TAUS Translation API – Version 2.0

TAUS Translation API User Guide - A Common Translation Services API

May 2014

AUTHORS
Klemens Waldhör, TAUS
Achim Ruopp, TAUS

DOCUMENT VERSION:
1.0 from 17.06.2014

Table of Content

1. Introduction

The translation industry, unlike other sectors underpinned by technology, is defined by a lack of the type of open standards that have fueled innovation. In this guide and the accompanying specification, we describe a strategy for developing an open and flexible web services API (Application Programming Interface) that addresses many common use cases in translation, but is also easy and inexpensive for vendors to implement.

At a time when translation is moving to the cloud, the approach proposed in this guide has the potential to significantly advance the interoperability agenda in the translation industry.

Most of the efforts in defining standards for translation processes have focused on file formats, XLIFF for example. This makes sense when systems are not pervasively interconnected, but in today's environment where services are increasingly hosted in the cloud, file centric standards are missing an essential component - the standardization of system interactions. The TAUS Translation API provides this component.

Web APIs offer the ability to simplify the process of interacting with translation resources and services while also allowing complex tasks to be broken down into several small and simple requests. With feedback from across the industry we studied the basic set of features that should be included in a translation services API, and propose a specification for a standard API - the TAUS Translation API. The TAUS Translation API specification is a living document and we will continue to consult with the industry to expand the API to meet industry needs.

The TAUS Translation API focuses on the system interactions and allows for a broad variety of translation workload formats, including file centric standards like XLIFF. The industry has invested significantly in these file standards and the TAUS Translation API was designed in a way to carry these investments into the new world of cloud-based services.

2. Goals and Purpose

Most attempts at defining standards in the translation industry have tried to anticipate every possible use case. While this is a well-intentioned goal, the result is typically a standard or interface that requires considerable sophistication and effort to implement and test.

Popular web services, Twitter is a good example, take the opposite approach with their developer APIs. They provide a minimal set of simple interactions to cover the most common tasks. Because of this, developers can prototype new applications quickly, and can learn this type of API within a short period of time. The goal of the TAUS Translation API is to define something equally simple for translation.

This API is designed for software developers who have little or no knowledge about translation services, which is to say almost all software developers. Within the industry, we tend to have debates over technology choices that are meaningless to everyone else. So what do software developers need most? A simple way to request and get translations from a service in the “cloud”, in effect a command that means “translate this into French, call me back at www.foo.com/submit when you’re done. Thanks.”

The most basic API would simply define a standard way to request a translation and get the results back. With maybe a dozen other interactions, we can describe most of the things that software developers will need to do in building a multilingual website or application. Most applications will probably only use the most basic interaction, the method used to request a translation.

The design goal for the specification is simple. As a service provider, you should be able to do a partial implementation in a few hours. As a software developer, you should be able to build a “Hello World” application equally quickly, and master the overall API within a day or so without assistance.

To further facilitate API implementation an added design goal for the latest version 2.0 of the specification is to comply to RESTful web API design. This design principle is used in virtually all web APIs today and developers are very familiar with it.

3. Hello World - Requesting a Translation

Let’s start by describing a “hello world” example. In this case, we want to request a professional translation from a language service provider. We can do all of this using the same mechanism to post or put the resources of a form on a web page to the server.

In the following we assume that a translation server is available at `https://www.abctranslations.com` and that the server supports the TAUS API, e.g. using a node.js implementation. The server is run by “abctranslations”, an LSP.

The client requesting the translation creates a POST translation request. The translation POST request is responsible for creating an initial translation request and contains the parameters of the request in its HTTP body. Optionally it can contain files too.

The translation **POST** requests could look like this: `https://www.abctranslations.com/translation`

with the following JSON body:

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

`"id":`	`"2b575fdc-f6af-4b9e-850d-9dc0884c6595",`
`"sourceLanguage":`	`"DE-DE",`
`"targetLanguage":`	`"EN-EN",`
`"source":`	`"Hallo Welt",`
`"professional":`	`true,`
`"mt":`	`false,`
`"creationDatetime":`	`"2014-05-20T19:20+01:00"`
`"updateCounter":`	`0`
`"status":`	`"initial"`

```
   }
}
```

This resource is somewhere stored on the server in a database or similar data store. The resource created is uniquely identified by its id `2b575fdc-f6af-4b9e-850d-9dc0884c6595`. It can be retrieved by running a **GET** request like this: https://www.abctranslations.com/translation/2b575fdc-f6af-4b9e-850d-9dc0884c6595

After having the text translated the abctranstranslations translation server will reply with a **PUT** message

https://www.abctranslations.com/translation/2b575fdc-f6af-4b9e-850d-9dc0884c6595

modifying the request resource by adding the translation, updating the modification field and the update counter.

