

ONELINK OTX:

REAL-TIME TRANSLATION SERVICE

Version 2.0



Table of Contents

About this guide	3
Conventions used in this guide.....	3
Typographical	3
Icons	3
About OneLink® OTX	4
Getting Started	5
1. Problem	5
2. Solution	5
Choosing Your Hosting Model.....	6
OneLink® Platform using OTX directly	7
OTX Implementation	8
POST Request.....	8
POST Parameters	8
OTX Responses	9
Appendix: Code Samples	10
Appendix: OTX Testing	13
Conclusion.....	14

This documentation is proprietary and is protected by U.S. and international copyright laws and trade secret laws. Copyright © 2014 Translations.com, Inc. ("Translations.com") All rights reserved. No part of this documentation may be reproduced, copied, adapted, modified, distributed, transferred, translated, disclosed, displayed or otherwise used by anyone in any form or by any means without the express written authorization of Translations.com. Other names may be trademarks of their respective owners.

About this guide

This document illustrates a feature of the OneLink® Platform called OTX. It enables real-time querying of Translation Memory stores to feed other requirements — such as email content, browsers, and other third-party products and data stores that may require translation.

Conventions used in this guide




Typographical

This explains the typographical conventions used in this guide:

Convention	Application
Bold	Commands and keywords in body text.
<i>Italic</i>	Command input entered or selected by the user
< >	Command parameters that must be replaced by module-specific codes.

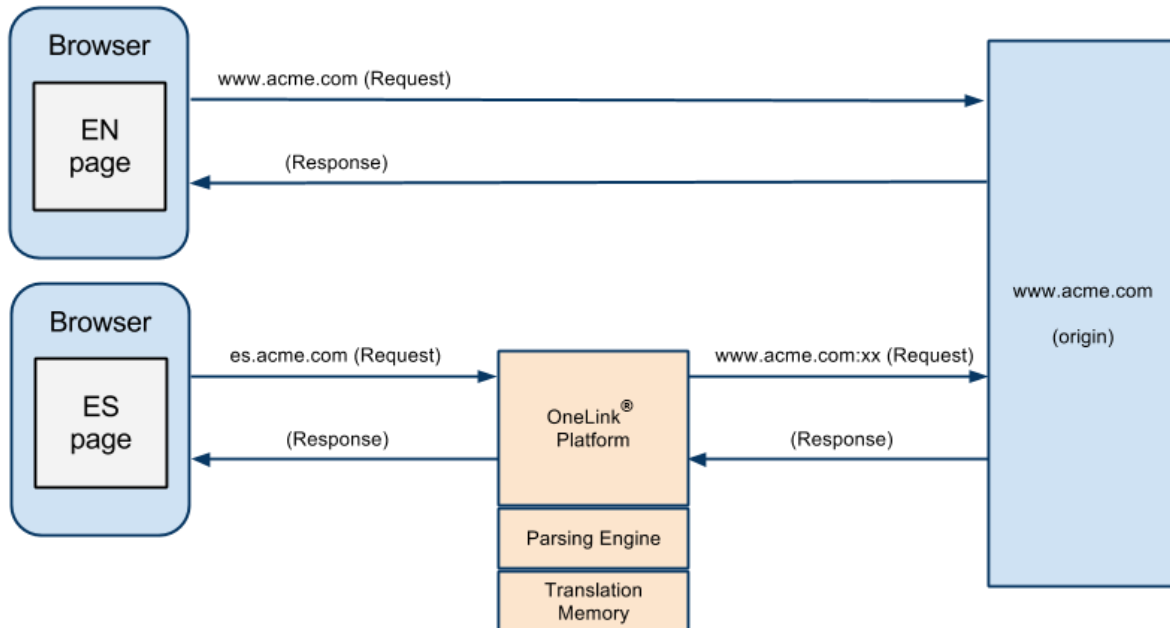
Icons

The explains the iconic conventions used in this guide:

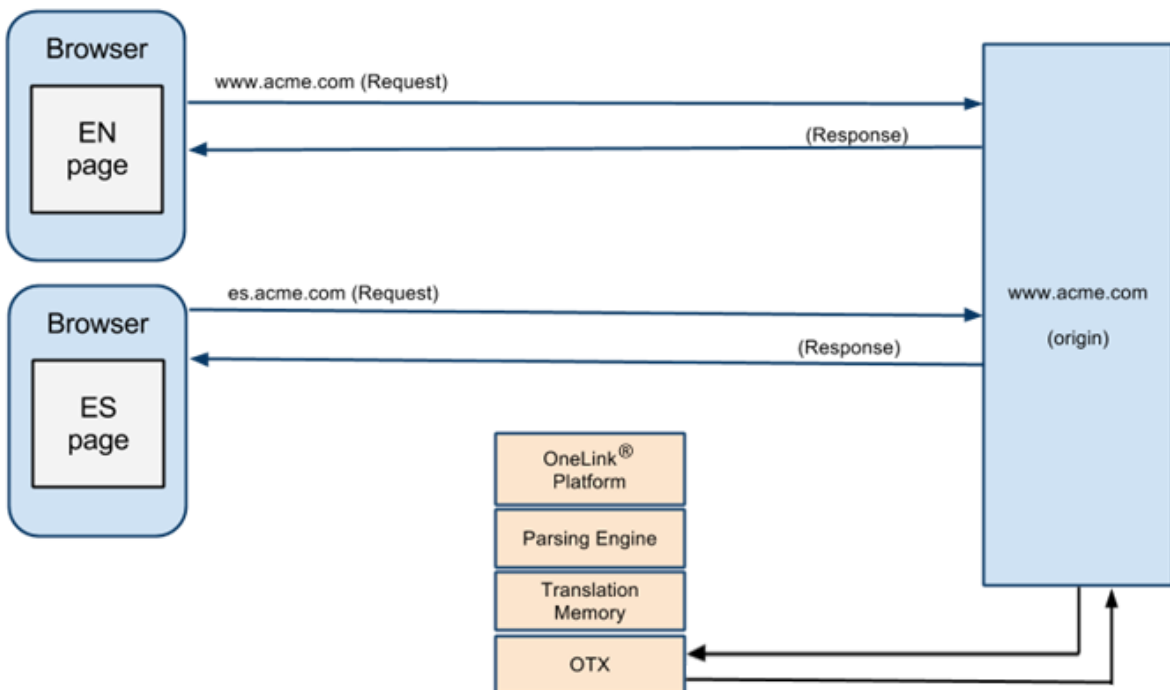
Convention	Application
	Note: This icon designates a note or helpful suggestion or reference relating to the surrounding text.
	Best Practice: This icon designates a suggestion for best practice relating to the surrounding text.
	Alert: This icon designates warning or alert relating to the surrounding text. In this situation, the user might do something that could have a negative result.

About OneLink® OTX

The GlobalLink OneLink® Platform can be configured as a proxy service to provide a translated version of a website, or invoked programmatically as a web service to provide translations into a number of receiving applications, such as email or both. This paper positions how OTX, the web service, can be used to return translations into documents, email, and other applications. Most clients of the GlobalLink OneLink® Platform use it to serve a proxy version of their website, as shown here:



Or the GlobalLink OneLink® Platform can also be configured as a web service like so:



1. Problem

Translation needs are very diverse within most companies. Requirements may come from a variety of sources and go to various target applications. Typically clients' website content and external applications like email are considered separately, yet both require translated content to function in a global environment. Website content once translated seems to span a company's entire ERP cycle from marketing and sales through to training and support.

