16L30

IBM Integration Bus

Integration with Microsoft .NET Applications

Using the .NETInput Node
to access MSMQ Queues
and using Cloned Nodes

Featuring:

The .NETInput node
Accessing MSMQ queues
Creating and using IIB Cloned Nodes

June 2016

Hands-on lab built at product
Version 10.0.0.5

Microsoft, Windows and SQL Server are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

1. Introduction

WebSphere Message Broker Version 8 introduced the capability to integrate with existing .NET applications using the .NETCompute node, but initially lacked the ability to start a Message Flow using .NET. This feature was introduced in IBM Integration Bus v9.0, and the feature has been used in this scenario with IBM Integration Bus version 10.

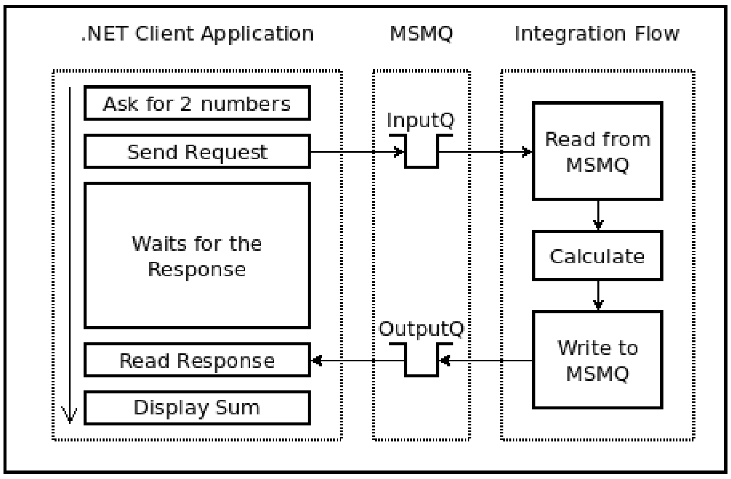
This lab will guide you through the steps needed to implement a Message Flow which starts an instance when a message arrives at a queue hosted on MSMQ (Microsoft Message Queuing). It responds to that by doing a simple mathematical calculation and sending a response to another MSMQ hosted queue.

1.1 Workshop system

If you are using the prepared VMWare image for the IIB workshop, this has been prepared with Microsoft Visual Studio Community Edition 2016. The screen captures in this lab guide have been taken from this version of Visual Studio. However, the scenario should work with other versions of Visual Studio.

1. Lab Objectives

. The scenario used in this lab is illustrated in the following diagram:



You will perform the following tasks

1. Create an application and message flow to contain the MSMQ Input nodes
2. Use the .NET Input Node to create an MSMQ Input Node for Event-based input
3. Create a Cloned Node based on the MSMQ Input Node
4. Create a new application and message flow which uses the new Cloned Node and two further .NET Compute nodes to perform the calculation and send the output back to an MSMQ queue.
5. Test the new message flow with a .NET Test Application
6. Perform steps 2-5 again, but creating a second MSMQ Input Node to use polling-based input.

The new nodes require some elements of new C# code. Some of this is generated by the Integration Bus Visual Studio plugins. These will automatically generate the following:

- A class to start an instance of a Message Flow by using the Polling technique.
- A class to start an instance of a Message Flow by using the Event-Driven approach.

In addition, the lab requires a small amount of bespoke C# code. This is provided for you.

2. Creating the MSMQ Queues

To enable the communication between the .NET Client Application and the Message Flow, you have to create the queues that they are going to use to talk to each other, and grant access on those queues to the user which the Integration Bus is being executed as.

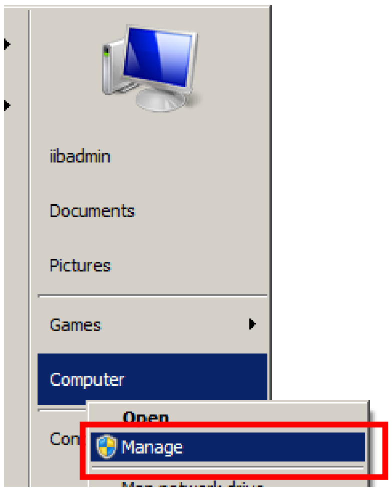
This section is included to show you how to create MSMQ queues, if you are not familiar with these tools.

2.1 Create the Input and the Output Queues

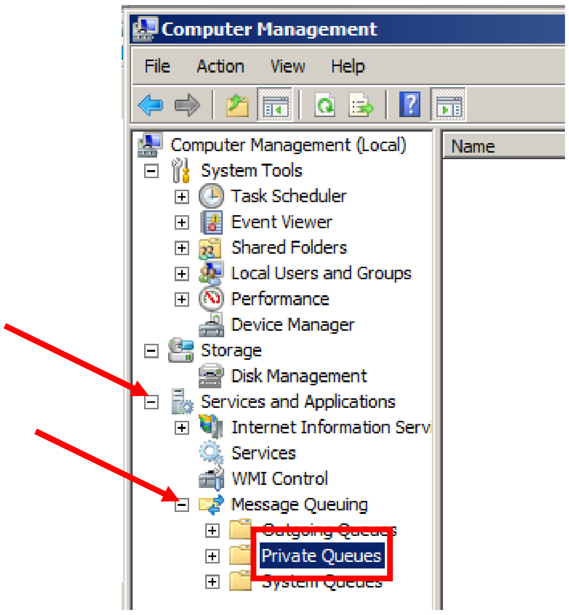
On the prepared VMWare Windows system, you may find these queues have already been created. You can still review the tools to create these queues, if you are not familiar with them.

1. Login to Windows with the user **iibuser**, password = **passw0rd**.

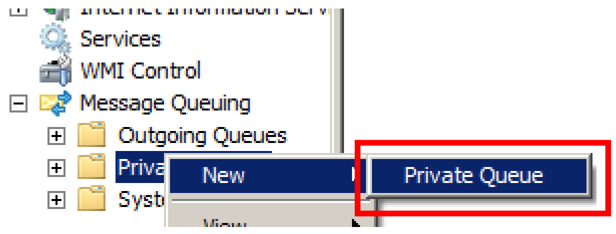
2. Click Windows Start, then right-click on the "**Computer**" item and click "**Manage**".



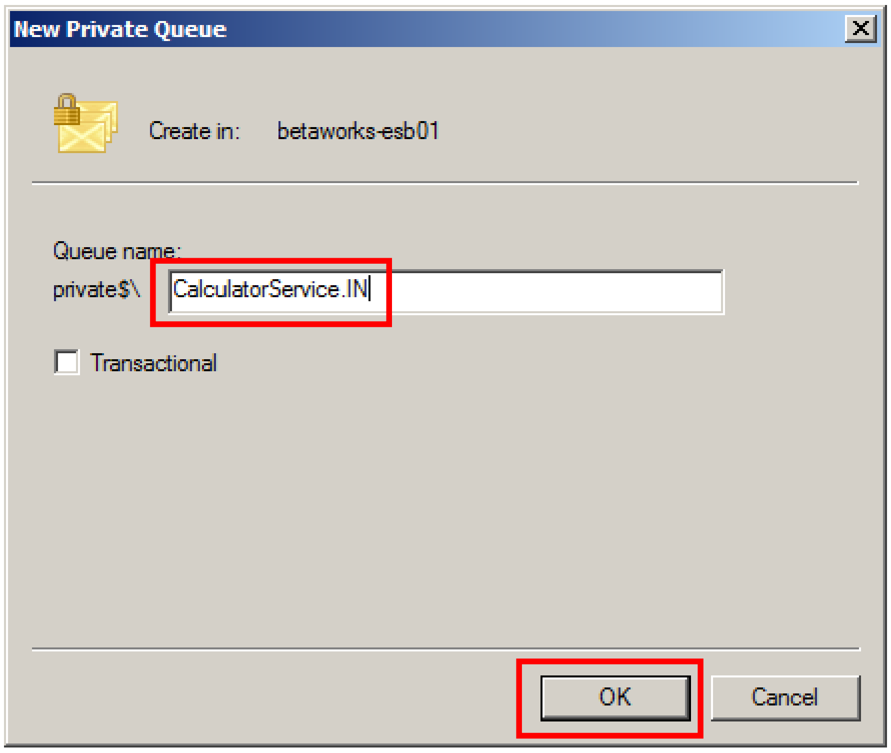
3. In the "Computer Management" application, expand "**Services and Applications**" and its "**Message Queuing**" sub-section, then select **Private Queues**.