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

"target":	"Hello world",
"modificationDatetime":	"2014-05-21T19:12+02:00",
"status":	"translated",
"updateCounter":	1

```
   }
}
```

The client can now retrieve the translated resource with the GET request `https://www.abctranslations.com/translation/2b575fdc-f6af-4b9e-850d-9dc0884c6595`.

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

"id":	"2b575fdc-f6af-4b9e-850d-9dc0884c6595",
"sourceLanguage":	"DE-DE",
"targetLanguage":	"EN-EN",
"source":	"Hallo Welt",
"professional":	true,
"mt":	false,
"creationDatetime":	"2014-05-20T19:20+01:00"
"modificationDatetime":	"2014-05-21T19:12+02:00",
"updateCounter":	1
"status":	"translated"

```
      }
}
```

This approach makes it trivial for a developer to build a script to generate and process requests, and therefore greatly reduces the time and cost to implement this interface (especially compared to XML centric approaches that require a lot of parsing on both ends of the connection), whereas this approach can be implemented using the standard HTTP services and standard libraries bundled with common web programming languages (e.g. node.js, PHP, Ruby on Rails, etc).

The key advantage of this API oriented approach is that it says nothing about how a request is processed or how data is stored within the server on either side of the connection, but only defines the forms based communication between client and server. This frees system developers to use whatever languages and tools are best suited to the problem they are solving, without forcing arbitrary standards (e.g. internal file formats) upon developers.

4. API History from Version 1.x to 2.0

The development of the TAUS API started in 2011 with the Barcelona Executive forum and resulted in the creation of a TAUS Lab researching and creating a first standard for the TAUS API. The first TAUS API was based on the ideas of Brian McConnell and published in a first paper. The paper described the first ideas, especially that the API should be based on a REST approach. These ideas where taken forward by TAUS and a first version presented at the XLIFF symposium in September 2011 by Klemens Waldhör. A first implementation was done which allowed potential interested parties checking their implementation for conformance with the API. Example implementations of the TAUS API were done by Lingo 24 and Nova. In 2011 and 2012 several industry surveys have been undertaken to understand the industry needs.

Several key players analyzed the API (esp. Jörg Schütz and Alan Melby) and returned very valuable comments for the improvement. Based on these comments the API version 1.2 was totally redesigned. The new version 2.0 is now fully REST oriented, esp. it follows the CRUD principles for restful APIs (create, read, update, delete). The new API also distinguishes between an API core and additional modules which add additional methods to the API (e.g. bidding).

End of May 2014 version 2.0 was published and opened for a comment phase. Version 2.0 will be presented at the TAUS Industry Leader Forum und LocWorld 2014 in Dublin. This will be followed by an extensive industry evaluation phase.

An example server version based on node.js will be published as well in the next months.

The following table sums up the main difference between versions 1.2 to version 2.0.

API Method v 1.4	API Method v 2.0	Service	Changes
/translation/ get submit	/translation/{id} /translation/*filter-query-string* GET POST PUT DELETE Further methods based on PUT verb /translation/accept /translation/reject /translation/cancel /translation/confirm	Get and submit translations	Methods adapted to CRUD approach. Parameter names unified. Support of submitting files and other resources. Callback parameter improved.
/im/*	**removed**	Real-time message translation	All methods removed due to overlap with translation
/comment/ submit get	/comment/{id} /comment/*filter-query-string* GET POST PUT DELETE	View and submit comments about translations	Methods adapted to CRUD approach.
/score/ submit get	/score/{id} / score/*filter-query-string* GET POST PUT DELETE	View and submit score for translations	Methods adapted to CRUD approach
/request/ status retrieve accept reject cancel confirm view	**removed**	View and manage task queue on the translation server	All methods associated with request methods v1.4 are obsolete and replaced by GET methods in v2.0
	callback	Run a callback method for a request	New methods which supports callbacks

5. API Design

5.1 Functional Coverage

The first step in designing a standard web API for translation services is to identify the most commonly encountered tasks and use cases. Rather than trying to address every possible situation, we want to cover 90-99% of real world use cases. This greatly simplifies the scope of work, while enabling service providers who need additional functionality to implement custom API calls to extend the basic functionality defined in this spec. We'll assume that if a customer has a special need, they will develop a custom interface or extend the API to address this.

