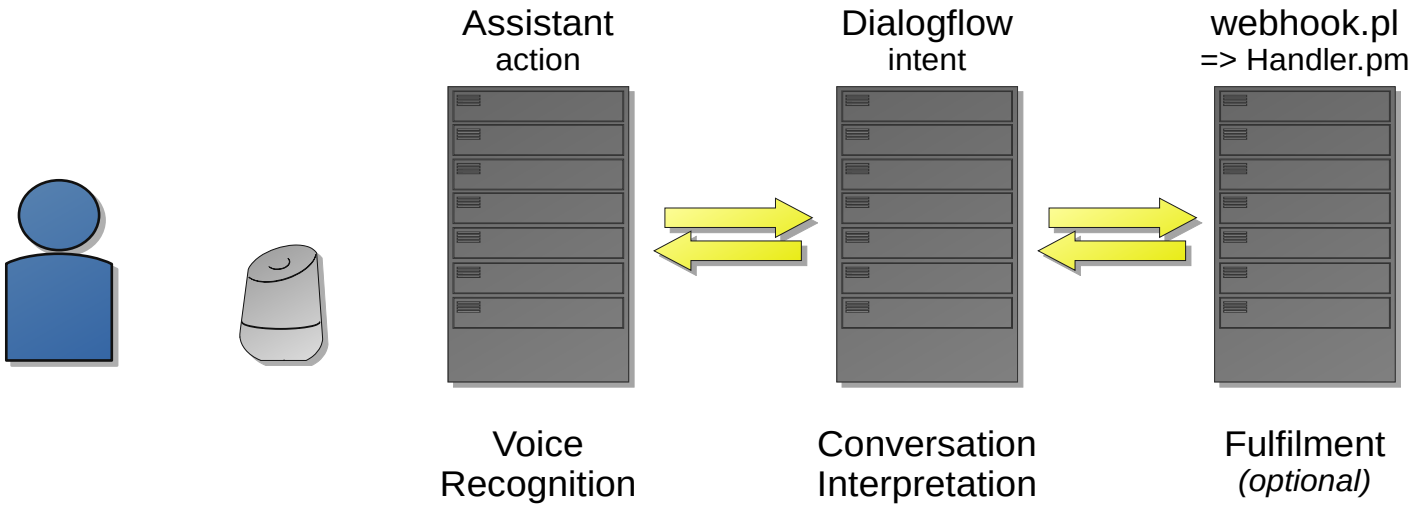
Google Dialogflow Perl Webhook API

v2019-11-30

Google Assistant Overview



Google Assistant Data Flow

Assistant

Requests for a specific application (or *action*, in Google terminology) spoken to a Google Home device are passed to the Google Assistant server, where speech recognition (and voice recognition, if configured) takes place.

Dialogflow

A JSON message is passed to the Dialogflow server for interpretation.

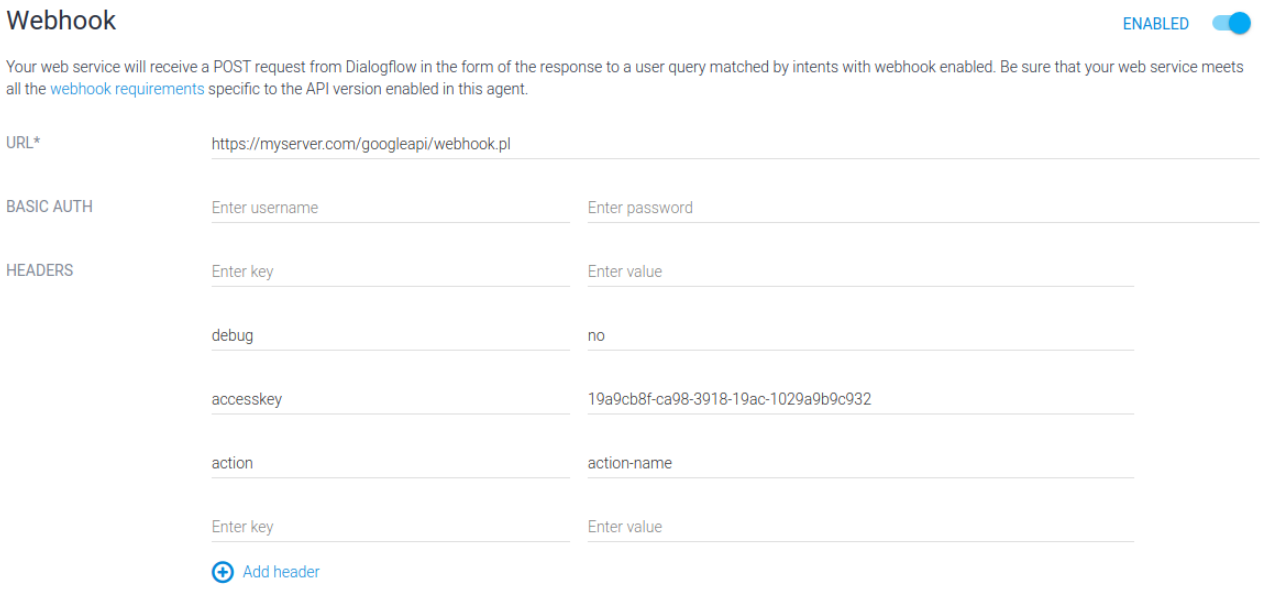
The Dialogflow server interprets the textual message received, and works out how to satisfy the response – this is by identifying an appropriate *intent*.
Intents are identified along with sample phrases and also any possible parameters, for example 'buy some cheese' could have 'cheese' identified as a parameter.
The *intent* can have an immediate response configured in Dialogflow, or it can be associated with a fulfilment script or function.