4. Right-click “**Private Queues**”, then click “New” --> “Private Queue”.



5. Name it “**CalculatorService.IN**”, and click **OK**.



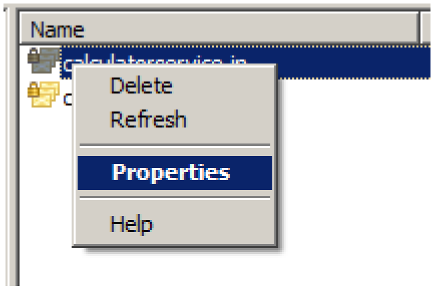
6. **Repeat** steps 3 and 4 to create the following queues. Note – if you are using the pre-built VM image, these have been created already:

- CalculatorService.OUT
- CalculatorService.IN.EVENT
- CalculatorService.IN.POLL

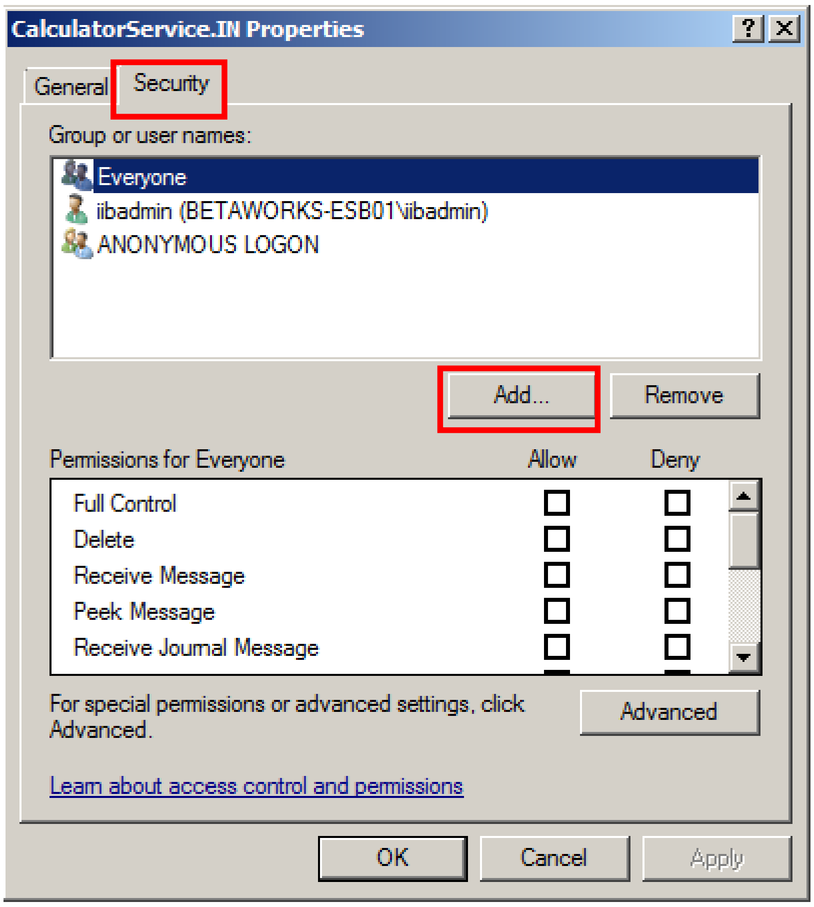
2.2 Grant permissions on the queues

Since the Integration Bus Node is running as the “Local System” account, you have to grant access to that user on the queues you just created, in order to allow the Integration Server to read and write messages to/from those queues.

1. Right-click the “**CalculatorService.IN**” queue, then click “**Properties**”.



2. Click the “**Security**” tab, then the “**Add**” button:


CalculatorService.IN Properties

General | Security

Group or user names:

- Everyone
- iibadmin (BETAWORKS-ESB01\iibadmin)
- ANONYMOUS LOGON

Add... | Remove

Permissions for Everyone	Allow	Deny
Full Control	☐	☐
Delete	☐	☐
Receive Message	☐	☐
Peek Message	☐	☐
Receive Journal Message	☐	☐

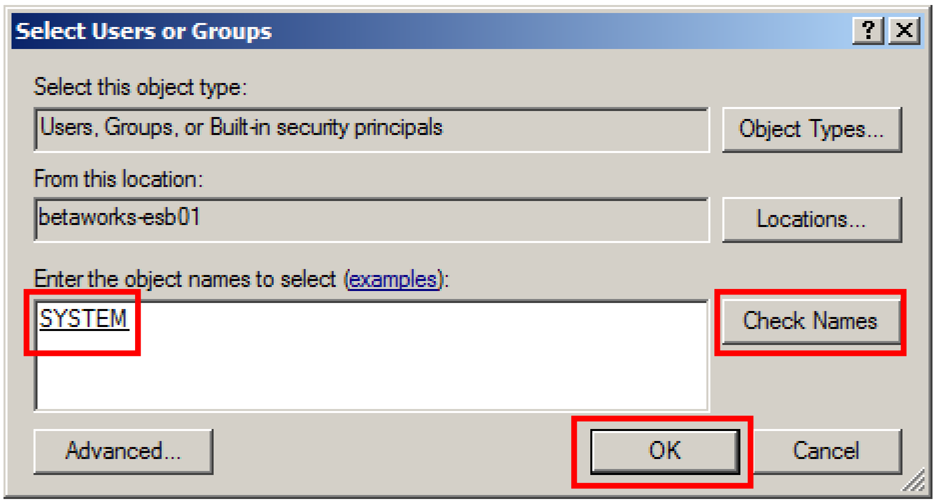
For special permissions or advanced settings, click Advanced.

Advanced

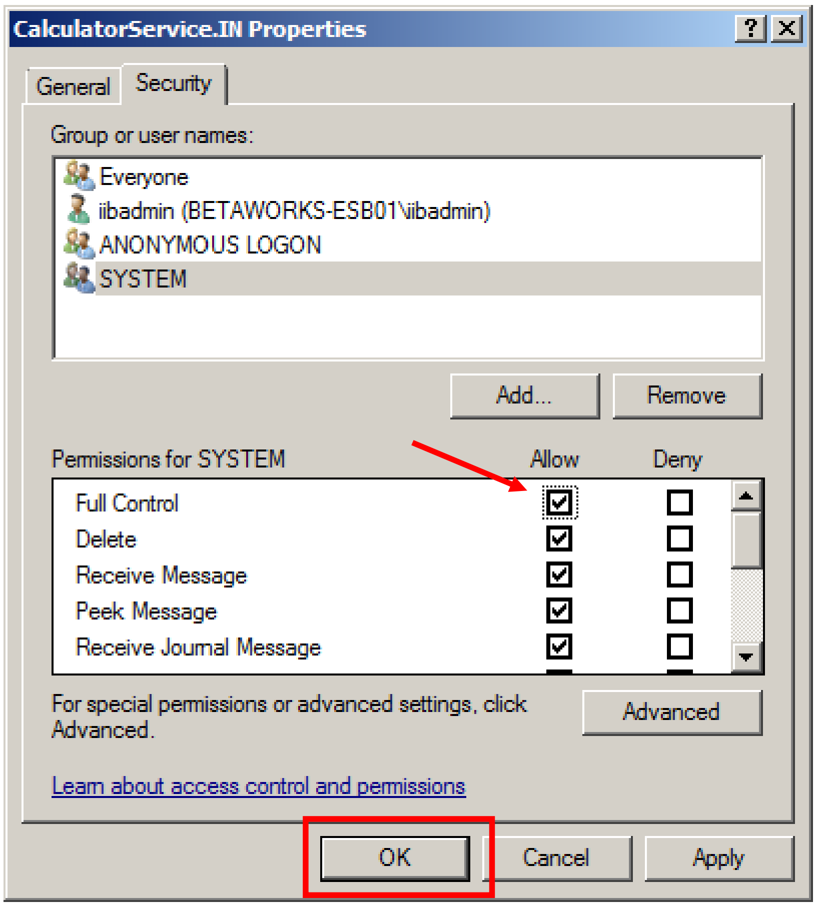
Learn about access control and permissions

OK | Cancel | Apply

3. Type "**SYSTEM**" in the text box, click the "**Check Names**" button and click "**OK**" to close the "Select Users or Groups" dialog.



4. Back in the "CalculatorService.IN" properties dialog, tick the "**Full Control**" check-box and click **OK**.



5. **Repeat** steps 1 to 4 to grant permissions on the remaining queues.

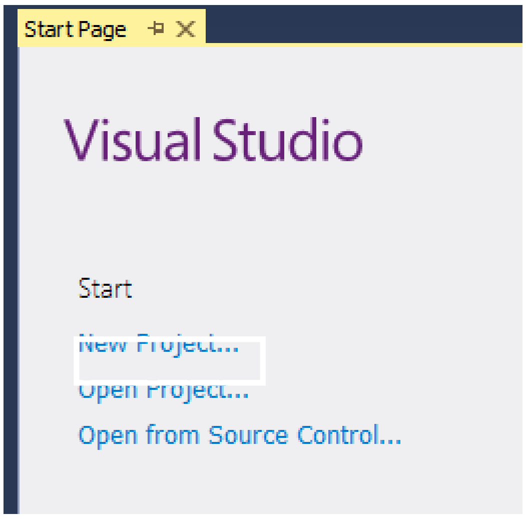
3. Use the .NETInput Node to create an MSMQ Input Node

The .NETInput node works in the same way as the .NETCompute node: The node is dropped onto the flow editor, and is then bound to a class within a .NET Assembly, which implements the logic you want to be executed by that node.

In this step, you will use Microsoft's Visual Studio 2015 Community Edition to implement the code behind the .NETInput node. This will create a specific instance of the .NET Input node for reading MSMQ queues.

