TMEngine

An Open Source Translation Memory Manager

Rodolfo M. Raya

rmraya@maxprograms.com

Maxprograms
https://www.maxprograms.com

Table of Contents

Overview

TMEngine

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

TMEngine can be used either as an embedded library that manages translation memories in a Java application or as a standalone TM server via its REST API.

Java Library

REST API

REST Methods

The REST methods that TMEngine's server support are:

- Create Memory
- List Memories
- Open Memory
- Close Memory
- Import TMX File
- Process Status
- Export TMX File
- Search Translations
- Search Translations
- 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

Field	Required	Content
`owner`	No	Text string used to identify the owner of the memory. Default value is the user name of the user running the server.
`type`	No	Type of engine to use. Possible values are: '`MapDbEngine`' (default) and '`SQLEngine`'

Example:

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

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

```
      "owner": "rmraya",
      "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 parameters 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]/create`

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

Send a '`POST`' request to the method end point with this parameters 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"
}
```

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

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

Example:

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

Process Status

End Point: `[TMEngine URL]/create`

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

Send a `POST` request to the method end point with this parameters 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 containing two or three fields.

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. Process outcome is provided 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 the process 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`

Search Translations

End Point: `[TMEngine URL]/create`

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

Concordance Search

End Point: `[TMEngine URL]/create`

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

Rename Memory

End Point: `[TMEngine URL]/create`

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

Delete Memory

End Point: `[TMEngine URL]/create`

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

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"
}
```