Relay Service Programming Guide

Version 2.1 – September 13, 2012

Abstract

The Project Hawaii Relay service provides a relay point in the cloud that mobile applications can use to communicate. The service enables applications to name endpoints and to store messages until the intended recipient can retrieve them.

This document introduces the managed-code interface for the relay service and walks you through a sample application that uses the service.

Contents

[Introduction 2](#_Toc335306800)

[Prerequisites 2](#_Toc335306801)

[Overview of the Relay Service 2](#_Toc335306802)

[Relay Client Library 4](#_Toc335306803)

[Walkthrough: Sample Applications 4](#_Toc335306804)

[Source Files 6](#_Toc335306805)

[Initialization 6](#_Toc335306806)

[Endpoint Creation 7](#_Toc335306807)

[Group Creation 9](#_Toc335306808)

[Sending Messages 11](#_Toc335306809)

[Receiving Messages 12](#_Toc335306810)

[Storing Relay Data 13](#_Toc335306811)

[Using the Relay Service in an Application 14](#_Toc335306812)

[Include the Required Libraries 14](#_Toc335306813)

[Reference the Namespace 15](#_Toc335306814)

[Set Up Your Authentication Credentials 15](#_Toc335306815)

[General Guidelines 15](#_Toc335306816)

[Resources 15](#_Toc335306817)

**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, 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 Relay service provides a relay point in the cloud that mobile applications can use to communicate. The Relay service enables applications to transmit and receive messages through registered endpoints, and to store messages for later retrieval. By grouping endpoints, an application can communicate with multiple targets in a single call.

This document introduces the managed-code interface for the relay service and walks you through a sample application that uses the service.

# Prerequisites

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

* Install 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 Relay Service

The Relay service enables mobile applications to transmit and receive messages through one or more registered endpoints. Each endpoint has a unique registration ID that identifies it as a target for sending or receiving messages.

The Relay service provides two important benefits for mobile clients:

* Enables connections to individual mobile devices that do not have static IP addresses.
* Provides storage for later retrieval of messages. The Relay service provides a first-in, first-out (FIFO) storage queue for each registration ID so that messages can be saved and retrieved later.

The registration ID remains constant if the IP address of the target changes, or if the target is behind a many-to-one network address translation (NAT).

For example, in Figure 1, although the NAT has only one IP address, the Relay service can use the registration ID to identify each mobile device that is behind the NAT.



Figure 1. Mobile Devices behind NAT

The Relay service also supports the creation of groups of endpoints. Each group of endpoints has both a name and a registration ID. An application can use the name instead of the ID to communicate with the group. Figure 2 shows how two mobile devices that are members of a group can use the Relay service.



Figure 2. Two Mobile Devices Using Hawaii Relay Service

In the figure, each device has an individual endpoint, and both endpoints belong to MyAppGroup. The devices can communicate by sending messages to MyAppGroup; they are not required to know the individual registration IDs of the other members of their group.

# Relay Client Library

The simplest way to communicate with the Hawaii Relay service is to use the Relay Client Library. This library implements an interface that enables a mobile application to communicate with the Hawaii Relay service. The source code for this library is installed with the Project Hawaii SDK in the following location:

* Source\ServiceClients\Relay

By using the Relay Client library, you can create and destroy endpoints and use an endpoint as the source or target of messages to and from other endpoints. An endpoint can join a group or remove itself from a group. Both groups and single endpoints can be the recipients of messages, thus enabling you to transmit not just to a single endpoint but to a collection of endpoints. The library also supports cloud storage of relay service data.

Applications access the Relay Client library through the **Microsoft.Hawaii.Relay.Client** namespace, which defines the following classes:

|  |  |
| --- | --- |
| Class | Description |
| **Endpoint** | Represents a communications endpoint of the relay service. |
| **EndpointResult** | Represents the results of relay service invocation. |
| **Group** | Represents a multicast group of communications endpoints. |
| **GroupResult** | Represents the results of relay service invocation. |
| **Groups** | A container class for **Group** objects. |
| **Message** | Represents a message received from the relay service. |
| **MessagingResult** | Represents the results of relay service invocation. |
| **RelayService** | Helper class that provides access to the Relay service. |
| **RelayStorage** | Helper class to store relay information in the mobile application's isolated storage area. |

# Walkthrough: Sample Applications

The Project Hawaii SDK includes two sample applications that demonstrate the use of the relay service:

* RelaySample
* RelayPivotSample

The samples are installed in the Samples\Relay subfolder of the Hawaii SDK installation directory.