1. From Windows Start, click the Visual Studio 2015 item.

2. When it has loaded, click the "**New Project**" link.



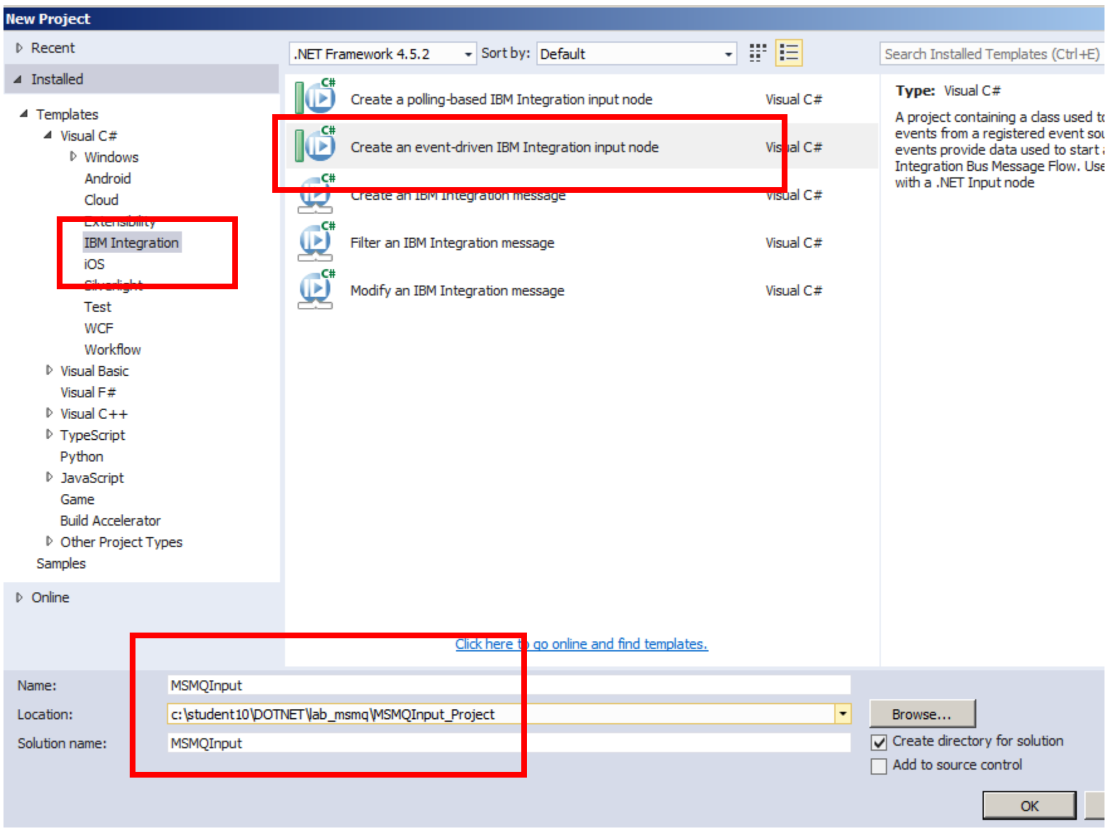
3. Expand the Visual C# template, and select “IBM Integration”.

   Highlight “Create an event-driven IBM Integration input node”.

   Set the Name of the project to MSMQInput.

   Set the Location to c:\student\DOTNET\lab_msmq\MSMQInput_Project.

   Click OK.



Create a polling-based IBM Integration input node	Visual C#
Create an event-driven IBM Integration input node	Visual C#
Create an IBM Integration message	Visual C#
Filter an IBM Integration message	Visual C#
Modify an IBM Integration message	Visual C#

Type: Visual C#

A project containing a class used t
events from a registered event sou
events provide data used to start
Integration Bus Message Flow. Use
with a .NET Input node

Name: MSMQInput

Location: c:\student10\DOTNET\lab_msmq\MSMQInput_Project

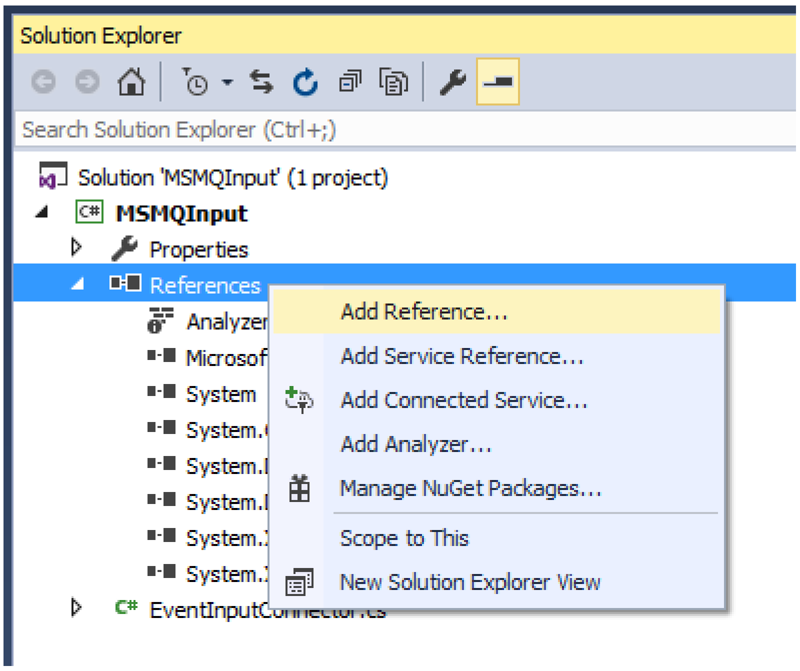
Solution name: MSMQInput

☑ Create directory for solution

☐ Add to source control

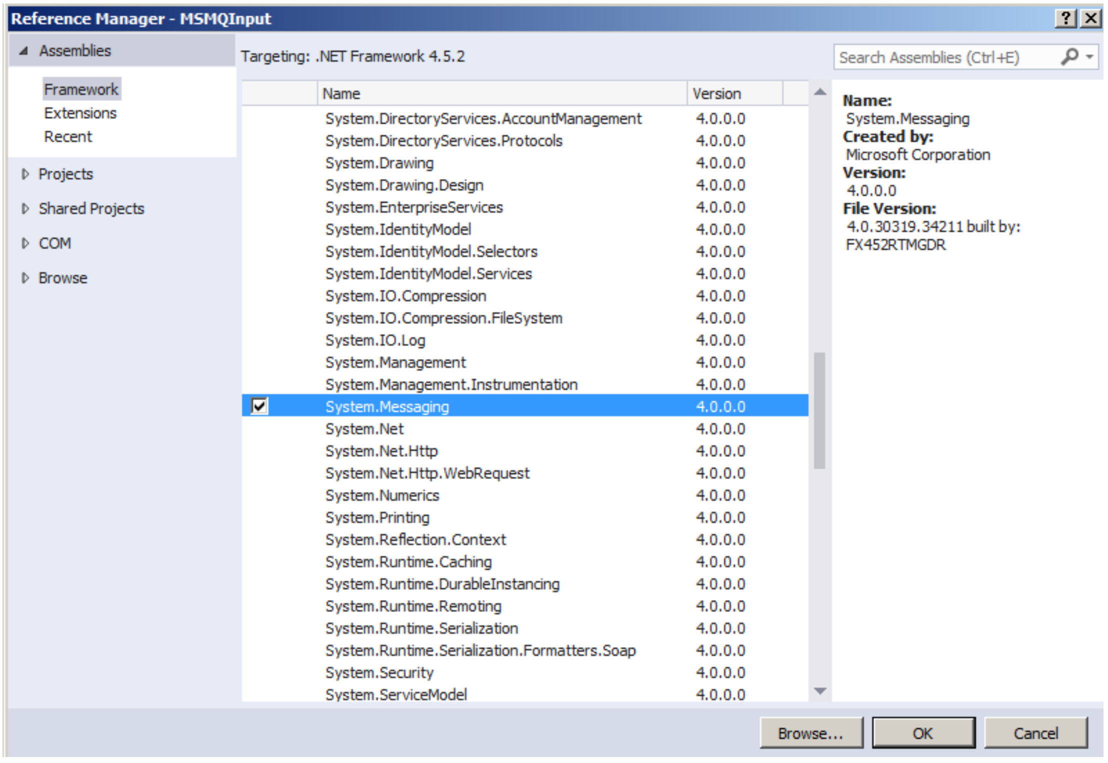
4. The default Visual Studio project does not contain a reference to System.Messaging, so you need to define a new reference for this.

   In the Solution Explorer (right pane), right-click References, and click “Add Reference”.



5. Scroll down to System.Messaging. As you highlight each line, the appropriate check-box will become visible.

   Place a tick in the System.Messaging checkbox, and click OK.



Targeting: .NET Framework 4.5.2

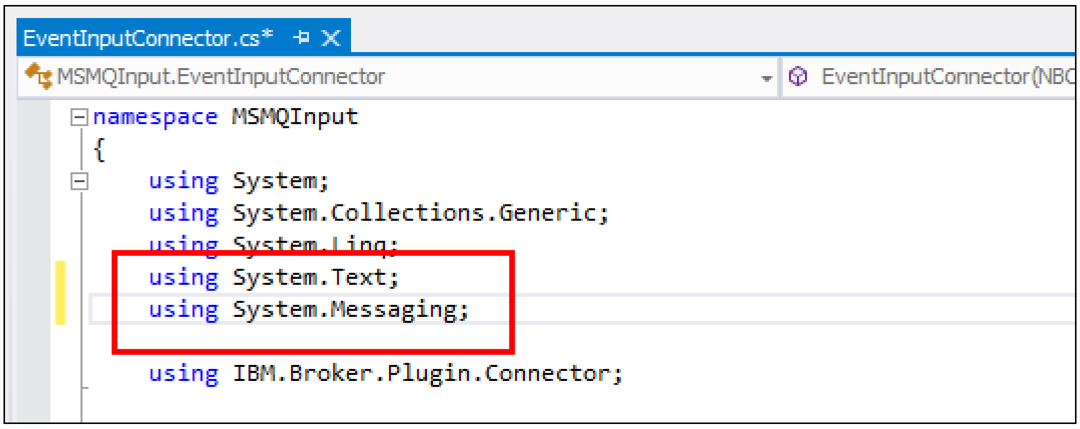
	Name	Version
	System.DirectoryServices.AccountManagement	4.0.0.0
	System.DirectoryServices.Protocols	4.0.0.0
	System.Drawing	4.0.0.0
	System.Drawing.Design	4.0.0.0
	System.EnterpriseServices	4.0.0.0
	System.IdentityModel	4.0.0.0
	System.IdentityModel.Selectors	4.0.0.0
	System.IdentityModel.Services	4.0.0.0
	System.IO.Compression	4.0.0.0
	System.IO.Compression.FileSystem	4.0.0.0
	System.IO.Log	4.0.0.0
	System.Management	4.0.0.0
	System.Management.Instrumentation	4.0.0.0
✓	System.Messaging	4.0.0.0
	System.Net	4.0.0.0
	System.Net.Http	4.0.0.0
	System.Net.Http.WebRequest	4.0.0.0
	System.Numerics	4.0.0.0
	System.Printing	4.0.0.0
	System.Reflection.Context	4.0.0.0
	System.Runtime.Caching	4.0.0.0
	System.Runtime.DurableInstancing	4.0.0.0
	System.Runtime.Remoting	4.0.0.0
	System.Runtime.Serialization	4.0.0.0
	System.Runtime.Serialization.Formatters.Soap	4.0.0.0
	System.Security	4.0.0.0
	System.ServiceModel	4.0.0.0

