t5memory - translate5 TM service - REST API

Overview and API introduction

In this document the translate5 TM service REST interface is described.

The translate5 TM service is build by using the OpenTM2 Translation Memory Engine.

It provides the following functionality:

- import new openTM2-TMs
- delete openTM2-TMs
- create new empty openTM2-TM
- import TMX
- open TM and close TM: not possible see extra section in this document.
- query TM for Matches: one query per TM, not quering multiple TMs at once.
- query TM for concordance search
- save new entry to TM

This can be achieved by the following specification of a RESTful HTTP Serive, the specification is given in the following form:

1. URL of the HTTP Resource, where servername and an optional path prefix is configurable.
2. HTTP Method with affected functionality
3. Brief Description
4. Sent and returned Body.

Request Data Format:

The transferred data in the requests is JSON and is directly done in the request body.

URL Format:

In this document, the OpenTM2 is always assumed under http://opentm2/.

To rely on full networking features (proxying etc.) the URL is configurable in Translate5 so that the OpenTM2 instance can also reside under http://xyz/foo/bar/.

Errors

For each request, the possible errors are listed below for each resource. In case of an error, the body should contain at least the following JSON, if it is senseful the attributes of the original representation can be added.

{

errors: [{errorMsg: 'Given tmxData is no TMX.'}]

}

Values	
%service%	Name of service(default - t5memory, could be changed in t5m3mory.conf file
%tm_name%	Name of Translation Memory
Example	http://localhost:4040/t5memory/examle_tm/fuzzysearch/?

Endpoints overview

1	Get the list of TMs	Returns JSON list of TMs	**GET**	/%service%/
2	Create TM	Creates TM with the provided name	**POST**	/%service%/
3	Create/Import TM in internal format	Import and unpack base64 encoded archive of .TMD, .TMI, .MEM files. Rename it to provided name	**POST**	/%service%/
4	Delete TM	Deletes .TMD, .TMI, .MEM files	**DELETE**	/%service%/%tm_name%/f
5	Import TMX into TM	Import provided base64 encoded TMX file into TM	**POST**	/%service%/%tm_name%/import
6	Export TMX from TM	Creates TMX from tm. Encoded in base64	**GET**	/%service%/%tm_name%/
7	Export in Internal format	Creates and exports archive with .TMD, .TMI, .MEM files of TM	**GET**	/%service%/%tm_name%/
8	Status of TM	Returns status\import status of TM	**GET**	/%service%/%tm_name%/status
9	Fuzzy search	Returns enrties\translations with small differences from requested	**POST**	/%service%/%tm_name%/fuzzysearch
10	Concordance search	Returns entries\translations that contain requested segment	**POST**	/%service%/%tm_name% /concordancesearch
11	Entry update	Updates entry\translation	**POST**	/%service%/%tm_name%/entry
12	Entry delete	Deletes entry\translation	**POST**	/%service%/%tm_name%/entrydelete
13	Save all TMs	Flushes all filebuffers(TMD, TMI files) into the filesystem	**GET**	/%service%_**service**/savetms
14	Shutdown service	Flushes all filebuffers into the filesystem and shutting down the service	**GET**	/%service%_**service**/shutdown
15	Test tag replacement call	For testing tag replacement	**POST**	/%service%_service/tagreplacement

Available end points

List of TMs

Purpose	Returns JSON list of TMs
Request	**GET** /%service%/
Params	-

Response

```
Response example:{
    [
        {name:examle_tm
        },
        {name:mem_gt_issue
        }
    ]
}
```

Create TM

Purpose	Creates TM with the provided name
Request	**Post** /%service%/%tm_name%/
Params	Required: name, sourceLang

Response

```
Request example
{    "name": "examle_tm",
     "sourceLang": "bg-BG"
   ["data": "base64_encoded_archive_see_import_in_internal_format"]
   ["loggingThreshold": 0]
}

Response example:Success:{
"name": "examle_tm",
}
TM already exists:
{
  "ReturnValue": 1,
  "ErrorMsg": ""
}
```

Create/Import TM in internal format

Purpose	Import and unpack base64 encoded archive of .TMD, .TMI, .MEM files. Rename it to provided name
Request	**POST** /%service%/%tm_name%/
Params	{ "name": "examle_tm", "sourceLang": "bg-BG" , "data":"base64EncodedArchive" }

Response

```
Request example:{ "name": "mem_internal_format", "sourceLang": "bg-BG", "data":"
UEsDBBQACAgIAPmrhVQAAAAAAAAAAAAAAAAWAAQAT1RNXy1JRDE3NS0wXzJfNV9iLk1FTQEAAADtzqEKgDAQgOFTEHwNWZ5swrAO0SBys6wfWx
FBDILv6uOI2WZQw33lr38GbvRIsm91baSiigzFEjuEb6XHEK\/myX0PXtXsyxS2OazwhLDWeVTaWgEFMMYYY\
/9wAlBLBwhEWTaSXAAAAAAAAAAACAAAAAAAAFBLAwQUAAgICAD5q4VUAAAAAAAAAAAAAAAAAFgAEAE9UTV8tSUQxNzUtMF8yXzVfYi5UTUQBAAA
A7d3Pa5JxHMDxz+Ns09phDAYdPfaDyQqWRcYjS9nGpoYZhBeZMCISW2v2g5o6VkqQONk\
/0KVzh4IoKAovnboUo1PHbuuwU8dSn8c9Pk2yTbc53y+R5\/P9fL7P1wf5Ps9zep5vIOy3iMiSiPLn0yPrQ7In+rStTQARi\
/bV9chEyHcxGPIKAGDnPonl21SsHNmUYNgfHZ70nnKNDo9ET0dHozFn2L+Ll9uxZPzazPz1mYQAAAAAAAAAAAAAAAAAAAAAAAAAANDtBkXRoj5
Zk7OqSFZ9q35Vn6khNa6W2wAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAdBKbKHK4Em1omT5DxV6J7FrmkKFypBKt9FczvYaKtr+2DLpiqPTWVayGi
q2uYjFUpC7VI6aElN8F8JPn\/QEAAAAAAAAAAAAAAAAAAAAAAAAAAAAA2ANW7U0Ag9Iv60MnT4j8uLBZ\
/X5+7dxn1ztX6Uy5AgAAAAAAAAAAAAAAAAgA6nL1qFjmc1rAO2IwNN9bL9u4ulVUeEfcQqQAfxSNtltshZaytB7jalZZ2a5KhFGT3Qr\
/ztv1pkzAnP1v06+F7UxL22tRzSNf6aFq08MdoiY078\/znmkTZo5Qm2YdoOSLSyDdbaVUop\/Cj3cDm14I6\
/uqf++nDUN1u4lS+k9MbKXL4QK72+775U+phOpp8sucdK728X5nK5hVT+weJqbTiHjMiNzWG1yNxWvI8rvxZ9cTfycj71NH1nsZgbf54uJlKry
Wy6GFlueBT6xHrzJRupDqkPXc9eyyduJmbLkf6\/mlYRDgQDPtO++3\/uYvsazANfYHx68vLEsSvOKedxqa\/hAGowD4Jh\/1X\
/dH1X5sEBZpoH6E6\
/AVBLBwj3gRyzjAIAAAAAAAAAAAEAAAAAAFBLAwQUAAgICAD5q4VUAAAAAAAAAAAAAAAAFgAEAE9UTV8tSUQxNzUtMF8yXzVfYi5UTUkBAAAA7
d3PS9NhHMDxz\/Y1nbp0zfw2Vw6CEjooJkkFPs9DZZaFCiIRHRxKoJUIFXk06iB0kS5Fvw6dhDp28FDgOSqiIKQ\
/ICQMhIIuYVnJt2f7eK2M2Ps1xp49b8Y+fP6ArXegJy4iV0RiPx6BNAXyT6ysrKhXlLZ49PwlkKP9hw\
/19XcKAOD3PZX42+PDP0+JWN9AT765u3P33vbm1nxbvj0\
/3DLQ0y3r5uClsZGhC2eGxgUAAAAAAAAAAAAAAAAAAAAAAAAAgFKXllh0ahQbLHeInDb3Xc6NWrF77Jibcr22zC2YY6bVLNoX5qp97Pa5SbPc8
ci8sqHpd1k7a2+ZN+6eFQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAD4YxISk8bVUyq6eVa905dtqtxO3fBlqyqnkrW+ZFVZCGp8aVDl9ZeELxlVjhR
NsEWVa+UffAlVuf78rC\/1eoK20JfNqnzt3OhLnSp1DZW+bFJl\/467vqRUuVxV5UutKts\
/JX2pUWUyXvie9OopE5U7QWEHSfWZXdmPvlSr8i75xJcqVT7fPOdLpSqj5+t9Sahy8UBhOxWqLEph6nJVHhZNvUFPXbS3MlXyYWFvgSon3xf2F
ldlpGiCmCoPiiYQVbLR3or\
/ZT0tS04AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAMC6K4t+ZSAtOWkKQpOSeTfnZty0m3CDrsu1uNB9swv2pZ21IlN23J6w1uZsuV0y82bOzJhpM
2EGTZdpMaERAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAPjrUmteK0RypXifid5n1tyX6j7+9\
/vvUEsHCGo104BhAgAAAAAAAAAAQAAAAAAUEsBAgAAFAAICAgA912FVERZNpJcAAAAAAgAABYABAAAAAAAAAAALSBAAAAAE9UTV8tSUQxNzU
tMF8yXzVfYi5NRU0BAAAAUEsBAgAAFAAICAgA\
/F2FVPeBHLOMAgAAAAABABYABAAAAAAAAAAAALSBrAAAAE9UTV8tSUQxNzUtMF8yXzVfYi5UTUQBAAAAUEsBAgAAFAAICAgA\
/F2FVGo104BhAgAAAAABABYABAAAAAAAAAAAALSBiAMAAE9UTV8tSUQxNzUtMF8yXzVfYi5UTUkBAAAAUEsGBiwAAAAAAAAAHgMtAAAAAAAAAA
AAAwAAAAAAAADAAAAAAAAANgAAAAAAAAAOQYAAAAAAABQSwYHAAAAABEHAAAAAAAAAQAAAFBLBQYAAAAAAwADANgAAAA5BgAAAAA=" }
Response example:{
"name": "examle_tm"
}

TM already exists:
{
  "ReturnValue": 65535,
  "ErrorMsg": ""
}
```