The RelaySample application implements a simple interface with which you can create endpoints, manage groups of endpoints, and send and receive messages through the relay service. This brief walkthrough introduces the components of the sample and shows you how the sample calls methods in the **RelayClientService** class.

To compile and run RelaySample

1. In Visual Studio, open RelaySampleApp.csproj.

2. Open the HawaiiClient.cs file and set the **AdmClientId** and **AdmClientSecret** strings to your 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 RelaySample

* Click **Create** **Endpoint**. Note that the endpoint assigned to the application changes.
* To create a group of endpoints, click **Manage Groups**. The application displays the Groups page, which shows all the currently available groups and the groups to which this endpoint belongs. Click **Create** to create a group and then click **Join** to add your endpoint to it. To leave a group, select its endpoint in the **Groups joined by this Endpoint** box and then click **Leave**. Click **Back** to return to the RelayClient page.
* To send a message, click **Send Messages** on the Relay Client page. The application displays the Messaging page. You can send the default message or use the virtual keyboard to type your own message. To send to one of the groups to which the current endpoint belongs, select the group in the **Groups joined by this Endpoint** box and then click **Send to Group**. You can also send a message to a particular endpoint—for either an individual device or a group—by entering the endpoint ID in the text box. Click **Back** to return to the RelayClient page.
* To receive the messages that are queued for your endpoint, click **Receive Messages** on the RelayClient page. The application displays the Messages page, which shows the number of messages in the queue, the endpoint ID of the sender, and the endpoint ID that was the target of the message. Note that if your endpoint is a member of multiple groups to which the same message was sent, you will receive the same message more than once. Click **Clear Messages** to empty the queue and **Back** to return to the RelayClient page.

## Source Files

The following table lists the C# source files for the RelaySample application.

|  |  |
| --- | --- |
| Filename | Description |
| App.xaml.cs | Implements App as the Application class object and initializes the phone application. |
| GroupsPage.xaml.cs | Implements the GroupsPage class, which provides code for the **Groups** screen of the application. |
| HawaiiClient.cs | Implements the HawaiiClient class, which stores the Hawaii Application ID. |
| MainPage.xaml.cs | Implements the initial UI for the application and manages user input. |
| ReceiveMessagesPage.xaml.cs | Implements the ReceiveMessagesPage class, which provides code for the **ReceiveMessages** screen of the application. |
| SendMessagesPage.xaml.cs | Implements the SendMessagesPage class, which provides code for the **SendMessages** screen of 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 Relay service.

## Initialization

When the application starts, the constructor for the App class establishes an exception handler, sets up profiling information for debugging, initializes Silverlight, and initializes the phone application. This code appears in App.xaml and its code-behind file.

After application initialization is complete, Silverlight makes the main phone application page active and calls the **OnNavigatedTo** method for the page. In the MainPage.xaml.cs file, the sample overrides this method as follows:

protected override void OnNavigatedTo(NavigationEventArgs nea)

{

if (String.IsNullOrEmpty(HawaiiClient.AdmClientId) ||

String.IsNullOrEmpty(HawaiiClient.AdmClientSecret))

{

return;

}

if (this.RelayContext.Endpoint != null)

{

this.MyEndpointLabel.Text = "My Endpoint : " +   
 this.RelayContext.Endpoint.RegistrationId;

this.EndpointGroupsListBox.Items.Clear();

foreach (Group group in this.RelayContext.Endpoint.Groups)

{

this.EndpointGroupsListBox.Items.Add(group.RegistrationId);

}

int selectedIndex = -1;

selectedIndex = this.EndpointGroupsListBox.Items.Count - 1;

if (selectedIndex != -1)

{

this.EndpointGroupsListBox.SelectedIndex = selectedIndex;

}

}

base.OnNavigatedTo(nea);

}

The **OnNavigatedTo** method ensures that the application provides ADM credentials. It then displays the current endpoint for the device, lists the groups that the endpoint has joined, and selects the first group in the list.

