## **Summary**

In the [chapter 1](https://code.msdn.microsoft.com/scenarioDesign-a-web-page-10ceb941), we talked about that the Contoso Garment Factory will use Azure Service Bus for communication between different departments because of the growth of the company business.

We also talked about the advantage of Azure Service Bus in detail and how to manage the Azure Service Bus via the code. For example, how to create and delete the Azure Service Bus Namespace.

In this chapter, we will talk about how to exchange messages between Sales department and Production department [using Azure Service Bus Topic](https://azure.microsoft.com/en-us/documentation/articles/service-bus-dotnet-how-to-use-topics-subscriptions/).

Each message is processed by a single consumer, topics and subscriptions provide a one-to-many form of communication, in a publish/subscribe pattern. Messages are sent to a topic and delivered to one or more associated subscriptions, depending on filter rules that can be set on a per-subscription basis. The subscriptions can use additional filters to restrict the messages that they want to receive.

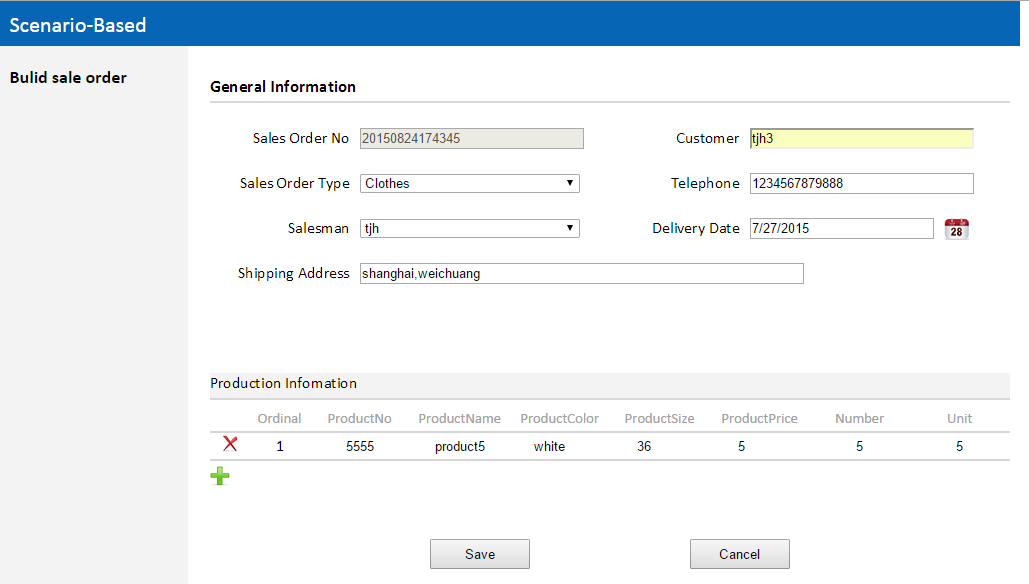
There are two module in this project, every module will be designed to a website. One is sending messages using the Azure service Bus. The other is receiving messages using the Azure service bus.

Links bellow are my project websites hosted in the Azure VM IIS.

<http://tjhvm.cloudapp.net:8080>

<http://tjhvm.cloudapp.net:8081>

The picture bellow is the sending messages module.



## **Objectives in this chapter**

* How to send messages using the Azure Service Bus Topic.
* How to receive messages using the Azure Service Bus Topic.

## **Step 1:** Install [Visual Studio 2013](https://www.visualstudio.com/en-us/downloads/download-visual-studio-vs.aspx), [Microsoft Azure SDK 2.6](https://www.microsoft.com/en-us/download/details.aspx?id=46892) .

Note: it is best to use Internet Explorer 11 when you run project.

## **Step 2:** Design two Azure SQL Database.

There are two Azure Sql databases; one is “Contoso Sales” which is used by Sales department. The other is “Product Database” which is used by Production department. There are two methods to design the Azure SQL Database.