Delete TM

Purpose	Deletes .TMD, .TMI, .MEM files
Request	**Delete** /%service%/%tm_name%/
Params	-

Response

```
Response example:%empty_anyway%
```

Response

```
Response example:%empty_anyway%
```

Import provided base64 encoded TMX file into TM

Purpose	Import provided base64 encoded TMX file into TM. Starts another thead for import. For checking import status use status call
Request	**POST** /%service%/%tm_name%/import
Params	{"tmxData": "base64EncodedTmxFile" }

Response

```
Request example:{
  "tmxData":   "PD94bWwgdmVyc2lvbj0iMS4wIiBlbmNvZGluZz0idXRmLTgiPz4KPHRteCB2ZXJzaW9uPSIxLjQiPgogIDxoZWFkZXIgY3
JlYXRpb250b29sPSJTREwgTGFuZ3VhZ2UgUGxhdGZvcm0iIGNyZWF0aW9udG9vbHZlcnNpb249IjguMCIgby10bWY9IlNETCBUTTggRm9ybWF0
IiBkYXRhdHlwZT0ieG1sIiBzZWd0eXBlPSJzZW50ZW5jZSIgYWRtaW5sYW5nPSJlbi1HQiIgc3JjbGFuZz0iYmctQkciIGNyZWF0aW9uZGF0ZT
0iMjAxNTA4MjFUMDkyNjE0WiIgY3JlYXRpb25pZD0idGVzdCIvPgogIDxib2R5PgogIPHR1IGNyZWF0aW9uZGF0ZT0iMjAxODAyMTZUMTU1MTA1
WiIgY3JlYXRpb25pZD0iREVTS1RPUC1SNTlCT0tCXFBDMiIgY2hhbmdlZGF0ZT0iMjAxODAyMTZUMTU1MTA4WiIgY2hhbmdlaWQ9IkRFU0tUT1
AtUjU5Qk9LQlxQQzIiIGxhc3R1c2FnZWRhdGU9IjIwMTgwMjE2VDE2MTMwNVoiIHVzYWdlY291bnQ9IjEiPgogICAgICA8dHV2IHhtbDpsYW5n
PSJiZy1CRyI+CiAgICAgICAgPHNlZz5UaGUgPHBoIC8+IGVuZDwvc2VnPgogICAgICA8L3R1dj4KICAgICAgPHR1diB4bWw6bGFuZz0iZW4tR0
IiPgogICAgICAgIDxzZWc+RxRoIDxwaCAvPiBkbmU8L3NlZz4KICAgICAgPC90dXY+CiAgICA8L3R1PgogIDwvYm9keT4KPC90bXg+Cg=="
}Response example:Error in case of errorFrom v0_2_15
{ "%tm_name%":"deleted"} in case of success
Check status of import using status call
```

Export TMX from TM

Purpose	Creates TMX from tm.
Request	**GET** /%service%/%tm_name%/
Headers	Accept - applicaton/xml

Response

```
Response example:<?xml version="1.0" encoding="UTF-8" ?>
<tmx version="1.4">
<header creationtoolversion="0.2.14" gitCommit="60784cf * refactoring and cleanup" segtype="sentence"
adminlang="en-us" srclang="en-GB" o-tmf="t5memory" creationtool="t5memory" datatype="xml" />
<body>
  <tu tuid="1" datatype="xml" creationdate="20190401T084052Z">
     <prop type="tmgr:segNum">10906825</prop>
     <prop type="tmgr:markup">OTMXML</prop>
     <prop type="tmgr:docname">none</prop>
     <tuv xml:lang="en-GB">
          <prop type="tmgr:language">English(U.K.)</prop>
          <seg>For > 100 setups.</seg>
     </tuv>
     <tuv xml:lang="de-DE">
          <prop type="tmgr:language">GERMAN(REFORM)</prop>
     <seg>Für > 100 Aufstellungen.</seg>
     </tuv>
     </tu>
   </body>
</tmx>
```

Export in internal format

Purpose	Creates and exports archive with .TMD, .TMI, .MEM files of TM
Request	**GET** /%service%/%tm_name%/
Headers	application/zip

Response

```
Response example:%binary_data%
```

Get the status of TM

Request	**GET** /%service%/%tm_name%/status
Params	-

Response

```
Response example:{
    "tmxImportStatus": "available"
}
The status could be "available", "import" or "failed" if the import had errors. If it's "failed" ErrorMsg
would be added
```

Fuzzy search

Purpose	Returns enrties\translations with small differences from requested
Request	**POST** /%service%/%tm_name%/fuzzysearch
Params	Required: source, sourceLang, targetLang iNumOfProposal - limit of found proposals - max is 20, if 0 use default value '5'

Response

Request example:

Request example:
{ "sourceLang":"en-GB",   "targetLang":"de",   "source":"For > 100 setups.",   ["documentName":"OBJ_DCL-0000000845-004_pt-br.xml"],   ["segmentNumber":""],   ["markupTable":"OTMXUXLF"],   ["context":"395_408"],   ["numOfProposals":20],   ["loggingThreshold": 0]}

Response example:

Success:
{
  "ReturnValue": 0,
  "ErrorMsg": "",
  "NumOfFoundProposals": 1,
  "results": [
  {
     "source": "For > 100 setups.",
     "target": "Für > 100 Aufstellungen.",
     "segmentNumber": 10906825,
     "id": "",
     "documentName": "none",
    "documentShortName": "NONE",
    "sourceLang": "en-GB",
    "targetLang": "de-DE",
    "type": "Manual",
    "matchType": "Exact",
    "author": "",
    "timestamp": "20190401T084052Z",
    "matchRate": 100,
    "markupTable": "OTMXML",
    "context": "",
   "additionalInfo": ""
   }
 ]
}

Not found:

{
"ReturnValue": 133,
"ErrorMsg": "OtmMemoryServiceWorker::concordanceSearch::"
}

Concordance search

Purpose	Returns entries\translations that contain requested segment
Request	**POST** /%service%/%tm_name%/concordancesearch
Params	Required: searchString - what we are looking for , searchType ["Source"|"Target"|"SourceAndTarget"] - where to look iNumOfProposal - limit of found proposals - max is 20, if 0 use default value '5'

Response

Request example:
```
{
    "searchString": "The",
    "searchType": "source",
    ["searchPosition": "",]
    ["numResults": 20,]
    ["msSearchAfterNumResults": 250,]
        ["loggingThreshold": 0]
}
```
Response example:Success:
```
{
  "ReturnValue": 0,
  "NewSearchPosition": null,
  "results": [
  {
     "source": "For > 100 setups.",
     "target": "Für > 100 Aufstellungen.",
     "segmentNumber": 10906825,
     "id": "",
     "documentName": "none",
     "documentShortName": "NONE",
     "sourceLang": "en-GB", rfc5646
     "targetLang": "de-DE", rfc5646
     "type": "Manual",
     "matchType": "undefined",
     "author": "",
     "timestamp": "20190401T084052Z",
     "matchRate": 0,
     "markupTable": "OTMXML",
     "context": "",
     "additionalInfo": ""
   }
  ],
 "ErrorMsg": ""
}
```

Success, but with NewSearchPosition - not all TM was checked, use this position to repeat search:
```
{
  "ReturnValue": 0,
  "NewSearchPosition": "8:1",
  "results": [
  {
     "source": "For > 100 setups.",
     "target": "Für > 100 Aufstellungen.",
     "segmentNumber": 10906825,
     "id": "",
    "documentName": "none",
    "documentShortName": "NONE",
    "sourceLang": "en-GB",
    "targetLang": "de-DE",
    "type": "Manual",
     "matchType": "undefined",
     "author": "",
     "timestamp": "20190401T084052Z",
     "matchRate": 0,
     "markupTable": "OTMXML",
     "context": "",
     "additionalInfo": ""
   }
  ],
 "ErrorMsg": ""
}
```
SearchPosition / NewSearchPositionFormat: "7:1"
First is segmeng\record number, second is target number
The NextSearchposition is an internal key of the memory for the next position on sequential access. Since it is an internal key, maintained and understood by the underlying memory plug-in (for EqfMemoryPlugin is it the record number and the position in one record),
no assumptions should be made regarding the content. It is just a string that, should be sent back to OpenTM2 on the next request, so that the search starts from there.
So is the implementation in Translate5: The first request to OpenTM2 contains SearchPosition with an empty string, OpenTM2 returns than a string in NewSearchPosition, which is just resent to OpenTM2 in the next

```
request.

Not found:{
"ReturnValue": 0,
"NewSearchPosition": null,
"ErrorMsg": ""
}TM not found:{
"ReturnValue": 133,
"ErrorMsg": "OtmMemoryServiceWorker::concordanceSearch::"
}
```

Update entry

Purpose	Updates entry\translation
Request	**POST** /%service%/%tm_name%/entry
Params	Only sourceLang, targetLang, source and target are required

Response

```
Request example:
{
    "source": "The end",
    "target": "The target",
    "sourceLang": "en",
    "targetLang": "de",
    ["documentName": "Translate5 Demo Text-en-de.xlf"],
    ["segmentNumber": 8,]
    ["author": "Thomas Lauria"],
    ["timeStamp": "20210621T071042Z"],
    ["context": "2_2"],
    ["addInfo": "2_2"],
    ["type": "Manual"],
    ["markupTable": "OTMXUXLF"],
        ["loggingThreshold": 0]
}
Response example:
```

Delete entry

Purpose	Deletes entry\translation
Request	**POST** /%service%/%tm_name%/entrydelete
Params	Only sourceLang, targetLang, source, and target are required Deleting based on strict match(including tags and whitespaces) of target and source

Response

```
Request example:
{
  "sourceLang": "bg",
  "targetLang": "en",
  "source": "The end",
  "target": "Eth dne"
  ["documentName": "my file.sdlxliff",]
  ["segmentNumber": 1,]
  ["markupTable": "translate5",]
  ["author": "Thomas Lauria",]
  ["type": "",]
  ["timeStamp": ""],
  ["context": "",]
   ["addInfo": ""] ,  ["loggingThreshold": 0]
}
```

Save all TMs

Purpose	Flushes all filebuffers(TMD, TMI files) into the filesystem. Reset 'Modified' flags for file buffers. Filebuffer is a file instance of .TMD or .TMI loaded into RAM. It provides better speed and safety when working with files.
Request	**GET** /%service%_**service**/savetms
Params	-

Response

```
Response example:{
'saved files': '/home/or/.t5memory/MEM/bg_internal_format.TMD; /home/or/.t5memory/MEM/bg_internal_format.TMI;
/home/or/.t5memory/MEM/mem_gt_issue.TMD; /home/or/.t5memory/MEM/mem_gt_issue.TMI; EQFSYSW.PRP; '
}
List of saved files
```

Shutdown service

Purpose	Safely shutting down the service with\without saving all loaded tm files to the disk
Request	**GET** /%service%_**service**/shutdown?dontsave=1
Params	dontsave=1(optional in address) - skips saving tms, for now value doesn't matter, only presence

If try to save tms before closing, would check if there is still import process going on
If there is some, would wait 1 second and check again.
Repeats last step up to 10 min, then closes service anyway.

