Real Time API

Brian S McConnell <brian@speaklike.com>
Updated April 20th, 2011

The SpeakLike real-time translation API enables you to translate communication sessions, such as instant messaging sessions, customer support chats and many other services. We provide three modes of interaction for real-time application development:

1. REST with asynchronous callback
2. XMPP with JSON messages
3. Asynchronous sockets for Javascript applications

Each interface is optimized for different types of systems and tasks.

If you are building a server-to-server integration, where translations are requested and processed by your server before messages are sent to clients, you should use the REST with asynchronous callback interface.

If your system is built around the XMPP messaging protocol, you can simply exchange JSON messages with speaklikeim@appspot.com

If you are building a client application in Javascript, for example to translate messages in a web chat application, you can use our asynchronous messaging API for Javascript. This pushes notifications to your app, so you do not need to poll for completed translations.

The API is currently in testing at speaklikeim.appspot.com and will also be ported to im.speaklike.com once testing is completed.

Table of Contents

REST API

The REST API exposes just three commands which are used to start and end sessions, and to request translations.

The API calls are:

- /realtime/start : start a session
- /realtime/stop : end a session
- /realtime/translate : request a translation
- /realtime/queue : fetch a list of recent translations (polling API)

The typical application flow is as follows:

1. Client application calls /realtime/start to initiate a session, and receives a session ID or error code in response
2. Whenever a translation is needed, calls /realtime/translate with the session id, text to be translated, meta data, and the callback url to send events back to
3. Client application calls /realtime/stop to end the session
4. SpeakLike posts asynchronous messages and completed translations to the callback URL you provided when you called /realtime/start , so you don't need to continually poll for translations (although you can poll for recent translations at /realtime/queue if you can't receive callbacks)

/realtime/start

This API handler expects a short list of parameters, which are:

- username : SpeakLike username
- pw : Speaklike pw or API key
- sl : source language code
- tl : target language codes (comma separated)
- mode : rest or socket (to initiate an asynchronous socket for event driven Javascript)
- callback_url : callback URL to send translations and events to (if omitted the service will return translations inline, or return an error code if none is received within 30 seconds)
- apikey : optional API key to require in callback requests
- key : session nickname or key (optional)
- ajax : y/n, if set to y, it will load a demo web page with Javascript that initiates the socket connection if mode=socket (you can take this code and add your own functions to it)

The API replies with a JSON dictionary with the system assigned session ID and other meta

data, or an HTTP error (e.g. 404 = translators not available)

You should expect the following fields in the JSON response:

- session : session ID/hash
- languages : languages supported

/realtime/queue

We strongly recommend that you use asynchronous callbacks with the REST API, so that translations are promptly transmitted to your system without delay. If you are unable to do so, because of a firewall policy for example, you can poll for translations using the /realtime/queue API. Just be aware that this will introduce noticeable latency to your application.

To use this method, call /realtime/start with mode=rest, and set callback=queue This will instruct the translation engine to store the translations locally. Then poll /realtime/queue with the following parameter:

- session : session ID

You will receive a JSON response with a list of recent translations (usually up to 50 records), each record will have the following fields:

- session : session ID
- guid : optional message ID or sequence number
- sl : source language
- tl : target language
- st : source text
- tt : translated text

While this will work, we don't recommend it unless you have no way of implementing asynchronous callback or the Javascript push notification, due to the increased latency this will introduce (up to a second or more).

/realtime/translate

This API handler expects the following list of parameters:

- session : session ID
- sl : source language code
- tl : target language code
- st : source text
- guid : unique identifier for message (optional, but recommended)

The API handler replies with ‘ok’ or an error code. You will receive an asynchronous callback or socket event when the translation is completed to the callback URL you specified when calling / realtime/start . The callback will be a POST form with the following fields:

- action : translation|event
- session : session ID
- apikey : optional user assigned API key for secure callbacks
- guid : message ID or sequence number
- state : event code (see below, if action=event)
- message : event message (if action=event)
- sl : source language code
- tl : target language code
- st : source text
- tt : translated text

If you initiated the session as an asynchronous Javascript socket session, the callback message will be sent over the persistent socket connection, and will trigger the onMessage() callback function in your Javascript app, which should expect a JSON message with the same fields as above.

NOTE: if you are translating non real-time messages, for example a customer support email, you can use the simpler /t API in a single “fire and forget” task without creating a session.
We require a session for chat translation so that we can reserve and hold translators until the conversation is completed.

/realtime/stop

This API handler is called to end a translation session, and expects one parameter:

- session : session ID

It will respond with ‘ok’ or an HTTP error. If you want to close all sessions associated with your user account, use the following parameters:

- session : “all”
  username : SpeakLike username
- pw : SpeakLike password

This will close out all real-time communication sessions in one transaction.