## Endpoint Creation

When a user clicks **Create** **Endpoint**, the application deletes the current endpoint if one already exists and then creates a new endpoint. The following shows the code, which appears in MainPage.xaml.cs:

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

{

// Check whether the active endpoint object is null. If null, create a

// new end point. If not null, delete the existing one and create a

// new end point.

if (this.RelayContext.Endpoint != null)

{

// Delete the active endpoint.

RelayService.DeleteEndPointAsync(

HawaiiClient.AdmClientId,

HawaiiClient.AdmClientSecret,

this.RelayContext.Endpoint,

null);

// Clear the created groups which were restored from storage.

this.RelayContext.Groups.Clear();

// Clear the items from list box.

this.EndpointGroupsListBox.Items.Clear();

}

// Create a new endpoint.

RelayService.CreateEndPointAsync(

HawaiiClient.AdmClientId,

HawaiiClient.AdmClientSecret,

MainPage.ApplicationName,

this.OnCompleteCreateEndPoint);

}

The sample calls **RelayService.DeleteEndPointAsync** to delete the existing endpoint. This method takes the following parameters:

* The ADM client ID and client secret, which the sample stores in the **HawaiiClient** object.
* The endpoint to delete, which the sample stores in the local RelayContext object.
* A pointer to a callback function that the Relay service calls when the **DeleteEndPointAsync** method has completed. The sample specifies **null** because it does not require callback.

The sample then clears the list of groups from the Relay Client page and calls **RelayService.CreateEndPointAsync** to create the new endpoint. This method takes the following parameters:

* The ADM client ID and client secret, which the sample stores in the **HawaiiClient** object.
* The name of the application.
* A pointer to a callback function that the Relay service calls when the **CreateEndPointAsync** method has completed. The Relay service returns the endpoint as a parameter to the callback function.

The following is the code for the callback function:

private void OnCompleteCreateEndPoint(EndpointResult result)

{

Debug.Assert(result != null, "result is null");

if (result.Status == Status.Success)

{

// Set the newly created endpoint as my active endpoint.

this.RelayContext.Endpoint = result.EndPoint;

this.Dispatcher.BeginInvoke(

delegate

{

this.MyEndpointLabel.Text = "My Endpoint : " +

result.EndPoint.RegistrationId;

});

}

else

{

// Display a message box if an error occurred.

this.DisplayMessage("Error creating a new endpoint.", "Error");

}

}

If endpoint creation succeeded, the **EndpointResult.Endpoint** member contains the new endpoint. Each **Endpoint** has three properties:

* **Groups**, which lists the groups to which the endpoint belongs.
* **RegistrationId**, which uniquely identifies the endpoint.
* **SecretKey**, which is returned only by **CreateEndPointAsync** and must be passed to the service to change an endpoint.

The sample saves the **Endpoint** value in its local RelayContext object and updates the user interface (UI) to show the new value. In Silverlight, you can access UI elements only on the main UI thread. By using the **Dispatcher.BeginInvoke** method, the sample ensures that the delegate runs on the current thread.

## Group Creation

When the user taps **Manage Groups**, the sample navigates to the Groups page and calls the associated OnNavigatedTo method. This code appears in the GroupsPage.xaml.cs file. To create a group, the sample calls **RelayService.DeleteEndPointAsync**, as follows:

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

{

RelayService.CreateGroupAsync(HawaiiClient.AdmClientId,

HawaiiClient.AdmClientSecret,

this.OnCompleteCreateGroup);

}

The **CreateGroupAsync** method has the following parameters:

* The ADM client ID and client secret, which the sample stores in the **HawaiiClient** object.
* A pointer to a callback function that the Relay service calls when the **CreateGroupAsync** method has completed. The Relay service returns the registration ID for the group as a parameter to the callback function.

The following shows the code for the callback function:

private void OnCompleteCreateGroup(GroupResult result)

{

Debug.Assert(result != null, "result is null");

if (result.Status == Status.Success)

{

// Add the new group into available group data object.

this.RelayContext.Groups.Add(result.Group);

// Refresh the list box to reflect the changes.

this.RefreshGroupsListBox(result.Group.RegistrationId);

}

else

{

this.DisplayMessage("Creating a new group failed.", "Error");

}

}

The Relay service returns the group registration ID in the **GroupResult.Group.RegistrationId** member. The sample then calls local methods that update the UI with the new group information.