Response

```
Response example:%Empty%
```

Test tag replacement call

Purpose	Updates entry\translation
Request	**POST /%service%_service/tagreplacement**
Params	Required: src, trg, Optional: req

Response

```
Fuzzy search tag replacement test:
Request example:
{
    "src": "Tap <ph x='1'/>View <ph x='2' />o<bpt i='1' x='3'/> get <ph x='4'>strong</ph>displayed<ph
x='5'>View</ph> two strong<ept i='1' x='6'/>US patents.",
    "trg": "View <ph x='1'/> tap <ph x='2' />to<bpt i='1' x='3'/> got <ph x='4'>strong</ph>dosplayd<ph
x='5'>Veiw</ph> two strong<ept i='1' x='6'/>US patents.",
    "req": "Tap <x id='123'/>View <x id='222' />o<g> get <x id='44'>strong</x>displayed<x id='51'>View</x>
two strong</g>US patents."
}

Response example:
//'1' - request result
//'2' - src result
//'3' - trg result
{
 '1' :'Tap <x id="123"/>View <x id="222"/>o<bx/> get <x id="44"/>displayed<x id="51"/> two strong<ex/>US
patents.',
 '2' :'Tap <x id="123"/>View <x id="222"/>o<g> get <x id="44"/>displayed<x id="51"/> two strong</g>US
patents.',
 '3' :'View <x id="123"/> tap <x id="222"/>to<g> got <x id="44"/>dosplayd<x id="51"/> two strong</g>US
patents.',
};

Import tag replacement test:
Request example:
{
    "src": "Tap <ph/>View <ph/>o<bpt/> get <ph>strong</ph>displayed<ph>View</ph> two strong<ept/>US patents.",
    "trg": "View <ph/> tap <ph/>to<bpt/> got <ph>strong</ph>dosplayd<ph>Veiw</ph> two strong<ept/>US
patents.",
}
Response example:
{
 '1' :'Tap <ph x="1"/>View <ph x="2"/>o<bpt x="3" i="1"/> get <ph x="4"/>displayed<ph x="5"/> two strong<ept
x="6" i="1"/>US patents.',
 '2' :'View <ph x="1"/> tap <ph x="2"/>to<bpt x="3" i="1"/> got <ph x="4"/>dosplayd<ph x="5"/> two strong<ept
x="6" i="1"/>US patents.',
};
```

Configuration of service

You can configure the service in ~/.t5service/t5memory.conf

Logging

Level	Mnemonic	Description
0	DEVELOP	could make code work really slow, should be used only when debugging some specific places in code, like binary search in files, etc.
1	DEBUG	logging values of variables. Wouldn't delete temporary files(In MEM and TMP subdirectories), like base64 encoded\decoded tmx files and archives for import\export
2	INFO	logging top-level functions entrances, return codes, etc. Default value.
3	WARNING	logging if we reached some commented or hardcoded code. Usually commented code here is replaced with new code, and if not, it's marked as ERROR level
4	ERROR	errors, why and where something fails during parsing, search, etc
5	FATAL	you shouldn't reach this code, something is really wrongOther values would be ignored. The set level would stay the same till you change it in a new request or close the app. Logs suppose to be written into a file with date\time name under ~/.OtmMemoryService/Logs and errors /fatal are supposed to be duplicated in another log file with *FATAL* suffices
6	TRANSACTION	- Logs only things like begin\end of request etc. No purpose to setup this hight

Logging could impact application speed very much, especially during import or export.
You can setup the logging level from the config file or in any POST JSON request by attaching a parameter to a JSON object
[loggingThreshold:"2"]
Like here

POST http://localhost:4040/t5memory/example_tm/

{
sourceLang: “en”, // the source language is required for a new TM
name: „TM Name“,
loggingThreshold:"2"
}
This would set the logging level to INFO just before the main work of creating mem endpoint starts

Or in t5memory.conf file in line
logLevel=0
Would set the log level to DEVELOP, **this would be applied only after restarting of service**

Working directory

Path	Description
~/.t5memory	The main directory of service. Should always be under the home directory. Consists of nested folders and **t5memory.conf** file(see **Config file**). All directories\files below are nested
LOG	IIncludes log files. It should be cleanup manualy. One session(launch of service) creates two files Log_Thu May 12 10:15:48 2022 .log and Log_Thu May 12 10:15:48 2022 .log_IMPORTANT Last have logs reduced to level Warning and higher.
MEM	Main data directory. All tm files is stored here. One TM should include .TMD(data file), .TMI(index file), .MEM(properties file) with the same name as TM name
TABLE	Services reserved readonly folder with tagtables, languages etc.
TEMP	For temporary files that were created for mainly import\export. On low debug leved(DEVELOP, DEBUG) should be cleaned manualy
t5memory.conf	Main config file(see config file)

Config directory should be located in a specific place

Config file

field	default	Description
name	t5memory	name of service that we use under %service% in address
port	8080	service port
timeout	3600	service timeout
threads	1	
logLevel	2	logLevel - > see logging
AllowedRAM_MB	1500	Ram limit to operate openning\closing TM(see Openning and closing TM) Doesn't include services RAM in Megabytes
TriplesThreshold	33	Level of pre-fuzzy search filtering based on combinations of triples of tokens(excluding tags). Could impact fuzzy search perfomance. For higher values service is faster, but could skip some segments in result. Not always corelated with resulted fuzzyRate

Config file should be located under **~/.t5memory/t5memory.conf**

Anyway, all field has default values so the service could start without the conf file

Reading\applying configs happen only once at service start

Once service started you should be able to see setup values in logs.
Config file example:

Response

```
name=t5memory
port=4040
timeout=3600
threads=1
logLevel=0
AllowedRAM_MB=200
TriplesThreshold=5
```

Conceptional information

Openning and closing TM

In first concept it was planned to implement routines to open and close a TM. While concepting we found some problemes with this approach:

- First one is the realization: opening and closing a TM by REST would mean to update the TM Resource and set a state to open or close. This is very awkward.
- Since in translate5 multiple tasks can be used to the same time, multiple tasks try to access one TM. Closing TMs is getting complicated to prevent race conditions in TM usage.
- Since OpenTM2 loads the whole TM in memory, OpenTM2 must control itself which TMs are loaded or not.

This leads to the following conclusion in implementation of opening and closing of TMs:

OpenTM2 has to automatically load the requested TMs if requested. Also OpenTM2 has to close the TMs after a TM was not used for some time. That means that OpenTM2 has to track the timestamps when a TM was last requested.

Concept endpoints, not implemented

http://opentm2/translationmemory/[TM_Name]/openHandle

GET – Opens a memory for queries by OpenTM2

Note: This method is not required as memories are automatically opened when they are accessed for the first time.

http://opentm2/translationmemory/[TM_Name]/openHandle

DELETE – Closes a memory for queries by OpenTM2

Note: This method is not required as memories are automatically opened when they are accessed for the first time.

For now we open TM in case of call to work with it. TM stays opened till the shutdown we wouldn't try to open more TM's, exceeding the RAM limit setupped in config file.
In that case we would close TM in order of longest not used, till we would fit in limit including TM that we try to open.
 TM size is calcucated basicaly as sum .TMD and .TMI files
Ram limit doesn't include service RAM and temporary files

TAG REPLACEMENT

Tag replacement

Pseudocode for tag replacement in import call:

TAG_REPLACEMENT PSEUDO CODE

This is the pseudo code, that was used as a discussion base for finding the right algorithm for implementation. It was not exactly implemented like this, but it's logic should be valid and can be used to understand, what should be going on.