Name:
System.Messaging
Created by:
Microsoft Corporation
Version:
4.0.0.0
File Version:
4.0.30319.34211 built by: FX452RTMGDR

6. You now need to include a reference to this in the generated class.

   In the EventInputConnector class, add a new line as follows

   `using System.Messaging;`



EventInputConnector.cs*

MSMQInput.EventInputConnector

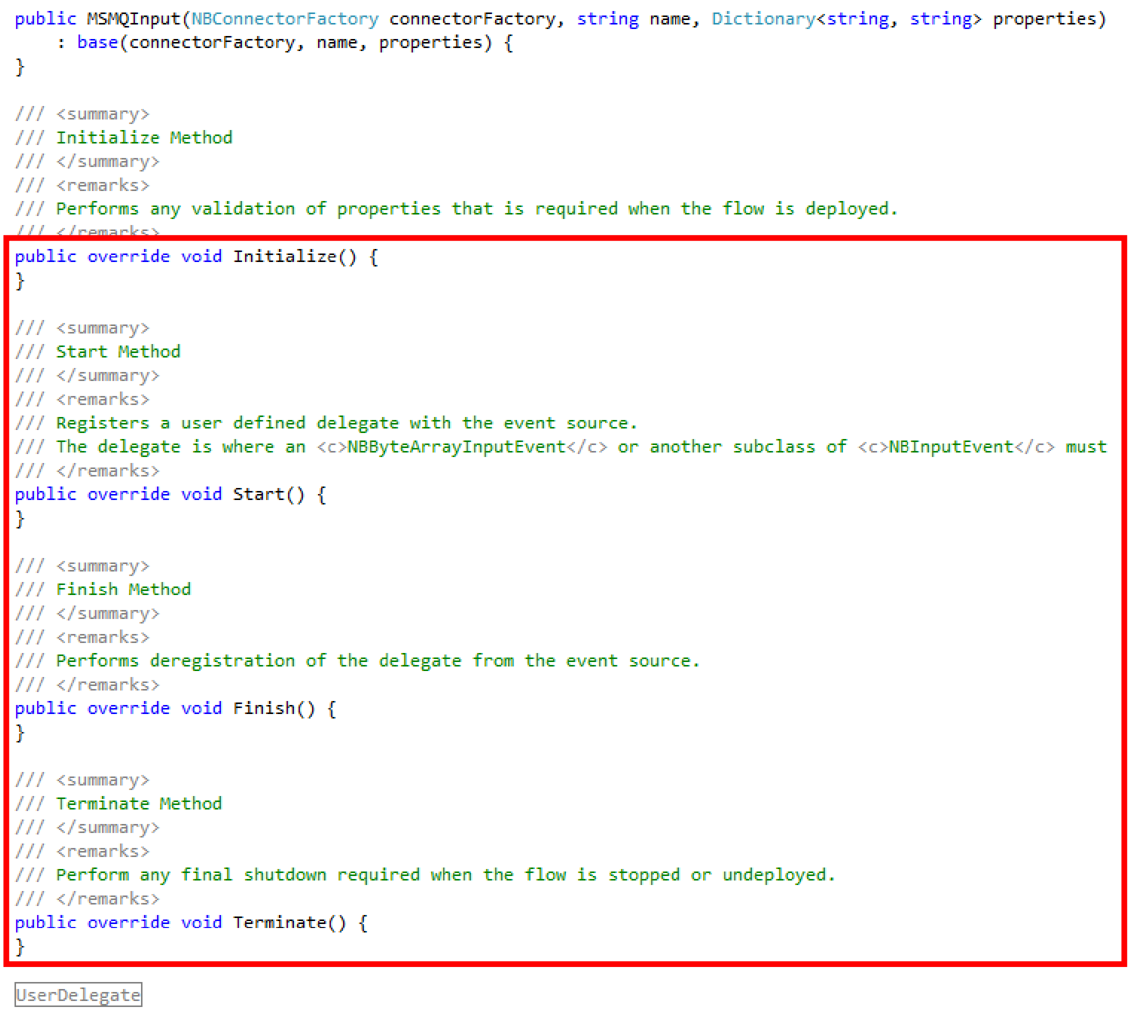
EventInputConnector(NBC

```
namespace MSMQInput
{
    using System;
    using System.Collections.Generic;
    using System.Linq;
    using System.Text;
    using System.Messaging;

    using IBM.Broker.Plugin.Connector;
```

7. Take a minute to examine the generated class; You'll notice that, in addition to the constructor, 4 methods and a #region were generated:

   - Initialize: This method is invoked when the flow is deployed to the Integration Server.
   - Start: This method is invoked when the flow is started (either by Node, Server or Flow startup).
   - Finish: This method is invoked when the flow is gracefully stopped (either by Flow, Server or Node stop).
   - Terminate: This method is invoked when the flow is removed (un-deployed) from the Integration Server.

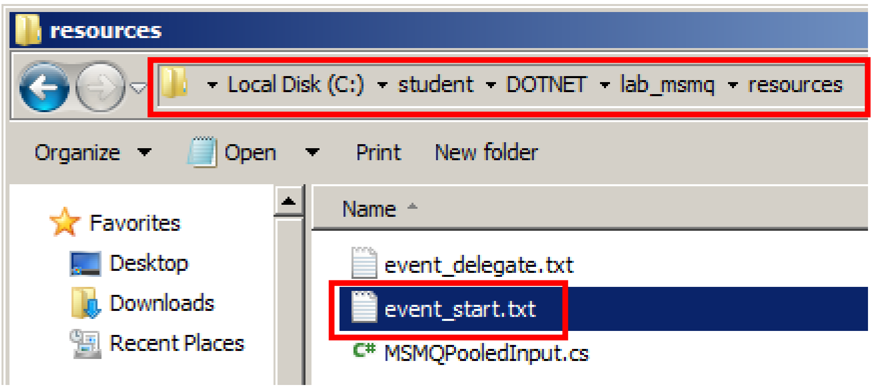
   Additionally, in the event-driven approach, you have to register the "delegate" method of your class (typically inserted into the "UserDelegate" region) that is going to be called every time a message flow instance has to be created. In this Lab, you will register a callback in a "System.Messaging.MessageQueue" instance that will call the MSMQInput class when a message arrives on the Input Queue. This registration must be done inside the Start() method and undone inside the Finish() method.



```
public MSMQInput(NBConnectorFactory connectorFactory, string name, Dictionary<string, string> properties)
    : base(connectorFactory, name, properties) {
}

/// <summary>
/// Initialize Method
/// </summary>
/// <remarks>
/// Performs any validation of properties that is required when the flow is deployed.
/// </remarks>
public override void Initialize() {
}

/// <summary>
/// Start Method
/// </summary>
/// <remarks>
/// Registers a user defined delegate with the event source.
/// The delegate is where an <c>NBByteArrayInputEvent</c> or another subclass of <c>NBInputEvent</c> must
/// </remarks>
public override void Start() {
}

/// <summary>
/// Finish Method
/// </summary>
/// <remarks>
/// Performs deregistration of the delegate from the event source.
/// </remarks>
public override void Finish() {
}

/// <summary>
/// Terminate Method
/// </summary>
/// <remarks>
/// Perform any final shutdown required when the flow is stopped or undeployed.
/// </remarks>
public override void Terminate() {
}

UserDelegate
```

8. You will now complete the implementation of the MSMQ Input Node by copying some prepared C# code into the new class.

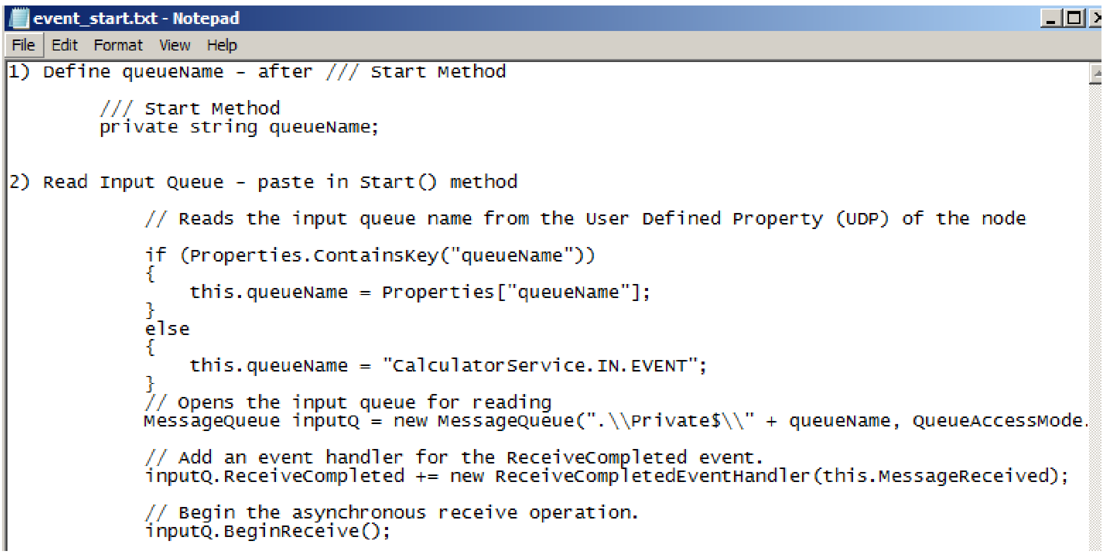
   In Windows Explorer, navigate to the "**C:\student\DOTNET\lab_msmq\resources**" folder. Open the file "**event_start.txt**" in Notepad.