2. Solution

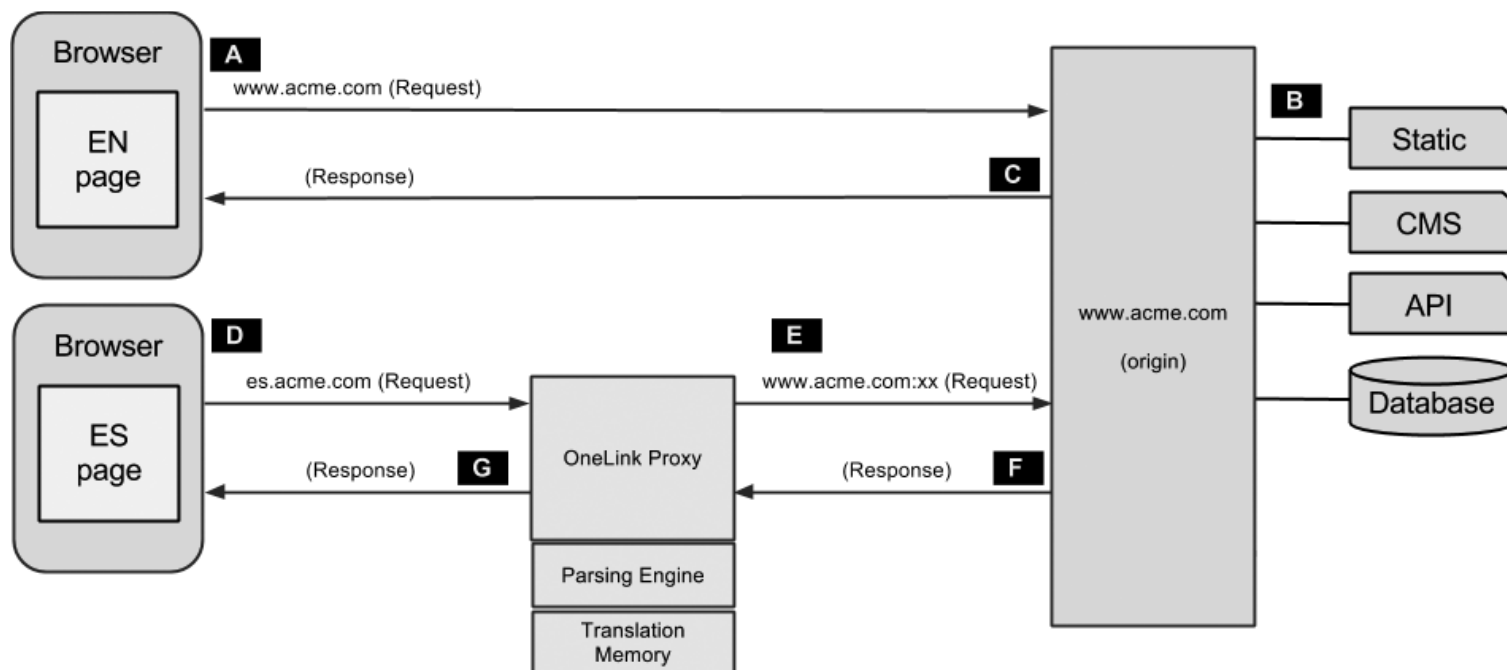
In response to these multiple requirements, Translations.com offers a variety of technologies. Translations.com's OneLink[®] offers both proxy-based website translation and an API for general purpose real-time translation. Translations can be sourced through OTX and delivered either to a website or to any number of external applications, as long as that content has been captured, translated, and resides in Translation Memory. OneLink[®] OTX is a function of the OneLink[®] Platform which accepts programmatic requests for translation through a web service. The OneLink[®] Platform is required to implement OTX, and there are various flexible ways to set up that platform to fulfill our clients' needs, requirements, and goals.

OneLink[®] OTX leverages the following components of the OneLink[®] Platform:

- Real-time parsing of web related mime-types (HTML, XML, JS, JSON, etc.),
- Real-time translation memory (TM) lookups for segment/sentence-level replacements from source to target language,
- Ability to mark-up source content with OneLink[®] classes to prevent certain content from being translated,
- Ability to capture segments that are missing from the TM and/or backfill with real-time machine translation, and
- Sub-second response time.

