TAUS Translation API – Version 2.0

TAUS Technical Specification -
A Common Translation Services API

May 2014

TAUS.NET

AUTHORS
Klemens Waldhör, TAUS

TAUS REVIEW TEAM
Achim Ruopp, TAUS
Chase Tingley, Spartan Consulting
Vinod Sudharshan, TAUS

REVIEWED AND COMMENTED BY
Rahzeb Choudhury, TAUS
Sanjay Bhansali, Vladimir Weinstein, Google
Gustavo Lucardi, Trusted Translations
Grant Straker, Straker Translations
Hans Timmerman, AVB Vertalingen
Marco Trombetti, Translated.net
Jörg Schütz, bioloom group
Alan Melby, Linport project

INITIAL AUTHOR
Brian McConnell, MyGengo

DOCUMENT VERSION:
1.1 from 20.09.2012
1.2 from 23.01.2013
2.0c from 23.04.2014
2.0dfrom 16.05.2014

Published by TAUS BV, De Rijp, The Netherlands.

For further information, please email api@taus.net.

Table of Content

Overview TAUS Translation API translation request methods 4
RESTful Webservices 6
Request IDs 6
Handling requests and parameter with conflicting requirements 6
Using files and URLs 7
File Handling 7
URL Handling 8
Call Back URLs 8
Request Definition and Structure 9
Attribute Naming 9
Attribute values 9
Language Codes 9
Translation request 9
Comment request 10
Score request 11
Callback request 12
Error Response 13
Important HTTP Status Codes 13
Error Response Object 13
TRANSLATION Request related Methods 14
GET 14
POST 15
PUT 15
DELETE 18
COMMENT related Methods 19
GET 19
POST 19
PUT 20
DELETE 20
SCORE related Methods 21
GET 21
POST 21
PUT 22
DELETE 22
CALLBACK related Methods 23
POST 23
Querying individual translation, comment and score attributes 23
US Translation API translation (V1.4) request methods 24
References 25

Overview TAUS Translation API translation request methods

The following table describes the basic requests supported by API Version 2.0. An overview of the requests methods of API version 1.4 is given at the end of the paper.

Request Method	Function / HTTP Verbs	Typical Use Cases	Required
translation	Get, submit, change, delete translations	This is the core method of the API and processes translation	Yes
	GET	Get a translation request based on an <id> or some filter criteria	
	POST	Create a new translation request	
	PUT	Change a translation request Esp. the following methods are supported: /translation/accept /translation/reject /translation/cancel /translation/confirm	
	DELETE	Remove a translation request	
comment	View and submit comments about translations	Crowd translation services, professional translation services that allow user feedback, comments or discussion	No
	GET	Get a comment request based on an <id> or some filter criteria	
	POST	Create a new comment request	
	PUT	Change a comment request	
	DELETE	Remove a comment request	
score	View and submit scores about translations	Crowd and professional translation services that allow user feedback and/or professional translator QA services, as well as moderated crowd translation systems	No
	GET	Get a score request based on an <id> or some filter criteria	
	POST	Create a new score request	
	PUT	Change a score request	
	DELETE	Remove a score request	
callback	Run a callback function	Call a function at requester side when changes happen on server side	No
	POST	Send request changes using a callback URL	
All methods	Get request attribute values	This allows retrieving specific attributes and their values	

Request Method	Function / HTTP Verbs	Typical Use Cases	Required
{request-method}/{attribute}/{id}		Return the value of a specific attribute for a request method above	No

* We might move this method out from the core

RESTful Webservices

The following methods (HTTP verbs; RFC 2616[1]) are based on the CRUD principle:

Operation	HTTP Method	Comment
CREATE	POST	All methods that create a new request on the server
READ	GET	All methods that return a request or other information; does not change anything on the server side
UPDATE	PUT	All methods that change a given request
DELETE	DELETE	All methods that remove requests

All parameters are passed through JSON MIME type (application/json). The following shows a JSON formatted example translation request:

```
{ "translationRequest":
      {
            "id": "2b575fdc-f6af-4b9e-850d-9dc0884c6595",
            "sourceLanguage": "DE-DE",
            "targetLanguage": "EN-US",
            "source": "Diesen Text übersetzen",
            "mt": true,
            "professional": false,
            "creationDatetime": "2014-05-14T19:20:30+01:00",
            "updateCounter": 0
      }
}
```

Request IDs

Each request has a unique id {id} associated with it. Unique ids have to be generated by the originator (requester) of the request. Request ids follow the GUID-format (Globally Unique Identifier[2]). Example: `2b575fdc-f6af-4b9e-850d-9dc0884c6595`. GUIDs can be easily generated by various programming languages. Request IDs should not be changed by applications working with the request once the GUID has been generated.