9. This file has the implementation of the **Start()** method of the **MSMQInput** class. It reads the name of the Input Queue from a "User Defined Property", creates an instance of the Queue and Registers the delegate method to be called when a new message arrives.

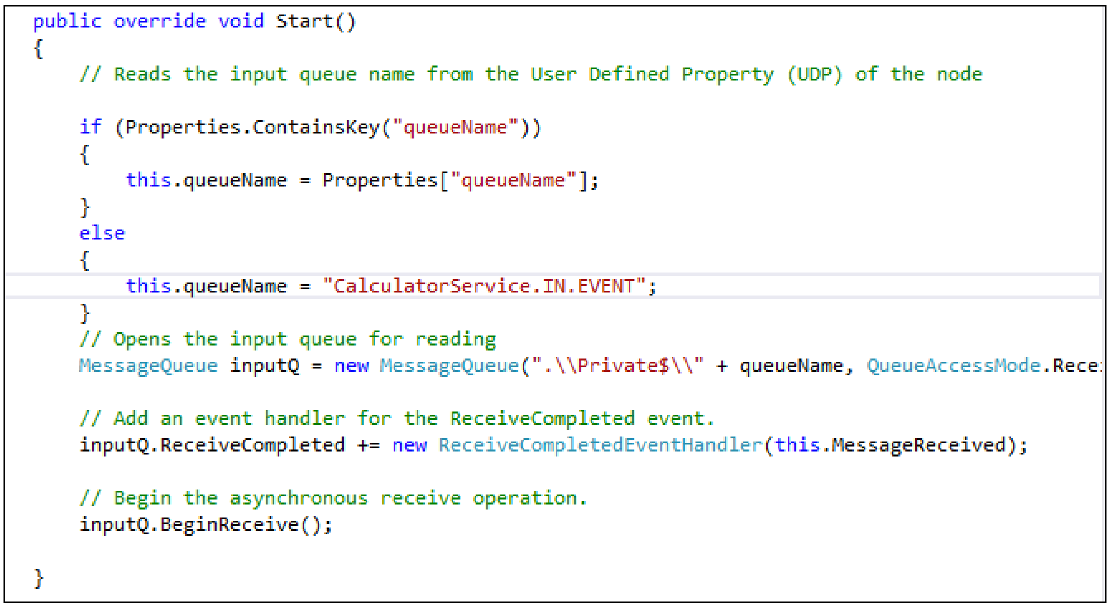
   Take a moment to examine its logic on your notepad instance.

   Copy the contents in section (2) of the file.



```
1) Define queueName - after /// Start Method

        /// Start Method
        private string queueName;

2) Read Input Queue - paste in Start() method

            // Reads the input queue name from the User Defined Property (UDP) of the node

            if (Properties.ContainsKey("queueName"))
            {
                this.queueName = Properties["queueName"];
            }
            else
            {
                this.queueName = "CalculatorService.IN.EVENT";
            }
            // Opens the input queue for reading
            MessageQueue inputQ = new MessageQueue(".\\Private$\\" + queueName, QueueAccessMode.

            // Add an event handler for the ReceiveCompleted event.
            inputQ.ReceiveCompleted += new ReceiveCompletedEventHandler(this.MessageReceived);

            // Begin the asynchronous receive operation.
            inputQ.BeginReceive();
```

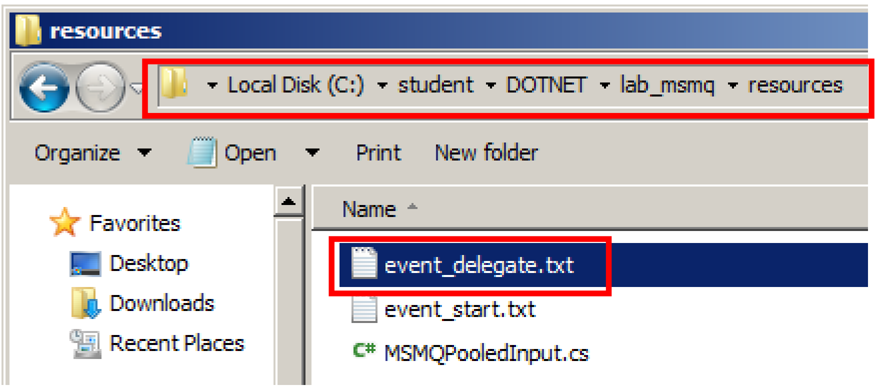
10. Paste the copied text inside of the **Start()** method of the EventInputConnector class. Your result should look like this:



```
public override void Start()
{
    // Reads the input queue name from the User Defined Property (UDP) of the node

    if (Properties.ContainsKey("queueName"))
    {
        this.queueName = Properties["queueName"];
    }
    else
    {
        this.queueName = "CalculatorService.IN.EVENT";
    }
    // Opens the input queue for reading
    MessageQueue inputQ = new MessageQueue(".\\Private$\\" + queueName, QueueAccessMode.Rece

    // Add an event handler for the ReceiveCompleted event.
    inputQ.ReceiveCompleted += new ReceiveCompletedEventHandler(this.MessageReceived);

    // Begin the asynchronous receive operation.
    inputQ.BeginReceive();

}
```

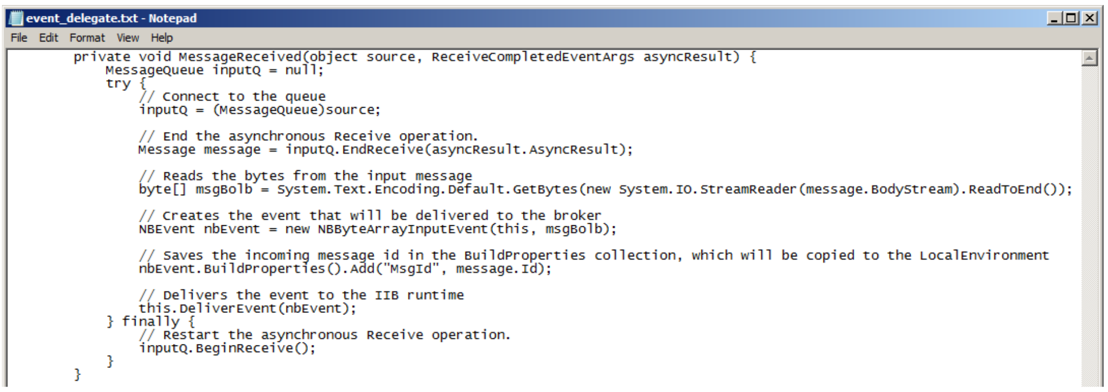
WARNING!
Be aware that if your notepad has the "**Word Wrap**" option enabled, it **will** insert some line breaks in the text when you copy it. If that's the case, either disable that function before copying or adjust the code after pasting, according to the one showed above.

11. Back in Windows Explorer, open "**event_delegate.txt**" file in Notepad.



12. This file has the implementation of the **UserDelegate** method of the **MSMQInput** class. It reads the incoming message into an unparsed byte array and saves its MessageId header into the LocalEnvironment, in order to allow the response message to have this value in its Correlation Id header.

    Copy all the contents of the file:

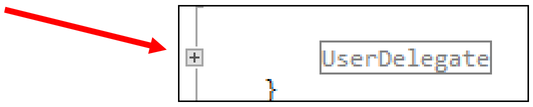


event_delegate.txt - Notepad

File Edit Format View Help

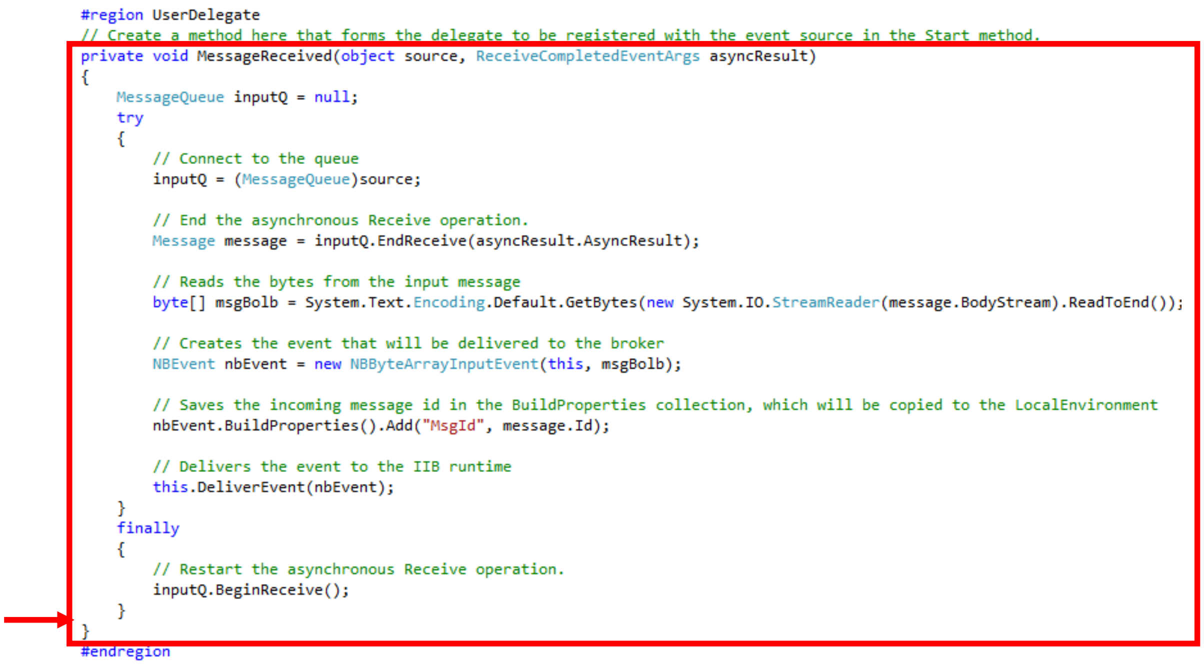
```
        private void MessageReceived(object source, ReceiveCompletedEventArgs asyncResult) {
            MessageQueue inputQ = null;
            try {
                // Connect to the queue
                inputQ = (MessageQueue)source;

                // End the asynchronous Receive operation.
                Message message = inputQ.EndReceive(asyncResult.AsyncResult);

                // Reads the bytes from the input message
                byte[] msgBolb = System.Text.Encoding.Default.GetBytes(new System.IO.StreamReader(message.BodyStream).ReadToEnd());

                // Creates the event that will be delivered to the broker
                NBEvent nbEvent = new NBByteArrayInputEvent(this, msgBolb);

                // Saves the incoming message id in the BuildProperties collection, which will be copied to the LocalEnvironment
                nbEvent.BuildProperties().Add("MsgId", message.Id);

                // Delivers the event to the IIB runtime
                this.DeliverEvent(nbEvent);
            } finally {
                // Restart the asynchronous Receive operation.
                inputQ.BeginReceive();
            }
        }
```