Asynchronous Events

The SpeakLike service will additionally transmit asynchronous event messages about translations in progress, to the same callback URL registered when starting the session. The callback handler, if it is designed to monitor events (in a simple implementation, it can ignore them) should watch for the following parameters:

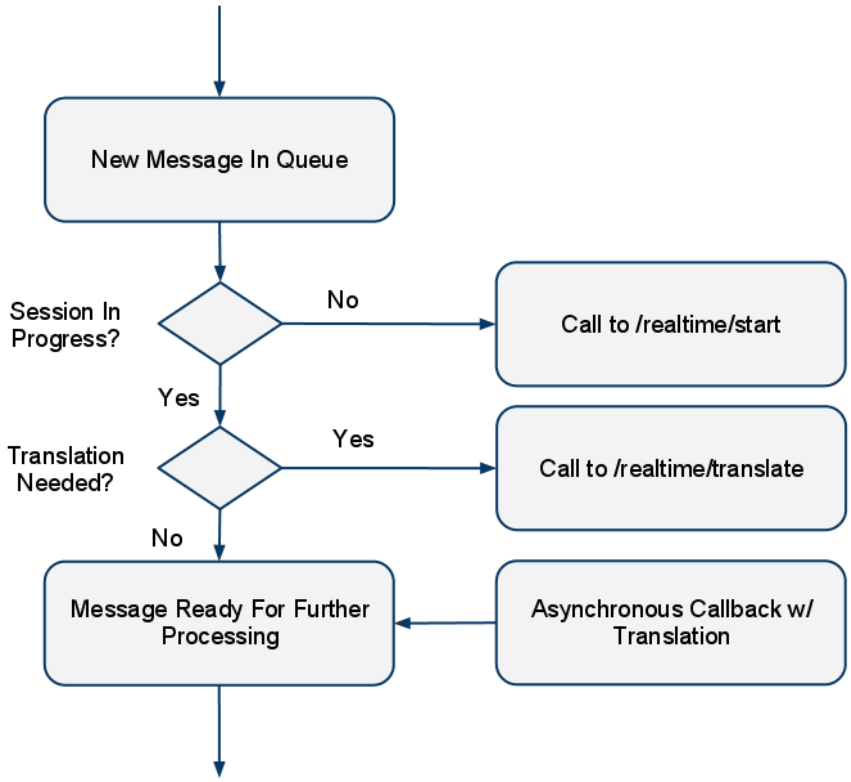
- action : ‘event’
- session : session ID
- guid : message ID
- state : event code (see below)
- message : optional message

Event messages are most frequently used to transmit state information, so that parties know when a translator is typing, or when translators have become unavailable. The application may expect one of the following messages in the state field:

- NO_TRANSLATOR
- TRANSLATOR_TYPING
- TRANSLATION_PROGRESS_NN%
- TRANSLATION_COMPLETE
- TRANSLATION_RESUMED

Application Flow

The flow for a basic implementation is quite simple, as shown in the flowchart below.



XMPP Interface

The XMPP interface provides exactly the same functionality as the REST callback API, except messages are relayed via a bidirectional XMPP session. This is an especially useful tool for chat/messaging system vendors because it is fast, but can also be tested manually. You just open a chat session to speaklikeim@appspot.com and start sending commands, so you can test application behaviors using your IM client (much like web developers can use Telnet to test servers by hand). This interface will also be particularly useful for chat/messaging developers since these systems often run on XMPP, so working with XMPP sessions is easy.

Initiating A Connection

This is simple, just open an XMPP chat session to speaklikeim@appspot.com

Commands and events are sent back and forth as JSON dictionaries, which are easy to parse and work with. The typical flow is the same as the REST callback API, where the client starts a session, then starts sending requests for translation, and then stops the session when done. Events and translations are sent asynchronously via the return path.

Starting A Session

Simply send the following JSON dictionary to speaklikeim@appspot.com

> {“action”:”start”,”username”:”speaklike_login”,”pw”:”speaklike_pw”,”languages”:”lang1,lang2...”}

The XMPP service will reply with a JSON dictionary as follows:

> {“status”:”ok”, “session”:”session_id”}
> or
> {“status”:”error”,”reason”:”error message”}

Requesting A Translation

Simply send the following JSON dictionary to speaklikeim@appspot.com

> {“action”:”translate”,”sl”:”lang1”,”tl”:”lang2”,”session”:”session id”,”st”,”text to translate”,”guid”:”message id”}

The XMPP service will reply with a JSON dictionary as follows:

> {“session”:”session id”,”guid”:”message id”,”status”,”ok|error”}

Ending A Session

Simply send the following JSON dictionary to speaklikeim@appspot.com

> {“action”:”stop”,”session”:”session id”}

The XMPP service will reply with a JSON dictionary as follows:

> {“status”:”ok|error”}

Receiving Events And Translations

The XMPP service will send events and translations back to you as they arrive.

Events are sent as a JSON dictionary in the following form:

> {“msgtype”:”event”,”session”:”session id”,”guid”:”message id”,”state”:”state label”}

Translations are sent as a JSON dictionary in the following form:

> {“mgtype”:”translation”,”session”:”session id”,”guid”:”message id”,”sl”,”source lang”,”tl”:”target lang”,”st”:”source text”,”tt”:”translated text”}

Best Practices

Use UTF-8 encoding

The system uses UTF-8 as the default character encoding. If you are supporting customers in multiple languages, your system should already be set up this way by default.

Listen for messages

Asynchronous messages are helpful in improving the user experience since the delays in translation can be noticeable. We send event messages to notify you when a translator is typing, and also about approximate progress, information you can incorporate into your user interface so users know that something is happening. While you don't need to implement this when building a first prototype, we do recommend implementing this as you go to production.

Be Aware Of Communication Delays

The translation process typically adds several seconds of latency to message transmittal. It is helpful if your application communicates this to users in a way that they understand a person is involved in translating their session. Otherwise the user may think that the session has died, or that the other party is unresponsive. You can look for event messages (action=event) which contain information about session activity (e.g. translator is typing), or you can fake this in your client application.