Handling requests and parameter with conflicting requirements

If conflicts occur between parameter values specified in the request and values specified in the payload of a request the application has to return an http error code **409** (Conflict in resource) together with an error

[1] http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html

[2] http://pubs.opengroup.org/onlinepubs/9629399/apdxa.htm; http://tools.ietf.org/html/rfc4122

object. A payload is defined here as any part of the request (attribute – value pairs) or files attached to the request. A typical payload could be represented by an XLIFF[3] file or a TIPP[4] package.

Example: Different values in the request for source language and the language specified in a XLIFF file.

Using files and URLs

Depending on the type of translation required clients can optionally submit documents as a files or URL based. This is an alternative instead of sending the texts in the `source` and `target` field of a translation request.

In the following Request Body describes the structure of the request body if required. If a "*" is used for content type the "*" needs to be replaced with the correct format (for example xml or similar). The Request Body only describes the essential parts, not all required parts for a HTTP request (like content-length and so on).

The type `multipart/form-data`[5] should be used for `POST` and `PUT`. Multipart forms require that each part of the multipart message is divided by boundary string (section 4.1[6] of the multipart/form-data description). In the examples below "`aboundarystring`" is used to separate the different parts of the message.

File Handling

The following naming schemes must be applied if files are used:

Names for files (attached) are `sourceDocument` and/or `targetDocument`.

```
Content Type: multipart/form-data; boundary=aboundarystring

--aboundarystring
Content Type: application/json: name="translationRequest"
translation request

--aboundarystring
Content Type: application/*;name="sourceDocument"
source document

--aboundarystring
Content Type: application/*;name="targetDocument"
target document

--aboundarystring
```

[3] https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xliff

[4] https://code.google.com/p/interoperability-now/

[5] http://www.ietf.org/rfc/rfc2388.txt

[6] "As with other multipart types, a boundary is selected that does not occur in any of the data.", p 1, section 4.1of http://www.ietf.org/rfc/rfc2388.txt .

URL Handling

In many cases documents, more general any kind of payloads cannot be sent as part of the message (e.g. because of the size of the document). In this case URLs representing a resource can be used instead. URLs can be used to reference resources on the web which should be used as part of the translation process. The service is then responsible for retrieving these resources. If URLs[7] are used the following option should be used:

```
Content-type: message/external-body; access-type=URL;
      URL=http://www.foo.com/file
```

Thus requires adding some information if source or target is referenced by the URL:

```
Content-type: message/external-body; access-type=URL;
      URL=http://www.taus.net/upload/myfileen.docx”;
      type=”sourceDocument”;
Content-type: message/external-body; access-type=URL;
      URL=http://www.taus.net/upload/myfileen.docx”;
      type=”targetDocument”;
```

or using ftp[8]:

```
Content-type: message/external-body; access-type=URL;
      URL=ftp://www.taus.net/upload/myfileen.docx”;
      type=”targetDocument”;

Content-type: message/external-body;
      access-type=ANON-FTP;
      site="ftp.taus.org";
      mode="binary";
      directory="/pub/documentation"; name=" myfileen.docx ";
      type=”targetDocument”;
```

Call Back URLs

The requests support the specification of a callback URL. This URL will used by the receiver of the request to inform the requester about the progress of the request. As there are no real standards out there several formats could be used. Here are some examples formats:

a) URLs following the x-callback-url specification[9]

b) Web hooks[10]: Basically some kind of URLs which can be called by the server (e.g. using a HTTP POST message). There is no real standard definition out there. So it is up to the implementation how this link looks like.

A callback is implemented as a service on the resp. side (e.g. client). If a server calls a callback it sends a `CALLBACK POST` request to the requester of the message. The body of the CALLBACK method contains

---

7 http://tools.ietf.org/html/rfc2017

8 http://www.iana.org/assignments/access-types/access-types.xhtml

9 http://x-callback-url.com

10 http://en.wikipedia.org/wiki/Webhook

the attributes of the original request, a comment field and callbackStatus field describing the changes, actions etc. applied by the server.