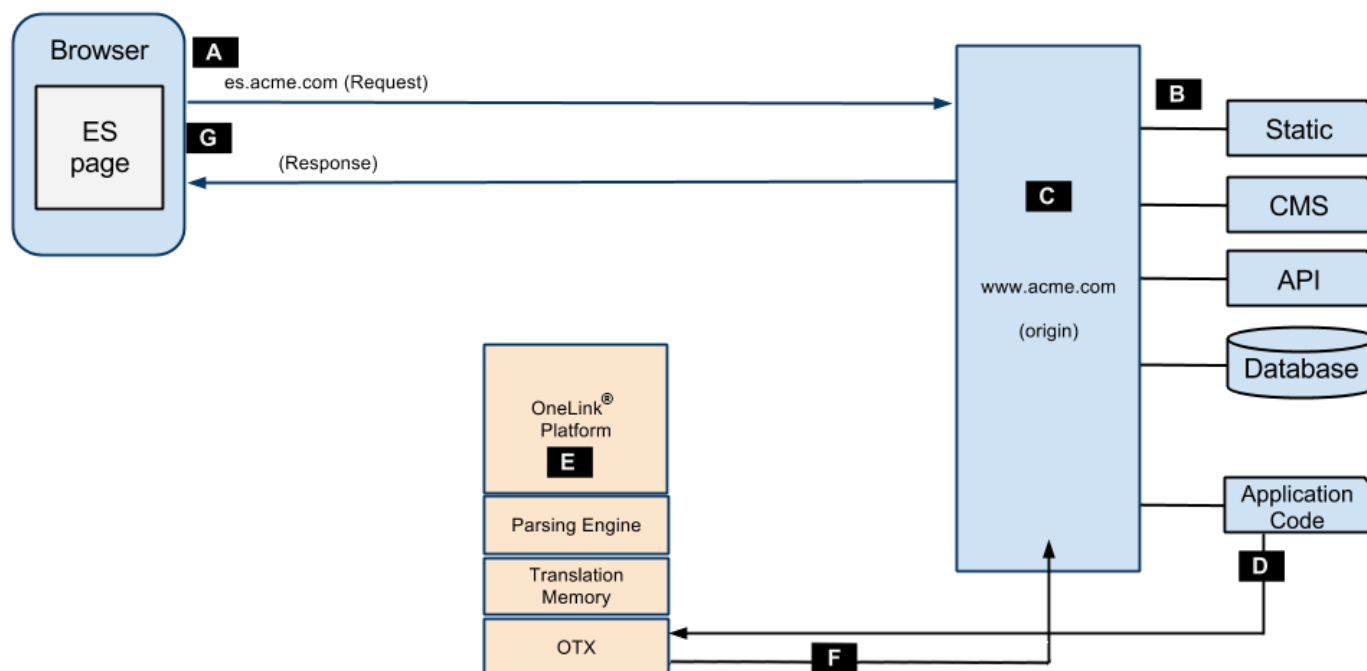
Choosing Your Hosting Model

The standard implementation of the OneLink® Platform looks like this:



Represented first is an English page request **A** that goes directly to the server, is composed **B** and returns to the client browser **C**. Then a Spanish page request **D** is forwarded onto the host in the same fashion as the English page **E**, and is composed by the server as a standard English response **F**, but is intercepted on its return to the browser by OneLink®, parsed and translated based on localization rules and subsequently forwarded onto the Spanish browser **G**. In this model the OneLink® Platform functions solely as a proxy server, standing in between an origin server and a client's browser. In doing so it reflects the origin server pages into as many languages as have been prepared for the client browser, much the same way as a movie projector projects content onto a screen. If a blue filter is applied at the projector, a blue image appears on the screen. The projector and filter do not fundamentally change the image, only its appearance. Similarly, the translated page is identical to the source except it now appears translated. This model typically is hosted and maintained by OneLink® in our colocation facilities as a service to our clients.

OneLink® Platform using OTX directly



In this model, a Spanish page request is made by the browser **A**, the origin host responds by composing an English page **B**, built using normal processes (i.e., CMS, API, Static Page, etc.). The composed page is sent to OTX application code **C**. The OTX code makes the appropriate requests from Translation Memory and the Parsing Engine to swap Spanish for English content **D**. The OneLink® Platform composes the Spanish page **E**, and returns it to the origin server **F**. The origin server renders the Spanish page in the browser **G**.

