Software Setup Instructions –“WCF NetMessagingBinding: In Azure Cloud and On-Prem” Blog Example Code

George Stevens

|  |  |  |
| --- | --- | --- |
| **version** | **date** | **author** |
| 0.1 | 5/7/2015 | George Stevens |
| 0.2 | 5/10/15 | George Stevens |
|  |  |  |
|  |  |  |

# Introduction

This document explains how to setup the WcfNQueueSMEx Visual Studio solution which demonstrates the WCF NetMessagingBinding for a WCF service that can be hosted in both Azure and on-premises. You need Visual Studio 2013 or later, a connection to the internet, and an Azure account.

There are 2 phases to this setup, each of them having step by step instructions:

1. The **ServiceModelEx Setup Phase** -- Setup the WcfNQueueSMEx solution to include ServiceModelEx, which you’ll have to download from [IDesign](http://www.idesign.net) via the link below in the detailed setup instructions.
2. The **Azure Setup Phase** -- Setup the WcfNQueueSMEx solution for Azure, including creating the Azure items you’ll need if they do not already exist.

From this point onward I assume that you have downloaded the WcfNQueueSMEx solution, and placed an unzipped version of it you’ll be working with in a development directory of your choice.

# The ServiceModelEx Setup Phase

WcfNQueueSMEx solution has several “using ServiceModelEx” statements in the code that require the projects containing these statements have a reference to the ServiceModelEx dll. Due to copyright restrictions on republishing, please download the ServiceModelEx source and dll from the [IDesign](http://www.idesign.net) site, and make sure to read the usage license and agree with it. Here is how to download ServiceModelEx and then manually add it to the WcfNQueueSMEx solution:

1. **Download** ServiceModelEx source from this link: <http://www.idesign.net/Downloads/GetDownload/1887>
2. Once received, **unzip** the downloaded file into a temporary directory. Notice that in the unzipped top level directory there are several ServiceModelEx subdirectories. The one you want is named “ServiceModelEx” with no suffixes. Disregard the other ServiceModelEx directories having suffixes for the purposes of this example.
3. Copy the above non-suffixed ServiceModelEx subdirectory into the WcfNQueueSMEx top level WcfNQueueSMEx solution directory so that the top level solution directory has a subdirectory called ServiceModelEx, in addition to the other directories for projects in WcfNQueueSMEx.

1. **Add the ServiceModelEx project** (in the ServiceModelEx directory) to the WcfNQueueSMEx solution.
   1. Right click WcfNQueueSMEx solution.
   2. Add
   3. Existing Project.
2. **Build** the ServiceModelEx project. You will get lots of errors, like ServiceBus.dll missing.
3. **Update via Nuget** – The proper Microsoft ServiceBus dll needs to be obtained. Note that you need to be connected to the internet so that NuGet will download the required packages.
   1. Right click on the WcfNQueueSMEx solution and select Manage Nuget Packages for solution.
   2. Open the Installed Packages list
   3. Select the Microsoft Azure Service Bus item and click the Manage button.
   4. Make sure ServiceModelEx is checked. Then click on OK. Nuget will install some packages.
   5. Close the Package Manager
4. **Build** the ServiceModelEx project again. The build should have no errors now.
5. **Build** the entire solution. There will be errors. Do the following to get rid of the errors.
6. **Make references to ServiceModelEx.dll**. For each of the following projects, add the ServiceModelEx reference. ServiceModelEx is referenced in the following projects:
   1. DataIngestionCloudWorker
   2. DataIngestionServiceHost
   3. DataSourceSimulatorClien
   4. Shared
7. **Build** the entire solution. This should be a clean build with no errors.

Even though ServiceModelEx is present, the solution will not run yet since it is not setup for Azure. Since the solution uses the Azure Service Bus, it must be setup to interact with Azure.

# The Azure Setup Phase

First, here is an overview of what preparation you have to do to get the WcfNQueueSMEx solution configured so it can use an Azure Service Bus Queue. The details of each step are presented subsequently.