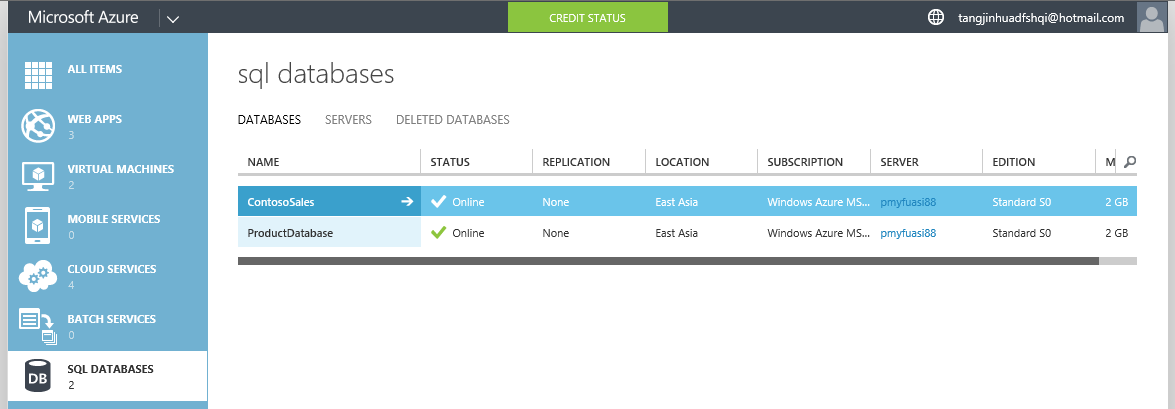
* Import a saved database from your Blob storage account.

1. Upload the database backup under the “Database” folder of the solution to the Azure Blob Container.
2. Open the <https://manage.windowsazure.com> .
3. Click the “New” on the bottom and then select the” DATA SERVICES “and then select “SQL DATABASE “and then click the “IMPORT” to import a database.

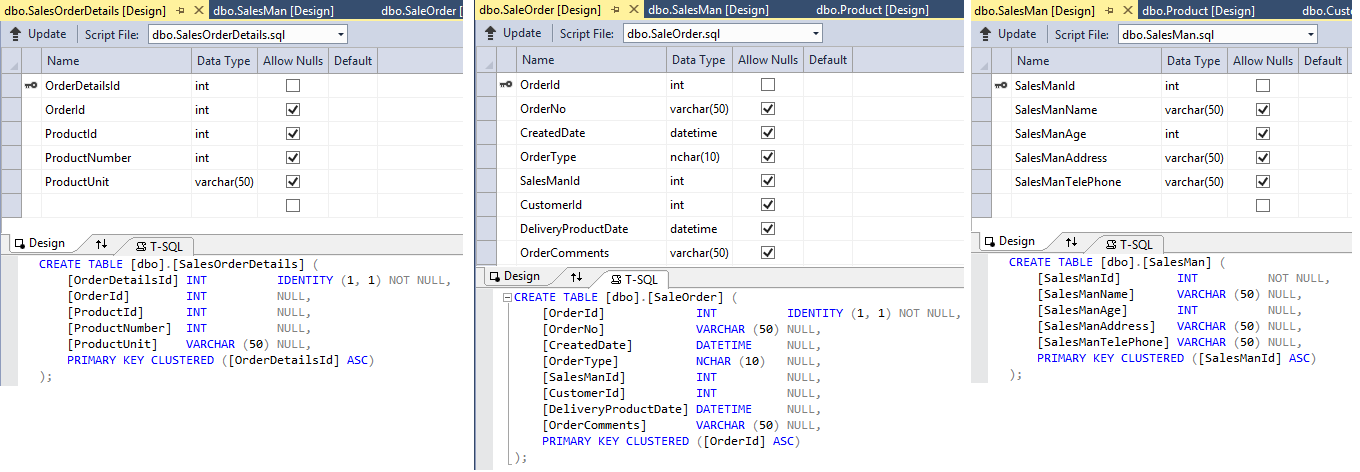
* Create a new database.