This model achieves anonymity from the client-server relationship since it is not connected to the exchange between browser and server. The server itself appears in every way as the source of the translated site. When content is requested from the server it is served directly to the client's browser with no interim contact with a OneLink® proxy. Few circumstances mandate implementation of this sort of architecture, but it is available to clients who wish to isolate OneLink® completely from that dialog. Those who implement this alternative are required to host the OneLink® Platform directly inside their IT architecture, which entails a deeper involvement of internal IT resources than may be desirable for many clients.

OTX Implementation

POST Request

Your application code sends POST requests to the OneLink® OTX using parameters that are specified on the next page. Note that you will need to understand where the OTX server lives. If the OTX server is installed on your LAN the request may look something like this:

`https://10.20.30.40/OneLinkOTX/page/requesting/data.html`

If the OTX server is installed in one of our colocation facilities, the request may look like this:

`https://es.acme.com/OneLinkOTX/page/requesting/data.html`

Using your application code, you should create a POST request with the following request headers and post parameters:

Required Request Headers:

Host: *your-virtual-host* (see below)

Content-Type: application/x-www-form-urlencoded (all POST requests have this header)

Required Post Parameters:

otx_account=account number, account password

otx_mimetype=(see below)

otx_service=(see below)

otx_content=(see below)

POST Parameters

otx_account: This contains the account number and password for OTX.

otx_service: *Optional. Defaults to "tx".* Values are as follows:

tx: Performs normal translation as defined for the virtual host.

smt: Performs SMT (simple machine translation).

wmt: Performs WebMT (WorldLingo machine translation).

tx+smt: Performs normal translation, but any segments not found in the TM are translated using simple machine translation

tx+wmt: Performs normal translation, but any segments not found in the TM are translated using WorldLingo machine translation

parse: Parses and outputs segmentation and token logic (used primarily for debugging)

otx_mimetype: The mime-type of the content being uploaded. This tells the OneLink® Proxy how to interpret the incoming information. The only supported types as of right now are:

text/html: This page will be parsed as a normal HTML page by the rules defined for the virtual host.

text/xml: This data will be parsed normally as XML content following the logic defined for the virtual host.

text/javascript: Incoming data will have our custom JavaScript parser applied.

text/json or **text/plain:** The JSON data will be filtered through our custom JSON parser logic.

text/segment or **application/json**: This refers to a single segment of text that will not be parsed with rules, it will only be translated.

Note: any foreign mimetype not listed above will cause OTX to respond with an HTTP-205 and the original document untouched. The same thing will happen if the mimetype is not set.

otx_content: the content to be translated. Note that in the event that the content contains non-ascii characters, the base encoding must be UTF-8 (not Windows-1252, nor ISO-8859). In order to be passed as POST data, the content must also be “form-url-encoding.” Most application frameworks will automatically form encode data as long as they know you are sending a POST request.

OTX Responses

The response is a standard HTTP response and the status code is any one of the following:

HTTP-200: Success

HTTP-205: The mime type not supported, original document is returned untouched

HTTP-401: Protocol error: missing or malformed required header

HTTP-402: Feature is not enabled for specified host

HTTP-403: Invalid account number or invalid account password

HTTP-404: Translation Memory (TM) is not started or service is not available

HTTP-405: Attempted to use HTTP instead of HTTPS from a public IP address

Following an HTTP-200 response, the rest of the response header will look like the following:

Content-Type: The mime type of the response (usually the same as the mime type that came in)

Content-Length: Number of bytes in the translated document

Encoding: gzip, etc. (based on site configuration rules)

The following response headers are included whenever the request includes **X-OneLink-Headers** and “translation” is in the comma-delimited list of flags:

X-OneLinkSegments: *nn* (the number of translatable text segments in the content)

X-OneLinkTranslated: *nn* (the number of text segments translated)

X-OneLinkTxPercent: *nn* (percentage of text segments translated)