Pseudocode for tag replacement in import call:

```
TAG_REPLACEMENT PSEUDOCODE

struct TagInfo
 {
    bool fPairTagClosed = true;           // false for bpt tag - waiting for matching ept tag. If we'll find
matching tag -> we'll set this to true
    bool fTagAlreadyUsedInTarget = false; // would be set to true if we would already use this tag as matching
for target

    // this we generate to save in TM. this would be saved as <{generated_tagType} [x={generated_x}] [i=
{generated_i}]/>.
    // we would skip x attribute for generated_tagType=EPT_ELEMENT and i for generated_tagType=PH_ELEMENT
    int generated_i = -1;                 // for pair tags - generated identifier to find matching tag. the same
as in original_i if it's not binded to other tag in segment
    int generated_x = -1;                 // id of tag. should match original_x, if it's not occupied by other
tags
    TagType generated_tagType = UNKNOWN_ELEMENT; // replaced tagType, could be only PH_ELEMENT, BPT_ELEMENT,
EPT_ELEMENT

    // this cant be generated, only saved from provided data
    int original_i = -1;        // original paired tags i
    int original_x = -1;        // original id of tag
    TagType original_tagType = UNKNOWN_ELEMENT;  // original tagType, could be any tag
  };
}

TagType could be one of the values in enum:

[

  BPT_ELEMENT EPT_ELEMENT G_ELEMENT  HI_ELEMENT SUB_ELEMENT BX_ELEMENT  EX_ELEMENT

  //standalone tags
  BEGIN_STANDALONE_TAGS PH_ELEMENT  X_ELEMENT  IT_ELEMENT  UT_ELEMENT
]

we use 3 lists of tags
SOURCE_TAGS
TARGET_TAGS
REQUEST_TAGS
as id we understand one of following attributes(which is present in original tag) : 'x', 'id'
as i  we understand one of following attributes(which is present in original tag) : 'i', 'rid'
all single tags we understand as ph_tag
all opening pair tags we understand as bpt_tag
all closing pair tags we understand as ept_tag
-1 means that value is not found/not used/not provided etc.
for ept tags in generated_id  we would use generated_id from matching bpt tag
    if matching bpt tag is not found -> ???

TagType could be set to one of following values


TAG REPLACEMENT USE CASES {
    IMPORT{
        SOURCE_SEGMENT{
            <single tags> -> would be saved as <ph>{ // for ph and all single tags
                if(type == "lb"){
                    replace with newline
                }else{
                    generate next generated_id incrementally
                    ignore content and attributes(except id) if provided
                    set  generated_tagType to PH_ELEMENT

                    save original_tagType for matching
                    if id provided -> save as original_id for matching

                    save tag to SOURCE_TAGS
                }
            }

            <opening pair tags> -> would be saved as <bpt>{
                original type is <bpt>{
                    generate generated_i  incrementally in source segment
                    generate generated_id incrementally

                    set  generated_tagType to BPT_ELEMENT
                    save original_i (should that always be provided??)
                    save original_id if provided (should that always be provided??)
                    set  fPairTagClosed to false; // it would be set to true if we would use this tag as matching
                    set  original_type as BPT_ELEMENT

                    save tag to SOURCE_TAGS
                }
```

```
                original type is <bx>{
                    generate generated_i  incrementally in source segment
                    generate generated_id incrementally
                    set  generated_tagType to BPT_ELEMENT

                    save original_i (should that always be provided??)
                    save original_id if provided (should that always be provided??)
                    set  fPairTagClosed to false; // it would be set to true if we would use this tag as matching
                    set  original_type as BX_ELEMENT

                    save tag to SOURCE_TAGS
                }

                original type is other openning pair tags(like <g>){
                    generate generated_i  incrementally in source segment
                    generate generated_id incrementally
                    set  generated_tagType to BPT_ELEMENT

                    set  fPairTagClosed to false; // it would be set to true if we would use this tag as matching
                    save tag type as original_tagType;

                    save tag to SOURCE_TAGS
                }

            }

            <closing pair tags> -> would be saved as <ept>{
                original type is <ept>{
                    search for matching bpt_tag in saved tags

                        //should we look in reverse order?
                        looking in SOURCE_TAGS for matchingTag which have  [
                                                        matchingTag.fPairTagClosed    == false
                                                  AND matchingTag.generated_tagType == BPT_ELEMENT
//all OPENING PAIR TAGs always has BPT_ELEMENT here
                                                  AND matchingTag.original_tagType  == BPT_ELEMENT
                                                  AND matchingTag.original_i        == our_ept_tag.original_i
                                                ]
                    if found
                        set matchingTag.fPairTagClosed to true to eliminate matching one opening tag for
different closing tags
                        set our_ept_tag.i  to matchingTag.i
                        set our_ept_tag.id to matchingTag.id
                    else
                        generate next our_ept_tag.generated_i  incrementally in source segment // in every segment
(target, source, request) i starts from 1
                        generate next our_ept_tag.generated_id incrementally                  // should be
unique across target, source and request segments

                    save tag in SOURCE_TAGS
                }

                original type is <ex>{
                    search for matching bpt_tag in saved tags

                        //should we look in reverse order?
                        looking in SOURCE_TAGS for matchingTag which have  [
                                                        matchingTag.fPairTagClosed    == false
                                                  AND matchingTag.generated_tagType == BPT_ELEMENT
//all OPENING PAIR TAGs has BPT_ELEMENT here
                                                  AND matchingTag.original_tagType  == BX_ELEMENT
                                                  AND matchingTag.original_i        == our_ept_tag.original_i
                                                ]
                    if found
                        set matchingTag.fPairTagClosed to true to eliminate matching one opening tag for
different closing tags
                        set our_ept_tag.i  to matchingTag.i
                        set our_ept_tag.id to matchingTag.id
                    else
                        generate next our_ept_tag.generated_i  incrementally in source segment  // in every
segment(target, source, request) i starts from 1
                        generate next our_ept_tag.generated_id incrementally                   // should be
unique across target, source and request segments

                    save tag in SOURCE_TAGS
                }

                original type is others closing pair tags(like </g>){
                    search for matching bpt_tag in saved tags:
                        looking in SOURCE_TAGS in REVERSE for matchingTag which have
                                                                [       matchingTag.fPairTagClosed    == false
                                                                  AND matchingTag.generated_tagType ==
BPT_ELEMENT         //OPENING_PAIR_TAG
```

```
                                                                    AND matchingTag.original_tagType  ==
our_tag.original_tagType
                                                                ]
                    if found
                        set matchingTag.fPairTagClosed to true to eliminate matching one opening tag for
different closing tags
                        set our_tag.generated_i  to matchingTag.i
                        set our_tag.generated_id to matchingTag.id
                    else
                        generate next our_tag.generated_i  incrementally in source segment // in every segment
(target, source, request) i starts from 1
                        generate next our_tag.generated_id incrementally                  // should be unique
across target, source and request segments

                    save tag in SOURCE_TAGS
                }

            }
        }



        TARGET_SEGMENT{
            <single tags> -> would be saved as <ph>{ // for ph and all single tags
                if(type == "lb"){
                    replace with newline
                }else{
                    ignore content and attributes(except id) if provided
                    save original_tagType for matching

                    if id provided -> save as original_id for matching

                    search for matching ph_tag in saved tags
                        looking in SOURCE_TAGS for matchingTag which have  [
                                                    matchingTag.fTagAlreadyUsedInTarget == false
                                                AND matchingTag.generated_tagType == PH_ELEMENT
//SINGLE TAG
                                                AND matchingTag.original_tagType  == our_ph_tag.
original_tagType
                                                AND matchingTag.original_id       == our_ph_tag.original_id
                                               ]
                    if found
                        set matchingTag.fTagAlreadyUsedInTarget = true
                        set our_ph_tag.generated_id = matchingTag.generated_id // use id generated for source
segment
                    else
                        generate new our_ph_tag.generated_id incrementally(should be unique for SOURCE and
TARGET)

                    save tag in TARGET_TAGS // we should track only opening pair tags in target, so theoretically
can skip this step
                }
            }

            <opening tags> -> would be saved as <bpt>{
                original type is <bpt>{
                    set  generated_tagType to BPT_ELEMENT
                    save original_i (should that always be provided??)
                    save original_id if provided (should that always be provided??)
                    set  fPairTagClosed to false; // it would be set to true if we would use this tag as matching
                    set  original_type as BPT_ELEMENT

                    try to found matching source tag to get generated id:
                        looking in SOURCE_TAGS for matchingTag which have  [
                                                                                matchingTag.
fTagAlreadyUsedInTarget    == false
                                                                              AND matchingTag.
generated_tagType          == BPT_ELEMENT         //all OPENING PAIR TAGs always has BPT_ELEMENT here
                                                                              AND matchingTag.
original_tagType           == BPT_ELEMENT
                                                                              AND matchingTag.
original_id                == our_bpt_tag.original_id
                                                                             ]
                    if found:
                        set  matchingTag.fTagAlreadyUsedInTarget to true
                        generate our_bpt_tag.generated_i  incrementally in target segment
                        set      our_bpt_tag.generated_id to matchingTag.generated_id
                    else:
                        generate our_bpt_tag.generated_i  incrementally // unique between all segments
                        generate our_bpt_tag.generated_id incrementally // unique between all
segments

                    save tag in TARGET_TAGS
```

```
                }

                original type is <bx>{
                    set  generated_tagType to BPT_ELEMENT
                    save original_i (should that always be provided??)
                    save original_id if provided (should that always be provided??)
                    set  fPairTagClosed to false; // it would be set to true if we would use this tag as matching
                    set  original_type as BX_ELEMENT

                    try to found matching source tag to get generated id:
                        looking in SOURCE_TAGS for matchingTag which have  [
                                                                                    matchingTag.
fTagAlreadyUsedInTarget    == false
                                                                                  AND matchingTag.
generated_tagType          == BPT_ELEMENT        //all OPENING PAIR TAGs always has BPT_ELEMENT here
                                                                                  AND matchingTag.
original_tagType           == BX_ELEMENT
                                                                                  AND matchingTag.
original_id                == our_bpt_tag.original_id
                                                                                 ]
                    if found:
                        set  matchingTag.fTagAlreadyUsedInTarget to true
                        generate our_bpt_tag.generated_i  incrementally in target segment
                        set      our_bpt_tag.generated_id to matchingTag.generated_id
                    else:
                        generate our_bpt_tag.generated_i  incrementally // unique between all segments
                        generate our_bpt_tag.generated_id incrementally // unique between all
segments

                    save tag in TARGET_TAGS
                }

                original type is other openning pair tags(like <g>){
                    set  generated_tagType to BPT_ELEMENT
                    we never have here original i attribute
                    save original_id if provided (should that always be provided??)
                    set  fPairTagClosed to false; // it would be set to true if we would use this tag as matching
                    save  original_type

                    try to found matching source tag to get generated id:
                        looking in SOURCE_TAGS for matchingTag which have  [
                                                                                    matchingTag.
fTagAlreadyUsedInTarget    == false
                                                                                  AND matchingTag.
generated_tagType          == BPT_ELEMENT        //all OPENING PAIR TAGs always has BPT_ELEMENT here
                                                                                  AND matchingTag.
original_tagType           == our_tag.original_tagType
                                                                                  AND matchingTag.
original_id                == our_tag.original_id
                                                                                 ]
                    if found:
                        set  matchingTag.fTagAlreadyUsedInTarget to true
                        generate our_tag.generated_i  incrementally in target segment
                        set      our_tag.generated_id to matchingTag.generated_id
                    else:
                        generate our_tag.generated_i  incrementally // unique between all segments
                        generate our_tag.generated_id incrementally // unique between all
segments

                    save tag in TARGET_TAGS
                }

            }

            <closing tags> -> would be saved as <ept>{
                original type is <ept>{
                    try to found matching bpt tag in TARGET_TAGS
                        looking in TARGET_TAGS for matchingTag which have  [
                                                                                    matchingTag.
fPairTagClosed             == false
                                                                                  AND matchingTag.
generated_tagType          == BPT_ELEMENT        //all OPENING PAIR TAGs always has BPT_ELEMENT here
                                                                                  AND matchingTag.
original_tagType           == BPT_ELEMENT
                                                                                  AND matchingTag.original_i
      == our_tag.original_i
                                                                                 ]
                    if found:
                        set      matchingTag.fPairTagClosed to true
                        set      our_tag.generated_id to matchingTag.generated_id
                        set      our_tag.generated_i  to matchingTag.generated_i
                    else:
                        generate our_tag.generated_i  incrementally // unique between all segments
                        generate our_tag.generated_id incrementally // unique between all segments
```

```
                    save tag in TARGET_TAGS // we should track only opening pair tags in target, so theoretically
can skip this step
                }

                original type is <ex>{
                    try to found matching bpt tag in TARGET_TAGS
                        looking in TARGET_TAGS for matchingTag which have  [
                                                                                matchingTag.
fPairTagClosed             == false
                                                                              AND matchingTag.
generated_tagType          == BPT_ELEMENT         //all OPENING PAIR TAGs always has BPT_ELEMENT here
                                                                              AND matchingTag.
original_tagType           == BX_ELEMENT
                                                                              AND matchingTag.
original_i                 == our_tag.original_i
                                                                            ]
                    if found:
                        set      matchingTag.fPairTagClosed to true
                        set      our_tag.generated_id to matchingTag.generated_id
                        set      our_tag.generated_i  to matchingTag.generated_i
                    else:
                        generate our_tag.generated_i  incrementally // unique between all segments
                        generate our_tag.generated_id incrementally // unique between all segments

                    save tag in TARGET_TAGS // we should track only opening pair tags in target, so theoretically
can skip this step
                }

                original type is others closing pair tags(like </g>){

                    search for matching bpt_tag in saved tags:
                        looking in TARGET_TAGS in REVERSE for matchingTag which have
                                                                    [       matchingTag.fPairTagClosed    == false
                                                                      AND matchingTag.generated_tagType ==
BPT_ELEMENT         //OPENING_PAIR_TAG
                                                                      AND matchingTag.original_tagType  ==
our_tag.original_tagType
                                                                    ]
                    if found:
                        set matchingTag.fPairTagClosed to true to eliminate matching one opening tag for
different closing tags
                        set our_tag.generated_i  to matchingTag.i
                        set our_tag.generated_id to matchingTag.id
                    else :
                        generate next our_tag.generated_i  incrementally in target segment // in every segment
(target, source, request) i starts from 1
                        generate next our_tag.generated_id incrementally                   // should be unique
across target, source and request segments

                    save tag in TARGET_TAGS // we should track only opening pair tags in target, so theoretically
can skip this step
                }
            }
        }
    }

}
```

