Translator Service Programming Guide

Version 2.1 – September 26, 2012

Abstract

The Project Hawaii Translator Service provides translation and text-to-speech services for mobile applications. In essence, the Translator service provides an interface to the Microsoft Translator.

This document provides an introduction to the Translator Service and walks you through a simple Windows Phone application that uses the service.

Contents

[Introduction 2](#_Toc335306444)

[Prerequisites 2](#_Toc335306445)

[Overview of the Translator Service 2](#_Toc335306446)

[Walkthrough: Translator Sample 3](#_Toc335306447)

[Contents of the Sample 4](#_Toc335306448)

[Obtaining a List of Languages 5](#_Toc335306449)

[Requesting Translation 7](#_Toc335306450)

[Speaking a Language 8](#_Toc335306451)

[Using the Translator Service in an Application 10](#_Toc335306452)

[Add Required Assemblies 10](#_Toc335306453)

[Reference the Namespace 10](#_Toc335306454)

[Set Up Your Authentication Credentials 10](#_Toc335306455)

[Resources 10](#_Toc335306456)

**Disclaimer:** This document is provided “as-is”. Information and views expressed in this document, including URL and other Internet Web site references, may change without notice. You bear the risk of using it.

This document does not provide you with any legal rights to any intellectual property in any Microsoft product. You may copy and use this document for your internal, reference purposes.

© 2012 Microsoft Corporation. All rights reserved.

Microsoft, Visual Studio, Silverlight, XNA, Windows Azure, and Windows are trademarks of the Microsoft group of companies. All other trademarks are property of their respective owners.

# Introduction

The Project Hawaii Translator Service provides translation and text-to-speech services for mobile applications. The translation service translates text from one language to another, and the text-to-speech service returns an audio stream that renders a string in a spoken language. In essence, the Translator service wraps an interface to Microsoft Translator.

This document provides an introduction to the Translator Service and walks you through a simple Windows Phone application that uses the service.

# Prerequisites

Before you can build an application that uses the Translator service, you must:

* Install and build the Project Hawaii SDK.
* Build the Project Hawaii SDK.
* Obtain Project Hawaii authentication credentials.

For information about installation, build procedures, and credentials, see “Hawaii Installation Guide,” which is installed with the SDK and is available on the web, as listed in “Resources” at the end of this document.

In addition, you should be familiar with the following:

* Windows Communication Foundation (WCF)
* Microsoft Silverlight®
* Windows Phone 7 SDK

# Overview of the Translator Service

The Project Hawaii Translator service provides an interface through which a mobile application can:

* Request the translation of a string from one language to another.
* Request a stream that contains a spoken rendition of a string in a particular language.

The Microsoft Translator API provides the translation capabilities that the Translator service exposes. The source and target languages are therefore the same as those supported by Microsoft Translator. You can find complete information on Microsoft Translator in MSDN, as listed in “Resources.” For Project Hawaii participants, a subscription to the Microsoft Translator is not required to use the service.

Applications access the service through the Translator Client library. The **Microsoft.Hawaii.Translator.Client** namespace defines the following classes:

|  |  |
| --- | --- |
| Class | Description |
| **GetLanguagesForSpeakResponse** | Response for the **GetLanguagesForSpeakAsync** method. |
| **GetLanguagesForSpeakResult** | Result for the **GetLanguagesForSpeakAsync** method. |
| **GetLanguagesForTranslateResponse** | Response for the **GetLanguagesForTranslateAsync** method. |
| **GetLanguagesForTranslateResult** | Result for the **GetLanguagesForTranslateAsync** method. |
| **Language** | A class to represent the language. |
| **SpeakResponse** | Response for the **SpeakAsync** method. |
| **SpeakResult** | Response for the **SpeakAsync** method. |
| **TranslateResponse** | Response for the **TranslateAsync** method. |
| **TranslateResult** | Response for the **TranslateAsync** method. |
| **TranslatorService** | Helper class that provides access to the Translator service and defines the **GetLanguagesForSpeakAsync**, **GetLanguagesForTranslateAsync**, **SpeakAsync**. and **TranslateAsync** methods. |

# Walkthrough: Translator Sample

The TranslatorSample application is installed with the Project Hawaii SDK. The sample implements a simple interface that translates typed text from one language to another and speaks text in your choice of languages. This brief walkthrough introduces the components of the sample and shows you how the sample uses the Translator service.