Among the uses we will commonly encounter when interacting with a language service provider are:

Typical Translation Tasks
Request a translation for a text from an LSP or translator
Forward a request from an LSP to a translator
Forward a request to an automated service (e.g. MT)
Get a translation back from a translator
Submit a translation back to the requester
Request for review or score for a translation
Check the status of a request
Cancel a request
Accept or reject a completed translation/task
Request a list of pending requests/tasks

Table 3: Typical service fields for language service provider

This is not intended to be a complete list, but a representative list of regular tasks. We will develop this list further based on feedback from stakeholders. Different stakeholders have different needs. In order to support these needs the API is divided into different layers. The core layer (layer 0) defines the methods which most of the stakeholders need and represent a common understanding of all stakeholders.

5.2 RESTful APIs

Restful APIs have become a quasi-standard in last years within the software industry as they are easy to implement a based on the HTTP protocol. The TAUS API is based on the CRUD principles and HTTP verbs. CRUD defines a set of operations which operate on resources and allow creating, modifying, querying and deleting resources. Within the localization industry typical resources are translation projects with their associated artifacts like translation files (for example XLIFF based), payloads like TIPP packages, language related information, required translation quality etc. Requests are the key methods APIs operate on. Given the HTTP protocol CRUD operations are mapped towards HTTP verbs as the table below explains:

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

Based on this a set of methods and request have been defined for the localization industry. The technical details of the API and its parameters can be found in the TAUS Technical Specification - A Common Translation Services API (2014). The following table describes the basic requests supported by API Version 2.0.

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 translations.	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 In addition the following subverbs are supported which modify an existing request: /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	

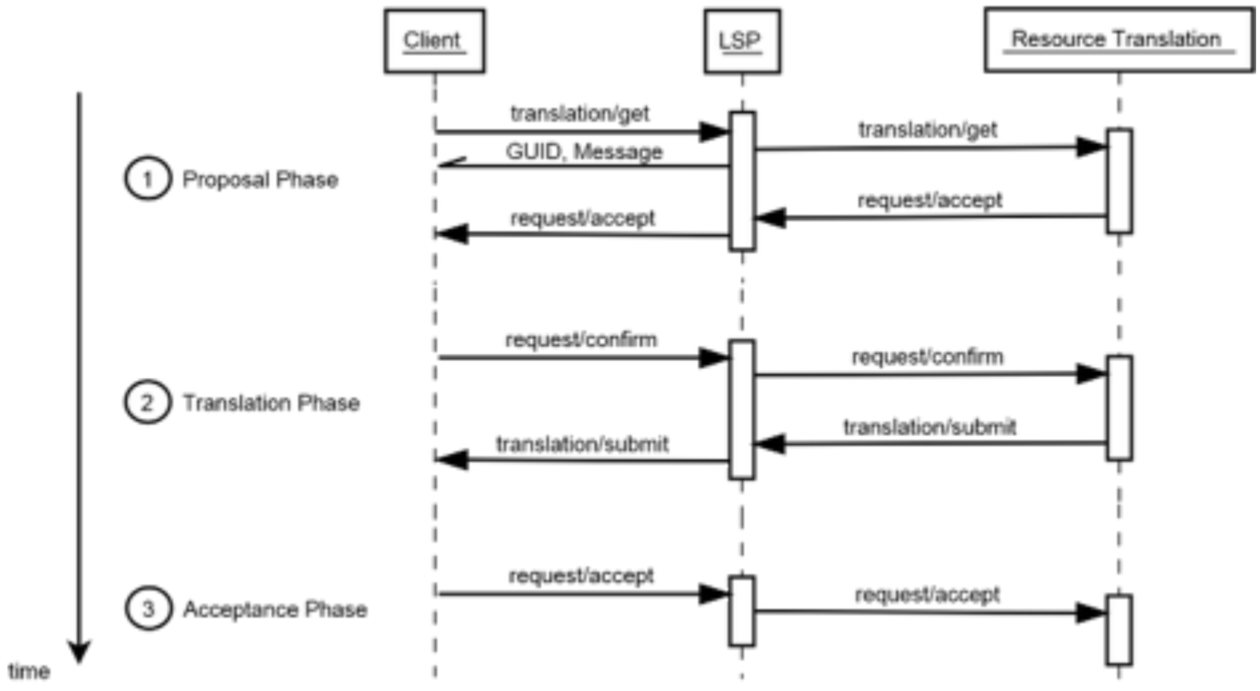
Request Method	Function / HTTP Verbs	Typical Use Cases	Required
	POST	Create a new comment request	
	PUT	Change a comment request Esp. the following methods are supported: accept, reject, confirm, cancel	
	DELETE	Remove a comment request	
score	View and submit scores for translations	Crowd and professional translation services that allow user feedback and/ or professional translator QA services, as well as moderated crowd translation systems.	No
	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 call back URL	
All methods	Get request attribute values	This allows retrieving specific attributes and their values	
{request-method}/ {attribute}/{id}	GET	Return the value of a specifies attribute for a request method above.	No
	/translation/accept /translation/reject /translation/cancel /translation/confirm		

5.3 Process Examples

The following examples show typical use cases for the TAUS Translation API.