Request Definition and Structure

Attribute Naming

Attribute naming follows Google JSON Style Guide[11].

Attribute values

Attribute values are in general any legal JSON attribute value. The legal values are given with attributes. If a request does not specify a value, **null**[12] is assumed. Null is also returned if an attributes is queried which was not set for the request. In case of default values the default value has to be used. Default values are given in bold. YES and NO are represented using the JSON boolean values true and false.

Language Codes

Several requests require specifying a language code. Language codes should be based on BCP47[13] conventions. Examples are: de for German, de-DE for German for Germany, sr-Cyrl for Serbian written using the Cyrillic script.

Translation request

A translation request is represented as JSON formatted string and it can be either used for submitting a new or changed request or as return result. Attributes in *italics* are optional. Bold attributes are generated by the POST or PUT request from the server.

```
{ "translationRequest":
      {
```

"id":	"<id>",	Translation request id generated when a new request is created (POST) or used when request should be changed (PUT). The requester creates the id.
"callbackURL":	call back url	A URL that will be used by the receiver to inform the requester about the progress of the request.
"sourceLanguage":	"<source language code>",	Can be omitted if application determines language or part of the payload
"targetLanguage":	"<target language code>",	Can be omitted if application determines language or part of the payload

11 https://google-styleguide.googlecode.com/svn/trunk/jsoncstyleguide.xml#Property_Name_Guidelines

12 Empty string and null are different values. The comparison null == "" will return false

13 http://tools.ietf.org/html/bcp47

"source":	"<source text>",	Can be omitted if text is part of the payload (file, URL …). Text is interpreted as plain text.
"target":	"<target text>",	Can be omitted if text is part of the payload (file, URL …). Text is interpreted as plain text.
"mt":	<true/false>,	Machine translation yes / no
"crowd":	<true/false>,	Crowd translation required yes / no
"professional":	<true/false>,	Professional translation required yes / no
"postedit":	<true/false>,	Post editing of translation required yes / no
"comment":	"<string>",	Any comment
"translator":	"<string>",	Translator of target
"owner":	"<string>",	Owner of request
"creationDatetime":	ISO 8601 formatted string,*	Date and time of request; generated when request is created
"modificationDatetime":	ISO 8601 formatted string,	Date and time of request; generated when request is modified
"updateCounter":	<int>,	Number of changes applied to request
"status":	"<string>"	Status of request

```
        }
}
```

If `source` and/or `target` are empty source and target documents can be submitted as part of a multipart/ form-data message with name: sourcedocument and/or: targetdocument.

* http://de.wikipedia.org/wiki/ISO_8601

Comment request

```
{ "commentRequest":
        {
```

"id":	"<id>",	Comment request id generated when a new request is created (POST) or used when request should be changed (PUT).
"referenceId":	"<requestid>",	Id of the request the common references; it can be a translation, comment or scoreq request

"callbackURL":	x-callback-url formatted url	A URL that will be used by the receiver to inform the requester about the progress of the request. URLs should follow the x-callback-url specification*
"language":	"<language code>",	Language of comment
"text":	"<string>"	Comment text; format is a plain text string; no formatting information)html) should be used
"creationDatetime":	ISO 8601 formatted string	Date and time of request; generated when request is created

}
}

Score request

{ "scoreRequest":
{

"id":	"<id>",	Score request id generated when a new request is created (POST) or used when request should be changed (PUT).
"requestId":	"<translation-requestid>",	Id of the translation request for the score
"callbackURL":	x-callback-url formatted url	A URL that will be used by the receiver to inform the requester about the progress of the request. URLs should follow the x-callback-url specification*.
"score":	<int>,	Score value
"crowd":	<true/**false**>,	Crowd translation required yes/no
"professional":	<true/**false**>,	Professional translation required yes/no
"text":	"<string>",	Score comment text
"creationDatetime":	ISO 8601 formatted string	Date and time of request; generated when request is created

}
}

* http://x-callback-url.com

Callback request

```
{ "callbackRequest":
      {
```

"id":	"<id>",	Callback request id generated when a new request is created (POST) or used when request should be changed (PUT).
"referenceId":	"<requestid>",	Id of the request the comment references; it can be a translation, comment or score request
"callbackText":	"<string>"	Textual information about the changes done by the server
"callbackStatus":	"<string>"	Status information about the changes done by the server
"callBackcreationDatetime":	ISO 8601 formatted string	Date and time of callback request; generated when request is created
attributes…	<attributes of the origin request>	This is a copy of the attributes of the original request; the id of the original request must be removed and goes into the referenceId attribute. Any file or URL attachments are not copied.

```
      }
}
```

Error Response

Errors are signalled using standard HTTP codes. If more specific error messages are used an error json object should be returned.

