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. Maybe we need trigger to flush tm to the disk, but also it could be done in some specific cases...
- query TM for Matches: one query per TM, not quering multiple TMs at once.
- query TM for concordance search
- save new entry to TM
- delete entry from TM
- localy clone TM
- reorganize TM
- get some statistics about service
- also you can use tagreplacement endpoint to test tag replacement mechanism

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. It's should be pretty json and ends with **'\n}"** symbol, because of bug in proxygen that caused garbage after valid data.

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					default endpoint/example	Is async?
1	Get the list of TMs	Returns JSON list of TMs	**GET**	/%*service*%/	/t5memory/	
2	Create TM	Creates TM with the provided name	**POST**	/%*service*%/	/t5memory/	
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*%/	/t5memory/	
4	Clone TM Localy	Makes clone of existing tm	**POST**	/%*service*%/%*tm_name*%/clone	/t5memory/**my+TM**/clone (**+**is placeholder for whitespace in tm name, so there should be '**my TM.TMD**' and '**my TM.TMI**'(and in pre **0.5.x** '**my TM.MEM**' also) files on the disk) **tm name IS case sensetive in url**	
5	Reorganize TM	Reorganizing tm(replacing tm with new one and reimporting segments from tmd) - async	**GET**	/%*service*%/%*tm_name*%/reorganize	/t5memory/my+other_tm/reorganize	+ in 0.5.x and up
5	Delete TM	Deletes .TMD, .TMI files	**DELETE**	/%*service*%/%*tm_name*%/	/t5memory/%*tm_name*%/	
6	Import TMX into TM	Import provided base64 encoded TMX file into TM - async	**POST**	/%*service*%/%*tm_name*%/import	/t5memory/%*tm_name*%/import	+
7	Export TMX from TM	Creates TMX from tm. Encoded in base64	**GET**	/%*service*%/%*tm_name*%/	/t5memory/%*tm_name*%/	
8	Export in Internal format	Creates and exports archive with .TMD, .TMI files of TM	**GET**	/%*service*%/%*tm_name*%/	/t5memory/%*tm_name*%/status	
9	Status of TM	Returns status\import status of TM	**GET**	/%*service*%/%*tm_name*%/status	/t5memory/%*tm_name*%/status	
10	Fuzzy search	Returns entries\translations with small differences from requested	**POST**	/%*service*%/%*tm_name*%/fuzzysearch	/t5memory/%*tm_name*%/fuzzysearch	
11	Concordance search	Returns entries\translations that contain requested segment	**POST**	/%*service*%/%*tm_name*%/concordancesearch	/t5memory/%*tm_name*%/concordancesearch	

12	Entry update	Updates entry\translation	**POST**	/%*service*%/%*tm_name*%/entry	/t5memory/%*tm_name*%/entry	
13	Entry delete	Deletes entry\translation	**POST**	/%*service*%/%*tm_name*%/entrydelete	/t5memory/%*tm_name*%/entrydelete	
14	Save all TMs	Flushes all filebuffers(TMD, TMI files) into the filesystem	**GET**	/%*service*%_**service**/savetms	/t5memory_service/saveatms	
15	Shutdown service	Flushes all filebuffers into the filesystem and shutting down the service	**GET**	/%*service*%_**service**/shutdown	/t5memory_service/shutdown	
16	Test tag replacement call	For testing tag replacement	**POST**	/%*service*%_**service**/tagreplacement	/t5memory_service/tagreplacement	
17	Resources	Returns resources and service data	**GET**	/%*service*%_**service**/resources	/t5memory_service/resources	
18	Import tmx from local file(in removing lookuptable git branch)	Similar to import tmx, but instead of base64 encoded file, use local path to file	**POST**	/%*service*%/%*tm_name*%/importlocal	/t5memory/%*tm_name*%/importlocal	+
19	Mass deletion of entries(from v0.6.0)	It's like reorganize, but with skipping import of segments, that after checking with provided filters combined with logical AND returns true.	**POST**	/%service%/%tm_name%/entriesdelete	/t5memory/tm1/entriesdelete	+
20	New concordance search(from v0.6.0)	It's extended concordance search, where you can search in different field of the segment	**POST**	/%service%/%tm_name%/search	/t5memory/tm1/search	

Available end points

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

Returns list of open TMs and then list of available(excluding open) in the app.

Response

```
Response example:
{
    "Open": [
        {
            "name": "mem2"
        }
    ],
    "Available on disk": [
        {
            "name": "mem_internal_format"
        },
        {
            "name": "mem1"
        },
        {
            "name": "newBtree3"
        },
        {
            "name": "newBtree3_cloned"
        }
    ]
}open - TM is in RAM, Available on disk - TM is not yet loaded from disk
```

Create TM

Purpose	Creates TM with the provided name(tmd and tmi files in/MEM/ folder)
Request	**Post** /%service%/%tm_name%/
Params	Required: name, sourceLang

Response

```
Request example
{    "name": "examle_tm",  // this name would be used as filename for .TMD and .TMI files
   {  "sourceLang": "bg-BG"}  // should match lang in languages.xml
   {"data": "base64_encoded_archive_see_import_in_internal_format"}
   ["loggingThreshold": 0]
}
this endpoint could work in 2 ways, like creation of new tm (then sourceLang is required and data can be
skipped) or importing archived .tm(then sourceLang can be skipped, but data is required)it's possible to add
memDescription in this stage, but this should be explored more if needed

Response example:Success:{
"name": "examle_tm",
}
TM already exists:
{
    "ReturnValue": 7272,
    "ErrorMsg": "::ERROR_MEM_NAME_EXISTS:: TM with this name already exists: examle_tm1; res = 0"
}
```

Create/Import TM in internal format

Purpose	Import and unpack base64 encoded archive of .TMD, .TMI, .MEM(in pre 0.5.x versions) files. Rename it to provided name
Request	**POST** /%service%/
Params	{ "name": "examle_tm", "sourceLang": "bg-BG" , "data":"base64EncodedArchive" }

Do not import tms created in other version of t5memory. Starting from 0.5.x tmd and tmi files has t5memory version where they were created in the header of the file, and different middle version(0.**5**.x) or global version(**0**.5.x) would be represented as
version mismatch. Instead export tmx in corresponding version and create new empty tm and import tmx in new version.

This would create example_tm.TMD(data file) and example.TMI(index file) in MEM folder
If there are "data" provided, no "sourceLang" required and vice versa - base64 data should be base64 encoded .tm file(which is just archive that contains .tmd and .tmi files
If there are no "data" - new tm would be created, "sourceLang" should be provided and should be match with lang in languages.xml