The first example shows a typical scenario where client requires a translation for a document. The client creates a request and sends the request (POST) to the LSP server. The LSP sends the request to one of his translators, possible doing some changes to the parameters of the request, e.g. adding a comment. The translator might accept the requests and answers with a accept request (PUT). Once the LSP has received the acceptance of the job from the translator he sends also an acceptance message to the client (PUT). The client confirms (PUT) the job, the LSP confirms the job for the translator (PUT) and the translator creates the translation and returns it (PUT). The translation is forwarded to the client by the LSP (PUT). The client responds with the acceptance of the translation (PUT**),** the LSP responds now with the acceptance of the translation to the translator (PUT).

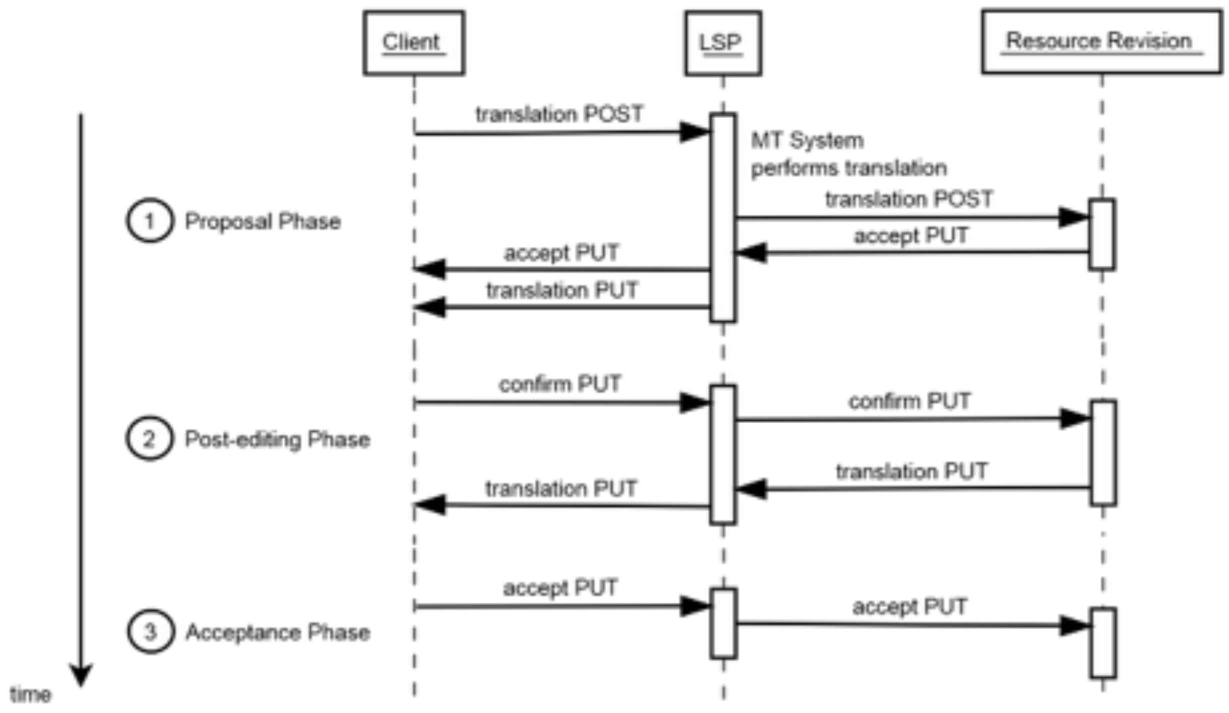
Human Translation Scenario: client requests translation through LSP, translation is performed by human translator and accepted by client



Please note that for readability reasons the IDs are not shown in the messages. It is also up to the LSP if he uses the original request sent in by the client or if he creates a new request for the translator. The example assumes that only one request is created and changed during the translation process.