Tag replacement for fuzzy request pseudocode:

TAG_REPLACEMENT PSEUDOCODE

```
struct TagInfo
 {
    bool fPairTagClosed = true;           // false for bpt tag - waiting for matching ept tag. If we'll find
matching tag -> we'll set this to true
    bool fTagAlreadyUsedInTarget = false; // would be set to true if we would already use this tag as matching
for target

    // this we generate to save in TM. this would be saved as <{generated_tagType} [x={generated_x}] [i=
{generated_i}]/>.
    // we would skip x attribute for generated_tagType=EPT_ELEMENT and i for generated_tagType=PH_ELEMENT
    int generated_i = -1;                 // for pair tags - generated identifier to find matching tag. the same
as in original_i if it's not binded to other tag in segment
```

```
    int generated_x = -1;                // id of tag. should match original_x, if it's not occupied by other
tags
    TagType generated_tagType = UNKNOWN_ELEMENT; // replaced tagType, could be only PH_ELEMENT, BPT_ELEMENT,
EPT_ELEMENT

    // this cant be generated, only saved from provided data
    int original_i = -1;        // original paired tags i
    int original_x = -1;        // original id of tag
    TagType original_tagType = UNKNOWN_ELEMENT;  // original tagType, could be any tag
  };
}

we use 3 lists of tags
SOURCE_TAGS
TARGET_TAGS
REQUEST_TAGS
as id we understand one of following attributes(which is present in original tag) : 'x', 'id'
as i  we understand one of following attributes(which is present in original tag) : 'i', 'rid'
all single tags we understand as ph_tag
all opening pair tags we understand as bpt_tag
all closing pair tags we understand as ept_tag
-1 means that value is not found/not used/not provided etc.
for ept tags in generated_id  we would use generated_id from matching bpt tag
    if matching bpt tag is not found -> ???

TagType could be set to one of following values


TAG REPLACEMENT USE CASES {

REQUEST{
   basically we convert request segment to tmx tags(similar as we generate ph, bpt and ept tags at import), but with saving original data
      then we try to find matching tags from the source to generated from the request. In matching source tags we replace data with original from request
(tagType, id and i attributes)
      then do the same with target segment\tags

   REQUEST_SEGMENT{
        are we sending only xliff? so ph, bpt and ept tag shouldn't be handled here?

        <single tags> {  // for ph and all single tags
           // here we can have PH, X, IT, UT tags, right?
           generate generated_id incrementally
           set  generated_tagType to PH_ELEMENT

           save original_id if provided (should that always be provided??)
           save tag type as out_tag.original_tagType

           save tag in REQUEST_TAGS
        }

        <opening tags> {
           //this would be never send from translate5, right?
           original type is <bpt>{
              save tag in REQUEST_TAGS
           }

           original type is <bx>{
              generate generated_i  incrementally in source segment
              generate generated_id incrementally
              set  generated_tagType to BPT_ELEMENT

              save original_i (should that always be provided??)
              save original_id if provided (should that always be provided??)
              set  fPairTagClosed to false; // it would be set to true if we would use this tag as matching
              set  fTagAlreadyUsedInTarget to false;
              set  original_type as BX_ELEMENT

              save tag to REQUEST_TAGS
           }


           original type is <g>{
              generate generated_i  incrementally in source segment
              generate generated_id incrementally
              set  generated_tagType to BPT_ELEMENT

              we don't have original_i provided here, only original_id, right?
              save original_id if provided (should that always be provided??)
              set  fPairTagClosed to false; // it would be set to true if we would use this tag as matching
              set  fTagAlreadyUsedInTarget to false;
              set  original_type as G_ELEMENT
```

```
      save tag in REQUEST_TAGS
   }

   original type is <hi>{
      generate generated_i  incrementally in source segment
      generate generated_id incrementally
      set  generated_tagType to BPT_ELEMENT

      we don't have original_i provided here, only original_id, right?
      save original_id if provided (should that always be provided??)
      set  fPairTagClosed to false; // it would be set to true if we would use this tag as matching
      set  fTagAlreadyUsedInTarget to false;
      set  original_type as HI_ELEMENT

      save tag in REQUEST_TAGS
   }

   original type is <sub>{
      generate generated_i  incrementally in source segment
      generate generated_id incrementally
      set  generated_tagType to BPT_ELEMENT

      we don't have original_i provided here, only original_id, right?
      save original_id if provided (should that always be provided??)
      set  fPairTagClosed to false; // it would be set to true if we would use this tag as matching
      set  fTagAlreadyUsedInTarget to false;
      set  original_type as HI_ELEMENT

      save tag in REQUEST_TAGS
   }

}

<closing tags> {
   //this would be never send from translate5, right?
   original type is <ept>{
      save tag in REQUEST_TAGS
   }

   original type is <ex>{
      search for matching tag in saved tags:
         looking in REQUEST_TAGS in REVERSE for matchingTag which have
                                          [      matchingTag.fPairTagClosed    == false
                                            AND matchingTag.generated_tagType == BPT_ELEMENT        //OPENING_PAIR_TAG
                                            AND matchingTag.original_tagType  == BX_ELEMENT  // our_tag.original_tagType
                                            AND matchingTag.original_i == our_tag.original_i
                                          ]
      if found
         set matchingTag.fPairTagClosed to true to eliminate matching one opening tag for different closing tags
         set our_tag.generated_i  to matchingTag.i
         set our_tag.generated_id to matchingTag.id
      else
         generate next our_tag.generated_i  incrementally in request segment // in every segment(target, source, request) i starts from 1
         generate next our_tag.generated_id incrementally                    // should be unique across target, source and request segments

      save tag in REQUEST_TAGS

   }

    original type is </g>{
      search for matching tag in saved tags:
         looking in REQUEST_TAGS in REVERSE for matchingTag which have
                                          [      matchingTag.fPairTagClosed    == false
                                            AND matchingTag.generated_tagType == BPT_ELEMENT        //OPENING_PAIR_TAG
                                            AND matchingTag.original_tagType  == G_ELEMENT // our_tag.original_tagType
                                          ]
      if found
         set matchingTag.fPairTagClosed to true to eliminate matching one opening tag for different closing tags
         set our_tag.generated_i  to matchingTag.i
         set our_tag.generated_id to matchingTag.id
      else
         generate next our_tag.generated_i  incrementally in request segment // in every segment(target, source, request) i starts from 1
         generate next our_tag.generated_id incrementally                    // should be unique across target, source and request segments

      save tag in REQUEST_TAGS
   }

   original type is </hi>{
```

```
        search for matching tag in saved tags:
            looking in REQUEST_TAGS in REVERSE for matchingTag which have
                                        [       matchingTag.fPairTagClosed    == false
                                          AND matchingTag.generated_tagType == BPT_ELEMENT         //OPENING_PAIR_TAG
                                          AND matchingTag.original_tagType  == HI_ELEMENT // our_tag.original_tagType
                                        ]
            if found
                set matchingTag.fPairTagClosed to true to eliminate matching one opening tag for different closing tags
                set our_tag.generated_i  to matchingTag.i
                set our_tag.generated_id to matchingTag.id
            else
                generate next our_tag.generated_i  incrementally in request segment // in every segment(target, source, request) i starts from 1
                generate next our_tag.generated_id incrementally                    // should be unique across target, source and request segments

            save tag in REQUEST_TAGS
        }

        original type is </sub>{
            search for matching tag in saved tags:
                looking in REQUEST_TAGS in REVERSE for matchingTag which have
                                        [       matchingTag.fPairTagClosed    == false
                                          AND matchingTag.generated_tagType == BPT_ELEMENT         //OPENING_PAIR_TAG
                                          AND matchingTag.original_tagType  == SUB_ELEMENT // our_tag.original_tagType
                                        ]
            if found
                set matchingTag.fPairTagClosed to true to eliminate matching one opening tag for different closing tags
                set our_tag.generated_i  to matchingTag.i
                set our_tag.generated_id to matchingTag.id
            else
                generate next our_tag.generated_i  incrementally in request segment // in every segment(target, source, request) i starts from 1
                generate next our_tag.generated_id incrementally                    // should be unique across target, source and request segments
            save tag in REQUEST_TAGS
        }
    }
}
```

!!!CONSIDER THAT WE SHOULD HAVE IN SOURCE SEGMENT ONLY 3 TYPES OF TAGS - PH_ELEMENT, BPT_ELEMENT and EPT_ELEMENT, because all of them was regenerated with their attributes at import stage