Important HTTP Status Codes

200	Request id found
201	A new resource was created
400	Bad Request
401	Authentication Error/Access Denied
404	Request id not found
405	Method not allowed
409	Conflict in resource
415	Media Type or Output Format Not Supported
422	Semantic error in request
500	Server Failure
501	Service/Method Not Implemented

Error Response Object

The API defines some basic error attributes[14] which a method should return if an error occurs, esp. for http status code > 400.

```
{ "error":
      {
```

"id":	"<id>",	Error request id
"requestId":	"<translation-requestid>",	Id of the request that generated the error
"errorMessage":	"<string>",	Human readable error message
"httpCode":	<int>,	The http status code returned by the method
"datetime":	ISO 8601 formatted string	Date and time of error; generated when error request is created
userMessage	"<string>",	A message for the end user
developerMessage	"<string>",	A message for the developer
errorCode	<int>,	An application specific error code
…	"<string>",	More attributes generated by the server (e.g. stack traces, …). The naming scheme is up to the implementation.

```
      }
}
```

[14] https://blog.apigee.com/detail/restful_api_design_what_about_errors

TRANSLATION Request related Methods

GET

URL	translation
Method	GET
Request Body	application/json
Returns	200 Request id found 401 Authentication Error/Access Denied 404 Request id not found 415 Media Type or Output Formal Not Supported 422 Semantic error in request 500 Server Failure 501 Service/Method Not Implemented
Comment	Returns a JSON array with all translation request ids. The request body may contain attributes which filter the requests to be returned, e.g. by specifying sourceLanguage “de-de” wich would return only those request ids of sourceLanguage “de-de”. An example URL with a filter may look like this. `translation?sourceLanguage=de-de&targetLanguage=en-us`

URL	translation / {id}
Method	GET
Request Body	application/json
Returns	200 Request id found 401 Authentication Error/Access Denied 404 Request id not found 415 Media Type or Output Formal Not Supported 422 Semantic error in request 500 Server Failure 501 Service/Method Not Implemented
Comment	Returns the translation request with id {id}

URL	status / {id}
Method	GET
Request Body	application/json
Returns	200 Request id found 401 Authentication Error/Access Denied 404 Request id not found 415 Media Type or Output Formal Not Supported 422 Semantic error in request 500 Server Failure 501 Service/Method Not Implemented

Comment	Returns the status of translation request with id {id}

URL	**view / {id}**
Method	GET
Request Body	application/json
Returns	200 Request id found 401 Authentication Error/Access Denied 404 Request id not found 415 Media Type or Output Formal Not Supported 422 Semantic error in request 500 Server Failure 501 Service/Method Not Implemented
Comment	Returns a list of matching translation requests with id {id} where id is a regular expression or criteria specified in the body

POST

URL	**translation**
Method	POST

Request Body	`No files:`
	`Content Type: application/json`
	
	`With files:`
	`Content Type: multipart/form-data; boundary=aboundarystring`
	
	`--aboundarystring`
	`Content Type: application/json: name="translationRequest"`
	`translation request`
	
	`--aboundarystring`
	`Content Type: application/*;name="sourcedocument"`
	`source document`
	
	`--aboundarystring`
	`Content Type: application/*;name="targetdocument"`
	`target document`
	
	`--aboundarystring`

Returns	201 Translation request successfully created 401 Authentication Error/Access Denied 415 Media Type or Output Formal Not Supported 500 Server Failure 501 Service/Method Not Implemented
Comment	Creates the translation request with id {id}; the request body contains the translation request content. It is ok to submit or target document only.