The **GroupsPage** also supports the ability to join, leave, or delete a group. The application calls the following methods to perform these actions:

* **RelayService.JoinGroupAsync**
* **RelayService.LeaveGroupAsync**
* **RelayService.DeleteGroupAsync**

Like **CreateGroupAsync**, these methods operate asynchronously, and the Relay service calls the specified application’s callback function when the operation is complete.

The sample calls the overloaded versions of these methods, each of which takes a caller-defined *stateObject* as the final parameter. In each call, the sample passes the local group variable as the *stateObject*. For example:

RelayService.DeleteGroupAsync(

HawaiiClient.AdmClientId,

HawaiiClient.AdmClientSecret,

group,

this.OnCompleteDeleteGroup,

group);

The **GroupResult** that the Relay service passes to the completion callback contains this object in the **StateObject** property. The completion callback can access it as in the following example:

private void OnCompleteDeleteGroup(GroupResult result)

{

Debug.Assert(result != null, "result is null");

if (result.Status == Status.Success)

{

// Get the group object that we passed through DeleteGroupAsync.

Group group = (Group)result.StateObject;

Debug.Assert(group != null, "StateObject is null");

// Remove the group from available groups object.

this.RelayContext.Groups.Remove(group.RegistrationId);

// Remove the group from groups container of MyEndpoint.

if (this.RelayContext.Endpoint.Groups.Exists(group.RegistrationId))

{

this.RelayContext.Endpoint.Groups.Remove(group.RegistrationId);

}

// We just updated the data. Refresh it

this.RefreshGroupsListBox();

this.RefreshEndpointGroupsListBox();

}

else

{

this.DisplayMessage("Deleting a group failed.", "Error");

}

}

## Sending Messages

The Relay service enables an application to send a message to one or more endpoints or to any groups that the sending endpoint has joined. An endpoint must be a member of a group to send a message to the group. The SendMessagesPage.xaml.cs file contains the code that implements the Messaging page of the sample application. the user can

To send a message to more than one endpoint or group, the application must pass the target registration IDs in a comma-delimited string. The following code snippet shows how the sample creates such a string:

List<string> groupIds = new List<string>();

foreach (Group group in this.RelayContext.Endpoint.Groups)

{

groupIds.Add(group.RegistrationId);

}

string recipientIds = string.Join(",", groupIds.ToArray());

// Send the message.

this.SendMessage(this.RelayContext.Endpoint, recipientIds);

The internal SendMessage method formats the message as a Unicode byte string and calls the **RelayService.SendMessageAsync** method to send the message, as follows:

private void SendMessage(Endpoint from, string recipients)

{

// Read the message from message content textbox control

string messageContent = this.MessageContentTextBox.Text;

if (string.IsNullOrEmpty(messageContent))

{

this.DisplayMessage("Message content is empty", "Error");

return;

}

byte[] message = Encoding.Unicode.GetBytes(messageContent);

RelayService.SendMessageAsync(

HawaiiClient.AdmClientId,

HawaiiClient.AdmClientSecret,

from,

recipients,

message,

this.OnCompleteSendMessage);

}

The completion callback is straightforward. It simply checks status and indicates whether the operation succeeded:

private void OnCompleteSendMessage(MessagingResult result)

{

Debug.Assert(result != null, "result is null");

if (result.Status == Status.Success)

{

this.DisplayMessage("Sending message to group(s) succeeded.", "Info");

}

else

{

this.DisplayMessage("Sending message to group(s) failed.", "Error");

}

}

## Receiving Messages

When the user taps **Receive Messages** on the Relay Client page, the sample application navigates to the Messages page and checks for new messages. This code appears in the ReceiveMessagesPage.xaml.cs file. The following shows the **OnNavigatedTo** method for this page:

protected override void OnNavigatedTo(NavigationEventArgs nea)

{

if (this.RelayContext.Endpoint != null)

{

if (this.RelayContext.Endpoint == null)

{

MessageBox.Show("No endpoint is created.", "Error",

MessageBoxButton.OK);

}

else

{

this.Dispatcher.BeginInvoke(

new DisplayMessagesDelegate(this.DisplayMessages));

RelayService.ReceiveMessagesAsync(

HawaiiClient.AdmClientId,

HawaiiClient.AdmClientSecret,

this.RelayContext.Endpoint,

string.Empty,

this.ReceiveMessages);

}

}

base.OnNavigatedTo(nea);

}

The call to **RelayService.ReceiveMessagesAsync** initiates message retrieval. This method takes the following arguments:

* The ADM client ID and client secret, which the sample stores in the **HawaiiClient** object.
* The caller’s endpoint, which the sample stores in the local RelayContext object.
* A string that contains the registration IDs from which to receive messages. The sample passes an empty string to get all messages that are waiting for it.
* A completion callback that receives the results of the operation.

