Rendezvous Service  
Programming Guide

Version 2.1 – September 13, 2012

Abstract

The Project Hawaii Rendezvous service enables an application to associate a friendly name with a registration ID that the Hawaii Relay service has previously assigned to an endpoint. The application can then communicate with the endpoint by using the name instead of the registration ID.

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

Contents

[Introduction 2](#_Toc335306639)

[Prerequisites 2](#_Toc335306640)

[Overview of the Rendezvous Service 2](#_Toc335306641)

[Walkthough: Rendezvous Sample Application 3](#_Toc335306642)

[Registering a Name 4](#_Toc335306643)

[Unregistering a Name 6](#_Toc335306644)

[Associating a Name with a Registration ID 7](#_Toc335306645)

[Disassociating a Name from a Registration ID 8](#_Toc335306646)

[Looking Up a Name 9](#_Toc335306647)

[Using the Rendezvous Service in an Application 11](#_Toc335306648)

[Add Required Assemblies 11](#_Toc335306649)

[Reference the Namespace 11](#_Toc335306650)

[Set Up Your Authentication Credentials 11](#_Toc335306651)

[Resources 11](#_Toc335306652)

**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 Rendezvous service enables an application to associate a friendly name with a registration ID that the Hawaii Relay service has previously assigned to an endpoint. The application can then communicate with the endpoint by using the name instead of the registration ID.

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

# Prerequisites

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

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

For information about installation, build procedures, andcredentials, 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:

* Project Hawaii Relay Service
* Windows Communication Foundation (WCF)
* Microsoft Silverlight®
* Windows Phone 7 SDK

For more information about the Relay service, see “Relay Service Programming Guide,” which is installed with the Project Hawaii SDK.

# Overview of the Rendezvous Service

The Rendezvous service enables an application to associate a friendly name with a registration Id that the Relay service assigns, similar to the way that the domain name system (DNS) associates a machine name with an IP address. The Rendezvous service provides the following features:

* Register a friendly name.
* Unregister a friendly name.
* Associate a registration ID with an existing friendly name.
* Disassociate a registration ID from a friendly name.
* Look up the registration ID associated with a friendly name.
* Store a name and registration ID with the secret key for later retrieval.

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

* Source\ServiceClients\Rendezvous

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

|  |  |
| --- | --- |
| Class | Description |
| **NameRegistration** | Represents a communication endpoint or group that is associated with a friendly name. |
| **NameRegistrationResult** | Contains the results of a Rendezvous service method. |
| **RendezvousService** | Helper class that provides access to the Rendezvous service. |
| **RendezvousStorage** | Helper class to store a name and its secret key in the mobile device's isolated storage. |

# Walkthough: Rendezvous Sample Application

The Project Hawaii SDK includes the RendezvousSample application, which demonstrates the features of the Rendezvous service. The application is installed in the Samples\Rendezvous subfolder of the Project Hawaii SDK installation directory.

The RendezvousSample application implements a simple interface with which you can register and unregister names, look up names, and associate names with Registration IDs that the Relay service assigned. This brief walkthrough introduces the components of the sample and shows you how the sample calls methods in the **RendezvousClientService** class.

To compile and run RendezvousSample

1. In Visual Studio, open RendezvousSampleApp.csproj.

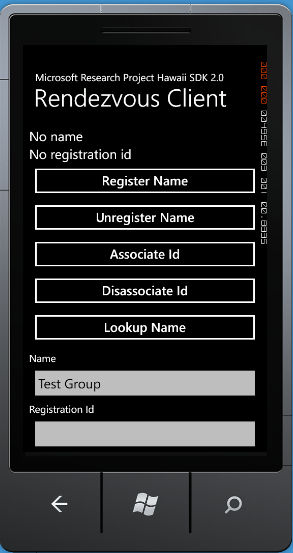
2. Open the HawaiiClient.cs file and set the **AdmClientId** and **AdmClientSecret** strings to your Azure Data Market (ADM) credentials.

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 RendezvousSample

* To register a friendly name, type the name in the Name box and tap **Register Name**.
* To associate a name with a Registration ID, type the ID in the Registration id box and tap **Associate Id**. The sample program accepts an invented ID.
* To break the association between the name and the ID, tap **Disassociate Id**.
* To look up a name, type the name in the Name box and tap **Lookup Name**.