In summary, WcfNQueueSMEx requires:

1. A Service Bus Namespace
2. A queue in the above Service Bus Namespace
3. A Cloud Service

The main steps in the Azure setup for WcfNQueueSMEx are as follows. The detailed steps presented subsequently:

1. Get an Azure account if you do not have one. **Login** to your Azure account.
2. **Setup a Service Bus Namespace** named AzExploreSbNs – This namespace will be used by the following solution items to access the shared Service Bus Queue:
   1. The on-prem client (DataSourceSimulatorClient) that enqueues data items.
   2. The on-prem WCF Service Host (DataIngestionServiceHost) and its hosted WCF service (DataIngestionManager) that removes items from the shares queue.
   3. The Azure in-cloud service (DataIngestionCloudWorker) that also removes items from the queue.
3. Within the new Service Bus Namespace, **create the ingestionqueue** – This is the Service Bus Queue shared by the DataSourceSimulatorClient (enqueues messages) and the WCF service (dequeues messages).
4. **Setup a Cloud Service** called AzExploreCloudSvc – It will contain the WorkerRole that acts as the WCF service host in the cloud. In the on-prem version the service host is a Console App.
5. **Modify the shared access signature** (SAS) in the app.config files. The SAS allows access to anything in the above Service Bus Namespace, including the ingestionqueue. The ingestionqueue does NOT have its own SAS.
   1. WARNING -- Do not do things this way in a production app, and do not put the SAS keys in app.config files. Both are big security holes. Its only OK in a demo.
6. **Run the app** in on-prem mode!

If you already have an existing Service Bus Namespace, a queue, and a Cloud Service then you can use these items (and their names) instead of the below items. However, you will have to figure out on your own how to modify the solution to support your existing Azure items, and thus do more editing of the WcfNQueueSMEx solution. **I suggest that you just follow the below instructions which minimize the amount of changes you have to make to the solution since it is set up for specific names for the Service Bus Namespace, queue, and Cloud Service**.

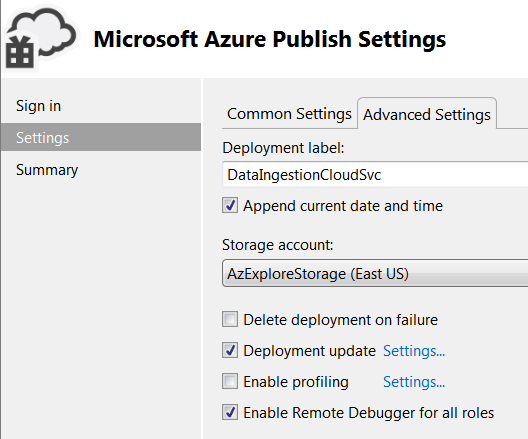
Step by step Setup:

1. **Dealing with the Visual Studio solution**.
   1. Note that running the WcfNQueueSMEx solution requires you **run Visual Studio as administrator** AND that you **are connected to the internet**. This is required to access the Service Bus.
   2. To make any changes to the solution it is best to **run Visual Studio as administrator**.
2. **Login to the Azure Portal** – This is the current portal as of May 2015, not the new portal.
3. **Setup the Service Bus Namespace** that will hold the shared queue we use.
   1. In Portal main window click on Service Bus
   2. Click on Create in lower center. This brings up “Create a namespace” dialog. Fill out as follows
      1. Set namespace name to: AzExploreSbNs
      2. Set the Region to the one nearest you from the items in the dropdown.
      3. Set Type to: Messaging
      4. Set Messaging Tier to: Standard
      5. Click the check mark in the lower right to save your info.
   3. Wait a few seconds while it is activating.
4. Click on the AzExploreSbNs. You will see the “Access connection information” screen. Here is where you will get the SAS name and connection string information you will need later when preparing the app.config files in the several projects in the solution.
   * 1. We’ll be back for this later.
5. Click on the X in the upper right to close this connection info screen.
6. Done with creating the SB namespace.
7. **Setup the queue** within the new Service Bus Namespace, AzExploreSbNs.
8. In the service bus main screen, displaying all service bus namespaces, click on the AzExploreSbNs item.
9. Then, from the top horizontal list of items a service bus namespace can have, click “Queues”.
10. Click “Create new queue”.
11. Then click “Quick create”
12. Set the Queue Name to: ingestionqueue
13. Leave the other 2 fields, Region and Namespace as is.
14. Click “Create new queue”.
15. You will see ingestionqueue has been added to the list of queues in the SB namespace.
16. Done with creating the queue.
17. Click the “Back” arrow in the upper left of the display so as to navigate to the top level portal main screen that displays all the different kinds to goodies offered by Azure.
18. **Setup the Cloud Service** that contains the WorkerRole which hosts the WCF service.
19. From the left hand list of items offered by Azure, click on the Cloud services item.
20. In the lower left hand corner of the Cloud Services main page, click the “+ New” item to add a new Cloud Service to hold the WorkerRole.
21. The Compute and the Cloud Service items should be highlighted. If not, select them.
    1. Click Quick Create.
    2. Set the URL field to: AzExploreCloudSvc
    3. Set the Region to the one nearest you from the items in the dropdown.
    4. Click the check mark in the lower right to save your info.
22. Done creating the Cloud Service
23. Modify the WcfNQueueSMEx solution so it will run.
24. First open the solution in Visual Studio running it as administrator.
25. Copy the Service Bus Namespace Shared Access Signature key value from the Azure Portal into the 3 app.config files.
    1. In the Azure Portal, navigate to the Service Bus namespaces, i.e. click Service Bus in the left hand list on the main page.
       1. Note we are NOT using the queue’s SharedAccessSignature. Instead we are using that of the Service Bus Namespace.
    2. In the Service Bus main page click on the AzExploreSbNs item to open it.
    3. At the bottom of the just opened azexploresbns page, click on the key image labeled Connection Information.
       1. Note in the SAS section there is one name, RootManageSharedAccessKey.
       2. Beside this there is a connection string. Copy the connection string and past it into Notepad.
       3. Next in Notepad, in the part of the connection string containing SharedAccessKey=Kos+RqH2ap42sAtTdvftyqY2PftyuUjVH7ep8cGXK9/I= copy everything to the right of “SharedAccessKey=” and paste it into a new line in Notepad. Note that the characters that you copy will NOT the same as those above!!!
          1. These characters are the key and it will definitely be different from what you see above since your key is unique. That key value was mine, but it is changed now.
       4. Copy the key.
    4. Now paste your key into the “key” field of the sharedAccessSignature element of the tokenProvider in each of the 3 app.config files
       1. DataIngestionCloudWorker app.config
       2. DataIngestionServiceHost app.config
       3. DataSourceSimulatorClient app.config.
       4. The tokenProvider element contains something like the following. Just copy your key over that value.

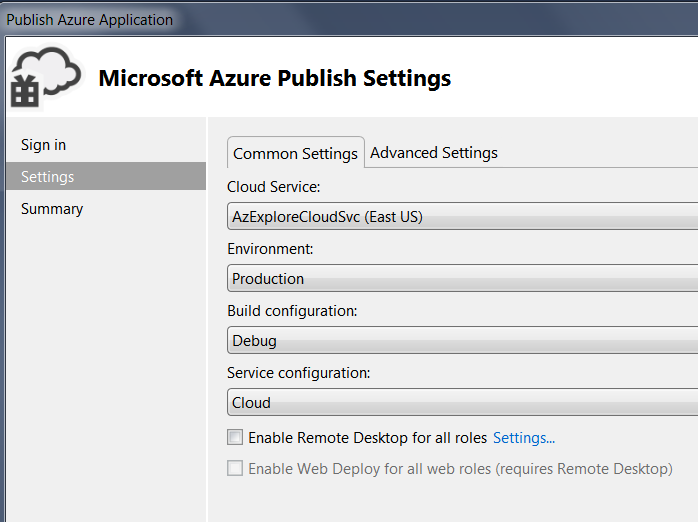
<sharedAccessSignature keyName="RootManageSharedAccessKey"