Response

```
Request example:{ "name": "mem_internal_format", "data":"
UEsDBBQACAgIAPmrhVQAAAAAAAAAAAAAAAAWAAQAT1RNXy1JRDE3NS0wXzJfNV9iLk1FTQEAAADtzqEKgDAQgOFTEHwNWZ5swrAO0SBys6wfWx
FBDILv6uOI2WZQw33lr38GbvRIsm91baSiigzFEjuEb6XHEK\/myX0PXtXsyxS2OazwhLDWeVTaWgEFMMYYY\
/9wAlBLBwhEWTaSXAAAAAAAAAAACAAAAAAAAFBLAwQUAAgICAD5q4VUAAAAAAAAAAAAAAAAAFgAEAE9UTV8tSUQxNzUtMF8yXzVfYi5UTUQBAAA
A7d3Pa5JxHMDxz+Ns09phDAYdPfaDyQqWRcYjS9nGpoYZhBeZMCISW2v2g5o6VkqQONk\
/0KVzh4IoKAovnboUo1PHbuuwU8dSn8c9Pk2yTbc53y+R5\/P9fL7P1wf5Ps9zep5vIOy3iMiSiPLn0yPrQ7In+rStTQARi\
/bV9chEyHcxGPIKAGDnPonl21SsHNmUYNgfHZ70nnKNDo9ET0dHozFn2L+Ll9uxZPzazPz1mYQAAAAAAAAAAAAAAAAAAAAAAAAAANDtBkXRoj5
Zk7OqSFZ9q35Vn6khNa6W2wAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAdBKbKHK4Em1omT5DxV6J7FrmkKFypBKt9FczvYaKtr+2DLpiqPTWVayGi
q2uYjFUpC7VI6aElN8F8JPn\/QEAAAAAAAAAAAAAAAAAAAAAAAAAAAAA2ANW7U0Ag9Iv60MnT4j8uLBZ\
/X5+7dxn1ztX6Uy5AgAAAAAAAAAAAAAAAAAgA6nL1qFjmc1rAO2IwNN9bL9u4ulVUeEfcQqQAfxSNtltshZaytB7jalZZ2a5KhFGT3Qr\
/ztv1pkzAnP1v06+F7UxL22tRzSNf6aFq08MdoiY078\/znmkTZo5Qm2YdoOSLSyDdbaVUop\/Cj3cDm14I6\
/uqf++nDUN1u4lS+k9MbKXL4QK72+775U+phOpp8sucdK728X5nK5hVT+weJqbTiHjMiNzWG1yNxWvI8rvxZ9cTfycj71NH1nsZgbf54uJlKry
Wy6GFlueBT6xHrzJRupDqkPXc9eyyduJmbLkf6\/mlYRDgQDPtO++3\/uYvsazANfYHx68vLEsSvOKedxqa\/hAGowD4Jh\/1X\
/dH1X5sEBZpoH6E6\
/AVBLBwj3gRyzjAIAAAAAAAAAAAEAAAAAAFBLAwQUAAgICAD5q4VUAAAAAAAAAAAAAAAAAFgAEAE9UTV8tSUQxNzUtMF8yXzVfYi5UTUkBAAAA7
d3PS9NhHMDxz\/Y1nbp0zfw2Vw6CEjooJkkFPs9DZZaFCiIRHRxKoJUIFXk06iB0kS5Fvw6dhDp28FDgOSqiIKQ\
/ICQMhIIuYVnJt2f7eK2M2Ps1xp49b8Y+fP6ArXegJy4iV0RiPx6BNAXyT6ysrKhXlLZ49PwlkKP9hw\
/19XcKAOD3PZX42+PDP0+JWN9AT765u3P33vbm1nxbvj0\
/3DLQ0y3r5uClsZGhC2eGxgUAAAAAAAAAAAAAAAAAAAAAAAAAAgFKXllh0ahQbLHeInDb3Xc6NWrF77Jibcr22zC2YY6bVLNoX5qp97Pa5SbPc8
ci8sqHpd1k7a2+ZN+6eFQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAD4YxISk8bVUyq6eVa905dtqtxO3fBlqyqnkrW+ZFVZCGp8aVDl9ZeELxlVjhR
NsEWVa+UffAlVuf78rC\/1eoK20JfNqnzt3OhLnSp1DZW+bFJl\/467vqRUuVxV5UutKts\
/JX2pUWUyXvie9OopE5U7QWEHSfWZXdmPvlSr8i75xJcqVT7fPOdLpSqj5+t9Sahy8UBhOxWqLEph6nJVHhZNvUFPXbS3MlXyYWFvgSon3xf2F
ldlpGiCmCoPiiYQVbLR3or\
/ZT0tS04AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAMC6K4t+ZSAtOWkKQpOSeTfnZty0m3CDrsu1uNB9swv2pZ21IlN23J6w1uZsuV0y82bOzJhpM
2EGTZdpMaERAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAPjrUmteK0RypXifid5n1tyX6j7+9\
/vvUEsHCGo104BhAgAAAAAAAAAAQAAAAAAUEsBAgAAFAAICAgA912FVERZNpJcAAAAAAgAABYABAAAAAAAAAAAALSBAAAAAE9UTV8tSUQxNzU
tMF8yXzVfYi5NRU0BAAAAUEsBAgAAFAAICAgA\
/F2FVPeBHLOMAgAAAAABABYABAAAAAAAAAAAALSBrAAAAE9UTV8tSUQxNzUtMF8yXzVfYi5UTUQBAAAAUEsBAgAAFAAICAgA\
/F2FVGo104BhAgAAAAABABYABAAAAAAAAAAAALSBiAMAAE9UTV8tSUQxNzUtMF8yXzVfYi5UTUkBAAAAUEsGBiwAAAAAAAAAHgMtAAAAAAAAAA
AAAwAAAAAAAAADAAAAAAAAANgAAAAAAAAOQYAAAAAAABQSwYHAAAAABEHAAAAAAAAAQAAAFBLBQYAAAAAAwADANgAAAA5BgAAAAA=" }
Response example:{
"name": "examle_tm"
}

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

Clone TM localy

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

Endpoint is sync(blocking)

Response

```
Request example
{    "newName": "examle_tm" // when cloning, cloned tm would be renamed to this name(source tm is in url)
}

Response example:
Success:
{
    "msg": "newBtree3_cloned2 was cloned successfully",
    "time": "5 ms"
}

 Failure:
{
    "ReturnValue": -1,
    "ErrorMsg": "'dstTmdPath' = /home/or/.t5memory/MEM/newBtree3_cloned.TMD already exists; for request for
mem newBtree3; with body = {\n    \"newName\": \"newBtree3_cloned\"\n}"
}
```

Delete TM

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

Response

```
Response example:
success:
{
    "newBtree3_cloned2": "deleted"
},
```

Response

```
Response example:
failed:
{
    "newBtree3_cloned2": "not found"
}
```

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" } ▪ additional: "framingTags": "saveAll" - default behaviour, do nothing "skipAll" - skip all enclosing tags, including standalone tags "skipPaired" - skip only paired enclosing tags

TM must exist
It's async, so check status using status endpoint, like with reorganize in 0.5.x and up

Handling if framing tag situation differs from source to target - for skipAll or skipPaired

If framing tags situation is the same in source and target, both sides should be treated as described above.

If framing tags only exist in source, then still they should be treated as described above.

If they only exist in target, then nothing should be removed.

Response

```
Request example:{
   ["framingTags": "skipAll"["skipPaired", "saveAll"],]
   "tmxData":   "PD94bWwgdmVyc2lvbj0iMS4wIiBlbmNvZGluZz0idXRmLTgiPz4KPHRteCB2ZXJzaW9uPSIxLjQiPgogIDxoZWFkZXIgY
3JlYXRpb250b29sPSJTREwgTGFuZ3VhZ2UgUGxhdGZvcm0iIGNyZWF0aW9udG9vbHZlcnNpb249IjguMCIgby10bWY9IlNETCBUTTggRm9ybWF
0IiBkYXRhdHlwZT0ieG1sIiBzZWd0eXBlPSJzZW50ZW5jZSIgYWRtaW5sYW5nPSJlbi1HQiIgc3JjbGFuZz0iYmctQkciIGNyZWF0aW9uZGF0Z
T0iMjAxNTA4MjFUMDkyNjE0WiIgY3JlYXRpb25pZD0idGVzdCIvPgogIDxib2R5PgoJPHR1IGNyZWF0aW9uZGF0ZT0iMjAxODAyMTZUMTU1MTA
1WiIgY3JlYXRpb25pZD0iREVTS1RPUC1SNTlCT0tCXFBDMiIgY2hhbmdlZGF0ZT0iMjAxODAyMTZUMTU1MTA4WiIgY2hhbmdlaWQ9IkRFU0tUT
1AtUjU5Qk9LQlxQQzIiIGxhc3R1c2FnZWRhdGU9IjIwMTgwMjE2VDE2MTMwNVoiIHVzYWdlY291bnQ9IjEiPgogICAgICA8dHV2IHhtbDpsYW5
nPSJiZy1CRyI+CiAgICAgICAgPHNlZz5UaGUgPHBoIC8+IGVuZDwvc2VnPgogICAgICA8L3R1diB4bWw6bGFuZz0iZW4tR
0IiPgogICAgICAgIDxzZWc+RXRoIDxwaCAvPiBkbmU8L3NlZz4KICAgICAgPC90dXY+CiAgICA8L3R1PgogIDwvYm9keT4KPC90bXg+Cg=="
}Response example:Error in case of errorFrom v0_2_15
{ "%tm_name%":""} in case of success
Check status of import using status call
TMX import could be interrupted in case of invalid XML or TM reaching it's limit. For both cases check status
request to have info about position in tmx file where it was interrupted.
```

Reorganize TM

Purpose	Reorganizes tm and fixing issues.
Request	**GET** /%service%/%tm_name%/reorganize
Headers	Accept - applicaton/xml

up to v0.4.x reorganize is sync, so t5memory
starting from 0.5.x is async, so you can check status of reorganize similar to how you can check status for importTMX

Under the hood it creates new tm with $Org- prefix, then reimport all segments one-by-one, and then deletes original TM and rename reorganized TM to replace original.
This request should flush tm(from RAM to the disk) before reorganizing

reorganize would check this condition

```
if (fValidXmlInSrc && fValidXmlInTrg && (pProposal->getSourceLen() != 0) && (pProposal->getTargetLen() != 0) &&
(szTargetLanguage[0] != EOS) && (szTagTable[0] != EOS) )
```

, and in case if this condition is true and then it passes segment to putProposal function, which is also used by UpdateRequest and ImportTmx request, so other
issues could be connected to updating new tm.

In 0.4.48 reorganize responce would look like this