PUT

URL	translation / {id}
Method	PUT
Request Body	No files: Content Type: application/json With files: Content Type: multipart/form-data; boundary=aboundarystring --aboundarystring Content Type: application/json: name="translationRequest" translation request --aboundarystring Content Type: application/*;name="sourcedocument" source document --aboundarystring Content Type: application/*;name="targetdocument" target document --aboundarystring
Returns*	200 Translation request successfully changed 401 Authentication Error/Access Denied 404 Request id not found 415 Media Type or Output Formal Not Supported 500 Server Failure 501 Service/Method Not Implemented
Comment	Changes the translation request with id {id}; the request body contains the translation request content with the required changes If request does not exist 404 has to be returned. It is ok just to submit just a source or a target document.

* “If a new resource is created, the origin server MUST inform the user agent via the 201 (Created) response. If an existing resource is modified, either the 200 (OK) or 204 (No Content) response codes SHOULD be sent to indicate successful completion of the request.“ - http://tools.ietf.org/html/rfc2616#section-9.6

URL	accept / {id}
Method	PUT
Request Body	application/json
Returns	200 Translation request successfully changed 401 Authentication Error/Access Denied 404 Request id not found 415 Media Type or Output Formal Not Supported 500 Server Failure 501 Service/Method Not Implemented
Comment	Accepts the translation request with id {id} and changes the state (`status`) of the request to “accepted“.

URL	**reject / {id}**
Method	PUT
Request Body	application/json
Returns	200 Translation request successfully rejected 401 Authentication Error/Access Denied 404 Request id not found 415 Media Type or Output Formal Not Supported 500 Server Failure 501 Service/Method Not Implemented
Comment	Rejects the translation request with id {id} and changes the state (`status`) of the request to rejected

URL	**confirm / {id}**
Method	PUT
Request Body	application/json
Returns	200 Translation request successfully changed 401 Authentication Error/Access Denied 404 Request id not found 415 Media Type or Output Formal Not Supported 500 Server Failure 501 Service/Method Not Implemented
Comment	Confirms the translation request with id {id} and changes the status (`status`) of the request to confirmed

URL	**cancel / {id}**
Method	PUT
Request Body	application/json
Returns	204 No content (successfully deleted) 401 Authentication Error/Access Denied 404 Request id not found 415 Media Type or Output Formal Not Supported 500 Server Failure 501 Service/Method Not Implemented
Comment	Cancel the translation request with id {id}; and changes the state (`status`) of the request to “cancelled“. NOTE: I used PUT instead of DELETE as deleting the request may be a follow up operation.

DELETE

URL	translation / {id}
Method	DELETE
Request Body	application/json
Returns	204 No content (successfully deleted) 401 Authentication Error/Access Denied 404 Request id not found 415 Media Type or Output Format not Supported 500 Server Failure 501 Service/Method Not Implemented
Comment	Deletes the translation request with id {id}; the request body contains the translation request id

COMMENT related Methods

GET

URL	comment
Method	GET
Request Body	application/json
Returns	200 Request id found 401 Authentication Error/Access denied 404 Request id not found 415 Media Type or Output Format Not Supported 422 Semantic error in request 500 Server Failure 501 Service/Method Not Implemented
Comment	Return a JSON array with all comment request ids. the request body may contain attributes which filter the requests to be returned.

URL	comment / {id}
Method	GET
Request Body	application/json
Returns	200 Comment id found 401 Authentication Error/Access Denied 404 Request id not found 415 Media Type or Output Format Not Supported 500 Server Failure 501 Service/Method Not Implemented
Comment	Returns the comment request with id {id}

POST

URL	comment
Method	POST
Request Body	application/json
Returns	201 Comment request successfully created 401 Authentication Error/Access Denied 415 Media Type or Output Format Not Supported 500 Server Failure 501 Service/Method Not Implemented
Comment	Creates a comment for request with id {id}; the request body contains the comment content.

PUT

URL	comment / {id}
Method	PUT
Request Body	application/json
Returns	200 Comment request successfully changed 401 Authentication Error/Access Denied 404 Request id not found 415 Media Type or Output Format Not Supported 500 Server Failure 501 Service/Method Not Implemented
Comment	Changes the comment with id {id}, the request body contains the translation request content with the required changes

DELETE

URL	comment / {id}
Method	DELETE
Request Body	application/json
Returns	204 No content (successfully deleted) 401 Authentication Error/Access Denied 404 Request id not found 415 Media Type or Output Format Not Supported 500 Server Failure 501 Service/Method Not Implemented
Comment	Deletes the comment with id {id}; the request body contains the request id. If request does not exist 404 has to be returned.