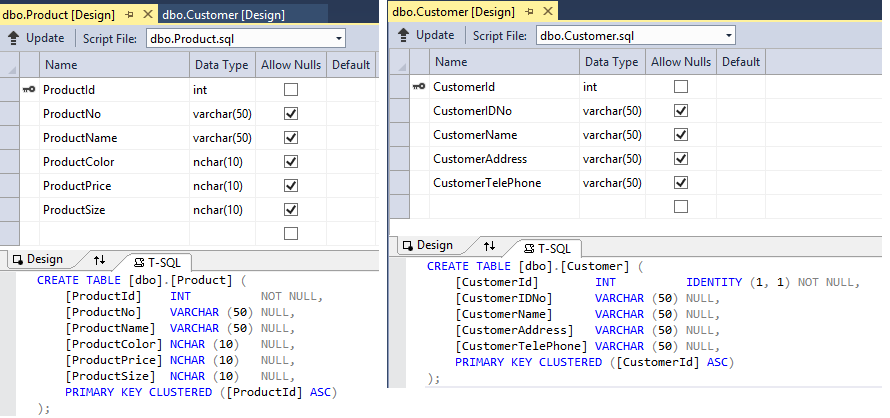
1. Create an Azure Sql database.

* Open the <https://manage.windowsazure.com> .
* Click the “New” on the bottom and then select the” DATA SERVICES “and then select “SQL DATABASE “and then click the “QUICK CREATE” to create the new database.
* Click the “SQL DATABASE” on the left sidebar to check the database created.