```
{
"too_long_reorg_0_4": "reorganized",
"time": "37 sec",
"reorganizedSegmentCount": "9424",
"invalidSegmentCount": "0"
}
```

so if there is invalid segments, inspect t5memory log

Export TMX from TM

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

This endpoint should flush tm before execution

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

returns archive(.tm file) consists with .tmd and .tmi files
This should flush tm before execution

Response

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

Get the status of TM

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

Would return status of TM. It could be 'not found', 'available' if it's on the disk but not loaded into the RAM yet, and 'open' with additional info. In case if there was at least one try to import tmx or reorganize tm since it was loaded into the RAM, additional fields would appear and stay in the statistics till memory would be unloaded.

Response

Response example:
{//just opened tm, without import\reorganize called
    "status": "open",
    "lastAccessTime": "",
    "creationTime": "20230703T122212Z",
    "tmCreatedInT5M_version": "0:5:1"
}

{// after reorgainize was called
    "status": "open",
    "reorganizeStatus": "available",
    "reorganizeTime": 100,
    "reorganizeTime": "Overall reorganize time is   : 0:00:02\n",
    "segmentsReorganized": 1112,
    "invalidSegments": 10,
    "invalidSegmentsRCs": "5005:10; ",
    "firstInvalidSegments": "123; 432; 554; 623; 659; 675; 741; 742; 753; 755; ",
    "invalidSymbolErrors": -1,
    "reorganizeErrorMsg": "",
    "lastAccessTime": "",
    "creationTime": "20230810T095233Z",
    "tmCreatedInT5M_version": "0:5:10"
}   {//not opened but available on the disk
        "status": "available"
}
{//not found tm {
    "status": "not found",
    "res": 48 // 48- both tmi and tmd files are no found, 16- only TMD file not found, 32 - only TMI file not found
}
  The tmxImportStatus could be "available", "import" or "failed" if the import had errors. If there were at least one import to that tm, new fields would appear
{//tm in process of import
    "status": "open",
    "tmxImportStatus": "import",
        "importProgress" : 56,
        "importTime": "00:00:13",
        "segmentsImported": 1356,
        "invalidSegments": 23,
        "invalidSymbolErrors": 2,
        "importErrorMsg": "",
    "lastAccessTime": "%lastAccessTime",
    "creationTime": "20230703T122212Z",
    "tmCreatedInT5M_version": "0:5:1"
}// in case if internal error happened, like t5memory would have error 5034 or 5035 which indicates, that tm size is reached it's limit and you should create new one to save new segments or part that left from tmx that you tried to import, status would look like this
{
"status": "open",
"tmxImportStatus": "failed",
"importProgress": 100,
"importTime": "Overall import time is : 0:00:19\n",
"segmentsImported": 445,
"invalidSegments": 1,
"invalidSymbolErrors": 0,
"importErrorMsg": "Warning: encoding 'UTF-16' from XML declaration or manually set contradicts the auto-sensed encoding; ignoring at column 40 in line 1; \n Fatal internal Error at column 6 in line 9605, import stopped at progress = 0%, errorMsg: TM is reached it's size limit, please create another one and import segments there, rc = 5034; aciveSegment = 1834\n\nSegment 1834 not imported\r\n\nReason = \nDocument = none\nSourceLanguage = de-DE\nTargetLanguage = en-GB\nMarkup = OTMXUXLF\nSource = in Verbindung mit Befestigungswinkel MS-...-WPE-B zur Wandmontage eines Einzelgeräts\nTarget = In combination with mounting bracket MS-...-WPE-B for wall mounting an individual component ",
"lastAccessTime": "",
"ErrorMsg": " Fatal internal Error at column 6 in line 9605, import stopped at progress = 0%, errorMsg: TM is reached it's size limit, please create another one and import segments there, rc = 5034; aciveSegment = 1834\n\nSegment 1834 not imported\r\n\nReason = \nDoc"
}

So you would have info about last segment which interrupted tm import

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:

```
{ // required fields
  "sourceLang":"en-GB",    // langs would be checked with languages.xml
  "targetLang":"de",
  "source":"For > 100 setups.",

 // optional fields
  ["documentName":"OBJ_DCL-0000000845-004_pt-br.xml"],
  ["segmentNumber":15],
  ["markupTable":"OTMXUXLF"],    //if there is no markup, default OTMXUXLF would be used.
                                                                  //Markup tables should be located inside ~/.
t5memory/TABLE/%markup$.TBL
  ["context":"395_408"],
  ["numOfProposals":20],   // num of expected segments in output. By default it's 5
  ["loggingThreshold": 0]
}
```

Response example:

Success:


```
{
    "ReturnValue": 0,
    "ErrorMsg": "",
    "NumOfFoundProposals": 1,
    "results": [
        {
            "source": "The end",
            "target": "The target",
            "segmentNumber": 0,
            "id": "",
            "documentName": "Te2.xlf",
            "sourceLang": "de-DE",
            "targetLang": "EN-GB",
            "type": "Manual",
            "author": "THOMAS LAURIA",
            "timestamp": "20231228T171821Z",
            "markupTable": "OTMXUXLF",
            "context": "2_3",
            "additionalInfo": "",
            "internalKey": "7:1",
            "matchType": "Fuzzy",
            "matchRate": 50,
            "fuzzyWords": 0,
            "fuzzyDiffs": 0
        }
    ]
```

```
}

example 2
{
  "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", // could be exact or fuzzy
    "author": "",
    "timestamp": "20190401T084052Z",
    "matchRate": 100,
     "fuzzyWords": -1, // for exact match it would be -1 here and in diffs
     "fuzzyDiffs": -1, // otherwise here would be amount of parsed words and diffs that was
                                          // used in fuzzy matchrate calculation
         "markupTable": "OTMXML",
     "context": "",
     "additionalInfo": ""
   }
 ]
}

Not found:

{
"ReturnValue": 133,
"ErrorMsg": "OtmMemoryServiceWorker::concordanceSearch::"
}
For exact match used function that's comparing strings ignoring whitespaces. First normalized strings(without
tags).
If it's the same string, then t5memory is checking string with tags and could return 100 or 97 match rate
depending on result.

Then it's checking context match rate and if document name is the same(non case sensitive)

Then it's checking and modifying exactMatchRate according to code in code block below.
After that it would store exact matches only with usMatchLevel>=100. If there would be no exact matches,
fuzzy match calculations would begin.
In case if there is at least one exact match, any fuzzy matches would be skipped.
In case if we have only one exact exact match, it's rate would be set to 102


For equal matches with 100% word matches but different whitespaces/newlines, each whitespace/newline diffs
would be count as -1%. For punctuation, at least for 0.4.50, each punctuation would count as word token. This
would be changed in future to count punctuation as whitespaces.

For fuzzy calculation tags would be removed from text, except t5:np tags, which would be replaced with their
"r" attribute to be counted as 1 word per tag.


For fuzzy rate calculation we count words and then diffs in normalized string(without tags), using this
formula:
  if (usDiff < usWords )
  {
    *pusFuzzy = (usWords != 0) ? ((usWords - usDiff)*100 / usWords) : 100;
  }
  else
  {
    *pusFuzzy = 0;
  } /* endif */ Regarging Number Protection feature, tags from number protection would be replaced with their
regexHashes from their attributes, so they would be count as 1 word each. NP with the same regex would be
```

```
counted as equal
 To count diffs, t5memory go throuht both segments to find matching tokens, to find something called snake-
line of matching tokens.
 Then It marks unmatched as INSERTED or DELETED tokens, and based on that it calculates diffs.

if it's 100% rate, we add tags and compare it again
if then it's not equal, here is how match rate would be changed - probably this would never happens, because
we have exact match test before fuzzy,
and we do exact test even if triplesHashes is different(which is pre-fuzzy calculation and if it's equal, it
could be flag that trigger exact test)

  if ( !fStringEqual )
  {
    if ( usFuzzy > 3 )
    {
      usFuzzy -= 3;
    }
    else
    {
       usFuzzy = 0;
    } /* endif */
    usFuzzy = std::min( (USHORT)99, usFuzzy );
  } /* endif */

then depending on type of translation it could tweak rate
if ( (usModifiedTranslationFlag == TRANSLFLAG_MACHINE) && (usFuzzy < 100) )
{
  // ignore machine fuzzy matches
}
else if ( usFuzzy > TM_FUZZINESS_THRESHOLD )
{
  /*****************************************************/
  /* give MT flag a little less fuzziness */
  /*****************************************************/
  if ( usModifiedTranslationFlag == TRANSLFLAG_MACHINE )
  {
    if ( usFuzzy > 1 )
    {
      usFuzzy -= 1;
    }
    else
    {
      usFuzzy = 0;
    } /* endif */
  } /* endif */
  if (usFuzzy == 100 && (pGetIn->ulParm & GET_RESPECTCRLF) && !fRespectCRLFStringEqual )
  { // P018279!
    usFuzzy -= 1;
  }
   add to resulting set
} /* endif */
} /* endif */


