhost:8000/TMServer/ (adjust port number if you change it).

Stopping the Server

A running TMEngine server can be stopped using the Stop Server method from its REST API.

Simply visit "`http://localhost:8000/TMServer/stop`" using a browser or open a connection to that URL when the server is embedded in a Java application. Adjust the port number if necessary.

REST API

The REST methods that TMEngine's server supports are:

- Create Memory
- List Memories
- Open Memory
- Close Memory
- Import TMX File
- Get Process Status
- Export TMX File
- Search Translations
- Concordance Search
- Rename Memory
- Delete Memory
- Stop Server

Default TMEngine URL is '`http://localhost:8000/TMServer/`'.

> **Note**
> It is possible to select a custom port for the server, passing the '`-port`' parameter to the script used for launching it.

All methods return a JSON object with a '`status`' field. Applications must watch this field and verify that it is set to '`OK`'.

In case of error, the JSON response includes a field named '`reason`' that contains the error cause.

Create Memory

End Point: `[TMEngine URL]/create`

Default: `http://localhost:8000/TMServer/create`

Send a '`POST`' request to the method end point with these parameters in a JSON body:

Field	Required	Content
`id`	No	ID of the memory to create. The value of '`id`' must be unique. Default value is current server time represented as the number of milliseconds since January 1, 1970, 00:00:00 GMT
`name`	Yes	A meaningful name to identify the memory
`owner`	No	Text string used to identify the owner of the memory. Default value is the login name of the user running the server.
`type`	No	Type of engine to use. Possible values are: • '`MapDbEngine`' (default) • '`SQLEngine`'
`serverName`	No	Name or IP of the server running MySQL or MariaDB. Required for `SQLEngine`. Defaut value: '`localhost`'
`port`	No	Port in which MySQL or MariaDB listens for requests. Required for `SQLEngine`. Default value: `3306`
`userName`	No	ID of of the MySQL or MariaDB user creating the database. Required for `SQLEngine`.
`password`	No	Password of the MySQL or MariaDB user creating the database. Required for `SQLEngine`.

Example:

```
{
  "name": "First Memory",
  "type": "MapDbEngine"
}
```

```
{
  "name": "MariaMemory",
  "type": "SQLEngine",
  "serverName": "localhost",
  "port": 3306,
  "userName": "root",
  "password": "secret123!"
}
```

The server responds with a JSON object containing two fields.

On success, field '`status`' is set to '`OK`' and field '`id`' contains the ID assigned to the new memory.

Example:

```
{
  "status": "OK",
  "id": "1234567890987"
}
```

On error, field '`status`' is set to '`failed`' and field '`reason`' contains the error cause.

Example:

```
{
  "status": "failed",
  "reason": "Duplicated id"
}
```

List Memories

End Point: `[TMEngine URL]/list`

Default: `http://localhost:8000/TMServer/list`

Send a '`GET`' request to the method end point.

The server responds with a JSON object containing two fields. On success, field '`status`' is set to '`OK`' and field '`memories`' contains an array with memory details.

```
{
  "memories": [
     {
       "owner": "manager",
       "isOpen": false,
       "name": "Fluenta Localization",
       "id": "fluenta",
       "type": "MapDbEngine",
       "creationDate": "2019-09-10 21:54:13 UYT"
     },
```

```
    {
      "owner": "manager",
      "isOpen": false,
      "name": "First Memory",
      "id": "1568163112478",
      "type": "MapDbEngine",
      "creationDate": "2019-09-10 21:51:52 UYT"
    }
  ],
  "status": "OK"
}
```

On error, field '`status`' is set to '`failed`' and field '`reason`' contains the error cause.

Example:

```
{
  "status": "failed",
  "reason": "Error reading memories"
}
```

Open Memory

End Point: `[TMEngine URL]/create`

Default: `http://localhost:8000/TMServer/open`

Send a '`POST`' request to the method end point with this parameter in a JSON body:

Field	Required	Content
`id`	Yes	ID of the memory to open

Example:

```
{
  "id": "1568163112478"
}
```

The server responds with a JSON object. On success, field '`status`' is set to '`OK`'.

```
{
  "status": "OK"
}
```

On error, field '`status`' is set to '`failed`' and field '`reason`' contains the error cause.