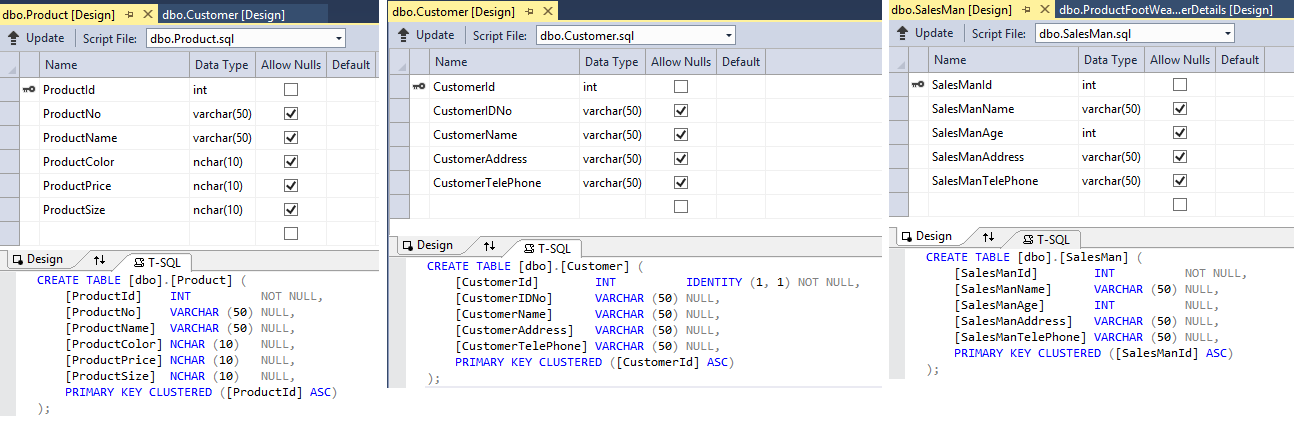


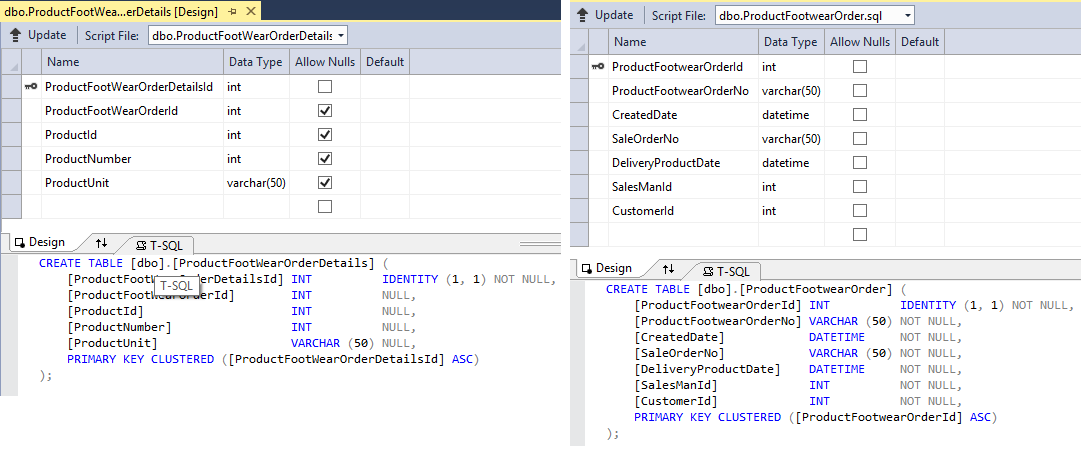
1. Create a new table of the database created.
2. Open the Visual Studio 2013 and then open the Server Explorer.
3. Select the Azure and then select the “ContosoSales” database created.
4. Right click on database selected and then select “Open in SQL Server Object Explorer”.
5. In the SQL Server Object Explorer, select the Tables of the database selected and right click on it and then select the “Add the New Table to create a new table. There are five tables in the “ContosoSales” database.

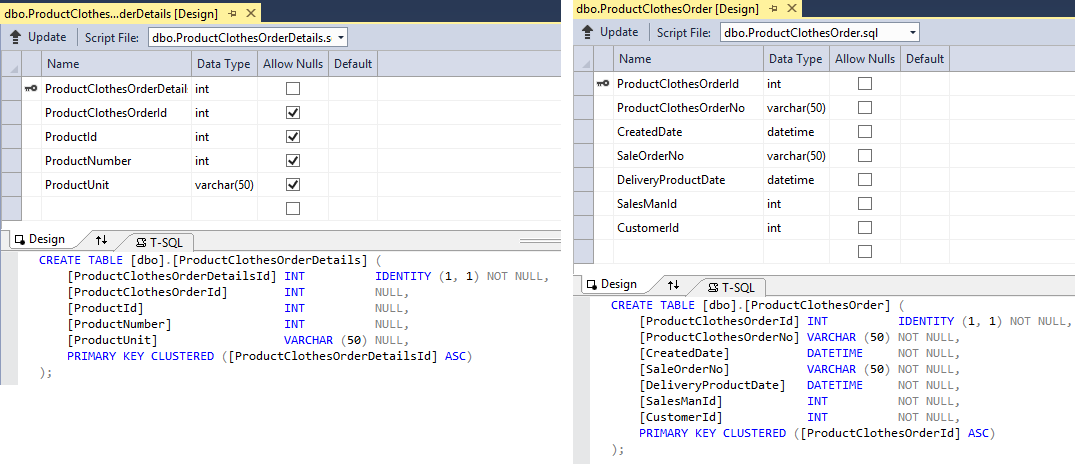




1. Select the “ProductDatabase” database. There are seven tables in the this database.







## **Step 3:** Open the” CSAzureSalesSendMessagesViaTopic” project with Visual Studio 2013.

* Open the Server Explorer and select the Azure Service Bus created in the [chapter 1](https://code.msdn.microsoft.com/scenarioDesign-a-web-page-10ceb941).
* Get the Azure Service Bus Topic name created in the [chapter 1](https://code.msdn.microsoft.com/scenarioDesign-a-web-page-10ceb941).
* Open the “DALClass”file and then replace the “StrTopic” value with the got.

-Code block start-

--C# code snippet start--

public static string StrTopic

{

get

{

strTopic = "TopicName";

return DALClass.strTopic;

}

set { DALClass.strTopic = value; }

}

--C# code snippet end--

-Code block end-

* Right click the Azure Service Bus selected and then select the “Properties” and get the property of ConnectionString.
* Open the “DALClass” file and then replace the “strServiceBusConnection” value with the got.

-Code block start-

--C# code snippet start--

public static string StrServiceBusConnect

{

get

{

strServiceBusConnection = "ConectionString";

return DALClass.strServiceBusConnection;

}

set { DALClass.strServiceBusConnection = value; }

}

--C# code snippet end--

-Code block end-

* Open SQL Server Object Explorer and select the “ContosoSales”.
* Right click the database selected and then select the “Properties” and get the property of Connection String.
* Open the “DALClass” file and then replace the “strDataBaseConnection” value with the got.