At this point we read the source and target segments "as is", without any tag replacement in lists. so original_id would be id, that was generated_id at import stage.

```
SOURCE_SEGMENT{
   <ph x="1" />{
      search for matching tag in saved tags:
         looking in REQUEST_TAGS in REVERSE for matchingTag which have
                                                matchingTag.generated_tagType == PH_ELEMENT //or our_tag.original_tagType
                                              AND matchingTag.generated_id  ==   our_tag.original_id
                                             ]
      if found
         set our_tag.generated_tagType = matchingTag.original_tagType
         set our_tag.generated_id = matchingTag.original_id
         use that that data to generate tag like <our_tag.generated_tagType id="{our_tag.generated_id}" />
      else
         maybe just return <x/> tag?

        save tag in SOURCE_TAGS

   }

   <bpt i="1" x="2"/> {
      search for matching tag in saved tags:
         looking in REQUEST_TAGS in REVERSE for matchingTag which have
                                             [       matchingTag.generated_tagType == BPT_ELEMENT //or our_tag.original_tagType
                                               AND  matchingTag.generated_id  ==   our_tag.original_id
                                             ]
      if found
         set our_tag.generated_tagType = matchingTag.original_tagType
         set our_tag.generated_id = matchingTag.original_id
         set our_tag.generated_i  = matchingTag.original_i

         if matchingTag.original_tagType == BX_ELEMENT // do BX_ELEMENT always have id and rid attributes provided?
            use that that data to generate tag like <our_tag.generated_tagType id="{our_tag.generated_id}" rid="{our_tag.generated_id}" />
         else:
            [rid="{our_tag.generated_id}"] - means optional, so for example if it's bigger than 0, then we should add this attribute
            use that that data to generate tag like <our_tag.generated_tagType [id="{our_tag.generated_id}"] [rid="{our_tag.generated_id}"] >
      else
         maybe just return <bx/> tag?

       save tag in SOURCE_TAGS
   }

   <ept i="1" /> {
      search for matching tag in saved tags:
         looking in REQUEST_TAGS in REVERSE for matchingTag which have
                                             [       matchingTag.generated_tagType == EPT_ELEMENT //or our_tag.original_tagType
                                               AND  matchingTag.generated_id  ==   our_tag.original_id  // id should hold information about paired
BPT_ELEMENT, or it's absence
                                             ]
      if found
         set our_tag.generated_tagType = matchingTag.original_tagType
         set our_tag.generated_id = matchingTag.original_id
         set our_tag.generated_i  = matchingTag.original_i
         use that that data to generate tag like <our_tag.generated_tagType id="{our_tag.generated_id}" rid="{our_tag.generated_id}" />

         if matchingTag.original_tagType == EX_ELEMENT // do EX_ELEMENT always have id and rid attributes provided?
            use that that data to generate tag like <our_tag.generated_tagType id="{our_tag.generated_id}" rid="{our_tag.generated_id}" />
         else:
            [rid="{our_tag.generated_id}"] - means optional, so for example if it's bigger than 0, then we should add this attribute
            use that that data to generate tag like </our_tag.generated_tagType>
      else
         maybe just return <ex/> tag? or add some specific attributes?

       save tag in SOURCE_TAGS
   }
}

}
```

NEW PSEUDO CODE

This is the code, actually implemented

Tag replacement feature implementation is splited into 2 functions:
GenerateReplacingTag - input - tagType, attributeList
output - tagInfo
this function would generate tagInfo data structure that saves original data(tagType, attributes(i\rid and x\id only) and would generate new data that suits context\segment
PrintTag - input - tagInfo
- output - text representation of tag with attributes depending on context
this function would print tag with attributes(if they exist(bigger than 0). If it's fuzzy call, would replace for source and target segments tags with matching tags from fuzzy search request.
If matching tag not found - would generate new tag in xliff format with id or rid attributes that rising starting from biggest id and rid values +1 that was present in requested segment
for fuzzy search request segment this function would pring tag with generated data - that is never used in production, but can be used to find out how mechanism normalized input fuzzy search request segment
(we base tag matching on this normalization.)

////////////////////////////////////

```
struct TagInfo
 {
   bool fPairTagClosed = true;           // false for bpt tag - waiting for matching ept tag. If we'll find matching tag -> we'll set this to true
   bool fTagAlreadyUsedInTarget = false; // would be set to true if we would already use this tag as matching for target

   // this we generate to save in TM. this would be saved as <{generated_tagType} [x={generated_id}] [i={generated_i}]/>.
   // we would skip x attribute for generated_tagType=EPT_ELEMENT and i for generated_tagType=PH_ELEMENT
   int generated_i = -1;                 // for pair tags - generated identifier to find matching tag. the same as in original_i if it's not binded to other tag in segment
   int generated_id = -1;                // id of tag. should match original_id, if it's not occupied by other tags
   TagType generated_tagType = UNKNOWN_ELEMENT; // replaced tagType, could be only PH_ELEMENT, BPT_ELEMENT, EPT_ELEMENT

   // this cant be generated, only saved from provided data
   int original_i = -1;        // original paired tags i
   int original_id = -1;       // original id of tag
   TagType original_tagType = UNKNOWN_ELEMENT;  // original tagType, could be any tag
  };
}
```

TagType could be one of the values in enum:

```
[
  BPT_ELEMENT EPT_ELEMENT G_ELEMENT  HI_ELEMENT SUB_ELEMENT BX_ELEMENT  EX_ELEMENT
  //standalone tags
  BEGIN_STANDALONE_TAGS PH_ELEMENT  X_ELEMENT  IT_ELEMENT  UT_ELEMENT
]
```

We make normalization process to tags which means to replace original xliff\tmx tags\attributes with only 3 tags:

```
   <ph x='1' />
   <bpt x='2' i='1' />
   <ept i='1' />
```

which means that we would regenerate id\x in source, target and request segments to make them unified

for source\target segments this replacement is done at import process, for fuzzy search request we do tag replacement, then look for matches between source and request segments(this happens in PringTag function), then replace tag from source with original tag that was in request
then we do the same with target segment - we try to find matches of target tags with generated tags in request, and then replace tags in target with original tags from fuzzy search request

for example, we have this segments in import process

```
   'source':"Select the <hi>net<ph/>work <g>BLK360</g> tag </hi>",
   'target':"Select the <hi>net<ph/>work <g>BLK360</g> tag </hi>",
```

after tag replacement we would have this saved in tm:

```
   'source' :'Select the <bpt x="1" i="1"/>net<ph x="2"/>work <bpt x="3" i="2"/>BLK360<ept i="2"/> tag <ept i="1"/>',
   'target' :'Select the <bpt x="1" i="1"/>net<ph x="2"/>work <bpt x="3" i="2"/>BLK360<ept i="2"/> tag <ept i="1"/>',
```

then if we would have fuzzy request call with segment:

```
   "Select the <g>net<x/>work <g>BLK360</g> tag </g>"
```

after normalization we would get this:

```
   "Select the <bpt x="1" i="1"/>net<ph x="2"/>work <bpt x="3" i="2"/>BLK360<ept i="2"/> tag <ept i="1"/>"
```

and then we would try to find matching tags in source and normalized request segments and in case of match-replace tag in src with original from fuzzy search request and then do the same with target and request
in response we should have:

```
'source' :'Select the <g>net<x/>work <g>BLK360</q> tag </g>',
 'target' :'Select the <g>net<x/>work <g>BLK360</q> tag </g>',
```