SCORE related Methods

GET

URL	score
Method	GET
Request Body	application/json
Returns	200 Request id found 401 Authentication Error/Access Denied 404 Request id not found 415 Media Type or Output Format Not Supported 422 Semantic error in request 500 Server Failure 501 Service/Method Not Implemented
Comment	Returns a JSON array with all score request ids. The request body may contain attributes that filter the request to be returned

URL	score / {id}
Method	GET
Request Body	application/json
Returns	200 score id found 201 401 Authentication Error/Access Denied 404 Request id not found 415 Media Type or Output Format Not Supported 500 Server Failure 501 Service/Method Not Implemented
Comment	Returns a score request with id {id}

POST

URL	score
Method	POST
Request Body	application/json
Returns	201 Comment request successfully created 401 Authentication Error/Access Denied 415 Media Type or Output Format Not Supported 500 Server Failure 501 Service/Method Not Implemented
Comment	Creates a score request for request with id {id}; the request body contains the score content

PUT

URL	score / {id}
Method	PUT
Request Body	application/json
Returns	200 score request successfully changed 401 Authentication Error/Access Denied 404 Request id not found 415 Media Type or Output Format Not Supported 500 Server Failure 501 Service/Method Not Implemented
Comment	Change the score request with id {id}; the request body contains the translation request content with the required changes; if request does not exist 404 has to be returned.

DELETE

URL	comment / {id}
Method	DELETE
Request Body	application/json
Returns	204 No content (successfully deleted) 401 Authentication Error/Access Denied 404 Request id not found 415 Media Type or Output Format Not Supported 500 Server Failure 501 Service/Method Not Implemented
Comment	Deletes the score request with id {id}; the request body contains the score request id.

CALLBACK related Methods

POST

URL	callback
Method	POST
Request Body	application/json
Returns	201 CALLBACK request successfully created 401 Authentication Error/Access Denied 415 Media Type or Output Format Not Supported 500 Server Failure 501 Service/Method Not Implemented
Comment	The request body contains the field of the original request where the id of the original request is named “callbackId” a callBackStatus attribute and callbackText is added which identifies the changes done to the request by the server. If files or URLs are part of the original request they are not part of the request body.

Querying individual translation, comment and score attributes

A server is free to implement a specific access method for each attribute of the three requests (translation, comment, score).

URL	{request-method} / {attribute} / {id}
Method	GET
Request Body	application/json
Returns	200 Request id found 401 Authentication Error/Access Denied 404 Request id not found 415 Media Type or Output Format Not Supported 500 Server Failure 501 Service/Method Not Implemented
Comment	Returns the attribute of the translation request with id {id} for {request-method} where request-method is one of the following: translation, score or comment. Attribute is an allowed attribute name. Example: `score/crowd/2b575fdc-f6af-4b9e-850d-9dc0884c6595` returns the crowd attribute value of score with id 2b575fdc-f6af-4b9e-850d-9dc0884c6595.

US Translation API translation (V1.4) request methods[15]

The following table describes the basic methods supported by API Version 1.4. This section is only contained for historical reasons. Prior version can be found at https://labs.taus.net/interoperability/taus-translation-api.

API Domain	Service	Typical Use Cases	Required
/translation/*	Get and submit translations	Almost all systems will implement this API	Yes
/im/*	Real-time message translation	IM, SMS and chat translation services	No
/comment/*	View and submit comments about translations	Crowd translation service, professional translation services that allow user feedback, comments or discussion	No
/score/*	View and submit score for translations	Crowd and professional translation services that allow user feedback and/or professional translator QA services, as well as moderated crowd translation systems	No
/request/*	View and manage task queue on the translation server	This is used to expose the request queue to client applications, for example a web app that enables translators to see pending requests, or enables a client application to monitor task progress	No

Note: im was removed in the last version! Functionality part of REQUEST. We might move the score method out from the core. **Request** is not required anymore as its functionality is tight to translation (requests)

[15] This box gives an overview from the previous methods supported, see https://labs.taus.net/interoperability/taus-translation-api

References

http://tools.ietf.org/html/rfc2616

http://tools.ietf.org/html/rfc2017

http://www.ietf.org/rfc/rfc2388.txt

https://labs.taus.net/interoperability/taus-translation-api

https://blog.apigee.com/detail/restful_api_design_what_about_errors

http://www.iana.org/assignments/access-types/access-types.xhtml