-Code block start-

--C# code snippet start--

public static string StrDataBaseConnection

{

get

{

strDataBaseConnection = "ConectionString";

return DALClass.strDataBaseConnection;

}

}

--C# code snippet end--

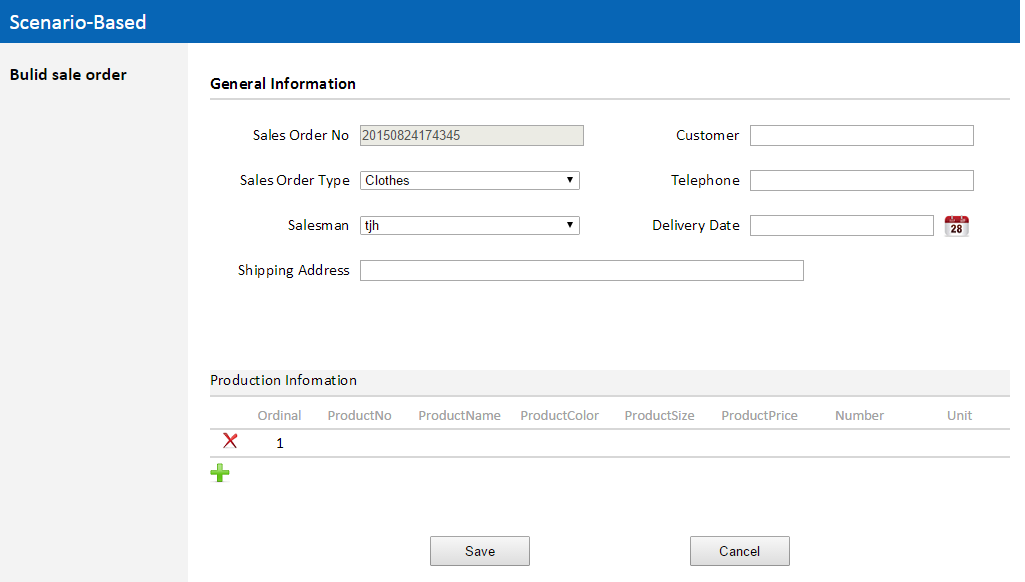
-Code block end-

* Rebuild the project.

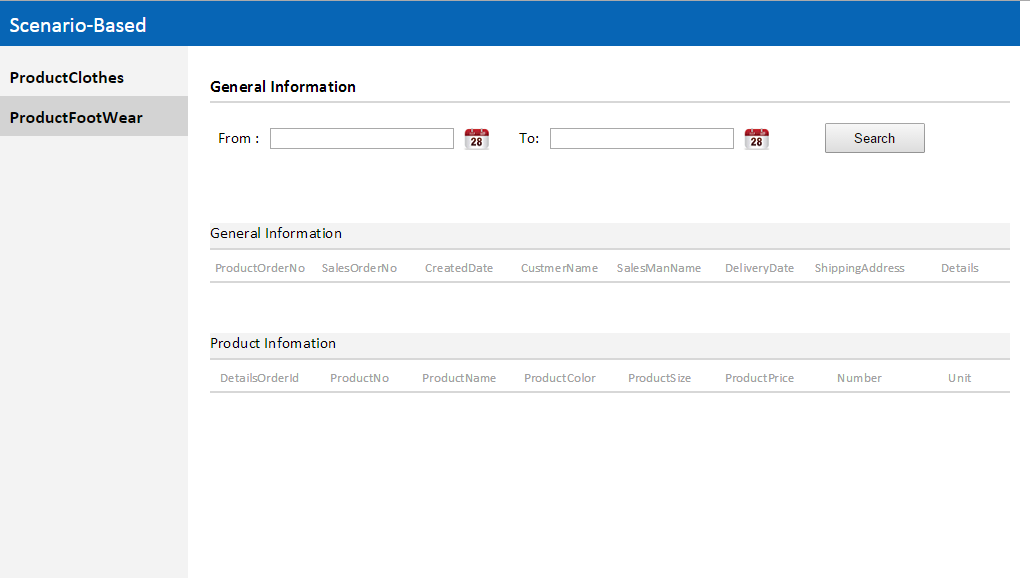
## **Step 4:** Open the” CSAzureReceiveSalesMessageViaTopic” project with Visual Studio 2013.