The fulfilment can be on a Google server, or associated with an external HTTPS server.
For this application, fulfilment is satisfied with the *webhook.pl* script.

The same *webhook.pl* script is capable of serving several different google actions, and in order to support this, the fulfilment in Dialogflow is configured to have a fulfilment *header key* of '*action*' with a value which uniquely identifies the action.

In addition, it is recommended that the Dialogflow fulfilment also has a *header key* of '*accesskey*' with a secret password, such that only the Dialogflow server will be able to use your *webhook.pl* script.

Finally, the Dialogflow fulfilment can contain an optional *header key* of '*debug*', which if set to '*yes*' will cause each transaction to be logged in the 'Log' folder.



Example Fulfilment Webhook Configuration

Webhook.pl

The *webhook.pl* script is placed on any HTTPS accessible server, and Dialogflow fulfilment is configured to point to it.
Dialogflow passes a request JSON message to *webhook.pl*, which then dispatches the request to the appropriate handler.
Webhook.pl is responsible for dispatching to the handler, and reporting any errors encountered.
The directory structure for the API, and handlers is as follows:

google-api-install-folder/
webhook.pl
Actions/*action-name*/*Handler.pm*
Lib/Conversation.pm
Log/
Manual/usermanual.pdf

The main processing is performed by *Handler.pm*.

Conversation.pm is the perl module which contains all of the intent conversation functions (to interpret the request JSON, and construct the response JSON).

Log is an optional folder, which is used to directly store the requests and responses from the *webhook pl*. If used, it is recommended that this folder is password protected.

Handler.pm

Headers and use statements. The use statements in italics below are not strictly necessary, however, if using an interactive editor, without these, it will not be possible to syntax check, because Lib::Conversation will not be found.

```
#
# Handler for: action-name
#

package Handler;

# Necessary for Handler.pm standalone compilation / syntax checking
use File::Basename;
use Cwd qw(abs_path);
use lib dirname ( abs_path(__FILE__) ) . "/../.." ;

use Lib::Conversation ;
```

A dispatch function called process is called from the webhook, and this is responsible for interpreting the intent, and passing execution to an appropriate function.

```
##############################
#  Intent Processing / Dispatch

sub process {

    my ($conv ) = @_ ;

    # Check the API accesskey (this dies if the key is not correct)
    $conv->checkAccessKey("19a9cb8f-ca98-3918-19ac-1029a9b9c932") ;

    # Get the current intent
    my $intent = $conv->intentName() ;

    if ( "$intent" eq "favorite-colour" ) {
          intent_favoritecolour($conv) ;

    } else {
          # Default message when an intent could not be found
          # This will only be called if you have created an intent in
          # Dialogflow, but not included a handler function.
          $conv->askSimple( {
                "en" => "You've asked something I can't handle, Sorry",
                "fr" => "Il ya un problème internal" } ) ;
    }
    return 1 ;
}
```