At the end fuzzy request replaces tags in proposal from TM with tags from request, and if matchRate >= 100,
it calculates whitespace diffs and apply matchRate-= wsDiffs
```

Response

```
ExactMatchRate calculation:so, before usExact is equal to 97 or 100, depending if strings with tags are equal
ignoring whitespaces  and then code do some tweaks.
 pClb is struct that have proposals from TM, pGetIn is fuzzy requests data

 // loop over CLBs and look for best matching entry
{
  LONG lLeftClbLen; // left CLB entries in CLB list
  PTMX_TARGET_CLB pClb; // pointer for CLB list processing
  #define SEG_DOC_AND_CONTEXT_MATCH 8
  #define DOC_AND_CONTEXT_MATCH 7
```

```
  #define CONTEXT_MATCH 6
  #define SAME_SEG_AND_DOC_MATCH 5
  #define SAME_DOC_MATCH 4
  #define MULT_DOC_MATCH 3
  #define NORMAL_MATCH 2
  #define IGNORE_MATCH 1
  SHORT sCurMatch = 0;

  // loop over all target CLBs
  pClb = pTMXTargetClb;
  lLeftClbLen = RECLEN(pTMXTargetRecord) -
  pTMXTargetRecord->usClb;
  while ( ( lLeftClbLen > 0 ) && (sCurMatch < SAME_SEG_AND_DOC_MATCH) )
  {
    USHORT usTranslationFlag = pClb->bTranslationFlag;
    USHORT usCurContextRanking = 0; // context ranking of this match
    BOOL fIgnoreProposal = FALSE;
    // apply global memory option file on global memory proposals
   if ( pClb->bTranslationFlag == TRANSLFLAG_GLOBMEM ) // pClb it's segment in TM
   {
       if ( (pGetIn->pvGMOptList != NULL) && pClb->usAddDataLen ) // pGetIn it's fuzzy requests segment
       {

           USHORT usAddDataLen = NtmGetAddData( pClb, ADDDATA_ADDINFO_ID, pContextBuffer, MAX_SEGMENT_SIZE );
           if ( usAddDataLen )
           {
               GMMEMOPT GobMemOpt = GlobMemGetFlagForProposal( pGetIn->pvGMOptList, pContextBuffer );
               switch ( GobMemOpt )
               {
                  case GM_SUBSTITUTE_OPT: usTranslationFlag = TRANSLFLAG_NORMAL; break;
                  case GM_HFLAG_OPT : usTranslationFlag = TRANSLFLAG_GLOBMEM; break;
                  case GM_HFLAGSTAR_OPT : usTranslationFlag = TRANSLFLAG_GLOBMEMSTAR; break;
                  case GM_EXCLUDE_OPT : fIgnoreProposal = TRUE; break;
               } /* endswitch */
           } /* endif */
      } /* endif */

      if ( pClb == pTMXTargetClb )
     {
        usTargetTranslationFlag = usTranslationFlag;
     } /* endif *
  } /* endif */


  // check context strings (if any)
  if ((!fIgnoreProposal)
       && pGetIn->szContext[0]
       && pClb->usAddDataLen )
   {
       USHORT usContextLen = NtmGetAddData( pClb, ADDDATA_CONTEXT_ID, pContextBuffer, MAX_SEGMENT_SIZE );
       if ( usContextLen != 0 )
       {
            usCurContextRanking = NTMCompareContext( pTmClb, pGetIn->szTagTable, pGetIn->szContext,
pContextBuffer );
       } /* endif */
    } /* endif */


  // check for matching document names
  if ( pGetIn->ulParm & GET_IGNORE_PATH )
  {
     // we have to compare the real document names rather than comparing the document name IDs
     PSZ pszCLBDocName = NTMFindNameForID( pTmClb, &(pClb->usFileId), (USHORT)FILE_KEY );
     if ( pszCLBDocName != NULL )
     {
        PSZ pszName = UtlGetFnameFromPath( pszCLBDocName );
        if ( pszName == NULL )
        {
           pszName = pszCLBDocName;
         } /* endif */
      fMatchingDocName = stricmp( pszName, pszDocName ) == 0;
```

```
    }
    else
    {
       // could not access the document name, we have to compare the document name IDs
      fMatchingDocName = ((pClb->usFileId == usGetFile) || (pClb->usFileId == usAlternateGetFile));
    } /* endif */
  }
  else
  {
     // we can compare the document name IDs
     fMatchingDocName = ((pClb->usFileId == usGetFile) || (pClb->usFileId == usAlternateGetFile));
  } /* endif */


  if ( fIgnoreProposal )
  {
    if ( sCurMatch == 0 )
    {
      sCurMatch = IGNORE_MATCH;
    } /* endif */
  }
  else if ( usCurContextRanking == 100 )
  {
    if ( fMatchingDocName && (pClb->ulSegmId >= (pGetIn->ulSegmentId - 1)) && (pClb->ulSegmId <= (pGetIn-
>ulSegmentId + 1)) )
    {
      if ( sCurMatch < SEG_DOC_AND_CONTEXT_MATCH )
      {
         sCurMatch = SEG_DOC_AND_CONTEXT_MATCH;
        pTMXTargetClb = pClb; // use this target CLB for match
        usTargetTranslationFlag = usTranslationFlag;
        usContextRanking = usCurContextRanking;
      }
    }
    else if ( fMatchingDocName )
    {
    if ( sCurMatch < DOC_AND_CONTEXT_MATCH )
    {
      sCurMatch = DOC_AND_CONTEXT_MATCH;
      pTMXTargetClb = pClb; // use this target CLB for match
      usTargetTranslationFlag = usTranslationFlag;
      usContextRanking = usCurContextRanking;
     }
     else if ( sCurMatch == DOC_AND_CONTEXT_MATCH )
     {
       // we have already a match of this type so check if context ranking
       if ( usCurContextRanking > usContextRanking )
       {
          pTMXTargetClb = pClb; // use newer target CLB for match
          usTargetTranslationFlag = usTranslationFlag;
          usContextRanking = usCurContextRanking;
       }
       // use time info to ensure that latest match is used
       else if ( usCurContextRanking == usContextRanking )
       {
         // GQ 2015-04-10 New approach: If we have an exact-exact match use this one, otherwise use timestamp
for the comparism
         BOOL fExactExactNewCLB = fMatchingDocName && (pClb->ulSegmId >= (pGetIn->ulSegmentId - 1)) && (pClb-
>ulSegmId <= (pGetIn->ulSegmentId + 1));
         BOOL fExactExactExistingCLB = ((pTMXTargetClb->usFileId == usGetFile) || (pTMXTargetClb->usFileId ==
usAlternateGetFile)) &&
         (pTMXTargetClb->ulSegmId >= (pGetIn->ulSegmentId - 1)) && (pTMXTargetClb->ulSegmId <= (pGetIn-
>ulSegmentId + 1));
         if ( fExactExactNewCLB && !fExactExactExistingCLB )
         {
           // use exact-exact CLB for match
           pTMXTargetClb = pClb;
           usTargetTranslationFlag = usTranslationFlag;
           usContextRanking = usCurContextRanking;
         }
         else if ( (fExactExactNewCLB == fExactExactExistingCLB) && (pClb->lTime > pTMXTargetClb->lTime) )
```

```
        {
          // use newer target CLB for match
          pTMXTargetClb = pClb;
          usTargetTranslationFlag = usTranslationFlag;
          usContextRanking = usCurContextRanking;
        }
      } /* endif */
    } /* endif */
  }
  else
  {
    if ( sCurMatch < CONTEXT_MATCH )
    {
    sCurMatch = CONTEXT_MATCH;
    pTMXTargetClb = pClb; // use this target CLB for match
    usTargetTranslationFlag = usTranslationFlag;
    usContextRanking = usCurContextRanking;
    }
    else if ( sCurMatch == CONTEXT_MATCH )
    {
      // we have already a match of this type so check if context ranking
     if ( usCurContextRanking > usContextRanking )
     {
       pTMXTargetClb = pClb; // use newer target CLB for match
       usTargetTranslationFlag = usTranslationFlag;
       usContextRanking = usCurContextRanking;
     }
     // use time info to ensure that latest match is used
    else if ( (usCurContextRanking == usContextRanking) && (pClb->lTime > pTMXTargetClb->lTime) )
    {
      pTMXTargetClb = pClb; // use newer target CLB for match
      usTargetTranslationFlag = usTranslationFlag;
      usContextRanking = usCurContextRanking;
     } /* endif */
   } /* endif */
 } /* endif */
}
else if ( fMatchingDocName && (pClb->ulSegmId >= (pGetIn->ulSegmentId - 1)) && (pClb->ulSegmId <= (pGetIn-
>ulSegmentId + 1)) )
{
  // same segment from same document available
  sCurMatch = SAME_SEG_AND_DOC_MATCH;
  pTMXTargetClb = pClb; // use this target CLB for match
  usContextRanking = usCurContextRanking;
  usTargetTranslationFlag = usTranslationFlag;
}
else if ( fMatchingDocName )
{
   // segment from same document available
   if ( sCurMatch < SAME_DOC_MATCH )
   {
      sCurMatch = SAME_DOC_MATCH;
      pTMXTargetClb = pClb; // use this target CLB for match
      usTargetTranslationFlag = usTranslationFlag;
      usContextRanking = usCurContextRanking;
    }
    else if ( sCurMatch == SAME_DOC_MATCH )
    {
      // we have already a match of this type so
      // use time info to ensure that latest match is used
      if ( pClb->lTime > pTMXTargetClb->lTime )
      {
        pTMXTargetClb = pClb; // use newer target CLB for match
        usTargetTranslationFlag = usTranslationFlag;
        usContextRanking = usCurContextRanking;
      } /* endif */
    } /* endif */
  }
   else if ( pClb->bMultiple )
   {
      // multiple target segment available
```