* You can follow [step 3](#step3) to get and replace the Azure Service Bus Topic name created in the [chapter 1](https://code.msdn.microsoft.com/scenarioDesign-a-web-page-10ceb941).
* You can follow [step 3](#step3) to get and replace the property of Connection String of the Azure Service Bus which is created in [chapter 1](https://code.msdn.microsoft.com/scenarioDesign-a-web-page-10ceb941).
* You can follow [step 3](#step3) to get and replace the property of Connection String of the “ProductDatabase”.
* Rebuild the project.

## **Step 5:** Run the” CSAzureSalesSendMessagesViaTopic” project and the picture below shows when the project is ran.

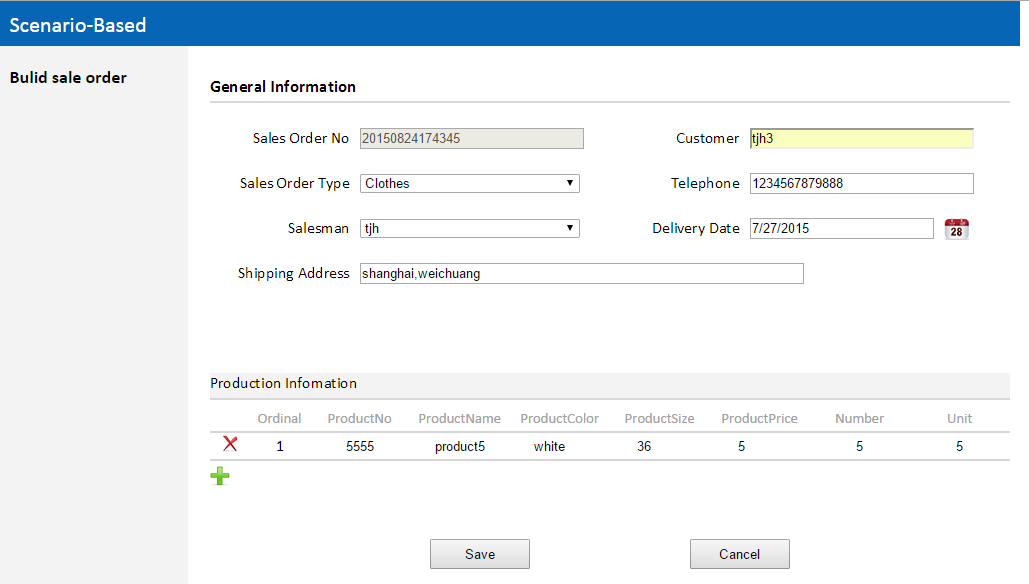


## **Step 6:** Run the” CSAzureReceiveSalesMessageViaTopic” project and the picture below shows when the project is ran.



## **Step 7:** Send messages via the Azure Service Bus Topic.

* Input the sales order information. For example, product information.



* Click the “Save “button when you finish inputting the sales order information.

1. The Data of the Customer table will be update if the data of the customer input is not same as it.
2. Save the data of the sales order into database.
3. Send the data of the sales order using the Azure Service Bus Topic when the data have saved into database. Serializes the data of the sales order to an XML before sending.

**Some code snippets:**

-Code block start-

--C# code snippet start--

/// <summary>

/// Sends the data of the sales order using Azure Service Bus Topic.