Example:

```
{
  "status": "failed",
  "reason": "Unknown memory type"
}
```

Close Memory

End Point: `[TMEngine URL]/close`

Default: `http://localhost:8000/TMServer/close`

Send a '`POST`' request to the method end point with this parameter in a JSON body:

Field	Required	Content
`id`	Yes	ID of the memory to close

Example:

```
{
  "id": "1568163112478"
}
```

The server responds with a JSON object. On success, field '`status`' is set to '`OK`'.

```
{
  "status": "OK"
}
```

On error, field '`status`' is set to '`failed`' and field '`reason`' contains the error cause.

Example:

```
{
  "status": "failed",
  "reason": "Unknown memory"
}
```

Get Languages

End Point: `[TMEngine URL]/languages`

Default: `http://localhost:8000/TMServer/languages`

Send a '`POST`' request to the method end point with this parameter in a JSON body:

Field	Required	Content
`id`	Yes	ID of the memory to query

Example:

```
{
  "id": "1568163112456"
}
```

The server responds with a JSON object containing two fields.

On success, field '`status`' is set to '`OK`' and field '`process`' contains the ID of the background query process that was initiated.

```
{
  "process": "1568222345683",
```

```
  "status": "OK"
}
```

On error, field '`status`' is set to '`failed`' and field '`reason`' contains the error cause.

```
{
  "status": "failed",
  "reason": "Unknown memory type"
}
```

After starting the query process, monitor its status using the Get Process Status method. On successful completion, the `data` field will contain a list of languages present in the memory.

Example:

```
{
  "result": "Completed",
  "data": {
    "languages": [ "es", "en" ]
  },
  "status": "OK"
}
```

Import TMX File

End Point: `[TMEngine URL]/import`

Default: `http://localhost:8000/TMServer/import`

Send a '`POST`' request to the method end point with these parameters in a JSON body:

Field	Required	Content
`id`	Yes	ID of the memory to populate with TMX data
`file`	Yes	Path to the TMX file being imported
`subject`	No	Name or identifier of the subject associated with the TMX file
`client`	No	Name or identifier of the client associated with the TMX file
`project`	No	Name or identifier of the project associated with the TMX file

Note

The TMEngine server must have access to the TMX file being imported. When importing a TMX file into a remote server, copy or upload the file to the server first and supply the right path in the JSON body.

Example:

```
{
  "id": "1568163112478",
  "file": "/Volumes/Data/segments.tmx",
  "project": "Main TM"
}
```

The server responds with a JSON object containing two fields.

On success, field '`status`' is set to '`OK`' and field '`process`' contains the ID of the background import process that was initiated.

```
{
  "process": "1568222345643",
  "status": "OK"
}
```

On error, field '`status`' is set to '`failed`' and field '`reason`' contains the error cause.

```
{
  "status": "failed",
  "reason": "Unknown memory type"
}
```

After starting the import process, monitor its status using the Get Process Status method. On successful completion, the result will contain the number of segments imported.

Example:

```
{
  "result": "Completed",
  "data": {
    "imported": "57678"
  },
  "status": "OK"
}
```

Get Process Status

End Point: `[TMEngine URL]/status`

Default: `http://localhost:8000/TMServer/status`

Send a `POST` request to the method end point with this parameter in a JSON body:

Field	Required	Content
`process`	Yes	ID of the background process to check

Example:

```
{
  "process": "1568223016762"
}
```

The server responds with a JSON object.

On successful status check, field '`status`' is set to '`OK`' and field '`result`' contains current status.

Example:

Field '`result`' may have these values:

- **Pending**: processing is still going on.

```
{
  "result": "Pending",
  "status": "OK"
}
```

- **Completed**: processing has finished. If the process produces any data, it is placed in the '`data`' field.

```
{
  "result": "Completed",
  "data": {
    "imported": "57678"
  },
  "status": "OK"
}
```

- **Failed**: processing failed. Failure reason is provided in '`reason`' field.

```
{
  "result": "Failed",
  "reason": "/Volumes/Data/something.tmx (No such file or directory)",
  "status": "failed"
}
```

If process status cannot be checked, the server omits the '`result`' field and provides a failure reason.

```
{
  "reason": "Missing 'process' parameter",
  "status": "failed"
}
```

Export TMX File

End Point: `[TMEngine URL]/create`

Default: `http://localhost:8000/TMServer/export`

Send a '`POST`' request to the method end point with these parameters in a JSON body:

Field	Required	Content
`id`	Yes	ID of the memory to populate with TMX data
`file`	Yes	Path to the TMX file being created
`langs`	No	JSON array contaning the list of languages to export

Field	Required	Content
`srcLang`	No	Language to set as source langage. The wildcard '`*all*`' is used by default
`properties`	No	JSON object with string properties to set in the exported file

> **Note**
> when exporting a TMX file on a remote server, make sure the TMEngine server has access to the specified location.