In addition to the prerequisites described earlier this document, the TranslatorSample program requires the Microsoft XNA® Framework, which is installed with the Project Hawaii SDK.

To compile and run TranslatorSample

1. In Visual Studio, open TranslatorSampleApp.csproj.

2. Open the HawaiiClient.cs file and set the **AdmClientId** and **AdmClientSecret** strings to your Azure Data Market (ADM) credentials.

3. Save the HawaiiClient.cs file.

4. Build the solution.

5. Run the sample with or without the debugger, as you prefer.

The following figure shows the initial Windows Phone emulator window for the sample:



To use TranslatorSample

To translate written text from one language to another:

* Tap the **Translated from** box to pop up a list of languages, flick to scroll to the source language, then tap the name of the language. Type the text below the **Translated from** box.
* Tap the **Translated to** box to pop up a list of languages, flick to scroll to the target language, then tap the name of the language in the list.
* Tap **Translate** to request translation.

To translate speech from one language to another:

* Flick to display the **Speech** screen.
* Tap the **Speak in** box to pop up a list of languages, flick to scroll to the desired language, then tap the name of the language. Type the text below the **Speak in** box.
* Tap **Speak** to hear the spoken text.

## Contents of the Sample

The following table lists the C# source files for the sample:

|  |  |
| --- | --- |
| Filename | Description |
| App.xaml.cs | Implements App as the Application class object and the PhoneApplicationFrame class. |
| HawaiiClient.cs | Implements the HawaiiClient class, which stores the Hawaii Application ID. |
| MainPage.xaml.cs | Implements the code for the application. |

Because this document assumes that you are familiar with Windows Phone application development, the walkthrough focuses on the aspects of the sample that are unique to the Hawaii Translator service.

## Obtaining a List of Languages

When you start the application, the constructor for the App object performs basic initialization tasks for the application. This code appears in App.xaml and its code-behind file.

After application initialization is complete, Silverlight makes the main phone application frame active and calls the MainPage constructor method. The verifies the Hawaii Application ID and then calls the Translator service to obtain a list of languages that the service can translate. The local BindLanguagePickers method calls the service. The code, which follows, appears in MainPage.xaml.cs, as does all the sample code shown in this document:

private void BindLanguagePickers()

{

this.StartAsync();

try

{

TranslatorService.GetLanguagesForTranslateAsync(

HawaiiClient.AdmClientId,

HawaiiClient.AdmClientSecret,

this.OnGetLanguagesForTranslateComplete,

null);

}

catch (ArgumentNullException ex)

{

this.EndAsync();

MessageBox.Show(ex.Message);

}

}

BindLanguagePickers first calls the local StartAsync method, which dims the screen and disables the buttons to prevent the user from making additional requests while the call to get languages is pending.

It then calls **TranslatorService.GetLanguagesForTranslateAsync** to obtain a list of languages. This call appears in a **try**-**catch** statement because the method throws an exception if the ADM client credentials are null.

**GetLanguagesForTranslateAsync** takes the following parameters:

* The ADM client ID, which is set in the HawaiiClient.cs file.
* The ADM client secret, which is set in the HawaiiClient.cs file.
* A pointer to a delegate for callback when the method has completed.
* A user-defined object, which the application does not use in this call.
* An optional locale variable, which determines the language in which to return the result of the method. The default locale is “en,”, so the list of languages is returned in English.

When the request is complete, the Translator service calls the OnGetLanguagesForTranslateComplete callback and passes a **GetLanguagesForTranslateResult** object that contains the results of the service call. The following shows the code for the callback function:

private void

OnGetLanguagesForTranslateComplete(GetLanguagesForTranslateResult result)

{

Dispatcher.BeginInvoke(() =>

{

if (result.Status == Status.Success)

{

List<Language> languagesFrom =

new List<Language>(result.SupportedLanguages);

languagesFrom.Insert(0, new Language()

{ Name = DetectedLanguageName, Code = DetectedLanguageCode });

List<Language> languagesTo =

new List<Language>(result.SupportedLanguages);

this.PickerLanguageFrom.ItemsSource = languagesFrom;

this.PickerLanguageTo.ItemsSource = languagesTo;

TranslatorService.GetLanguagesForSpeakAsync(

HawaiiClient.AdmClientId,

HawaiiClient.AdmClientSecret,

this.OnGetLanguagesForSpeakComplete,

null);

}

else

{

this.EndAsync();

MessageBox.Show(result.Exception.Message);

}

});

}

Whether or not the service call succeeds, OnGetLanguagesForTranslateComplete updates the user interface. Therefore, it uses **Dispatcher.BeginInvoke** to ensure that its code runs in the main UI thread, as Silverlight requires.