## Registering a Name

When the user taps **RegisterName**, the sample application reads the string from the Name text box and passes it to the **RendezvousService.RegisterNameAsync** method. The following snippet from the MainPage.xaml.cs file shows the code:

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

{

string name = this.NameTextBox.Text;

if (!string.IsNullOrEmpty(name))

{

RendezvousService.RegisterNameAsync(

HawaiiClient.AdmClientId, HawaiiClient.AdmClientSecret,

name, this.OnCompleteRegisterName, null);

}

else

{

this.DisplayMessage("Invalid name found. Enter a name and try again.",

"Error");

}

}

The **RegisterNameAsync** method 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 name to register, which is a string that the sample reads from **NameTextBox**. The maximum length for the name is 50 Unicode characters; **null** characters are not valid.
* A pointer to a callback function that the Rendezvous service calls when the method has completed.
* An application-defined object, which is **null** in this sample.

The Rendezvous service registers the name and returns a secret key, which the application must pass back to the service to deregister the name or associate a Registration ID with it. In the OnCompleteRegisterName callback function, the sample saves the key by using the **RendezvousStorage** class, as the following code shows:

private void OnCompleteRegisterName(NameRegistrationResult result)

{

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

if (result.Status == Status.Success)

{

this.Dispatcher.BeginInvoke(

delegate

{

this.NameRegistrationContext = new

NameRegistrationContext(result.NameRegistration);

// Store the secret key in isolated storage.

RendezvousStorage.SetSecretKey(result.NameRegistration.Name,

result.NameRegistration.SecretKey);

});

}

else

{

this.DisplayMessage("Name registration failed.", "Error");

}

}

The Rendezvous service passes a **NameRegistrationResult** object to the callback function. If the call succeeds, the sample creates a new instance of the NameRegistrationContext class, which is a container for the name, registration ID, and secret key. The code that implements that class appears in the NameRegistrationContext.cs file. The sample then calls the **RendezvousStorage.SetSecretKey** method to store the registered name and the secret key in isolated storage. **SetSecretKey** has two parameters:

* A string that represents the name.
* A string that represents the secret key.

The application can later use the name to retrieve the key by calling **RendezvousStorage.GetSecretKey**.

## Unregistering a Name

When the user taps **UnregisterName**, the sample application calls the Rendezvous service to remove the name from its registry. To unregister a name, an application must pass the secret key that was returned by the registration process. The following snippet from the MainPage.xaml.cs file shows the code:

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

{

string name = this.NameTextBox.Text;

if (!string.IsNullOrEmpty(name))

{

string secretKey = RendezvousStorage.GetSecretKey(name);

if (string.IsNullOrEmpty(secretKey))

{

. . . //Code omitted

}

NameRegistration nameRegistration = new NameRegistration()

{

Name = name,

SecretKey = secretKey

};

RendezvousService.UnregisterNameAsync(

HawaiiClient.AdmClientId, HawaiiClient.AdmClientSecret,

nameRegistration,

this.OnCompleteUnregisterName, null);

}

else

{

this.DisplayMessage("Invalid name found. Enter a name and try again.",

"Error");

}

}

In the example, the application passes the name to **RendezvousStorage.GetSecretKey**, which returns the secret key. It can then call **UnregisterNameAsync** to unregister the name. **UnregisterNameAsync** 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.
* An instance of the **NameRegistration** class, which contains the name and the secret key.
* A pointer to a callback function that the Rendezvous service calls when the method has completed.
* An application-defined object, which is **null** in this sample.

The callback function simply checks the returned status and clears the local NameRegistrationContext object.

## Associating a Name with a Registration ID

When the user taps **Associate ID**, the sample reads the contents of the **Name** and **Registration id** text boxes, retrieves the secret key that is associated with the name, and calls **RendezvousService.AssociateIdAsync** to associate the name with the ID. The following code is from MainPage.xaml.cs:

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

{

string name = this.NameTextBox.Text;

string registrationId = this.RegistrationIdTextBox.Text;

if (!string.IsNullOrEmpty(name) ||

!string.IsNullOrEmpty(registrationId))

{

string secretKey = RendezvousStorage.GetSecretKey(name);

if (string.IsNullOrEmpty(secretKey))

{

this.DisplayMessage("You are not the owner of this name or the name ownership details are lost. You can't perform this operation.", "Error");

return;

}

NameRegistration nameRegistration = new NameRegistration()

{

Name = name,

Id = registrationId,

SecretKey = secretKey

};

RendezvousService.AssociateIdAsync(HawaiiClient.AdmClientId,

HawaiiClient.AdmClientSecret,

nameRegistration, this.OnCompleteAssociateId, nameRegistration);

}

else

{

this.DisplayMessage("Invalid registration name/id found. Enter a name and registration id and try again.", "Error");

}

}

The sample passes the name to **RendezvousStorage.GetSecretKey** to retrieve the key from isolated storage. If the caller did not create the name, the call to **GetSecretKey** fails.