13. Expand the "**UserDelegate**" region, by clicking the "plus" icon on the left of the editor:



14. Paste the copied text inside of the "**UserDelegate**" part of the class.

    Your result must look like this:



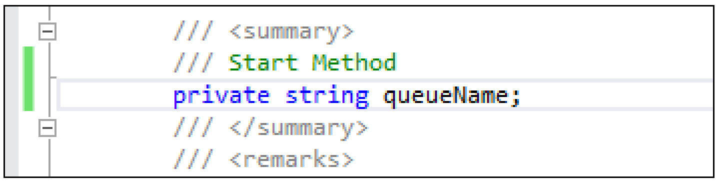
```
#region UserDelegate
// Create a method here that forms the delegate to be registered with the event source in the Start method.
private void MessageReceived(object source, ReceiveCompletedEventArgs asyncResult)
{
    MessageQueue inputQ = null;
    try
    {
        // Connect to the queue
        inputQ = (MessageQueue)source;

        // End the asynchronous Receive operation.
        Message message = inputQ.EndReceive(asyncResult.AsyncResult);

        // Reads the bytes from the input message
        byte[] msgBolb = System.Text.Encoding.Default.GetBytes(new System.IO.StreamReader(message.BodyStream).ReadToEnd());

        // Creates the event that will be delivered to the broker
        NBEvent nbEvent = new NBByteArrayInputEvent(this, msgBolb);

        // Saves the incoming message id in the BuildProperties collection, which will be copied to the LocalEnvironment
        nbEvent.BuildProperties().Add("MsgId", message.Id);

        // Delivers the event to the IIB runtime
        this.DeliverEvent(nbEvent);
    }
    finally
    {
        // Restart the asynchronous Receive operation.
        inputQ.BeginReceive();
    }
}
#endregion
```

WARNING!
Be aware that if your notepad has the "**Word Wrap**" option enabled, it **will** insert some line breaks in the text when you copy it. If that's the case, either disable that function before copying or adjust the code after pasting, according to the one showed above.

15. To be able to select the name of the input queue using a user-defined property, define a variable called queueName.

    In the Start section, define queueName as follows:

    ```
    private string queueName;
    ```

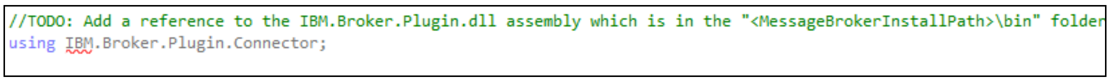


```
/// <summary>
/// Start Method
private string queueName;
/// </summary>
/// <remarks>
```

(or copy from the event_start.txt file, paragraph (1).

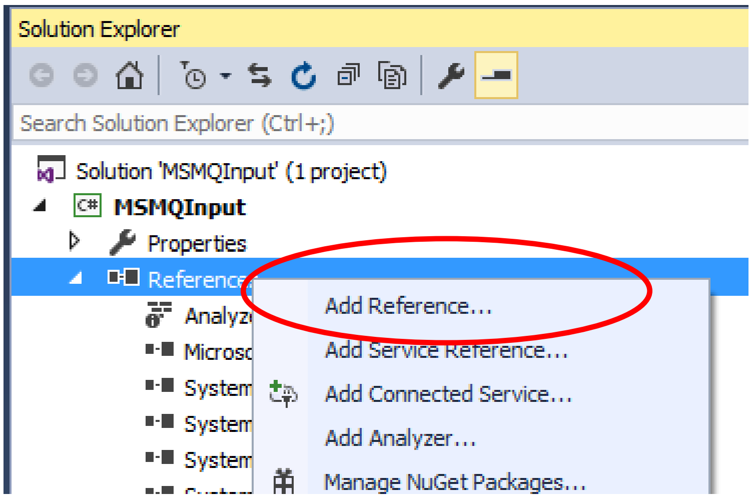
16. Finally, near the top of the generated code, you will see a comment

    //TODO: Add a reference to the IBM.Broker.Plugin.dll assembly .....

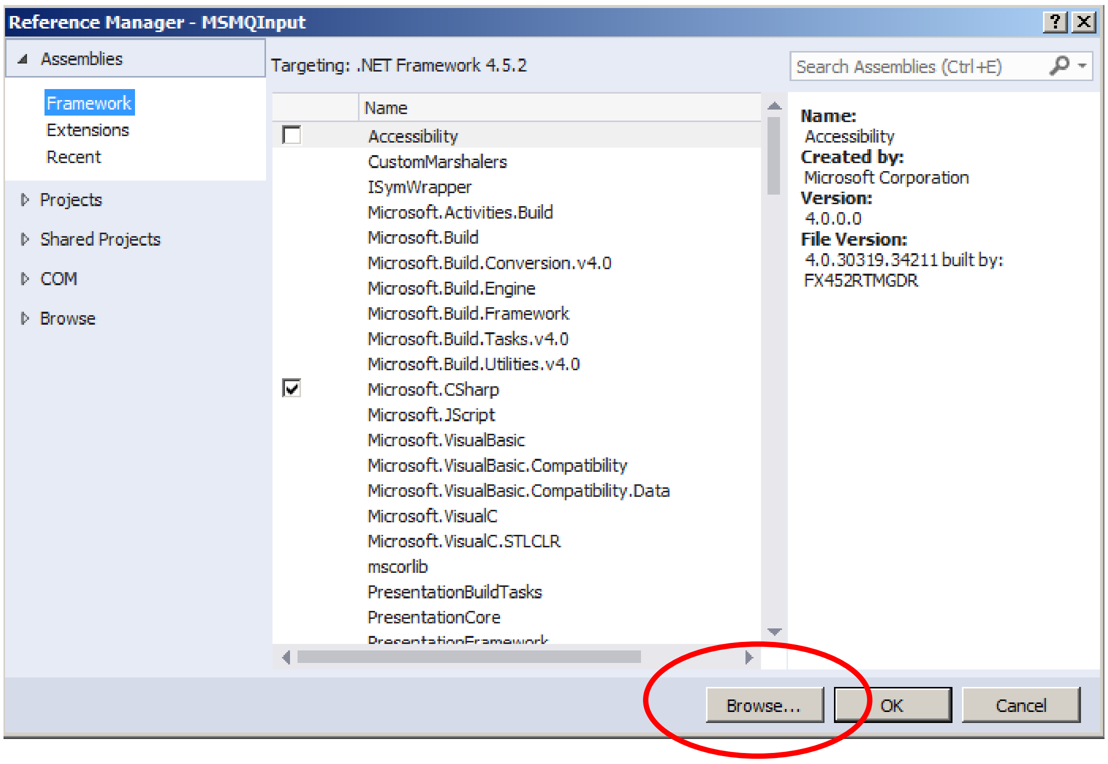


```
//TODO: Add a reference to the IBM.Broker.Plugin.dll assembly which is in the "<MessageBrokerInstallPath>\bin" folder
using IBM.Broker.Plugin.Connector;
```

To set the required reference, back in the Solution Explorer, right-click References, and select Add Reference.



17. In the Reference Manager, click Browse.



Targeting: .NET Framework 4.5.2

	Name
☐	Accessibility
	CustomMarshalers
	ISymWrapper
	Microsoft.Activities.Build
	Microsoft.Build
	Microsoft.Build.Conversion.v4.0
	Microsoft.Build.Engine
	Microsoft.Build.Framework
	Microsoft.Build.Tasks.v4.0
	Microsoft.Build.Utilities.v4.0
✓	Microsoft.CSharp
	Microsoft.JScript
	Microsoft.VisualBasic
	Microsoft.VisualBasic.Compatibility
	Microsoft.VisualBasic.Compatibility.Data
	Microsoft.VisualC
	Microsoft.VisualC.STLCLR
	mscorlib
	PresentationBuildTasks
	PresentationCore

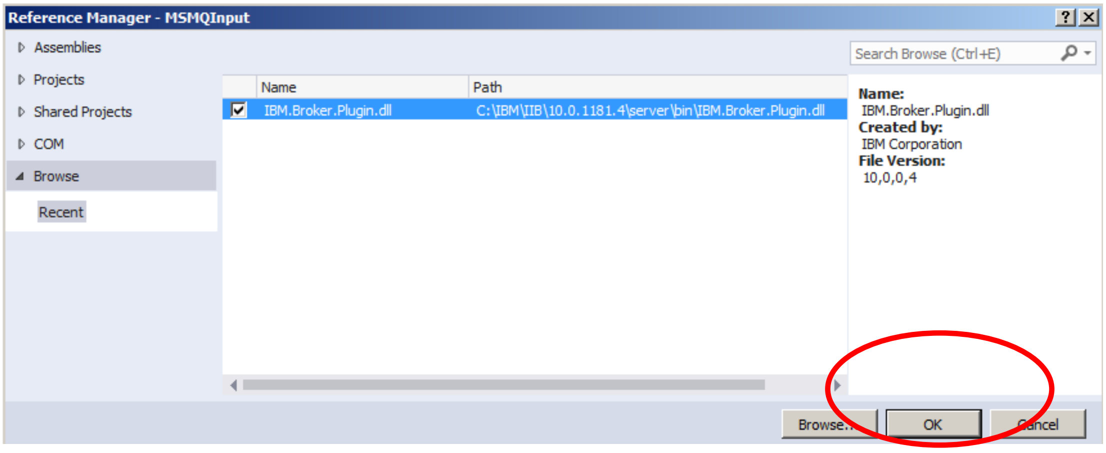
Name:
Accessibility
Created by:
Microsoft Corporation
Version:
4.0.0.0
File Version:
4.0.30319.34211 built by: FX452RTMGDR

18. Navigate to the file IBM.Broker.Plugin.dll.

This is located in the folder ..\IIB_*installFolder*\server\bin.