If the call succeeded, the list of languages is available in the **SupportedLanguages** member of the **GetLanguagesForTranslateResult** object. The sample populates both the list of source languages and the list of target languages from the **SupportedLanguage** member. Each item in the list is a **Language** object, which has two properties:

* **Name**, a string that contains the English name of the language.
* **Code**, a string that contains a brief code for the name.

You can find a list of all possible language codes on MSDN in “Translator Language Codes,” which is listed in “Resources.”

The callback then calls **TranslatorService.GetLanguagesForSpeakAsync** to obtain a list of languages that the translator can speak. This method takes the same parameters as **GetLanguagesForTranslateAsync**. The completion callback for this method calls the local EndAsync method, which returns the screen and buttons to its normal settings. The callback then updates the list of spoken languages. The following shows the code:

private void OnGetLanguagesForSpeakComplete(GetLanguagesForSpeakResult result)

{

Dispatcher.BeginInvoke(() =>

{

this.EndAsync();

if (result.Status == Status.Success)

{

List<Language> languages =

new List<Language>(result.SupportedLanguages);

languages.Insert(0, new Language()

{ Name = DetectedLanguageName, Code = DetectedLanguageCode });

this.PickerLanguageSpeakTo.ItemsSource = languages;

}

else

{

MessageBox.Show(result.Exception.Message);

}

});

}

## Requesting Translation

An application requests translation of text by calling the **TranslatorService.TranslateAsync** method. When a user taps the **Translate** button, the sample calls this method from its button-click event handler, as the following code shows:

private void BtnTranslate\_Click(object sender, RoutedEventArgs e)

{

this.StartAsync();

try

{

this.tbTextTranslatedTo.Text = string.Empty;

TranslatorService.TranslateAsync(

HawaiiClient.AdmClientId,

HawaiiClient.AdmClientSecret,

this.tbTextTranslatedFrom.Text.Trim(),

(this.PickerLanguageTo.SelectedItem as Language).Code,

this.OnTranslateComplete,

null,

(this.PickerLanguageFrom.SelectedItem as Language).Code);

}

catch (ArgumentNullException ex)

{

this.EndAsync();

MessageBox.Show(ex.Message);

}

}

**TranslateAsync** takes the following parameters:

* The ADM client ID, which is set in the HawaiiClient.cs file.
* The ADM client secret, which is set in the HawaiiClient.cs file.
* The text to translate, which the application reads from the tbTextTranslatedFrom text box.
* The code for the target language.
* A pointer to a delegate for callback when the method has completed.
* A user-defined object, which the application does not use in this call.
* The code for the source language.

When translation is complete, the service calls the OnTranslateComplete callback. The callback checks status and displays the result of the translation in the tbTextTranslatedTo text box. The following shows the code for this function:

private void OnTranslateComplete(TranslateResult result)

{

Dispatcher.BeginInvoke(() =>

{

this.EndAsync();

if (result.Status == Status.Success)

{

this.tbTextTranslatedTo.Text = result.TextTranslated;

}

else

{

MessageBox.Show(result.Exception.Message);

}

});

}

## Speaking a Language

An application requests speech by calling the **TranslatorService.SpeakAsync** method. **SpeakAsync** returns an audio stream that renders an input string in the language that the caller specifies; it does not perform translation. When a user taps the **Speak** button, the sample calls **SpeakAsync** from its button-click event handler, as the following code shows:

private void BtnSpeak\_Click(object sender, RoutedEventArgs e)

{

this.StartAsync();

try

{

TranslatorService.SpeakAsync(

HawaiiClient.AdmClientId,

HawaiiClient.AdmClientSecret,

this.tbTextSpeak.Text.Trim(),

this.OnSpeakComplete,

null,

(this.PickerLanguageSpeakTo.SelectedItem as Language).Code);

}

catch (ArgumentNullException ex)

{

this.EndAsync();

MessageBox.Show(ex.Message);

}

}

The sample calls **SpeakAsync** with the following parameters:

* The ADM client ID, which is set in the HawaiiClient.cs file.
* The ADM client secret, which is set in the HawaiiClient.cs file.
* The text to translate, which the application reads from the tbTextSpeak text box.
* A pointer to a delegate for callback when the method has completed.
* A user-defined object that is passed to the callback, for which the sample passes null.
* The code for the language in which to speak.