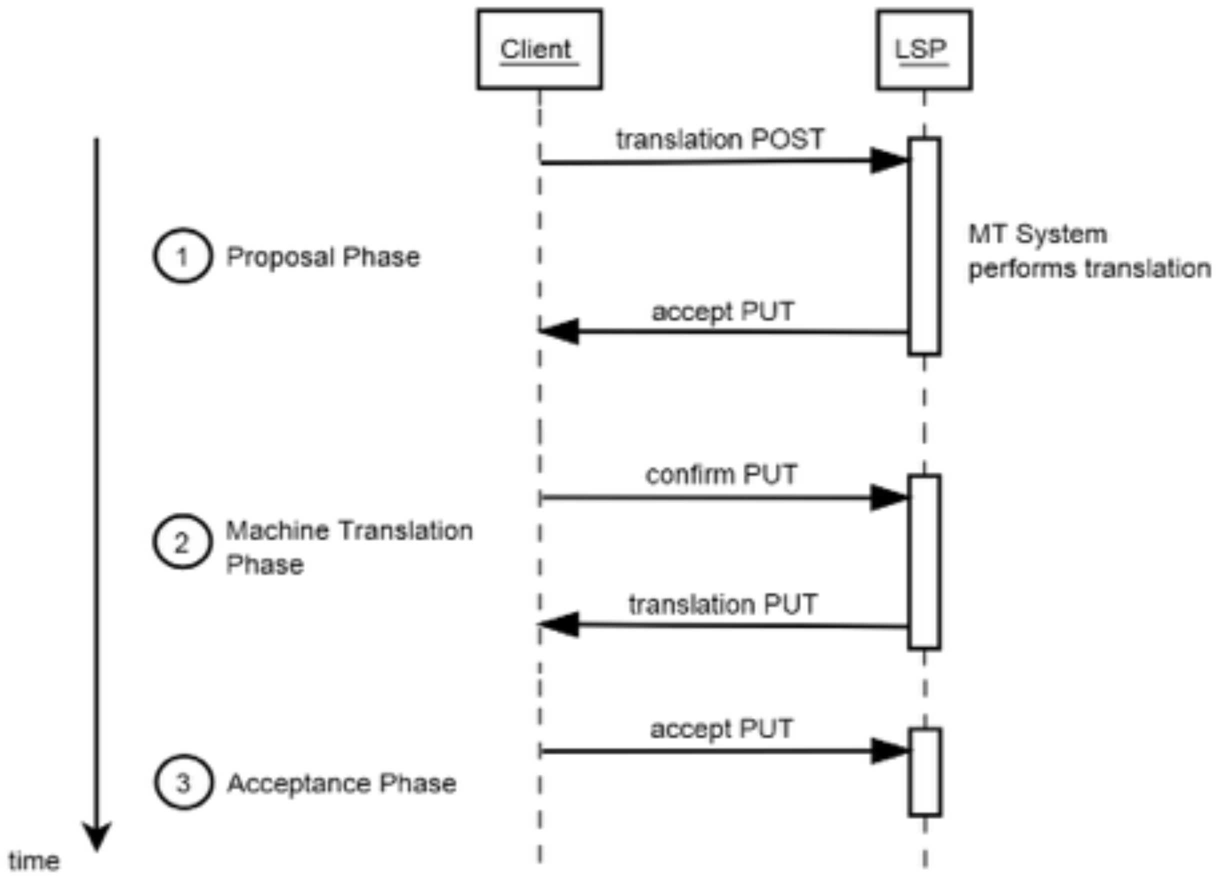
The following example shows the process of pre-translating with an MT system with subsequent editing by a human post-editor.

Machine Translation plus Post-editing Scenario: client requests machine translation with post-edit through LSP, machine translation is delivered to client, post-edit is performed by human editor and accepted by client



The following example shows a possible scenario for am MT provider. The translation process can be fully automated as the client just interacts with the MT system.

Machine Translation Scenario: client requests raw machine translation from LSP, machine translation is delivered to client and accepted by client



6. Authentication/Authorization

Like many other HTTP REST APIs the TAUS Translation API is designed to make use of web standard technologies for authentication and authorization. For authentication HTTP Basic or Digest authentication can be used, as can technologies like OpenID Connect and usual user name/password authentication. For authorized access to resources it is recommended to use the OAuth v2.0 authentication, a standard that is used by web APIs across the industry.

7. Call To Action

This API is designed to be very easy to implement. As a service provider, you should be able to implement the most basic functions, the /translation methods, in a few hours, especially if you already have a web API of some sort. We are calling on you to provide feedback on this discussion. We are also planning to release an open source test implementation.

7.1 Translation Companies

All you need to do for a partial implementation is to build a request handler for the /translation methods. Everything else in the API may be considered an extra. Most software developers will only use this one API call. So you can get started and try this out without committing much effort.

7.2 SDKs

If you have a favorite programming language, consider building a wrapper library for the API and sharing it with the developer community.