There are several intent handler functions.

```
##############################
# Favorite Colour Intent

sub intent_favoritecolour {
    my ($conv) = @_ ;

    my $last = $conv→getRequestContextParams(“colour”) ;
    my $colour = $conv->getParameter("colour") ;

    if ( $last→{colour} && $last→{colour} ne $colour ) {
          $conv->askSimple( {
                "en" => “I see you have changed your mind. I like $colour too.”,
                "fr" => "Moi, je n’aime pas le colour $colour” } );
    } else {
          $conv->askSimple( {
                "en" => “I like $colour too.”,
                "fr" => "Moi, je n’aime pas le colour $colour” } );
    }

    $conv→setResponseContext(“colour”, 100, { “colour” => $colour } ) ;

    end_processing($conv) ;
}
```

You can end an intent handler function with a simple return or $conv→close(), however, a separate end-processing function is useful as it allows you to differentiate between conversations that are one-off (i.e. ‘ask my application to do something’) and interactive (i.e. ‘let me talk to my application’).

```
##############################
# Next / Close

sub end_processing {
    my ($conv) = @_ ;

    if ($conv->isConversationNew()) {

          # New conversation without 'welcome'
          $conv->close() ;

    } else {

          # Current interactive conversation
          $conv->askSimple( {
                "en" => [
                      "What now?",
                      "How else can I help?",
                      "What's next?" ],
                "fr" => "Et maintenant?" } ) ;
    }
}

# Handler.pm must return true
1;
```

Conversation.pm

The functions supported by the Conversation module are identified in this section.

Constructor and Response

Constructor (Request)

This function creates a new conversation object, and returns a handle. This handle is then used in all future interactions with the conversation.
The constructor is provided the JSON formatted request string.

The constructor performs the following tasks:

- Decode the request JSON
- Extract userstorage variables from the request (these variables survive from conversation to conversation). If they are corrupt in any way, the userstorage variables are ignored.

If there is a problem, or the key does not match, the constructor will die and throw an error.

my $conv = Conversation::new($requestjson)	
Returns	Conversation Structure / Handle
$requestjson	This is the JSON request string received from the google server.

Response

This function returns the JSON formatted response string, which should be passed back to the requesting server.
The response JSON is constructed by making calls to the Conversation 'class', for example, calling $conv→askSimple() will create a RichResponse text response, and calling $conv→close() will ensure that the correct flags are set to shut down the conversation.

$responsejson = $conv->response()	
Returns	Response JSON string.

Getting Request Information

Check Access Key

This function checks the supplied access key against the access key set in the Google Dialogflow Fulfilment header parameter ‘accesskey’. If they do not match, an error is thrown.

$ok = $conv->checkAccessKey($accesskey)	
Returns	True if the accesskey is valid, if not an error is thrown
$accesskey	This is the access key string, which must match the ‘accesskey’ parameter in the Dialogflow Fulfilment header.

Intent Name

This function returns the conversation intent name. An error is thrown if the intent name cannot be found.
The intent name is the ‘action name’ specified in the intent (e.g. action-name.action-name-yes), otherwise it is the title of the intent itself if the action name is not provided.

$name = $conv->intentName()	
Returns	Intent name string.

Language

lang returns the current conversation two-letter language.

$lang = $conv→lang()	
Returns	Language code string, e.g. ‘en’ or ‘fr’

New Conversation

Returns true if the conversation is new

$new = $conv→isConversationNew()	
Returns	True if the conversation is new

Parameter

This function returns the given parameter in the current request.

$value = $conv->getParameter($param)	
Returns	The string value of the intent parameter
$param	The string name of the parameter.

Note that if the request has been processed in dialogflow with follow-up-requests, for example by asking for a yes/no confirmation, the parameters will not be associated with the final 'yes' response – instead, they will have been transferred to all followup contexts associated with the action.

$value = $conv→getContextParameter($context, $param)	
Returns	The string value of the intent parameter
$context	The string name of the current context – e.g. *action-name-followup*
$param	The string name of the parameter.

Verified User

Returns true if the user is verified (userStorage can be used)

$new = $conv→isUserVerified()	
Returns	True if the user is verified (by login or voice match)

Output Messages and Closing Conversation

Output Simple Text Message

askSimple places a speech (and optionally text) message into the response.
Any conversation response can accept up to two 'askSimple' response messages – any more, and the conversation handler will throw an error. The speech and text messages must be in the speechtext message format.