```
      if ( sCurMatch < MULT_DOC_MATCH )
      {
        // no better match yet
        sCurMatch = MULT_DOC_MATCH;
        pTMXTargetClb = pClb; // use this target CLB for match
        usTargetTranslationFlag = usTranslationFlag;
        usContextRanking = usCurContextRanking;
      } /* endif */
    }
    else if ( usTranslationFlag == TRANSLFLAG_NORMAL )
    {
       // a 'normal' memory match is available
       if ( sCurMatch < NORMAL_MATCH )
       {
          // no better match yet
          sCurMatch = NORMAL_MATCH;
          pTMXTargetClb = pClb; // use this target CLB for match
          usTargetTranslationFlag = usTranslationFlag;
          usContextRanking = usCurContextRanking;
        } /* endif */
    } /* endif */

   // continue with next target CLB
   if ( sCurMatch < SAME_SEG_AND_DOC_MATCH )
   {
     lLeftClbLen -= TARGETCLBLEN(pClb);
     if (lLeftClbLen > 0)
     {
       usTgtNum++;
       pClb = NEXTTARGETCLB(pClb);
     }
   } /* endif */
} /* endwhile */


{
  BOOL fNormalMatch = (usTargetTranslationFlag == TRANSLFLAG_NORMAL) ||
  (usTargetTranslationFlag == TRANSLFLAG_GLOBMEM) ||
  (usTargetTranslationFlag == TRANSLFLAG_GLOBMEMSTAR);
  switch ( sCurMatch )
  {
    case IGNORE_MATCH :
      usMatchLevel = 0;
       break;
  case SAME_SEG_AND_DOC_MATCH :
      usMatchLevel = fNormalMatch ? usEqual+2 : usEqual-1;
      break;
   case SEG_DOC_AND_CONTEXT_MATCH :
       usMatchLevel = fNormalMatch ? usEqual+2 : usEqual-1; // exact-exact match with matching context
       break;
    case DOC_AND_CONTEXT_MATCH :
       if ( usContextRanking == 100 )
       {
         // GQ 2015/05/09: treat 100% context matches as normal exact matches
         // usMatchLevel = fNormalMatch ? usEqual+2 : usEqual-1;
         usMatchLevel = fNormalMatch ? usEqual+1 : usEqual-1;
        }
        else
        {
          usMatchLevel = fNormalMatch ? usEqual+1 : usEqual-1;
        } /* endif */
       break;
  case CONTEXT_MATCH :
    if ( usContextRanking == 100 )
    {
      // GQ 2015/05/09: treat 100% context matches as normal exact context matches
      // usMatchLevel = fNormalMatch ? usEqual+2 : usEqual-1;
      // GQ 2016/10/24: treat 100% context matches as normal exact matches
      usMatchLevel = fNormalMatch ? usEqual : usEqual-1;
    }
    else
```

```
    {
      usMatchLevel = fNormalMatch ? usEqual : usEqual-1;
     } /* endif */
     break;
  case SAME_DOC_MATCH :
    usMatchLevel = fNormalMatch ? usEqual+1 : usEqual-1;
    break;
  case MULT_DOC_MATCH :
     usMatchLevel = fNormalMatch ? usEqual+1 : usEqual-1;
     break;
  default :
     usMatchLevel = fNormalMatch ? usEqual : usEqual-1;
     break;
  } /* endswitch */
}
}
```

New Concordance search

Purpose	Returns entries\translations that fits selected filters.
Request	**POST** /%service%/%tm_name%/search
Params	Required: NONE iNumOfProposal - limit of found proposals - max is 200, if 0 use default value '5'

Search is made segment-by segment, and it's checking segment if it fits selected filters. You can search for EXACT or CONCORDANCE matches in this fields:
source, target, document, author, addInfo, context
To set filter, use it's SearchMode field, otherwise filter would be disabled. So you have *sourceSearchMode, targetSearchMode, documentSearchMode, authorSearchMode, addInfoSearchMode, contextSearchMode*

Search mode should be set explicitly to CONTAINS/CONCORDANCE or EXACT, otherwise filter would be ignored. But also each searchMode could have additional search parameters "CONTAINS, caseinsensetive, WHITESPACETOLERANT, INVERTED", all that values is not important, as well as delimiter. By default search is case sensetive. If you add Inverted option, check for that filter would be reverted.
To check how filters would be parsed, check json in responce. Field with that info could look like this:

```
"Filters": "
Search filter, field: SOURCE FilterType::CONTAINS SearchStr: 'THE'; Options: SEARCH_FILTERS_NOT|SEARCH_CASEINSENSITIVE_OPT|SEARCH_WHITESPACETOLERANT_OPT|;\n
Search filter, field: TARGET FilterType::EXACT SearchStr: ''; Options: SEARCH_CASEINSENSITIVE_OPT|;\n
Search filter, field: ADDINFO FilterType::CONTAINS SearchStr: 'some add info'; Options: SEARCH_WHITESPACETOLERANT_OPT|;\n
Search filter, field: CONTEXT FilterType::EXACT SearchStr: 'context context'; Options: ;\nSearch filter, field: AUTHOR FilterType::CONTAINS SearchStr: ''; Options: ;\n
Search filter, field: DOCUMENT FilterType::CONTAINS SearchStr: 'evo3_p1137_reports_translation_properties_de_fr_20220720_094902'; Options: SEARCH_FILTERS_NOT|;\n
Search filter, field: TIMESTAMP FilterType::RANGE Range: 20000121T115234Z - 20240121T115234Z Options: ;\n",
```

It's possible to apply filter just with SearchMode, like if you would type "*authorSearchMode": "exact",*but there would be no *"author"* field, it would look for segments, where author field is empty.

Also there are timespan parameter, to set it, use this fields and format:

```
"timestampSpanStart": "20000121T115234Z",
"timestampSpanEnd": "20240121T115234Z",
```

You should set both parameters to apply filter, otherwise you would get error as return. Check output to see how it was parsed and applied.
By default all mentioned filters is applied in **logical and** combination, but you can change that globaly with adding

```
"logicalOr": 1,
```

Then all mentioned filters would be applied in **logical or** combination(please, use 1 to set this to true, boolean type is not supported by json parser in t5memory). Supported since 0.6.5

```
"onlyCountSegments": 1
```

Instead of returning segments, just count them and return counter in

```
"NumOfFoundSegments": 22741
```

Also there are lang filters, they would always be applied to selection of segments that passed previous filters, so value of `"logicalOr": 1,` wouldn't be applied to that.
To set language filters, use this fields:

```
"sourceLang":"en-GB",
"targetLang":"de",
```

Lang filters could be applied with major lang feature, so source lang in this case would be applied as exact filter for source lang, but target lang would check if langs is in the same lang group. That check is done in languages.xml file with isPreferred flag.
Lang filters and if filters is combined in logical or or logical and you can check in `GlobalSearchOptions` field of responce. It could look like this:

```
"GlobalSearchOptions": "SEARCH_FILTERS_LOGICAL_OR|SEARCH_EXACT_MATCH_OF_SRC_LANG_OPT, lang = en-GB|SEARCH_GROUP_MATCH_OF_TRG_LANG_OPT, lang = de",
```

Other that you can send is:

```
"searchPosition": "8:1",
"numResults": 2,
"msSearchAfterNumResults": 250
"loggingThreshold": 4 - check other requests,
```

So search position is position where to start search internaly in btree. This search is limited by num of found segment(set by numResults) or timeout(set by msSearchAfterNumResults), but timeout would be ignored in case if there are no segments in the tm to fit params. Max numResults is 200.

You can send empty json and search would work fine, but it would just return first 5 segments in tm
You can go through all segment with using this 2 fields
"searchPosition": "8:1",
"numResults": 200

and just updating searchPosition with NewSearchPosition
from responce.

Response