/// </summary>

/// <param name="customer"></param>

/// <param name="salesOrder"></param>

/// <param name="lstDetails"></param>

/// <returns></returns>

private bool SendOrderInfoViaTopic( Customer customer,SalesOrder salesOrder, List<SalesOrderProductDetails> lstDetails)

{

bool blnRet = false;

try

{

if(!string.IsNullOrEmpty(strServiceBusConnect)&&!string.IsNullOrEmpty(strTopic))

{

blnRet=CreateSubscription(salesOrder.StrSalesOrderType);

if(blnRet)

{

blnRet = false;

TopicClient topiclient = TopicClient.CreateFromConnectionString(strServiceBusConnect, strTopic);

BrokeredMessage message = new BrokeredMessage(" Message Date: " + DateTime.Now.ToString());

message.Properties.Add("Type", salesOrder.StrSalesOrderType);

string strCustomerContent = SerializeClass(typeof(Customer), customer);

string stSalesOrderContent = SerializeClass(typeof(SalesOrder), salesOrder);

string stSalesOrderDetailsContent = SerializeClass(typeof(List<SalesOrderProductDetails>), lstDetails);

message.Properties.Add("Customer", strCustomerContent);

message.Properties.Add("SalesOrder", stSalesOrderContent);

message.Properties.Add("SalesOrderDetails", stSalesOrderDetailsContent);

topiclient.Send(message);

blnRet = true;

}

}

}

catch(Exception ex)

{

//Throw ex;

}

return blnRet;

}

/// <summary>

/// Serializes the object to an XML.

/// </summary>

/// <param name="type"></param>

/// <param name="obj"></param>

/// <returns></returns>

private string SerializeClass(Type type, object obj)

{

string strContent = string.Empty;

string strFilePath = AppDomain.CurrentDomain.BaseDirectory + "XMLFile";

XmlSerializer xs = new XmlSerializer(type);

try

{

lock (lockObj)

{

if (!Directory.Exists(strFilePath))

{

Directory.CreateDirectory(strFilePath);

}

strFilePath = strFilePath + "\\Serialize+" + type.Name + ".xml";

if (File.Exists(strFilePath))

{

File.Delete(strFilePath);

}

}

}

catch (Exception ex)

{

//Throw ex;

}

try

{

using (FileStream stream = new FileStream(strFilePath, FileMode.OpenOrCreate, FileAccess.ReadWrite, FileShare.ReadWrite))

{

xs.Serialize(stream, obj);

stream.Close();

}

using (StreamReader sReader = new StreamReader(strFilePath, Encoding.GetEncoding("GB2312")))

{

strContent = sReader.ReadToEnd();

}

lock (lockObj)

{

if (File.Exists(strFilePath))

{

File.Delete(strFilePath);

}

}

}

catch (Exception ex)

{

//Throw ex;

}

return strContent;

}

--C# code snippet end--

-Code block end-

## **Step 8:** Receive messages via the Azure Service Bus Topic.

* There are two console applications to receive messages using the Azure Service Bus Topic.
* One is used receive the data of the Clothes sales order.
* The other is used receive data of the Footwear sales order.
* The data will be saved into the specified database when the application have finished receiving the data of the sales order. Serializes the xml content to object before saving.
* Set from date and to date and then click “Search “button, you can view the data received.

**Some code snippets:**

-Code block start-

--C# code snippet start--

string pathClothes = AppDomain.CurrentDomain.BaseDirectory + "bin" + "\\ConsoleReceiveSalesClothesMessage.exe";

string pathFootWear = AppDomain.CurrentDomain.BaseDirectory + "bin" + "\\ConsoleReceiveSalesFootWearMessage.exe";

string strClothesName = "";

string strFootwear = "";

/// <summary>

/// Starts two processes to receive messages that the sales department send.

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

protected void Application\_Start(object sender, EventArgs e)

{

ProcessStartInfo processClothes = new ProcessStartInfo(pathClothes, "");

processClothes.CreateNoWindow = true;

processClothes.UseShellExecute = false;

processClothes.WindowStyle = ProcessWindowStyle.Hidden;

Process myProcessClothes = Process.Start(processClothes);

strClothesName = myProcessClothes.ProcessName;

ProcessStartInfo processFootWear = new ProcessStartInfo(pathFootWear, "");

processFootWear.CreateNoWindow = true;

processFootWear.UseShellExecute = false;

processFootWear.WindowStyle = ProcessWindowStyle.Hidden;

Process myProcessFootWear = Process.Start(processFootWear);

strFootwear = myProcessFootWear.ProcessName;

}

--C# code snippet end--

-Code block end-

-Code block start-

--C# code snippet start--

/// <summary>

/// Receives messages that the sales department send when a Clothes sales order is built.