Sent to OTX:

```
Host: es.acme.com
```

```
Content-Type: application/x-www-form-urlencoded
```

```
Content-Length: 123
```

```
otx_account=ACME123,secret456&otx_mimetype=text/plain&otx_service=tx&otx_content=Hello%20world
```

Received from OTX:

```
HTTP/1.1 200 OK
```

```
Content-Type: text/plain
```

```
Content-Length: 12
```

```
Encoding: utf8
```

```
Hola Mundial
```

Appendix: Code Samples

Your programming environment may insulate you from what the request actually looks like but if you used a protocol analyzer like Wireshark (www.wireshark.org), it would look something like the following exchange.

Curl example:

```
% curl -k --header "host: es.acme.com" --request post
'https://es.acme.com/OneLinkOTX/wfonline/' --data
"otx_mimetype=text/html&otx_account=acme&otx_service=smt&otx_content=<html>
<head></head><body>Hello world</body></html>"
```

Python example:

```
#!/usr/bin/env python

# -*- coding: utf-8 -*-
# Import extensible library for opening URLs
import urllib2, urllib

# Segment for translation
contentForTx = "Acme Corp has the largest selection of dynamite and anvils on the market."

# The hostname of the project that contains the TM and OTX account
virtualHostName = "es.acme.com"

# The hostname of the server the project is contained on
physicalHostName = "stagex1.onelink-translations.com"

# Set the parameters of the request, encode with url encoding.
params = urllib.urlencode({'otx_mimetype': 'text/html',
                           'otx_account' : 'acme,acmePassw0rd',
                           'otx_service' : 'tx',
                           'otx_content' : contentForTx})

# Set the host header
headers = {"Host": virtualHostName}

# Make the request object passing in the parameters and headers
req = urllib2.Request("https:"+physicalHostName+"/OneLinkOTX/", params, headers)

# Execute the request
response = urllib2.urlopen(req)

# Read the response
htmlData = response.read()

print ":: Content for translation      : "+contentForTx
print ":: Received translated content: "+htmlData

$ ./otx-proof-of-concept.py
:: Content for translation      : Acme Corp has the largest selection of dynamite and anvils on
the market.
:: Received translated content: Acme Corp tiene la mayor selección de dinamita y yunques en el
mercado.
```

.NET Example:

```

using System;
using System.Net;

namespace OtxConsoleTestApp
{
    class Program
    {
        static void Main(string[] args)
        {
            // temporary solution that can be used to ignore invalid certs on staging
            sites: System.Net.ServicePointManager.ServerCertificateValidationCallback =
                ((sender, certificate, chain, sslPolicyErrors) => true);
            // Segment for translation
            string sContentForTx = "Acme Corp has the largest selection of dynamite
and anvils on the market.";

            // The hostname of the project that contains the TM and OTX account
            string sVirtualHostName = "es.acme.com";

            // The hostname of the server the project is contained on
            string sPhysicalHostName = "stagex1.onelink-translations.com";

            // The username and password for the OTX account
            string sAccount = "acme, acmePassWOrd";

            // The OTX service type (use 'tx' for actual translations, use 'smt' for
            simple machine translation for testing purposes)
            string sService = "tx";

            // The mime type of the content being sent to OTX for translation
            string sMimeType = "text/html";

            // ensure that post data is properly encoded:
            string sPostParms =
                "otx_account=" + Uri.EscapeUriString(sAccount)
                + "&otx_service=" + sService
                + "&otx_mimetype=" + sMimeType
                + "&otx_content=" + Uri.EscapeUriString(sContentForTx);

            string sTranslation;
            using (var wc = new WebClient{Encoding = Encoding.UTF8})
            {
                // set request headers:
                wc.Headers.Set("Host", sVirtualHostName);
                wc.Headers.Set("Content-Type", "application/x-www-form-urlencoded");

                try
                {
                    // attempt translation:
                    sTranslation = wc.UploadString("https://" + sPhysicalHostName +
"/OneLinkOTX", sPostParms);
                }
                catch (WebException e)
                {
                    // we got an error/non-200 response (see OTX spec for interpreting
the error codes):
                    sTranslation = e.Message;
                }
            }

            // print the translation (or the non-200 error response):
            Console.WriteLine(":: Content for translation : " + sContentForTx);
            Console.WriteLine(":: Received translated content: " + sTranslation);
            Console.ReadLine();
        }
    }
}