Example:

```
{
  "id": "1568163112478",
  "file": "/Volumes/Data/segments.tmx",
  "langs": [
      "en-US",
       "ja",
       "fr-FR",
       "it"
  ],
  "srcLang": "en-US",
  "properties": {
    "project": "Milky Way",
    "subject": "Astronomy Device"
  }
}
```

The server responds with a JSON object containing two fields.

On success, field '`status`' is set to '`OK`' and field '`process`' contains the ID of the background export process that was initiated.

```
{
  "process": "1568222345643",
  "status": "OK"
}
```

On error, field '`status`' is set to '`failed`' and field '`reason`' contains the error cause.

```
{
  "status": "failed",
  "reason": "Unknown memory type"
}
```

After starting the export process, monitor its status using the Get Process Status method.

Search Translations

End Point: `[TMEngine URL]/create`

Default: `http://localhost:8000/TMServer/search`

Send a '`POST`' request to the method end point with these parameters in a JSON body:

Field	Required	Content
`id`	yes	ID of the memory where the search should be performed
`text`	Yes	Text string to search
`srcLang`	Yes	Source language code
`tgtLang`	Yes	Target language code
`similarity`	Yes	Integer value indicating the lowest similarity percentage to include in results
`caseSensitive`	Yes	Boolean value indicating whether the search should be case sensitive or not

Example:

```
{
  "id": "1572538708492",
  "text": "tax compliance",
  "srcLang": "en-GB",
  "tgtLang": "fr-FR",
  "similarity": 70,
  "caseSensitive": false
}
```

The server responds with a JSON object containing two fields.

On success, field '`status`' is set to '`OK`' and field '`process`' contains the ID of the background search process that was initiated.

```
{
  "process": "1572531573026",
  "status": "OK"
}
```

On error, field '`status`' is set to '`failed`' and field '`reason`' contains the error cause.

```
{
  "status": "failed",
  "reason": "Unknown memory type"
}
```

After starting the search process, monitor its status using the Get Process Status method.

On successful completion, the result will contain an array of similar segments in the `data` field.

Example:

```
{
  "result": "Completed",
  "data": {
    "matches": [
```

```
    {
      "similarity": 71,
      "origin": "1572538708492",
      "source": "<tuv xml:lang="en-GB"><seg>Non-compliance</seg></tuv>",
      "target": "<tuv xml:lang="fr-FR"><seg>Violation</seg></tuv>",
      "properties": {
        "creationdate": "20070126T082848Z",
        "subject": "Taxes",
        "x-Origin": "TM",
        "project": "Main TM",
        "changedate": "20070126T082848Z",
        "tuid": "1546700322331",
        "creationid": "MC",
        "changeid": "MC",
        "lastusagedate": "20070126T082848Z",
        "customer": "ACME Auditors"
      }
    }, {
      "similarity": 73,
      "origin": "1572538708492",
      "source": "<tuv xml:lang="en-GB"><seg>Legal Compliance</seg></tuv>",
      "target": "<tuv xml:lang="fr-FR"><seg>Conformité légale</seg></tuv>",
      "properties": {
        "creationdate": "20160725T141611Z",
        "x-ConfirmationLevel": "ApprovedTranslation",
        "subject": "Taxes",
        "x-Origin": "TM",
        "project": "Main TM",
        "changedate": "20160727T093143Z",
        "tuid": "1546700366038",
        "creationid": "Aqcis9\Aqcis",
        "changeid": "FG",
        "lastusagedate": "20160727T093143Z",
        "customer": "ACME Auditors"
      }
    }, {
      "similarity": 100,
      "origin": "fluenta",
      "source": "<tuv xml:lang="en-GB"><seg>tax compliance</seg></tuv>",
      "target": "<tuv xml:lang="fr-FR"><seg>Conformité fiscale</seg></tuv>",
      "properties": {
        "creationdate": "20171004T111450Z",
        "subject": "Taxes",
        "project": "Main TM",
        "changedate": "20171004T111450Z",
        "tuid": "1546700370945",
        "changeid": "translator2",
        "usagecount": "1",
        "x-ConfirmationLevel": "Translated",
        "x-Origin": "TM",
        "creationid": "translator2",
```

```
          "lastusagedate": "20171006T103930Z",
          "customer": "ACME Auditors"
        }
      }
    ],
  },
  "status": "OK"
}
```