//////////////////////////TagReplacer class//////////////////////

tag normalization statements:
- all single tags we understand as ph_tag that have only x attribute, and looks like this: "<ph x="1"/>"
- all opening pair tags we understand as bpt_tag that always have both i and x attributes, and looks like this: "<bpt x="1" i="1"/>"
- all closing pair tags we understand as ept_tag that always have only i attribute looks like this: "<ept i="1"/>"
- we ignore/skip context within <bpt> and </bpt> and replace this with single <bpt/> type tag, same is true for <ph/> and <ept/>
- as id we understand one of following attributes(which is present in original tag) : 'x', 'id'
- as i we understand one of following attributes(which is present in original tag) : 'i', 'rid'

```
TagReplacer{
    // lists of tagInfo
    SOURCE_TAGS
    TARGET_TAGS
    REQUEST_TAGS

    activeSegment  //could be one of following SOURCE_SEGMENT(default value), TARGET_SEGMENT, REQUEST_SEGMENT. Tells us how we should handle tag replacement

    iHighestI   = 0;              // increments with each opening pair tags
    iHighestId  = 0;               // increments with each tag

    fFuzzyRequest = false;         // flag, that tracks if we are dealing with import or fuzzy request. Tells us how we should handle tag replacement

    //to track id and i attributes in request and then generate new values for tags in srt and trg that is not matching
    iHighestRequestsOriginalI  = 0; // during saving original data of tags in request segment we save here biggest original I and Id,
    iHighestRequestsOriginalId = 0; //     and in case if we couldn't find match in source segment, we would generate xliff tag([bx, ex ,x] or can we left [bpt, ept, ph]?)
    // with using and incrementing this values

    //functions
    // during parsing of tags by xercesc we call this function to
    // - collect and save original tagType and attributes(only 'id' and 'i')
    // - generate normalized tag data
    // - find matches between TARGET and SOURCE segment tags. If we have match - use generated data from SOURCE, if not - generate new unique data
    // - save generated tags in lists depends on activeSegment value
    // - returns tagInfo data structure
    GenerateReplacingTag(tagType, attributes);

    //accepts tagInfo data
    //depending on fFuzzyRequest and activeSegment values just prints generated normalized tags with generated attributes
    //    or try to find match for tag from SOURCE\TARGET to REQUEST and print matching tag from REQUEST, or, if no matched, generate new xliff tag with unique attributes
    PrintTag(tagInfo);
};



TagInfo{
    fPairedTagClosed = false;       // flag, set to false for bpt/ept tag - waiting for matching ept/bpt tag
    fTagAlreadyUsedInTarget = false; // flag, that we use only when we save tags from source segment and then try to match\bind them in target

    generated_i = 0;                // for pair tags - generated identifier to find matching tag. the same as in original_i if it's not binded to other tag in segment
    generated_id = 0;                // id of tag. should match original_id, if it's not occupied by other tags
    generated_tagType = UNKNOWN_ELEMENT;          // replaced tagType, could be PH_ELEMENT, BPT_ELEMENT, EPT_ELEMENT

    original_i = 0;       // original paired tags i
    original_id = 0;       // original id of tag
    original_tagType;  // original tagType
 };


//////////////////////////////////////////////////

GenerateReplacingTag{

   SOURCE_SEGMENT/REQUEST_SEGMENT
   //we handle SOURCE and REQUEST segments here the same way, but we
   // use variables activeTagList, that should point to SOURCE_TAGS or REQUEST_TAGS
   // to make code more generic
   {
       <single tags> -> would be saved as <ph>{ // for ph and all single tags
          if(type == "lb"){
             replace with newline
          }else{
             save original_tagType
             save original_id if provided

             if it's REQUEST_SEGMENT AND original_id > iHighestRequestsOriginalId
                save original_id as new iHighestRequestsOriginalId
```

```
            set  generated_tagType to PH_ELEMENT
            set fPairedTagClosed to true
            generate generated_id incrementally ( increment iHighestId value, then use it )

            save tag to activeTagList // SOURCE_TAGS or REQUEST_TAGS
         }
      }

      <opening pair tags> -> would be saved as <bpt>{
         save original_i  if provided
         save original_id if provided
         save original_tagType
         set  generated_tagType to BPT_ELEMENT

         //save biggest id and i attributes in request original data to generate new values
         // that wouldn't overlap with other tags in case we wouldn't have matches
         if it's REQUEST_SEGMENT AND original_i > iHighestRequestsOriginalI
            save original_i as new iHighestRequestsOriginalI

         if it's REQUEST_SEGMENT AND original_id > iHighestRequestsOriginalId
            save original_id as new iHighestRequestsOriginalId

         originalTagTypeToFind = UNKNOWN_ELEMENT // use this variable to identify which tag type we are looking for

         if generated_tagType is BPT_ELEMENT
            set originalTagTypeToFind to EPT_ELEMENT
         else if generated_tagType is BX_ELEMENT
            set originalTagTypeToFind to EX_ELEMENT
         else
            skip search, because other tags could never have wrong order between opening and closing tags
            that would be error in <xml> and parser would throw INVALID_XML error then

         if originalTagTypeToFind is not UNKNOWN_ELEMENT
            try to find matching ept tag in this segment
               looking in REVERSE order in activeTagList for matchingTag which have  [
                                    matchingTag.fPairTagClosed    == false
                                 AND matchingTag.generated_tagType == EPT_ELEMENT        //all CLOSING PAIR TAGs always has
BPT_ELEMENT here
                                 AND matchingTag.original_tagType  == originalTagTypeToFind
                                 AND matchingTag.original_i        == our_bpt_tag.original_i
                               ]



            if mathingTag found
               set generated_i to    mathingTag.generated_i
               set generated_id to  -mathingTag.generated_id // EPT_TAGS have negative id's/x's that is equal to matching -bpt.x
                                    // if there are no matching bpt, ept have unique, but still negative value.
                                    // negative values and 0 would never be printed in PrintTag
               set fPairTagClosed to true
               set matchingTag.fPairTagClosed to true
            else
               generate generated_i  incrementally ( increment iHighestI  value, then use it )
               generate generated_id incrementally ( increment iHighestId value, then use it )
               set  fPairTagClosed to false; // it would be set to true if we would use this tag as matching

            save tag to activeTagList
         }

      <closing pair tags> -> would be saved as <ept>{
         save original_i  if provided
         save original_id if provided
         save original_tagType
         set  generated_tagType to EPT_ELEMENT

         //save biggest id and i attributes in request original data to generate new values
         // that wouldn't overlap with other tags in case we wouldn't have matches
         if it's REQUEST_SEGMENT AND original_i > iHighestRequestsOriginalI
            save original_i as new iHighestRequestsOriginalI

         if it's REQUEST_SEGMENT AND original_id > iHighestRequestsOriginalId
            save original_id as new iHighestRequestsOriginalId

         originalTagTypeToFind = UNKNOWN_ELEMENT // use this variable to identify which tag type we are looking for


         if generated_tagType is EPT_ELEMENT
            set originalTagTypeToFind to BPT_ELEMENT
```

```
            else if generated_tagType is EX_ELEMENT
               set originalTagTypeToFind to BX_ELEMENT
            else
               skip search, because other tags could never have wrong order between opening and closing tags
               that would be error in <xml> and parser would throw INVALID_XML error then



            if originalTagTypeToFind is not UNKNOWN_ELEMENT
               try to find matching ept tag in this segment
                  looking in REVERSE order in activeTagList for matchingTag which have  [
                                          matchingTag.fPairTagClosed    == false
                                        AND matchingTag.generated_tagType == BPT_ELEMENT        //all CLOSING PAIR TAGs always has
BPT_ELEMENT here
                                        AND matchingTag.original_tagType  == originalTagTypeToFind
                                        AND matchingTag.original_i        == our_ept_tag.original_i
                                      ]



            if mathingTag found
               set generated_i to    mathingTag.generated_i
               set generated_id to  -mathingTag.generated_id // EPT_TAGS have negative id's/x's that is equal to matching -bpt.x
                                          // if there are no matching bpt, ept have unique, but still negative value.
                                          // negative values and 0 would never be printed in PrintTag
               set fPairTagClosed to true
               set matchingTag.fPairTagClosed to true
            else
               generate generated_i  incrementally ( increment iHighestI  value, then use it )
               generate generated_id incrementally ( increment iHighestId value, then  multiply it by *(-1) and use it )
               set  fPairTagClosed to false; // it would be set to true if we would use this tag as matching

            save tag to activeTagList
         }
      }

   TARGET_SEGMENT
   //here we try to find connections from original Target tags to original Source tags and use data,
   //   that was generated for matching SOURCE tag. If there are no matching SOURCE tag - generate new unique attributes
   {
      save original_tagType
      save original_id if provided
      save original_i  if provided
      set  generated_tagType to    - PH_ELEMENT if we have single tag
                                   - BPT_ELEMENT if we have opening pair tag
                                   - EPT_ELEMENT if we have closing pair tag


      try to find matching source tag
         looking in SOURCE_TAGS for matchingSourceTag which have  [
                                      matchingSourceTag.fAlreadyUsedInTarget    == false
                                    AND matchingSourceTag.original_tagType  == our_tag.original_tagType
                                    AND matchingSourceTag.original_id       == our_tag.original_id
                                  ]
      if found:
         set generated_i  to   matchingSourceTag.generated_i
         set generated_id to  matchingSourceTag.generated_id
          // maybe we should add here search for matching ept\bpt tag in TARGET_TAGS, to set valid fPairTagClosed for both
         set matchingSourceTag.fAlreadyUsedInTarget to true
      else
         if generated_tagType is PH_ELEMENT
            set fPairTagClosed = true
         else
            use matchingTagOriginalType and matchingTagGeneratedType to find matching tag in TARGET_TAGS
               if original_tagType is BPT_ELEMENT
                  set matchingTagOriginalType to EPT_ELEMENT

               else if generated_tagType is BX_ELEMENT
                  set matchingTagOriginalType to EX_ELEMENT

               else if original_tagType is EPT_ELEMENT
                  set matchingTagOriginalType to BPT_ELEMENT

               else if generated_tagType is EX_ELEMENT
                  set matchingTagOriginalType to BX_ELEMENT

               else
                  matchingTagOriginalType = original_tagType
```

```
            if our_tag.generated_tagType = BPT_ELEMENT
               set matchingTagGeneratedType to EPT_ELEMENT
            else
               set matchingTagGeneratedType to BPT_ELEMENT

         try to find matching pair tag in this segment
               looking in REVERSE order in TARGET for matchingPairTag which have  [
                                    matchingPairTag.fPairTagClosed    == false
                                 AND matchingPairTag.original_tagType  == matchingTagOriginalType
                                 AND matchingPairTag.generated_tagType == matchingTagGeneratedType
                                 AND matchingPairTag.original_i        == our_tag.original_i
                               ]
         if found:
            set generated_i to    mathingTag.generated_i
            set generated_id to  -mathingTag.generated_id // EPT_TAGS have negative id's/x's that is equal to matching -bpt.x
                                       // if there are no matching bpt, ept have unique, but still negative value.
                                       // negative values and 0 would never be printed in PrintTag
            set fPairTagClosed to true
            set matchingPairTag.fPairTagClosed to true
         else:
            if we dealing with pair tags -> generate generated_i  incrementally ( increment iHighestI value, then use it )
            generate generated_id incrementally ( increment iHighestId value, then  multiply it by *(-1) and use it )
            set  fPairTagClosed to false; // it would be set to true if we would use this tag as matching

      save tag in TARGET_TAGS
   }

}


PrintTag{
   variables: idToPrint = 0,
              iToPrint = 0,
              tagTypeToPrint = tag.generated_tagType
   flags: fClosedTag = true;   //for slash at the end of tags like <ph/>
          fClosingTag = false; //for slash at the beginning of tag like </g>

   if it's REQUEST_SEGMENT
      // we need this only to track how tag replacement normalized tags in request segment
      idToPrint = tag.generated_id
       iToPrint = tag.generated_i
   else
      try to find matching request tag
         looking in SOURCE_TAGS for matchingRequestTag which have  [
                                    matchingRequestTag.generated_id        == our_tag.generated_i
                                 AND matchingRequestTag.generated_tagType  == our_tag.generated_tagType
                               ]
      if found:
         set  idToPrint to      matchingRequestTag.original_id
         set   iToPrint to      matchingRequestTag.original_i
         set  tagTypeToPrint to matchingRequestTag.original_tagType
         set fClosingTag to     tag.generated_tagType == EPT_ELEMENT
                             AND tagTypeToPrint != EPT_ELEMENT
                             AND tagTypeToPrint != EX_ELEMENT
```

```
        else
            //generate new id and i
            generate idToPrint using iHighestRequestsOriginalId incrementally ( increment incrementally value and use it )
            if generated_tagType is not PH_ELEMENT
                //could be improved here if we need
                generate iToPrint using iHighestRequestsOriginalI incrementally ( increment incrementally value and use it )


    if fClosingTag is true
        return ["</" + tagTypeToPrint + ">"]
    else
        output = ["<" + tagTypeToPrint]
        if idToPrint > 0
            if fFuzzyRequest is true:
                append to output [' id="' + idToPrint + ""]
            else
                append to output [' x="' + idToPrint + ""]

        if idToPrint > 0
            if fFuzzyRequest is true:
                append to output [' rid="' + iToPrint + ""]
            else
                append to output [' i="' + iToPrint + ""]

        //tag that has slash at the end looks like this: <tag />
        fClosedTag   =  tagTypeToPrint == BPT_ELEMENT OR
                        tagTypeToPrint == EPT_ELEMENT OR
                        tagTypeToPrint ==  PH_ELEMENT OR
                        tagTypeToPrint ==  BX_ELEMENT OR
                        tagTypeToPrint ==  EX_ELEMENT OR
                        tagTypeToPrint ==   X_ELEMENT ; //  other tags could be only not closed(<g>) or closing(</g>)

        if fClosedTag is true
            append to output "/"
        append to output ">"
        return output
}
```