On the workshop IIB installation, this file will be in c:\IBM\IIB\10.0.0.5\server\bin.

Click OK when you see the window shown below.



	Name	Path
✓	IBM.Broker.Plugin.dll	C:\IBM\IIB\10.0.1181.4\server\bin\IBM.Broker.Plugin.dll

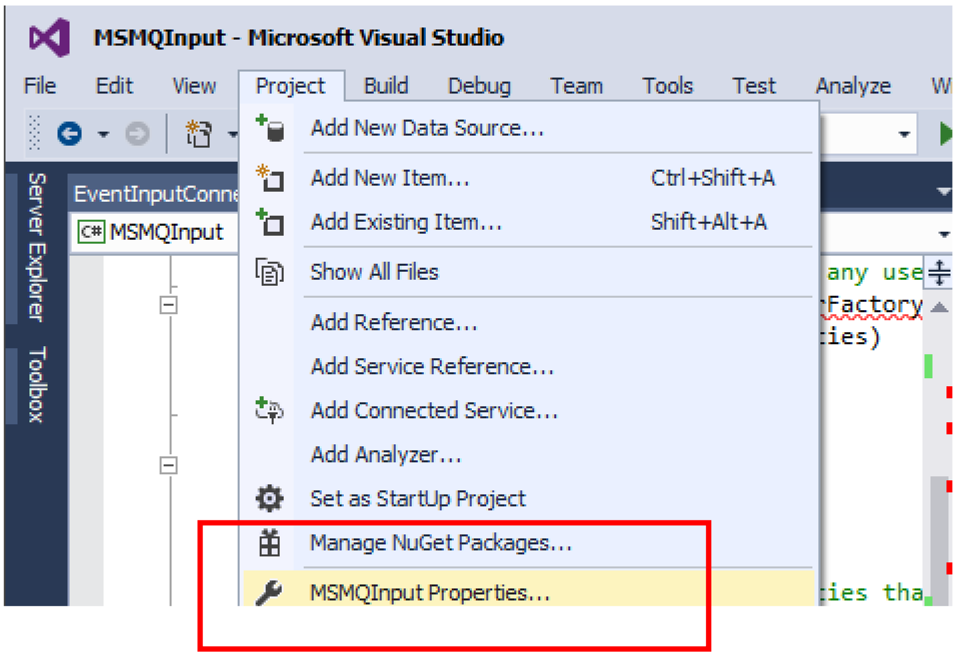
Name:
IBM.Broker.Plugin.dll
Created by:
IBM Corporation
File Version:
10,0,0,4

19. Save the EventInputConnector.cs file with Ctrl-S.

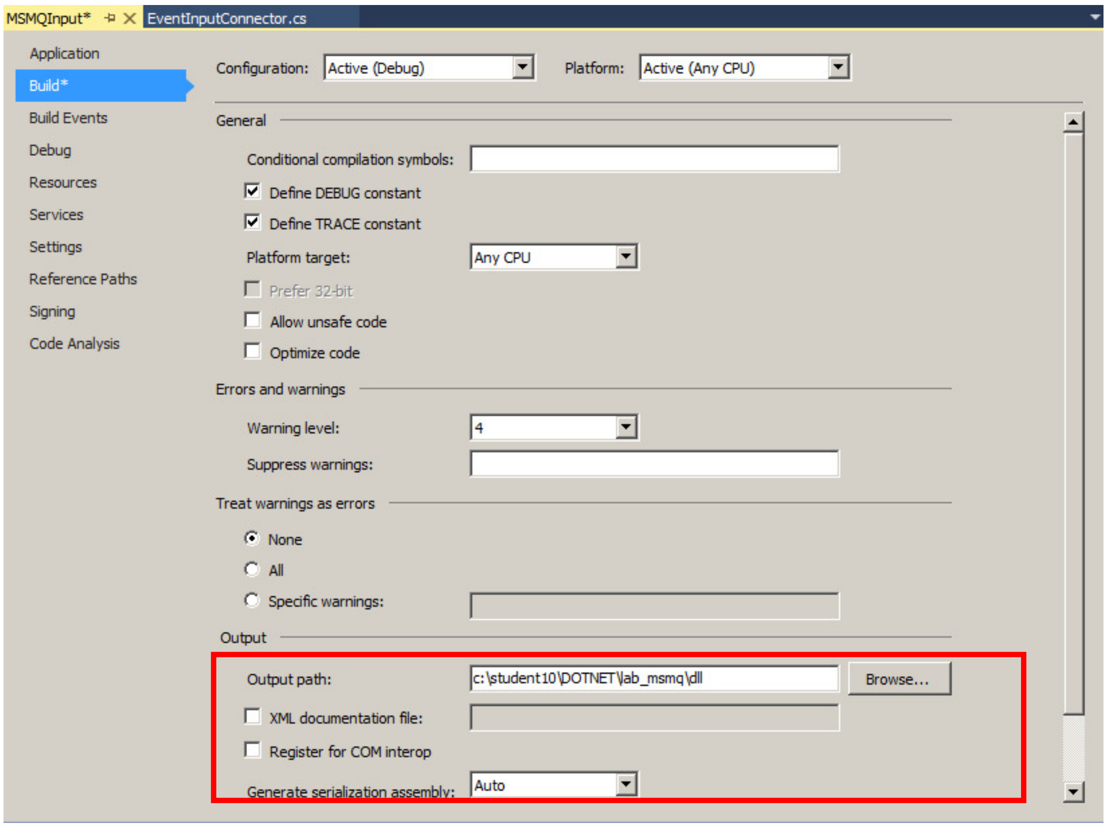
3.1 Build the .NET Application

1. First, you must change the output directory for the Visual Studio build tools. This is to make sure that the generated DLLs are placed in the correct location for the Integration Bus applications.

   To specify the directory for the Build output, on the Visual Studio Toolbar, click Project, and select MSMQInput Properties.



2. On the Build tab, set the Output Path to c:\student10\DOTNET\lab_msmq\dll.



Configuration: Active (Debug) Platform: Active (Any CPU)

General
Conditional compilation symbols:
[✓] Define DEBUG constant
[✓] Define TRACE constant
Platform target: Any CPU
[ ] Prefer 32-bit
[ ] Allow unsafe code
[ ] Optimize code

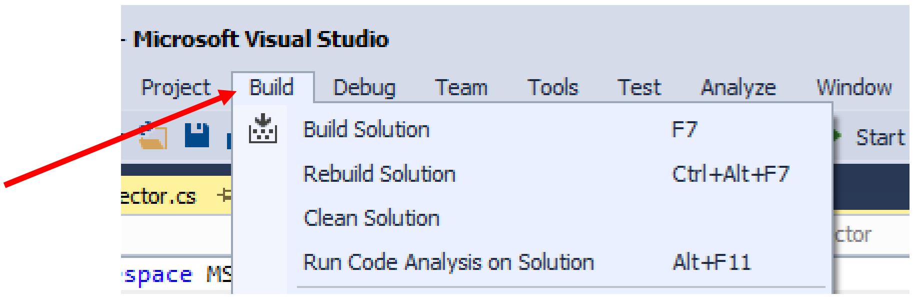
Errors and warnings
Warning level: 4
Suppress warnings:

Treat warnings as errors
[✓ None] All Specific warnings:

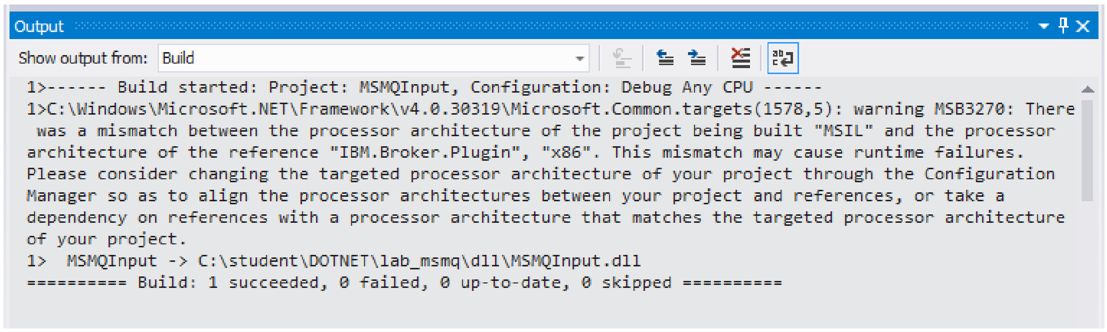
Output
Output path: c:\student10\DOTNET\lab_msmq\dll
[ ] XML documentation file:
[ ] Register for COM interop
Generate serialization assembly: Auto

Save this change (Ctrl-S), and then close the MSMQInput Properties.

3. Build the solution, either by pressing “F7” or using the “Build” menu:



4. The Output view is shown, with the following result. You can ignore the message regarding MSIL processor architectures.