Concordance Search

End Point: `[TMEngine URL]/concordance`

Default: `http://localhost:8000/TMServer/concordance`

Send a '`POST`' request to the method end point with these parameters in a JSON body:

Field	Required	Content
`id`	yes	ID of the memory where the search should be performed
`text`	Yes	Text string to search
`srcLang`	Yes	Source language code
`limit`	Yes	Integer value indicating the maximum number of matches to include
`isRegexp`	Yes	Boolean value indicationg wether the search text should be treated as a regular expression
`caseSensitive`	Yes	Boolean value indicating whether the search should be case sensitive or not

Example:

```
{
  "id": "fluenta",
  "text": "segment",
  "srcLang": "en",
  "limit": 5,
  "isRegexp": false,
  "caseSensitive": true
}
```

On success, field '`status`' is set to '`OK`' and field '`process`' contains the ID of the background search process that was initiated.

```
{
  "process": "1572531573026",
  "status": "OK"
}
```

On error, field '`status`' is set to '`failed`' and field '`reason`' contains the error cause.

```
{
  "status": "failed",
  "reason": "Unknown memory type"
}
```

After starting the search process, monitor its status using the Get Process Status method.

On successful completion, the result will contain an array of <tu> elements that contain the searched text in the `data` field.

Example:

```
{
  "result": "Completed",
  "data": {
    "entries": [
      "<tu creationid="rmraya" creationdate="20161225T150949Z" creationtool="Swordfish"
      creationtoolversion="3.3-8" tuid="-1247472893-0-1586928971">
      <prop type="project">Fluenta</prop>
      <tuv xml:lang="es"><seg>Hay segmentos con errores de etiquetas.</seg></tuv>
      <tuv xml:lang="en"><seg>There are segments with tag errors.</seg></tuv></tu>"
    ],
  },
  "status": "OK"
}
```

Rename Memory

End Point: `[TMEngine URL]/rename`

Default: `http://localhost:8000/TMServer/rename`

Send a '`POST`' request to the method end point with these parameters in a JSON body:

Field	Required	Content
`id`	Yes	ID of the memory to rename
`name`	Yes	New name for the memory

Note

Only memories of type '`MapDbEngine`' can be renamed.

Example:

```
{
  "id": "1568163112478",
  "name": "Updated Memory Name"
}
```

The server responds with a JSON object containing two fields.

On success, field '`status`' is set to '`OK`'.

Example:

```
{
  "status": "OK"
}
```

On error, field '`status`' is set to '`failed`' and field '`reason`' contains the error cause.

Example:

```
{
  "status": "failed",
  "reason": "Wrong memory type"
}
```

Delete Memory

End Point: `[TMEngine URL]/delete`

Default: `http://localhost:8000/TMServer/delete`

Send a '`POST`' request to the method end point with this parameter in a JSON body:

Field	Required	Content
`id`	Yes	ID of the memory to delete

Example:

```
{
  "id": "1568163112478"
}
```

The server responds with a JSON object. On success, field '`status`' is set to '`OK`'.

```
{
  "status": "OK"
}
```

On error, field '`status`' is set to '`failed`' and field '`reason`' contains the error cause.

Example:

```
{
  "status": "failed",
  "reason": "Unknown memory"
}
```

Stop Server

End Point: `[TMEngine URL]/create`

Default: `http://localhost:8000/TMServer/stop`

Send a '`GET`' request to the method end point.

The server responds with a JSON object. On success, field '`status`' is set to '`OK`'.

Example:

```
{
  "status": "OK"
}
```

On error, field '`status`' is set to '`failed`' and field '`reason`' contains the error cause.

Example:

```
{
  "status": "failed",
  "reason": "Error connecting to database"
}
```

Java Library

Java Applications

TMEngine can be embedded in a Java application that needs to deal with translation memories.

Dependencies

TMEngine is built on top of OpenXLIFF Filters, an open source project that provides support for managing the XML side of TMX documents.

OpenXLIFF includes the last Java release of MapDB, modified to work with modules in Java 11 or newer.

MariaDB JDBC driver is also included as an optional dependency. MariaDB code does not support mudularization at this moment.

Java 11 is used to compile and link TMEngine binaries for distribution. Newer versions of Java can also be used.

Supported Databases

Two options are currently available for storing TM data:

- **MapDB** databases
- **MySQL** or **MariaDB** databases.

ITmEngine Interface

The interface `com.maxprograms.tmengine.ITmEngine` provides the set of methods that applications can use to interact with TM data.

Applications that use TMEngine can work with the classes that implement `ITmEngine` or implement new versions that work with other database systems.