```

PHP example:

```
<?php
if($argc<2){
    echo "\n usage: getOTX.php content service\n";
    exit;
}

// Setup Defaults
$otx_account = 'otx,otxpass';
$otx_mimetype = "text/html";

// Fill from cli
$otx_content = $argv[1];
$otx_service = ( isset($argv[2])) ? $argv[2] : "smt";

// Setup data for http query
$url = 'https://es-otx.onelink-translations.com/OneLinkOTX/';
$data = array(
    'otx_account' => $otx_account
    , 'otx_service' => $otx_service
    , 'otx_mimetype' => $otx_mimetype
    , 'otx_content' => $otx_content
);

// Setup stream options
$options = array(
    'http' => array(
        'header' => "Content-type: application/x-www-form-urlencoded\r\n",
        'method' => 'POST',
        'content' => http_build_query($data),
    ),
);

// Create stream , get contents
$context = stream_context_create($options);
$result = file_get_contents($url, false, $context);

// Results
echo "\n content:" . $otx_content;
echo "\n translated:" . $result . "\n";

?>
```

Appendix: OTX Testing

We have an OTX service you can test with Spanish. It uses Machine Translation so be cautioned that the translation is not reliable and is used only for testing purposes.

The domain is: <http://es-otx.onelink-translations.com>

The username is: otx
and password is: otxpass

Rather than make separate calls for each segment you want to translate you can send the server JSON or XML.

A simple Curl test to the OTX Service:

```
curl -k --header 'Host:es-otx.onelink-translations.com' --request POST 'https://es-otx.onelink-translations.com/OneLinkOTX/' --data 'otx_mimetype=text/html&otx_account=otx,otxpass&otx_service=tx&otx_content=<p>I am going for a long walk</p>' ; echo
```

This is expected to return the translated segment:

```
<p>Yo soy va para un largo caminar</p>
```

You can also send several segments to the service for translation by using JSON or HTML

In this case the server must be configured to translate specific data. This is done because most servers use JSON and XML to transmit not only translatable segments but code and tag information.

For testing we have added a global 'otxtest' tag to the OneLink translation stack. Anytime it finds this in the tag stack it will translate the content inside it.

For example, this passes JSON to the server with 3 elements, because these are contained within an otxtest element it will translate:



```
curl -k --header 'Host:es-otx.onelink-translations.com' --request POST 'https://es-otx.onelink-translations.com/OneLinkOTX/' --data 'otx_mimetype=text/json&otx_account=otx,otxpass&otx_service=tx&otx_content={ otxtest: { "data1": "i see the cat", "data2": "chasing the dog", "data3": "in the yard" } }' ; echo
```

This is expected to return:

```
{ "otxtest": { "data1": "yo ver la gato", "data2": "persiguiendo la perro", "data3": "en la patio" } }
```

The same is true for XML (Note modified mime type)

```
curl -k --header 'Host:es-otx.onelink-translations.com' --request POST 'https://es-otx.onelink-translations.com/OneLinkOTX/' --data 'otx_mimetype=text/xml&otx_account=otx,otxpass&otx_service=tx&otx_content= <otxttest><foo>You have to learn the rules of the game.</foo><bar>And then you have to play better than anyone else.</bar></otxttest>' ; echo
```

It is expected to return:

```
<otxttest><foo>Usted tener a aprender la normas de la juego.</foo>  
<bar>Y entonces usted tener a jugar mejor de nadie más.</bar>  
</otxttest>
```

Conclusion

Though implementation of some of these concepts may prove challenging, OneLink[®] engineers are available to assist our clients with requirements they may have for specific applications. Please contact your Implementation Manager to discuss OTX implementation questions or requirements you may have.