The Relay service calls the ReceiveMessages completion callback with a **MessagingResult** object that contains the status of the operation in the **Status** member and the list of messages in the **Messages** member. The following shows the code for this method:

private void ReceiveMessages(MessagingResult result)

{

Debug.Assert(result != null, "result is null");

if (result.Status == Status.Success)

{

if (result.Messages == null || result.Messages.Count == 0)

{

this.DisplayMessage("No new messages available.", "Info");

}

else

{

foreach (Message message in result.Messages)

{

this.RelayContext.Messages.Add(message);

}

this.Dispatcher.BeginInvoke(new

DisplayMessagesDelegate(this.DisplayMessages));

}

}

else

{

this.DisplayMessage("Error receiving messages for active endpoint.",

"Error");

}

}

## Storing Relay Data

When the user closes the application, the sample saves the current endpoint, groups, and messages in isolated storage by calling methods in the **RelayStorage** class. The following code appears in the RelayContext.cs source file:

public void SaveToStorage()

{

try

{

RelayStorage.SaveEndpoint(this.Endpoint);

RelayStorage.SaveGroups(this.Groups);

RelayStorage.SaveMessages(this.Messages);

}

catch (Exception)

{

// Let's not crash the client.

}

}

The **SaveEndpoint**, **SaveGroups**, and **SaveMessages** methods each take a single parameter:

* **SaveEndpoint** takes an **Endpoint** object.
* **SaveGroups** takes a **Groups** object, which is a list of a groups.
* **SaveMessages** takes a list of **Messages**.

The application can read stored data in a similar way. The following also appears in RelayContext.cs:

public void ReadFromStorage()

{

try

{

this.Endpoint = RelayStorage.ReadEndpoint();

this.Groups = RelayStorage.ReadGroups();

this.Messages = RelayStorage.ReadMessages();

}

catch (Exception)

{

// Let's not to crash the client.

}

}

# Using the Relay Service in an Application

To use the Relay service in your own application, you must:

* Include the required assemblies to the Visual Studio project,
* Reference the namespace in your source code.
* Set up your authentication credentials.
* Follow the usage guidelines.

## Include the Required Libraries

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

* Microsoft.Hawaii.ClientBase.dll
* Microsoft.Hawaii.Relay.Client.dll

To include the libraries in your application

* Build the Hawaii SDK, as described in “Hawaii Installation Guide.”
* Add references to the following DLLs to your Visual Studio project:
* Microsoft.Hawaii.ClientBase.dll
* Microsoft.Hawaii.Relay.Client.dll

## Reference the Namespace

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

using Microsoft.Hawaii;

using Microsoft.Hawaii.Relay.Client;

## Set Up Your Authentication Credentials

Your application authenticates itself with the Relay service by using an Azure Data Market client ID and client secret. If you do not already have AMD credentials, obtain them as described in “Hawaii Installation Guide.”

The easiest way to use the the ADM credentials in your code is to 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**

## General Guidelines

The following general guidelines apply to use of the Relay service:

* Applications use a **RelayClient** instance to trigger an asynchronous call to the Hawaii Relay service.
* The application must have a registration ID for a valid endpoint before it can send a message. To register an endpoint, an application calls the **CreateEndpointAsync** method. Applications cannot pick their own registration IDs; the Relay service assigns Registration IDs randomly.
* A device or endpoint can have any number of registration IDs.
* An endpoint must be a member of a group to send a message to the group.
* Applications can use the **RelayStorage** class to save and retrieve endpoints, groups, and messages in isolated storage. When an application creates an endpoint or a group, it should store the returned **Endpoint** or **Group** to ensure that the **SecretKey** is available later. The application must pass the **SecretKey** to change the endpoint or group.

# Resources

This section provides links to additional information about the Project Hawaii Relay 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>

Project Hawaii on Facebook

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

MSDN

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>

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

**System.Io.IsolatedStorage**  
<http://msdn.microsoft.com/en-us/library/x7dzh4ws(v=vs.95)>