The classes that implement ITmEngine interface are:

- `com.maxprograms.tmengine.MapDbEngine`
- `com.maxprograms.tmengine.SQLEngine`

Interface Methods

The methods exposed by `ITmEngine` interface are:

- ```
  public abstract String getName();
  ```

  Returns the name of the engine instance.

- ```
  public abstract String getType();
  ```

  Returns the `ITmEngine` type used for storing data. Possible values are:

  - MapDbEngine
  - SQLEngine

- `public abstract void close() throws IOException, SQLException;`

  Closes the engine instance.

- ```
  public abstract int storeTMX(String tmxFile, String project, String customer,
                  String subject) throws SAXException, IOException,
                  ParserConfigurationException, SQLException;
  ```

  Imports a TMX document located at "`tmxFile`" and associates its data with "`project`", "`customer`" and "`subject`" properties

- ```
  public abstract void exportMemory(String tmxfile, Set<String> langs, String
                  srcLang, Map<String, String> properties) throws IOException,
                  SAXException, ParserConfigurationException, SQLException;
  ```

  Exports engine data to a TMX document located at "`tmxfile`".

  The "`langs`" argument may contain a set of language codes. If "`langs`" is not null, only the entries with the given language codes are exported.

  The "`srcLang`" argument indicates the source language assigned to the TMX file. It can be one of the languages from the data set (see "getAllLanguages()") or the string "`*all*`".

  The "`properties`" argument may contain a set of property-value pairs to be set in the exported TMX file.

- `public abstract void flag(String tuid) throws SQLException;`

  Adds the property "`x-flag`" and sets its value to "`SW-Flag`" to the translation unit identified by the "`tuid`" argument.

- `public abstract Set<String> getAllLanguages() throws SQLException;`

  Returns a collection containing all language codes used in the engine's data.

- `public abstract Set<String> getAllClients() throws SQLException;`

  Returns a collection containing all values assigned to the "`client`" property.

- `public abstract Set<String> getAllProjects() throws SQLException;`

  Returns a collection containing all values assigned to the "`project`" property.

- `public abstract Set<String> getAllSubjects() throws SQLException;`

  Returns a collection containing all values assigned to the "`subject`" property.

- ```
  public abstract List<Match> searchTranslation(String searchStr, String
                  srcLang, String tgtLang, int similarity, boolean
                  caseSensitive) throws IOException, SAXException,
                  ParserConfigurationException, SQLException;
  ```

  Returns a list of possible translations of the "`searchStr`" argument.

  The search result is restricted to entries with the source language indicated by "`srcLang`" and target language indicated by "`tgtLang`" whose similarity to the given text is greater or equal to the value of the "`similarity`" argument.
  ```

The "caseSensitive" argument indicates whether the search engine should consider letter case differences or not.

- ```
  public abstract List<Element> concordanceSearch(String searchStr,
                  String srcLang, int limit, boolean isRegexp, boolean
                  caseSensitive) throws IOException, SAXException,
                  ParserConfigurationException, SQLException;
  ```

  Returns a list of all translation units (<tu> elements) that contain the text indicated in "searchStr" argument.

  Searches are performed against the translation unit variant (<tuv> element) with language set to "srcLang".

  Search result contains at most "limit" entries. Returned data is in no particular order.

  Argument "isRegexp" indicates whether the "searchStr" parameter should be considered a regular expression that matches the whole segment.

  The "caseSensitive" argument indicates whether the search engine should consider letter case differences or not.

- ```
  public abstract void storeTu(Element tu) throws IOException, SQLException;
  ```

  Stores translation unit "tu" into the database, overwriting any existing <tu> element with the same "id" attribute.

- ```
  public abstract void commit() throws SQLException;
  ```

  Flushes to disk any data held in memory and not written yet.

- ```
  public abstract Element getTu(String tuid) throws IOException,
                  SAXException, ParserConfigurationException, SQLException;
  ```

  Returns the translation unit (<tu> element) that has the "id" attribute set to the "tuid" argument.

- ```
  public abstract void removeTu(String tuid) throws IOException, SAXException,
                  ParserConfigurationException, SQLException;
  ```

  Removes from the database the <tu> element that has the "id" attribute set to the "tuid" argument.

- ```
  public abstract void deleteDatabase() throws IOException, SQLException;
  ```

  - When used with "MapDbEngine" instances, closes the engine and removes all associated files from disk;
  - When used with "SQLEngine" instances, closes the engine and drops the associated database from the SQL server.