```
{
    "logicalOr": 1,
     "source":"the",
    "sourceSearchMode":"CONTAINS, CASEINSENSETIVE, WHITESPACETOLERANT, INVERTED",

    "target":"",
    "targetSearchMode":"EXACT, CASEINSENSETIVE",

    "document":"evo3_p1137_reports_translation_properties_de_fr_20220720_094902",
    "documentSearchMode":"CONTAINS, INVERTED",

     "author":"some author",
     "authorSearchMode":"CONTAINS",

    "timestampSpanStart": "20000121T115234Z",
    "timestampSpanEnd": "20240121T115234Z",

    "addInfo":"some add info",
    "addInfoSearchMode":"CONCORDANCE, WHITESPACETOLERANT",


    "context":"context context",
    "contextSearchMode":"EXACT",

    "sourceLang":"en-GB",
    "targetLang":"SV",
    "searchPosition": "8:1",
    "numResults": 2,
    "msSearchAfterNumResults": 25,
    "loggingThreshold": 3
}



So here search would be done in logical or way, so if any of source, target, document, context, author,
timestamp filters returns true, result would be added to set, which then would be filtered out by sourceLang
on exact match check and targetLang on groupLang check.
Search would start from position "8:1"(tm data start at "7:1" but if you wan't to start from the beggining,
just avoid that param.
numResuts:2 - so if there would be 2 segments found, search would end
"msSearchAfterNumResults": 25 - 25ms after first found segment, search would end, even if more segments was
found, responce would contain "NewSearchPosition": "10:1", which can be used in searchPosition to continue
search

Response example:Success:
example{
"Filters": "Search filter, field: SOURCE FilterType::CONTAINS SearchStr: 'THE'; Options:
SEARCH_FILTERS_NOT|SEARCH_CASEINSENSITIVE_OPT|SEARCH_WHITESPACETOLERANT_OPT|;\n
Search filter, field: TARGET FilterType::EXACT SearchStr: ''; Options: SEARCH_CASEINSENSITIVE_OPT|;\n
Search filter, field: ADDINFO FilterType::CONTAINS SearchStr: 'some add info'; Options:
SEARCH_WHITESPACETOLERANT_OPT|;\n
Search filter, field: CONTEXT FilterType::EXACT SearchStr: 'context context'; Options: ;\n
Search filter, field: AUTHOR FilterType::CONTAINS SearchStr: ''; Options: ;\n
Search filter, field: DOCUMENT FilterType::CONTAINS SearchStr:
'evo3_p1137_reports_translation_properties_de_fr_20220720_094902'; Options: SEARCH_FILTERS_NOT|;\n
Search filter, field: TIMESTAMP FilterType::RANGE Range: 20000121T115234Z - 20240121T115234Z Options: ;\n",
"GlobalSearchOptions": "SEARCH_FILTERS_LOGICAL_OR|SEARCH_EXACT_MATCH_OF_SRC_LANG_OPT, lang = en-
GB|SEARCH_GROUP_MATCH_OF_TRG_LANG_OPT, lang = sv",
"ReturnValue": 0,
```

```
"ReturnMessage": "FOUND",
"NewSearchPosition": "10:1",
"results": [
{
"source": "Congratulations on the purchase of a <ph x=\"101\"/> machine control system.",
"target": "Gratulerar till köpet av maskinstyrningsystemet <ph x=\"101\"/>.",
"segmentNumber": 5740419,
"id": "",
"documentName": "none",
"sourceLang": "en-GB",
"targetLang": "SV-SE",
"type": "Manual",
"author": "",
"timestamp": "20170327T091814Z",
"markupTable": "OTMXUXLF",
"context": "",
"additionalInfo": "",
"internalKey": "8:1"
},
{
"source": "The <ph x=\"101\"/> System is an ideal tool for increasing productivity in all aspects of the
construction earthmoving industry.",
"target": "Systemet <ph x=\"101\"/> är ett verktyg som lämpar sig perfekt för att öka produktiviteten inom
alla delar av bygg- och anläggningsområdet.",
"segmentNumber": 5740420,
"id": "",
"documentName": "none",
"sourceLang": "en-GB",
"targetLang": "SV-SE",
"type": "Manual",
"author": "",
"timestamp": "20170327T091814Z",
"markupTable": "OTMXUXLF",
"context": "",
"additionalInfo": "",
"internalKey": "9:1"
}
]
}
SearchPosition / NewSearchPositionFormat: "7:1"
First is segment\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::"
}
```


Here is search request with all possible parameters:


```
{
"logicalOr": 1,

 "source":"the",

  "sourceSearchMode":"CONTAINS, CASEINSENSETIVE, WHITESPACETOLERANT, INVERTED",

  "target":"", "targetSearchMode":"EXACT, CASEINSENSETIVE",

  "document":"evo3_p1137_reports_translation_properties_de_fr_20220720_094902",

   "documentSearchMode":"CONTAINS, INVERTED",
```

```
  "author":"some author",
  "authorSearchMode":"CONTAINS",

  "timestampSpanStart": "20000121T115234Z",

  "timestampSpanEnd": "20240121T115234Z",

  "addInfo":"some add info",

  "addInfoSearchMode":"CONCORDANCE, WHITESPACETOLERANT",

  "context":"context context",

  "contextSearchMode":"EXACT",

  "sourceLang":"en-GB",

  "targetLang":"SV",

  "searchPosition": "8:1",

  "numResults": 2,

   "msSearchAfterNumResults": 25,
   "loggingThreshold": 3
}
```


All fields is optional, but some depends on other, so error should be returned in case of not providing required field

So request with this body would also work:


```
{
}
```


Parameter	valueType	default value	possible values	requireField	description
sourceLang	string	""	langs that can be matched to langs in languages.xml	-	Filter segments on src/trg lang attribute, If specified lang is preffered, matching is done based on lang family, otherwise on exact match
targetLang					
searchPosition	string	"" (search would start from "7:1" then	"8:1" etc		point where to start search in tmd file
numResults	int	5	(0....200]		points how many matches return in current request
msSearchAfterNumResults		0	no check		sets how many ms should pass between first found segment and search stop, if it didn't reach the end yet.
loggingThreshold		-1	[0...6]		additional field to set log level on the run
logicalOr	int	0	0 for false, any other number as true, example: "logicalOr": 1, "onlyCountSegments": 1		by default source, target, document, author, context, addinfo, timestamp is combined in logical AND, but by sending here "OR" you can switch that to logical OR, any other value would left it in default AND state. Doesn't apply to sourceLang and targetLang filters, they are always in AND state
onlyCountSegments					instead of returning segment, would go in search till the end of tm and return total number of segments, that returns true with selected filters
source	string	""	any string, example "source": "data in the segment"	sourceSearchMode	Sets what to look for in source of the segments, based on type of search, specified in sourceSearchMode(exact, concordance). If sourceSearchMode is not specified, returns an error.
target				targetSearchMode	--//–(the same as above but for corresponding fields)
document				documentSearchMode	
author				authorSearchMode	
context				contextSearchMode	
addInfo				addInfoSearchMode	
timestampSpanStart	string		string with date in format "20240121T115234Z"	timestampSpanEnd	Sets filter for time. You need to provid both timestamps, or none, otherwise request would return an error. Could be used in "OR" combination in "logicalOr": 1,, but, maybe, it's better to change that behaviour to similar like with langs(Always AND)
timestampSpanEnd				timestampSpanStart	
sourceSearchMode	string	""	String with required **EXACT** or **CONCORDANCE** (or **CONTAINS**, what's equal to **CONCORDANCE**) words and some optional, like *CASEINSENSETIVE* for non case sensetive comparison, *WHITESPACETOLERANT* for modifying whitespaces(result of this actions you can see in filters in responce) *INVERTED* for applying filter in inverted state, so to return false on match and true if no match. Logical NOT Attributes is not case sensetive, Separator doen't matters	-	Sets type of search for corresponding field. If you set, for example, "authorSearchMode" = "EXACT", but don't provide any author in request, author field would be "", so request would look for segments, where author equals to "". The same is true for other fields Examples: 1) "source": "the text inside" "sourceSearchMode":"CONTAINS, CASEINSENSETIVE, WHITESPACETOLERANT, INVERTED", - search would be for all segments, which doesn't contains "the text inside" in non case sensetive mode and with normalizing whitespaces. 2) "author": "Ed Sheeran", "authorSearchMode" = "Exact", -search would be done on exact case sensetive matches with "Ed Sheeran" in author field 3) "author": "Ed Sheeran", "authorSearchMode" = "CASEINSENSETIVE", - ERROR, search mode(Exact\Contains) is not selected 4) "author": "Ed Sheeran", - ERROR, search mode(Exact\Contains) is not selected 5) "authorSearchMode" = "CONTAINS",- OK, filter would check if segment contains "", so every segment would return true then
sourceSearchMode					
targetSearchMode					
documentSearchMode					
authorSearchMode					
contextSearchMode					
addInfoSearchMode					

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", // could be Source, Target, SourceAndTarget - says where to do search
    ["searchPosition": "",]
    ["numResults": 20,]
    ["msSearchAfterNumResults": 250,]
        ["loggingThreshold": 0]
}
Response example:Success:
example_new{
  "ReturnValue": "ENDREACHED_RC",
  "NewSearchPosition": null,
  "results": [
    {
      "source": "The end",
      "target": "The target",
      "segmentNumber": 0,
      "id": "",
      "documentName": "Te2.xlf",
      "sourceLang": "de-DE",
      "targetLang": "EN-GB",
     "type": "Manual",
     "author": "THOMAS LAURIA",
     "timestamp": "20231228T171821Z",
      "markupTable": "OTMXUXLF",
      "context": "2_3",
      "additionalInfo": "",
      "internalKey": "7:1"
    }
  ]
}

example_old
{
  "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

This request would made changes only in the filebuffer(so files on disk would not be changed)
To write it to the disk just call request which would flush tm to the disk as part of execution(exportTMX, exportTM, cloneTM) or using SaveAllTms request

Response