**SpeakAsync** also supports two optional parameters, which the sample does not use:

* A string that specifies the audio format in which to return the spoken stream. Possible values are “audio/wav” and “audio/mp3”; “audio/wav” is the default format.
* A string that specifies the quality of the audio signal, either “MaxQuality” or “MinSize”, which is the default.

When SpeakAsync has completed, the Translator service calls the sample’s OnSpeakComplete callback, which creates a **System.Io.MemoryStream** object to hold the returned audio stream, and then calls the Microsoft XNA Framework to play the audio on the device’s speaker. The following shows the code:

private void OnSpeakComplete(SpeakResult result)

{

Dispatcher.BeginInvoke(() =>

{

this.EndAsync();

if (result.Status == Status.Success)

{

MemoryStream stream = new MemoryStream(result.Audio);

if (stream != null)

{

var effect =

Microsoft.Xna.Framework.Audio.SoundEffect.FromStream(stream);

Microsoft.Xna.Framework.FrameworkDispatcher.Update();

effect.Play();

}

}

else

{

MessageBox.Show(result.Exception.Message);

}

});

}

# Using the Translator Service in an Application

* Add required assemblies.
* Reference the namespace.
* Set up your authentication credentials.

## Add Required Assemblies

Applications that use the Translator service depend on the following libraries, which are built as part of the Project Hawaii SDK:

* Microsoft.Hawaii.ClientBase.dll
* Microsoft.Hawaii.Translator.Client.dll

To add the libraries to your application

* Build the Hawaii SDK, as described in “Getting Started with the Project Hawaii SDK.”
* Add references to the following DLLs to your Visual Studio project:
* Microsoft.Hawaii.ClientBase.dll
* Microsoft.Hawaii. Translator.Client.dll

## Reference the Namespace

The Translator service is defined in the **Microsoft.Hawaii.PathPrediction.Client** namespace. For ease of reference, include the following in your code:

using Microsoft.Hawaii;

using Microsoft.Hawaii.Translator.Client;

## Set Up Your Authentication Credentials

Your application authenticates itself with the service by using an ADM client ID and secret. If you do not already have ADM credentials, obtain them as described in “Getting Started with the Project Hawaii SDK.”

To set up the ADM credentials in your code, copy the HawaiiClient.cs file from one of the sample applications, add it to your project, and change the values of the following strings to your ADM client ID and secret, respectively:

* **AdmClientId**
* **AdmClientSecret**

# Resources

This section provides links to additional information about the Project Hawaii Translator Service and related topics.

Microsoft Research Project Hawaii

<http://research.microsoft.com/en-us/projects/hawaii/default.aspx>

Getting Started with the Project Hawaii SDK

<http://research.microsoft.com/en-US/projects/hawaii/docs.aspx>

Microsoft Research Project Hawaii on Facebook

<http://www.facebook.com/pages/Microsoft-Research-Project-Hawaii/164295863611699>

MSDN

How to: Create Your First Silverlight Application for Windows Phone  
<http://msdn.microsoft.com/library/ff402526(v=VS.92).aspx>

How to: Create Your First XNA Framework Application for Windows Phone  
<http://msdn.microsoft.com/en-us/library/ff472340(v=vs.92).aspx>

Microsoft Translator  
<http://msdn.microsoft.com/en-us/library/dd576287>

Programming Windows Phone 7  
<http://blogs.msdn.com/b/microsoft_press/archive/2010/10/28/free-ebook-programming-windows-phone-7-by-charles-petzold.aspx>

Translator Language Codes  
[http://msdn.microsoft.com/en-us/library/hh456380.aspx](http://msdn.microsoft.com/en-us/library/dd877180)