//

//

Previous documentation:

Response

```
http://opentm2/translationmemory/
POST - creating a new or importing an existing filebased binary OpenTM2 TM

The Parameter „name“ contains the TM Name as a string. The string has a maxlength of 256 chars. It can
contain any characters except the characters backslash (\), slash(/), colon (:), question mark (?), asterisk
(*), vertical line (|), less than sign (<), and greater than sign (>).

Uploading a file is optional, omitting a file means creating a empty TM only.

If an empty TM is created, the POST request contains only the JSON structure with the TM Name.

If an existing binary OpenTM2 file should be additionally imported to the new TM, the POST must be encoded as
multipart/form-data.

The JSON structure with the meta data will then be in the first chunk of the multiparted request, the chunk
must be named “meta”.

The second chunk contains the plain binary file content and must be named “data”. This binary data contains
the TM content
```

The resulting body contains the name of the TM, as given in the POST request.

To OpenTM2 – without data / creating an empty TM:

{
sourceLang: “en”, // the source language is required for a new TM
name: „TM Name“,
[loggingThreshold:"2"]
}

Raw POST to OpenTM2 – with provided import file:

```
POST http://opentm2/translationmemory HTTP/1.1
Content-Type: multipart/form-data; boundary="autogenerated"

-- autogenerated
Content-Type: application/json; charset=utf-8
Content-Disposition: form-data; name=meta

{"name":"TM Name", sourceLang:"en"}

--autogenerated
Content-Type: image/jpeg
Content-Disposition: form-data; name=data; filename=Original Filename.jpg
...TM content ...
--autogenerated--
```

In both cases from OpenTM2 - HTTP 200 OK:

{

name: „TM Name“

}

Errors:
400 Bad Request – if parameters are missing or are not well formed.
409 Conflict – if a memory with the given name already exists.
500 Server Error – for other technical problems.
http://opentm2/translationmemory/[TM_Name]/import
POST import a TMX file into an existing OpenTM2 TM

To OpenTM2:

multipart/form-data like on POST above, expect that no separate JSON section is needed here.

Call answers directly after the upload is done, but before the import starts with HTTP 201 – this means:
Import is created and will be started now.

From OpenTM2 - HTTP 201 OK:

{ // empty JSON object, since no data expected as result here!

}

Errors:
400 Bad Request – if parameters are missing or are not well formed.
404 Not Found – if the memory of the given name does not exist
500 Server Error – for other technical problems.
http://opentm2/translationmemory/[TM_Name]/status
GET status of a TM

To OpenTM2:

multipart/form-data like on POST above, expect that no separate JSON section is needed here.

From OpenTM2 - HTTP 200 OK:

{

‘status’:’import’ //allowed status values: import, available, error

}

Errors:
400 Bad Request – if parameters are missing or are not well formed.
404 Not Found – if the memory of the given name does not exist
500 Server Error – for other technical problems.
http://opentm2/translationmemory/
GET – retrieving a list of available TM Files

To OpenTM2: -

From OpenTM2 - HTTP 200 OK:

[{

name: 'my nice TM'

}]

Errors:
500 Server Error – for other technical problems.
http://opentm2/translationmemory/[TM_Name]/
TM_Name is URL-encoded
GET – retrieving a single TM File

To OpenTM2: -

From OpenTM2 - HTTP 200 OK:

Same as POST from OpenTM2 result.

Errors:
404 Not Found – if TM file to given [TMID] in URL was not found
500 Server Error – for other technical problems.
DELETE – deletes an existing TM File

Adressed by the given URL, no body needed.

Errors:
404 Not Found – if TM file to given [TMID] in URL was not found
500 Server Error – for other technical problems.
PUT – updating an existing TM File in one request

Currently not needed, would be only to change the TM name

GET – list of all segments from TM

Currently not needed.

http://opentm2/translationmemory/[TM_Name]/entry/
POST – creates a new entry or updates target entry if match pair already exists

This method updates an existing proposal when a proposal with the same key information (source text, language, segment number, and document name) exists.

Parameters sourceLang and targetLang are containing the languages as RFC5646.

Parameters source and target are containing the entry contents to be stored. Format? plain string?

Attribute Parameters:

documentName: contains the filename where the segment resides in Translate5.
context: evaluates to Translate5 segment mid.
markupTable: OpenTM2 gets a new markup table named „translate5“, so this is the value which is delivered by Translate5.
timestamp: this parameter is not set by translate5, but calculated automatically and delivered from OpenTM2 to translate5.
author: contains the named user which provides the update / new entry
In addition there are the following OpenTM2 Attributes currently not used by translate5:
segmentNumber
additional info
type

To OpenTM2:

{

sourceLang: 'de',

targetLang: 'en',

source: „Das ist das Haus des Nikolaus“,

target: „This is the house of St. Nicholas“,

documentName: 'my file.sdlxliff',

segmentNumber: ,

markupTable: 'translate5',

author: „Thomas Lauria“,

type: '',

timeStamp: '',

context: '123',

addInfo: '',

[loggingThreshold:"2"]

}

The result from the server contains the same data as posted to the server. No additonal ID is added, since the entries are identified by the whole source string instead by an ID, only the timestamp is added.

From OpenTM2 – HTTP 200 OK:

{

sourceLang: 'de',

targetLang: 'en',

source: „Das ist das Haus des Nikolaus“,

target: „This is the house of St. Nicholas“,

documentName: 'my file.sdlxliff',

segmentNumber: 123,

markupTable: 'translate5',

timestamp: '2015-05-12 13:46:12',

author: „Thomas Lauria“

}

Errors:
404 Not Found – if TM file to given [TM_Name] in URL was not found
500 Server Error – for other technical problems.
400 Bad Request – if JSON parameters are missing or are not well formed.

http://opentm2/translationmemory/[TM_Name]/fuzzysearch/
POST– Serves a memory lookup based on the provided search criteria

To OpenTM2:

{

sourceLang: 'de',

targetLang: 'en-US',

source: „Das ist das Haus des Nikolaus“,

documentName: 'my file.sdlxliff', // can be empty

segmentNumber: 123, // can be empty

markupTable: 'translate5', // can be empty

context: „xyz“, // can be empty

[loggingThreshold:"2"]

}

From OpenTM2 HTTP 200 OK:

{

'NumOfFoundProposals': 2,

'results':

[{

source: „Das ist das Haus des Nikolaus“,

target: „This is the house of St. Nicholas“,

sourceLang: 'de', rfc5646

targetLang: 'en', rfc5646

matchRate: '100',

documentName: 'my file.sdlxliff',

DocumentShortName: 'shortnam.txt',

id: 'identifier',

type: 'Manual',

matchType: 'Exact',

segmentNumber: 123,

markupTable: 'XYZ',

```
timestamp: '2015-05-12 13:46:12',

author: „Thomas Lauria“.

context: '',

addInfo: ''

},{

source: „Das ist das Haus des Nikolaus“,

target: „This is the house of St. Nicholas“,

sourceLang: 'de',  rfc5646

targetLang: 'en',  rfc5646

matchRate: '100',

documentName: 'my file.sdlxliff',

DocumentShortName: 'shortnam.txt',

id: 'identifier',

type: 'Manual',

matchType: 'Exact',

segmentNumber: 123,

markupTable: 'XYZ',

timestamp: '2015-05-12 13:46:12',

author: „Thomas Lauria“.

context: '',

addInfo: ''

}]}
```

Errors:
400 Bad Request – if search, query or language parameters are missing or are not well formed.
404 Not Found – if TM file to given [TM_Name] in URL was not found
500 Server Error – for other technical problems.

http://opentm2/translationmemory/[TM_Name]/concordancesearch /?
POST – Performs a context search of the given search string in the proposals contained in a memory. Returns one proposal per request.

To OpenTM2:

```
{

searchString: 'Haus des Nikolaus',
```

searchType: 'source', // values can be source or target

searchPosition: 123// can be empty; Position where a search should start in the memory, see below

numResults: 1,

msSearchAfterNumResults: 100 //number of milliseconds the search will continue, after the first result is found. All additional results that are found in this additional time will also be returned until numResults is reached. If numResults is reached before msSearchAfterNumResults is reached, the search will abort. If msSearchAfterNumResults is reached before numResults is reached, search is also aborted. All found results are delivered in both cases.

[loggingThreshold:"2"]

}

From OpenTM2 HTTP 200 OK:

{

NewSearchPosition: '123:54', /returns NULL, if end of TM is reached, see below

results:[{

source: „Das ist das Haus des Nikolaus“,

target: „This is the house of St. Nicholas“,

sourceLang: 'de', rfc5646

targetLang: 'en', rfc5646

matchRate: '100',

documentName: 'my file.sdlxliff',

DocumentShortName: 'shortnam.txt',

id: 'identifier',

type: 'Manual',

matchType: 'Exact',

segmentNumber: 123,

markupTable: 'XYZ',

timestamp: '2015-05-12 13:46:12',

author: „Thomas Lauria“.

context: '',

addInfo: ''

},{

source: „Das ist das Haus des Nikolaus“,

target: „This is the house of St. Nicholas“,

sourceLang: 'de', rfc5646

targetLang: 'en', rfc5646

matchRate: '100',

documentName: 'my file.sdlxliff',

```
DocumentShortName: 'shortnam.txt',

id: 'identifier',

type: 'Manual',

matchType: 'Exact',

segmentNumber: 123,

markupTable: 'XYZ',

timestamp: '2015-05-12 13:46:12',

author: „Thomas Lauria“.

context: '',

addInfo: ''

}]}
```

Errors:
400 Bad Request – if search, query or language parameters are missing or are not well formed.
404 Not Found – if TM file to given [TM_Name] in URL was not found
500 Server Error – for other technical problems.