```
Request example:
{
    "source": "The end",
    "target": "The target",
    "sourceLang": "en", // langs would be checked with languages.xml
    "targetLang": "de",
```

```
//additional field
    ["documentName": "Translate5 Demo Text-en-de.xlf"],
    ["segmentNumber": 8,]
    ["author": "Thomas Lauria"],
    ["timeStamp": "20210621T071042Z"], // if there is no timestamp, current time would be used
    ["context": "2_2"], // context and addInfo would be saved in TM in the same field
    ["addInfo": "2_2"],
    ["type": "Manual"], // could be GlobalMemory, GlobalMemoryStar, MachineTranslation, Manual, by default
Undefined
    ["markupTable": "OTMXUXLF"], //if there is no markup, default OTMXUXLF would be used.
                                                                    //Markup tables should be located inside ~/.
t5memory/TABLE/%markup$.TBL
    ["loggingThreshold": 0],
        ["save2disk": 0]   // flag if we need to flush tm to disk after update. by default is true
}

here are data struct used for search, so you can see max numbers of symbols
typedef struct _LOOKUPINMEMORYDATA
{
  char szMemory[260];
  wchar_t szSource[2050];
  wchar_t szTarget[2050];
  char szIsoSourceLang[40];
  char szIsoTargetLang[40];
  int lSegmentNum;
  char szDocName[260];
  char szMarkup[128];
  wchar_t szContext[2050];
  wchar_t szAddInfo[2050];
  wchar_t szError[512];
  char szType[256];
  char szAuthor[80];
  char szDateTime[40];
  char szSearchMode[40]; // only for concordance search
  char szSearchPos[80]; // only for concordance search
  int iNumOfProposals;
  int iSearchTime;
  wchar_t szSearchString[2050];
} LOOKUPINMEMORYDATA, *PLOOKUPINMEMORYDATA;

Response example:success:
example_new{
  "source": "The end",
  "sourceNPRepl": "The end",
  "sourceNorm": "The end",
  "target": "The target",
  "segmentNumber": 0,
  "id": "",
  "documentName": "Te2.xlf",
  "sourceLang": "DE-DE",
  "targetLang": "EN-GB",
  "type": "Manual",
  "author": "THOMAS LAURIA",
  "timestamp": "",
  "markupTable": "OTMXUXLF",
  "context": "2_3",
  "additionalInfo": "addInfo2",
  "internalKey": "8:1"
}

example_old
{
"sourceLang": "de-DE",
"targetLang": "en-GB",
"source": "The end",
"target": "The target",
"documentName": "Translate5 Demo Text-en-de.xlf",
"segmentNumber": 222,
"markupTable": "OTMXUXLF",
"timeStamp": "20210621T071042Z",
"author": "Thomas Lauria"
```

```
}

in case if similar record exists, t5memory comparing source text,
if it's the same, t5memory would compare docName,
if it's the same,t5memory would compare timestamps and would leave only newer one

in case if TM is alreade reached it's limit, you would get
{
"ReturnValue": 5034,
"ErrorMsg": ""
}or{
"ReturnValue": 5035,
"ErrorMsg": ""
}
```

UpdateEntry Pseudo code

```
Update entry pseudo code:update segment/import
{
  if we have triples equal match (candidate for exact match)
  {
    UpdateTmRecord
    if(updateFailed)
      AddToTMAsNewKey
      if(added) UpdateTmIndex
  }else{
    AddToTMAsNewKey
    if(added) UpdateTmIndex
  }
}

UpdateTmRecord{
  getListOfDataKeysFromIndexRecord
  sortThemByTriplesMatchesWithProposal(first have biggest match)

  foreach key untill fStop==true{
    readTmRecord // tm record is 16kB block in file, first number in "7:1"

    //compare tm record data with data passed in the get in structure
    CompareAndModifyPutData
    if(NO_ERROR) set fStop = true;
  }
}

CompareAndModifyPutData{
  if source strings are equal
    Delete old entry - with TMLoopAndDelTargetClb
  if fNewerTargetExists -> fStop = TRUE
  Loop thru target records
    loop over all target CLBs or until fStop
      if segment+file id found (exact-exact-found!)
        update time field in control block
        set fUpdate= fStop=TRUE
        update context info
      if not fStop
        goto next CLB
    endloop
    if no matching CLB has been found (if not fStop)
      add new CLB (ids, context, timestamp etc. )
    endloop
  endloop

  if fupdated, update TM record
  if !fStop (all target record have been tried & none matches )
    add new target record to end of tm record
  else
    return source_string_error // errcode for UpdateTmRecord to go to the next TM record in prepared list
}

TMLoopAndDelTargetClb{
  loop through all target records in tm record checking
    loop over all target CLBs or until fStop
      if lang + segment+file id found (exact-exact-found!)
        if entry is older
          delete it, fDel = TRUE
        else set fNewerTargetExists=TRUE(would be used in CompareAndModifyPutData)
          goon with search in next tgt CLB (control block)
      else
        goon with search in next tgt CLB (control block)
        endloop
      endif
    if not fDel
      position at next target record
  endloop
}
```

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

This request would made changes only in the filebuffer(so files on disk would not be changed)
To write it to the disk just call request which would flush tm to the disk as part of execution(exportTMX, exportTM, cloneTM) or using SaveAllTms request

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]
}
Responce example:
{
  "fileFlushed": 0,
  "results": {
     "source": "The tar",
     "target": "The target",
     "segmentNumber": 0,
     "id": "",
     "documentName": "Te2.xlf",
     "sourceLang": "de-DE",
     "targetLang": "EN-GB",
     "type": "Manual",
     "author": "THOMAS LAURIA",
     "timestamp": "20231229T125701Z",
     "markupTable": "OTMXUXLF",
     "context": "2_3",
    "additionalInfo": "",
    "internalKey": "7:1"
  }
}
```

Delete entries / mass deletion

Purpose	Deletes entries\translation

Request	**POST** /%service%/%tm_name%/entriesdelete
Params	This would start reorganize process which would remove like reorganize bad segments and also would remove segments that gives true when checking with provided filters combined with logical AND. So if you provide timestamps and addInfo, only segments within provided timestamp and with that addInfo would not be imported to new TM(check reorganize process). Every parameter is optional, so empty json would just start reorganize async process. If you provide one of timestamps you would get error - please provide both. To add parameter you should set it's SearchMode to be EXACT|CONCORDANCE(non case sensetive) If only searched string provided, but not search mode - you would get error.