$conv→askSimple($speech [,$text])	
Returns	Nothing. Throws an error if more than two askSimple calls are made per conversation.
$speech	Message to speak (type speechtext)– this can optionally contain SSML mark-up.
$text	Optional text (type speechtext) version of the message (to print).

Close Conversation

close places a close flag into the conversation response. The conversation will be terminated following this transaction.

$conv→close()	
Returns	Nothing

Permissions Functions

Ask for Permission

Sets the conversation response to contain a request for some access permissions. Note that this cannot be used in the same response as an askSimple call.
The server will prompt the user for acceptance, then call the handler with an *actions.intent.PERMISSION* intent.

$conv→askPermission($message, $permissions)	
Returns	Nothing
$message	This is a speechtext message which is provided along with a request for permission. It should be something like “in order to provide you with blah”.
$permissions	This is a space-separated string containing requested permissions, and can contain one or more of: NAME DEVICE_PRECISE_LOCATION DEVICE_COARSE_LOCATION UPDATE

Permission Granted

For use in the *actions.intent.PERMISSION* intent handler. This function returns true if permission was granted.

$conv→permissionGranted()	
Returns	True if permission was granted.

Store Permissions

This function stores any provided permissions in the conversation response into the user storage. It should be called in a handler for the *actions.intent.PERMISSION* intent.

The following user storage strings are set, if the information is provided in the response:

displayName	*latitude*	*city*
givenName	*longitude*	*zipcode*
familyName	*formattedAddress*	

$conv→storePermissionsInUserStorage()	
Returns	Nothing

User Storage Functions

Set User Storage

This function sets a value / string in the user storage in the response.
User storage is specific to both the user and the action, and is maintained across conversations.

$conv→setUserStorage($variable, $value)	
Returns	Nothing
$variable	The Variable string within the user storage to set
$value	The value to set. This can be a string, hash or an array

Get User Storage

This function retrieves a value / string from the user storage request.

$value = $conv→getUserStorage($variable)	
Returns	The previously set value. This can be a string, hash or an array
$variable	The Variable string within the user storage to set

Clear User Storage

This function clears the contents of the user storage in the response.

$conv→clearUserStorage()	
Returns	Nothing

Context Functions

Set Response Context

Sets a context within the next response. Contexts are set for a number of conversation interactions defined by lifespan. Contexts can optionally take a parameters hash, which can be returned in subsequent accesses.
After the lifespan, the context is automatically deleted.

$conv→setResponseContext($context, $lifespan [, $paramshash])	
Returns	Nothing
$context	The context name string that is set
$lifespan	The lifespan integer of the context (number of conversation interactions)
$paramshash	Optional hash of parameters to store with the context in the form { "param" => "value", ... }

Context Exists Check

Checks the request and returns true if the context exists.

$exists = $conv→isRequestContext($context)	
Returns	True if the context exists.
$context	The context name string that is set

Fetch Context Parameters

Returns the parameters hash associated with the given context.

$parameters = $conv→getRequestContextParams($context)	
Returns	The hash of parameters which were set in the setResponseContext call
$context	The context name string that is set

Fetch Context Count

Returns the context remaining lifespan count (integer).

$count = $conv→getRequestContextCount($context)	
Returns	The int representing the remaining lifespan count
$context	The context name string that is set

Translations, Speech and Text Messages

These variables contain text to say or print. There are several different formats the messages can take, supporting different languages, random and sequential response strings.

speechtext = simplemessage | simplelist | hashmessage | sequentialhashmessage

Simple Message

This is a single message, that is always the same.

```
$simplemessage = “Simple String” ;
```

Simple List

This is a list of messages, that are chosen for the response (by default, at random).

```
$simplelist = [ “Simple String”, “Alternative String” , ... ] ;
```

Hash Message

This is a hash of messages, which can be either Simple Messages or Simple Lists.
Each hash entry represents a different language, which is selected automatically in the conversation request.

```
$hashmessage = {
        “en” => $simplemessage,
        “fr” => $simplelist } ;
```

Sequential Hash Message

This is a hash of messages, which can be either Simple Messages or Simple Lists.
Each hash entry represents a different language. The difference between this and the previous Hash Message is that any Simple Lists are returned in sequence rather than randomly.
Ref is a unique reference mnemonic which is used to track where in the sequence the message output currently is.

```
$sequentialhashmessage = {
        “sequential” => “ref”,
        “en” => $simplelist,
        “fr” => $simplemessage } ;
```