The sample then populates a **NameRegistration** object with the name, registration ID, and key, and passes this object to the **RendezvousService.AssociateIdAsync** method. This method 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.
* An instance of the **NameRegistration** class, which contains the name, the ID, and the secret key.
* A pointer to a callback function that the Rendezvous service calls when the method has completed.
* An application-defined object, which the service passes to the completion callback.

If **AssociateIdAsync** succeeds, the OnCompleteAssociateId callback function creates and populates an internal NameRegistrationContext object, as follows:

private void OnCompleteAssociateId(NameRegistrationResult result)

{

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

if (result.Status == Status.Success)

{

NameRegistration nameRegistration =

(NameRegistration)result.StateObject;

this.Dispatcher.BeginInvoke(

delegate

{

this.NameRegistrationContext =

new NameRegistrationContext(nameRegistration);

});

}

else

{

. . . //code omitted

}

}

## Disassociating a Name from a Registration ID

When the user taps **Disassociate ID**, the sample application responds in much the same way as to create the associate. It reads the name and ID, looks up the secret key, and populates a **NameRegistration** object. It then calls **RendezvousService.DisassociateIdAsync** to break the association. The parameters to **DisassociateIdAsync** are the same as those to **AssociateIdAsync**. The following shows the code:

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

{

string name = this.NameTextBox.Text;

string registrationId = this.RegistrationIdTextBox.Text;

if (!string.IsNullOrEmpty(name) ||

!string.IsNullOrEmpty(registrationId))

{

string secretKey = RendezvousStorage.GetSecretKey(name);

if (string.IsNullOrEmpty(secretKey))

{

this.DisplayMessage("You are not the owner of this name or the name ownership details are lost. You can't perform this operation.", "Error");

return;

}

NameRegistration nameRegistration = new NameRegistration()

{

Name = name,

Id = registrationId,

SecretKey = secretKey

};

RendezvousService.DisassociateIdAsync(

HawaiiClient.AdmClientId, HawaiiClient.AdmClientSecret,

nameRegistration,

this.OnCompleteDisassociateId, nameRegistration);

}

else

{

this.DisplayMessage("Invalid registration name/id found. Enter a name and registration id and try again.", "Error");

}

}

The callback function simply checks status and saves the updated information in its local context object, as the following code shows:

private void OnCompleteDisassociateId(NameRegistrationResult result)

{

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

if (result.Status == Status.Success)

{

NameRegistration nameRegistration =

(NameRegistration)result.StateObject;

nameRegistration.Id = string.Empty;

this.Dispatcher.BeginInvoke(

delegate

{

this.NameRegistrationContext =

new NameRegistrationContext(nameRegistration);

});

}

else

{

this.DisplayMessage("Disassociating a registration id from a name failed.", "Error");

}

}

## Looking Up a Name

When the user taps **Lookup Name**, the sample reads the name from the **Name** text box and calls the **RendezvousService.LookupNameAsync** method as follows:

private void LookupNameButton\_Click(object sender, RoutedEventArgs ea)

{

string name = this.NameTextBox.Text;

if (!string.IsNullOrEmpty(name))

{

RendezvousService.LookupNameAsync(HawaiiClient.AdmClientId,

HawaiiClient.AdmClientSecret,

name, this.OnCompleteLookupName);

}

else

{

this.DisplayMessage("Invalid name found. Enter a name and try again.",

"Error");

}

}

**LookupNameAsync** takes four 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 string that contains the name to search for.
* A pointer to a callback function that the Rendezvous service calls when the method has completed.

**LookupNameAsync** succeeds if the name is registered with the Rendezvous service, regardless of the application that registered it. However, only an application that has the secret key can call methods in the RendezvousService to unregister the name or change its association with an ID.

The **OnCompleteLookupName** callback simply checks status and updates the UI with the name.

private void OnCompleteLookupName(NameRegistrationResult result)

{

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

if (result.Status == Status.Success)

{

this.Dispatcher.BeginInvoke(

delegate

{

this.NameRegistrationContext =

new NameRegistrationContext(result.NameRegistration);

});

}

else

{

this.DisplayMessage("Name lookup failed.", "Error");

}

}

# Using the Rendezvous Service in an Application

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

* Add required assemblies to the Visual Studio project.
* Reference the namespace in your source code.
* Set up your authentication credentials.

## Add Required Assemblies

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

* Microsoft.Hawaii.ClientBase.dll
* Microsoft.Hawaii.Rendezvous.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.Rendezvous.Client.dll

## Reference the Namespace

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

using Microsoft.Hawaii;

using Microsoft.Hawaii.Rendezvous.Client;

## Set Up Your Authentication Credentials

Your application authenticates itself with the Rendezvous 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 Project Hawaii 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

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)>