Response

```
Request example:
{
["addInfo": "ADD_INFO"],
["addInfoSearchMode" : "EXACT"],
["context": "CONTEXT"],
["contextSearchMode": "concordance"],
["author": "AUTHOR"],
["authorSearchMode": "exact"],
["document":"document"],
["documentSearchMode":"CONCORDANCE"],
["timestampSpanStart": "20000121T115234Z"],
["timestampSpanEnd": "20240121T115234Z"]
}
Responce example:
{
  "fileFlushed": 0,
  "results": {
     "source": "The tar",
     "target": "The target",
     "segmentNumber": 0,
     "id": "",
     "documentName": "Te2.xlf",
     "sourceLang": "de-DE",
     "targetLang": "EN-GB",
     "type": "Manual",
     "author": "THOMAS LAURIA",
     "timestamp": "20231229T125701Z",
     "markupTable": "OTMXUXLF",
     "context": "2_3",
    "additionalInfo": "",
    "internalKey": "7:1"
  }
}
```

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 4 files': '/home/or/.t5memory/MEM/mem2.TMD, /home/or/.t5memory/MEM/mem2.TMI, /home/or/.t5memory/MEM
/newBtree3.TMD, /home/or/.t5memory/MEM/newBtree3.TMI'
} 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. In t5memory there are 2 systems of logs - one from glog library and could be set in launch as commandline parameter and one is internal to filter out logs based on their level, can be set with every request that have json body with additional ["loggingThreshold": 0] parameter or at startup with flag.
[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. DEVELOP could be used in really low level debugging, but most of the time DEBUG log is more useful, since DEVELOP would log a lot of logs. Transaction logs have the highest level of severity but it's severity is also changes with -v parameter, so with --v=2 it would be the highest log level(this log is not used often, it's only to track something like end or start of request) but with default --v=0 it's severity is belowe WARNING

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

gLog part - it have it's own configuration with command line flags. you can see all possible flags for t5memory with ./t5memory --help command.
main parameter here is --v and you can set it to 2 or 0(default).
By default it set to 0, in that case all not-errors would be avoided in logs, except startup.
idea of --v=1 was to have logBuffer to keep log in some stream and in case of error show previous logs for that request, but it seems not so usefull, so it was not fixed and it's not working properly
--v=2 is basicaly disables that buffering, so
In case of error or fatalError, log would be written with info about what request caused that log to happen(but that info would be truncated to 3000 symbols, this is important for importTMX), but if there are second error with the same request, new logs would not have that requests info

Some parameters combinations:
Default - --t5loglevel=2(T5INFO), --v=0, in this case you could see only init messages and errors only, with info about requests that caused error to happen
Change only --v=2 - t5loglevel would be set by default to 2(T5INFO), so you could see T5INFO, T5WARNING, T5ERROR, T5FATAL, T5TRANSACTION messages
Debug production --t5loglevel=1(T5DEBUG), --v=2 - should be enough to have some info about issues, a lot of logs, but not as much as with Develop
Develop --t5loglevel=0(T5DEVELOP), --v=2 - all possible logs, includes entering to some functions, some step-by-step mechanisms logs(like how t5memory is parsing and hashing strings) etc. Useful only when you can reproduce issue so you don't get lost in logs from just normal behaviour or when it's crashing etc.

It's possible to change t5loglevel with some requests, so for example for some specific update request, you can set it to some lower log level and then set it back. It would affect other threads, but since in logs you have info about thread, it could be useful tool.

Seems like --v parameter it's not quite useful, maybe should be refactored, since with --v=0 you wouldn't get any messages with severity lower than T5ERROR, except init process.
But gLog library could be connected to some other libs in proxygen package

Here are all glog flags:
Flags from src/logging.cc:
-alsologtoemail (log messages go to these email addresses in addition to logfiles) type: string default: ""
-alsologtostderr (log messages go to stderr in addition to logfiles) type: bool default: false
-colorlogtostderr (color messages logged to stderr (if supported by terminal)) type: bool default: false
-drop_log_memory (Drop in-memory buffers of log contents. Logs can grow very quickly and they are rarely read before they need to be evicted from memory. Instead, drop them from memory as soon as they are flushed to disk.) type: bool default: true
-log_backtrace_at (Emit a backtrace when logging at file:linenum.) type: string default: ""
-log_dir (If specified, logfiles are written into this directory instead of the default logging directory.) type: string default: ""
currently: "/root/.t5memory/LOG/"
-log_link (Put additional links to the log files in this directory) type: string default: ""
-log_prefix (Prepend the log prefix to the start of each log line) type: bool default: true
-logbuflevel (Buffer log messages logged at this level or lower (-1 means don't buffer; 0 means buffer INFO only; ...)) type: int32 default: 0
-logbufsecs (Buffer log messages for at most this many seconds) type: int32 default: 30
-logemaillevel (Email log messages logged at this level or higher (0 means email all; 3 means email FATAL only; ...)) type: int32 default: 999
-logfile_mode (Log file mode/permissions.) type: int32 default: 436
-logmailer (Mailer used to send logging email) type: string default: "/bin/mail"
-logtostderr (log messages go to stderr instead of logfiles) type: bool default: false
-max_log_size (approx. maximum log file size (in MB). A value of 0 will be silently overridden to 1.) type: int32 default: 1800
-minloglevel (Messages logged at a lower level than this don't actually get logged anywhere) type: int32 default: 0
-stderrthreshold (log messages at or above this level are copied to stderr in addition to logfiles. This flag obsoletes --alsologtostderr.) type: int32 default: 2
-stop_logging_if_full_disk (Stop attempting to log to disk if the disk is full.) type: bool default: false

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 - obsolete - use commandline flags instead

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

TM files structure and other related info

Info below is actual for version 0_5_x

Starting from version 0_5_0 .mem file is excluded from TM files - tm now consists only with .tmd and .tmi files. That files have 2kb headers which have some useful information, like creation date and version in which that file was created. In general, changing mid_version number means binary incompatible files. During reorganize there would be created new empty tm and then segments would be reimported from previous, and then old files would be deleted and new ones would be renamed to replace old files. That means that reorganize would also update creation t5memory version of files to the newest.

TM file is just archive with tmi and tmd files.

tmd and tmi files should be flushed in a safe way - saved on disk with temporary filename and then replacing old files.(Should be implemented)

There is tmmanager(as singletone) which have list of tm, and one tm instance have two binary trees(for both (tmd)data and (tmi)index files), with each have own filebuffer instance(before there used to be a pool of filebuffers and it's files operation functions, like write, read, close and open was handling requests).

Request handler - it's an instance of class in request handler hierarhy classes. For each type of requests there is class to handle it. In general it have private functions "*parseJSON*"(would parse json if provided and would return error if json is invalid), "*checkData*"(whould check if all required fields was provided), "*requestTM*"(would request *readOnly*, *write* or *service* tm handlers. It would load tm if it is not loaded in RAM yet) and "*execute"* - original requests code. And also it has public function "*run*" which is stategy template to operate listed private function.

The TMs is saved in TMManager using smart pointers(it's pointer which track references to itself and call destructor automaticaly). That means that on request it's possible to clear list from some TM, while it would still be active in other thread(like in fuzzy search). Then ram would be freed at the end of last request handling that TM.
In case if in the middle of some request(like fuzzy search) there was a call to delete tm, first we clear TMlist(but we keep smart pointer in fuzzy requests thread, so this is not calling destructor yet, but would after fuzzy request would be done). Destructor would try to flush filebuffer into filesystem but because there is no files in the disk, filebuffers would not create them again and it would just clean the RAM(in that case log would be writen about filebuffer flush not founding file in the folder).

From TMManager, request could ask for one of 3 types of tm handers - readonly, write or service. ReadOnly\write requests here have it's name from inside-tm perspective(so operations with tm files in filesystem is service requests).
ReadOnly(concordance search, fuzzy search, exportTmx) would be provided if there is no write handlers, for write handlers(deleteEntry, updateEntry, importTmx) there should be no other write handlers and no readOnly handlers. Service handlers could mean different for different requests. For example status request should be able to access something like readonly handler, but it shouldn't be blocked if there is any write requests, since it's used for checking import\reorganize status and progress. For some filesystem requests(deleteTM, createTM, cloneTM, importTM, exportTM(internal format)) there should be other blocking mechanism, since most of them even doesn't require to load tm into the ram.

In case if tm is not in RAM, requesting handler from TMManager would try to load TM into the RAM, considering RAM limit explained in this document.

NUMBER PROTECTION TAGS (NP TAG, t5:n)

NP Feature is also implemented in tagReplacer, but it has other branch in code - for import it's just saves original id, r and n attributes, without generating new, for fuzzy requests it's just outputs original data without searching for mathing tag in src and trg. So NP tags is influence ID generation for other tags(or matching if it's trg segment).
For fuzzy requests TagReplacer would use GenerateNormalizedString to generate copy of string for src and input(from fuzzy request) where NP tags would be replaced with their **r** attribute(to be equal to 1 word in match) and then in fuzzy calculation other tags would be removed.
So
"Press the <t5:n id="5" r="**encodedRegex**" n="25th of 2043"/>, power button to turn on <bpt x="501" i="1"/>text<ept i="1"/>
for fuzzy requests would give you

"Press the **encodedRegex**, power button to turn on <bpt id="501" rid="1"/>text<ept rid="1"/>"
first and for fuzzy calculation it would become:
"Press the **encodedRegex**, power button to turn on text"

and saved segment would be:
"Press the <t5:n id="5" r="**encodedRegex**" n="2nd of 1999"/>, power button to turn on *the* <bpt id="501" rid="1"/>text<ept rid="1"/>"
would become
"Press the **encodedRegex**, power button to turn on *the* text"
And fuzzy match would give you 92% because it counted 13 words and 1 diff. [(13-1)/13 = 0.92]

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.