/// </summary>

/// <returns></returns>

static bool ReceiveMessage()

{

bool blnRet = false;

NamespaceManager namespaceManager = NamespaceManager.CreateFromConnectionString(strServiceBusConnect);

IEnumerable<SubscriptionDescription> lstSub = namespaceManager.GetSubscriptions(strTopic);

ThreadPool.SetMaxThreads(lstSub.Count(), lstSub.Count());

foreach (SubscriptionDescription sub in lstSub)

{

if (sub.MessageCount > 0 && sub.Name.Contains("Clothes"))

{

SubscriptionClient subClient = SubscriptionClient.CreateFromConnectionString(strServiceBusConnect, strTopic, sub.Name);

while (true)

{

try

{

BrokeredMessage message = null;

message = subClient.Receive();

if (message != null)

{

string strCustomer = "";

string strSalesOrder = "";

string strSalesOrderDetails = "";

if (message.Properties.ContainsKey("Customer") && message.Properties["Customer"] != null)

{

strCustomer = message.Properties["Customer"].ToString();

}

if (message.Properties.ContainsKey("SalesOrder") && message.Properties["SalesOrder"] != null)

{

strSalesOrder = message.Properties["SalesOrder"].ToString();

}

if (message.Properties.ContainsKey("SalesOrderDetails") && message.Properties["SalesOrderDetails"] != null)

{

strSalesOrderDetails = message.Properties["SalesOrderDetails"].ToString();

}

if (!string.IsNullOrEmpty(strCustomer) && !string.IsNullOrEmpty(strSalesOrder) && !string.IsNullOrEmpty(strSalesOrderDetails))

{

Customer customer = (Customer)Deserialize(strCustomer, typeof(Customer));

SalesOrder salesOrder = (SalesOrder)Deserialize(strSalesOrder, typeof(SalesOrder));

List<SalesOrderProductDetails> lstOrderDetails = (List<SalesOrderProductDetails>)Deserialize(strSalesOrderDetails, typeof(List<SalesOrderProductDetails>));

if (SaveMessage(customer, salesOrder, lstOrderDetails))

{

message.Complete();

}

}

}

}

catch (Exception ex)

{

//Throw ex;

}

}

}

}

return blnRet;

}

/// <summary>

/// Deserializes the XML document to an object.

/// </summary>

/// <param name="strContent"></param>

/// <param name="type"></param>

/// <returns></returns>

static object Deserialize(string strContent, Type type)

{

object obj = null;

try

{

byte[] array = Encoding.ASCII.GetBytes(strContent);

MemoryStream stream = new MemoryStream(array);

StreamReader reader = new StreamReader(stream);

XmlSerializer xs = new XmlSerializer(type);

if (type.Name == "Customer")

{

obj = xs.Deserialize(reader) as Customer;

}

else if (type.Name == "SalesOrder")

{

obj = xs.Deserialize(reader) as SalesOrder;

}

else

{

obj = xs.Deserialize(reader) as List<SalesOrderProductDetails>;

}

}

catch(Exception ex)

{

//Throw ex;

}

return obj;

}

/// <summary>

/// Saves messages revceived into the specified table.

/// </summary>

/// <param name="customer"></param>

/// <param name="salesOrder"></param>

/// <param name="lstOrderDetails"></param>

/// <returns></returns>

static bool SaveMessage(Customer customer, SalesOrder salesOrder, List< SalesOrderProductDetails> lstOrderDetails)

{

bool blnRet = false;

try

{

Customer customerNew = SaveCustomerInfo(customer);

if (!string.IsNullOrEmpty(customerNew.StrCustomerId))

{

salesOrder.StrCustomerId = customerNew.StrCustomerId;

blnRet= SaveClothesOrderInfo(salesOrder, lstOrderDetails);

}

}

catch(Exception ex)

{

//Throw ex;

}

return blnRet;

}

--C# code snippet end--

-Code block end-

Hope this makes things a bit easier in your developing Azure Service Bus.

Next chapter will show how to send or receive messages using Azure Service Bus Queue.

Stay tuned for more Azure Service Bus goodies next time.