key="XvfnIzGBYXDgtYwyUOYthz0oyWpKbnyqPtfd3rN9e45="/>

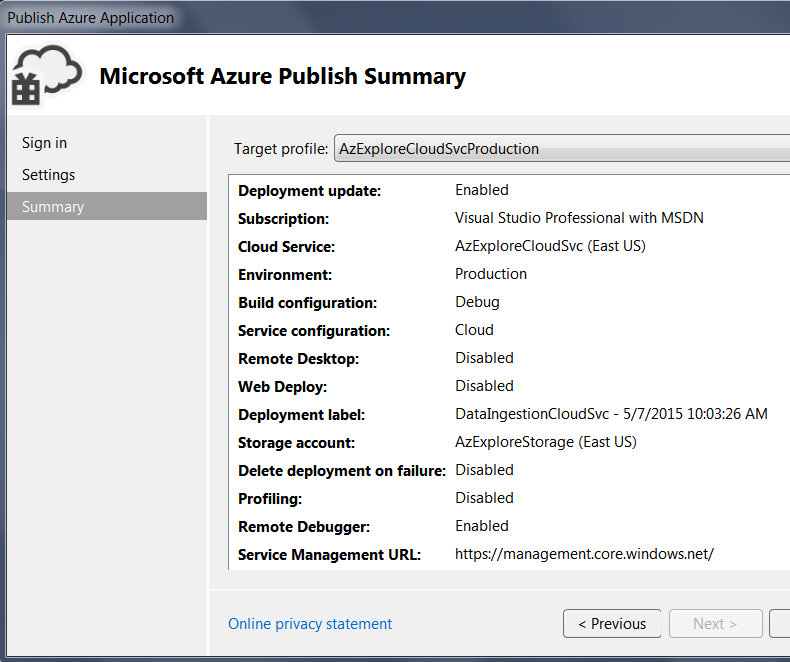
1. Now test it to see if it works, being sure to run Visual Studio as administrator and be connected to the internet.
   1. First, Rebuild the solution. Note that you need to be hooked up to the internet so that NuGet will get assemblies.
   2. Next, run the DataSourceSimulatorClient via Debug, Start new instance.
      1. When the Console Window comes up press Enter to send a message to the queue.
      2. There will usually be a 10 second delay, or so, after which the message sent will be displayed.
         1. If this successfully worked, then you can run the DataIngestionServiceHost and watch it empty the queue.
      3. If there is an error, you will see the exception on the Console Window within 30 seconds or so.
         1. In this case the first thing to do is to make sure you accurately copied and pasted the correct Service Bus Namespace key.
            1. Note we are NOT using the queue’s SharedAccessSignature. Instead we are using that of the Service Bus Namespace. In production situations it is more secure to use the queues Shared Access Signature.
2. If it works try enqueueing some items without the DataIngestionServiceHost running. Then look at the queue in the Azure Portal and you’ll see the queue count is non-zero. Then launch the DataIngestionServiceHost and watch it initially dequeue all the enqueued items as it starts.
   1. Try exploring the behavor of these 2 console apps running at the same time.
   2. Open the server explorer window. Right click on the HDInsight node and refresh it (this gets you logged in). Then refresh the Service Bus element. Drill down into it and find the ingestionqueue. Right click on it and select Properties. It will tell you how many items are in the queue, among other things.
3. Now run the WCF service in the DataIngestionCloudSvc locally, in the local Azure emulator.
   1. Right click on DataIngestionCloudSvc and select Debug, Start New Instance
      1. This will NOT show a console app.
      2. Instead open the Visual Studio Output window and you will see the output of the Trace statements embedded in the code of the WorkerRole and WCF Service.
      3. You can also set breakpoints.
   2. With the cloud service running, hosting the WCF Service, start the DataSourceSimulatorClient and enqueuer a few items. Watch the Output window to see the trace of their processing.
4. Once you get to the point where you can do the above, your solution is ready to be deployed to the Cloud.
5. **I am going to leave that to you to figure out how to do**. Please consult the Azure documentation, especially some of the Tutorials that show how to do this. Here is a sketch of my settings that work.
6. To publish and have it automatically run -- Right click on DataIngestionCloudSvc and select Publish
   1. In the Microsoft Azure Publish Summary window press the Previous button.
   2. The MS Azure Publish Settings are displayed. Make sure the following settings are set
      1. Cloud Service: AzExploreCloudSvc
      2. Environment: Production
      3. Build Configuration: Debug
      4. Service Configuration: Cloud
   3. Press on the Advanced tab
      1. Select the create storage account and fill in the settings as follows
         1. Name: AzExploreStorage
         2. Region or Affinity Group: Whatever is closest to you.
         3. Replication: Locally Redundant is likely good enough for testing, and cheaper.
   4. Here are what the Advanced settings look like that work for me.



* 1. Here are what my Common Settings look like



* 1. Here is what my Publish Summary looks like.



Thus ends the Azure Setup Phase and this setup document. I hope you learn as much as I did!

Revision History

V 0.1, 5/7/15, George Stevens. Initial document.

V 0.2, 5/10/15, George Stevens. Clean up ServiceModelEx setup instructions.