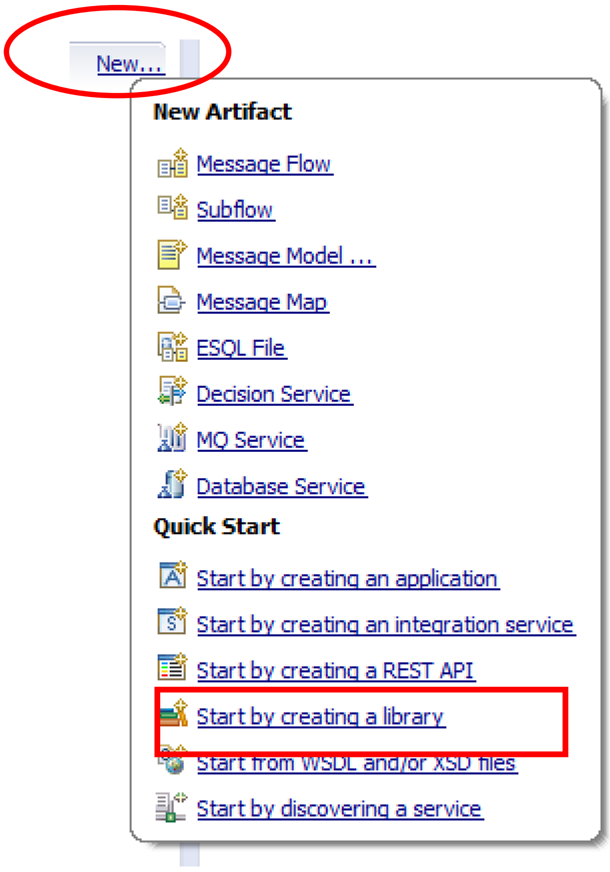
```
1>------ Build started: Project: MSMQInput, Configuration: Debug Any CPU ------
1>C:\Windows\Microsoft.NET\Framework\v4.0.30319\Microsoft.Common.targets(1578,5): warning MSB3270: There
 was a mismatch between the processor architecture of the project being built "MSIL" and the processor
architecture of the reference "IBM.Broker.Plugin", "x86". This mismatch may cause runtime failures.
Please consider changing the targeted processor architecture of your project through the Configuration
Manager so as to align the processor architectures between your project and references, or take a
dependency on references with a processor architecture that matches the targeted processor architecture
of your project.
1>  MSMQInput -> C:\student\DOTNET\lab_msmq\dll\MSMQInput.dll
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```

4. Create the MSMQ Cloned Input Node

You will now create an MSMQ Input Node by using the Cloned Node function with the MSMQinput class that you have just created.

1. Switch to the Integration Toolkit, and create a new workspace named "workspace_msmq".

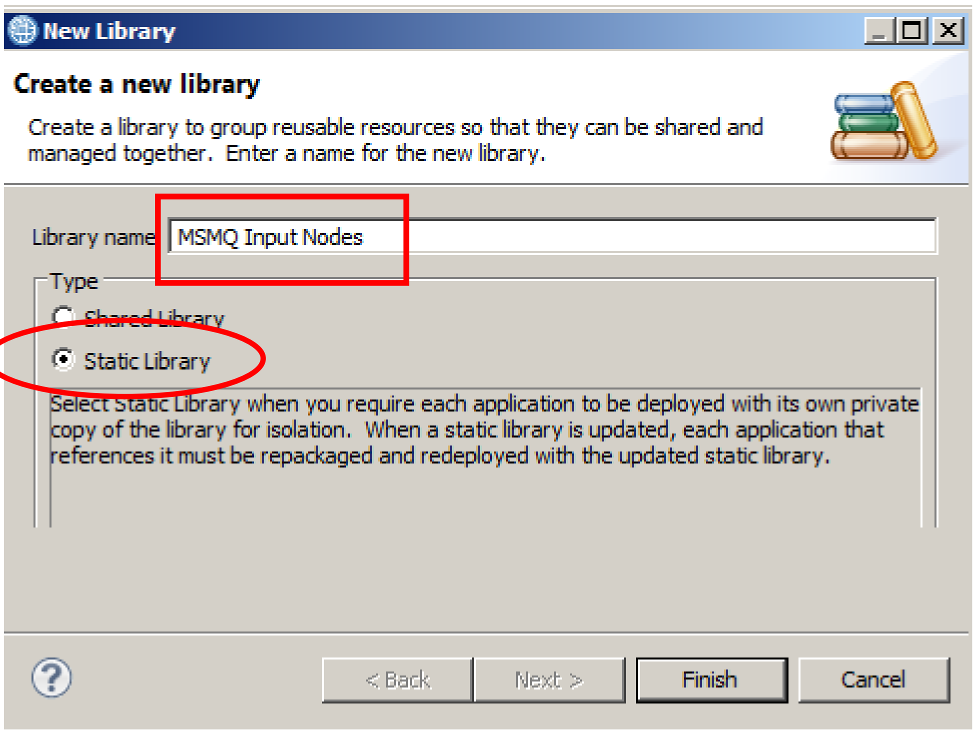
   Create a new library by clicking “New Library”, or Start by Creating a Library.


New…
New Artifact
Message Flow
Subflow
Message Model …
Message Map
ESQL File
Decision Service
MQ Service
Database Service
Quick Start
Start by creating an application
Start by creating an integration service
Start by creating a REST API
Start by creating a library
Start from WSDL and/or XSD files
Start by discovering a service

2. Name your library "**MSMQ Input Nodes**".

   Because this library will contain flow input nodes, you must create a static library (shared libraries cannot contain message flows).

   Ensure **Static Library** is selected, and click Finish.



New Library

Create a new library

Create a library to group reusable resources so that they can be shared and managed together. Enter a name for the new library.

Library name: MSMQ Input Nodes

Type

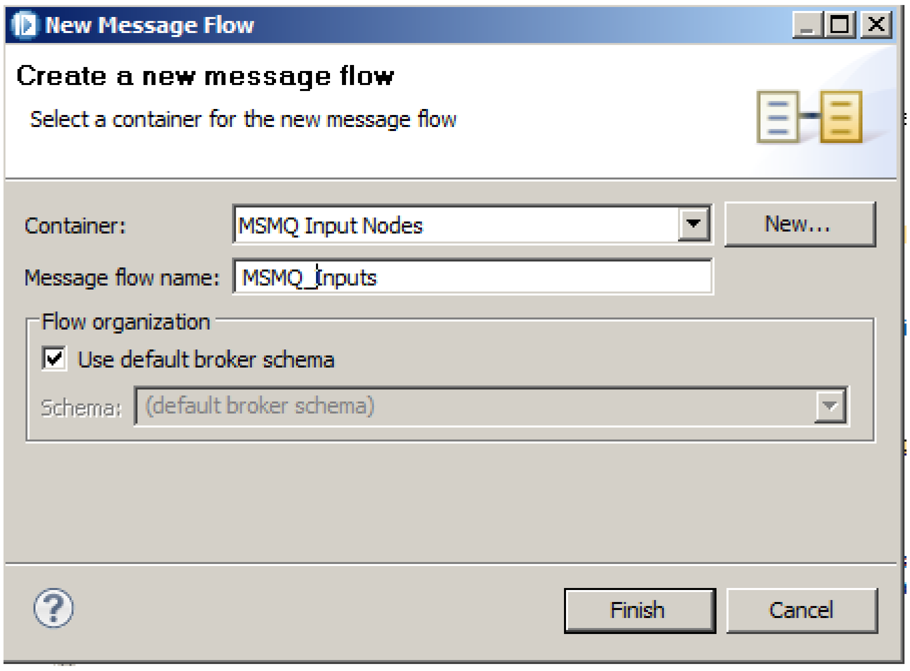
○ Shared Library

● Static Library

Select Static Library when you require each application to be deployed with its own private copy of the library for isolation. When a static library is updated, each application that references it must be repackaged and redeployed with the updated static library.

3. In the new library, create a new message flow.

   Name the flow MSMQ_Inputs



New Message Flow

Create a new message flow

Select a container for the new message flow

Container: MSMQ Input Nodes

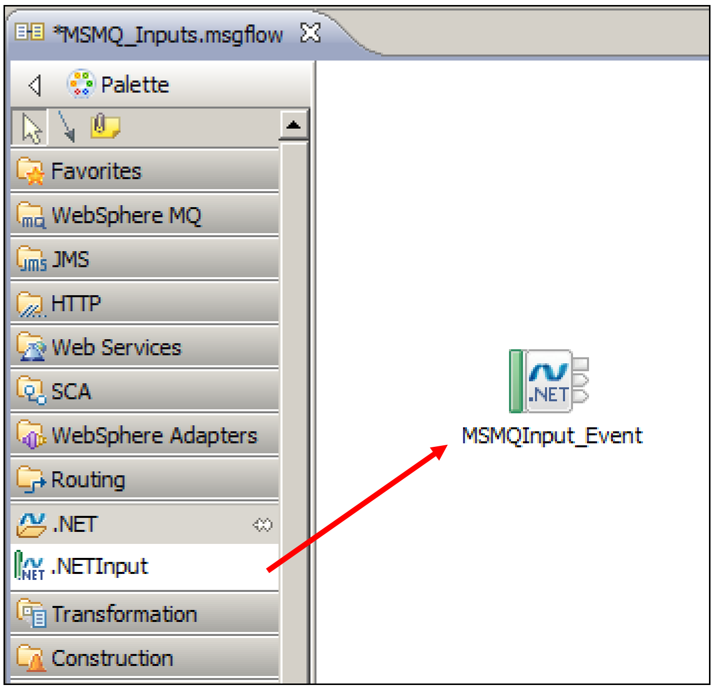
Message flow name: MSMQ_Inputs

Flow organization

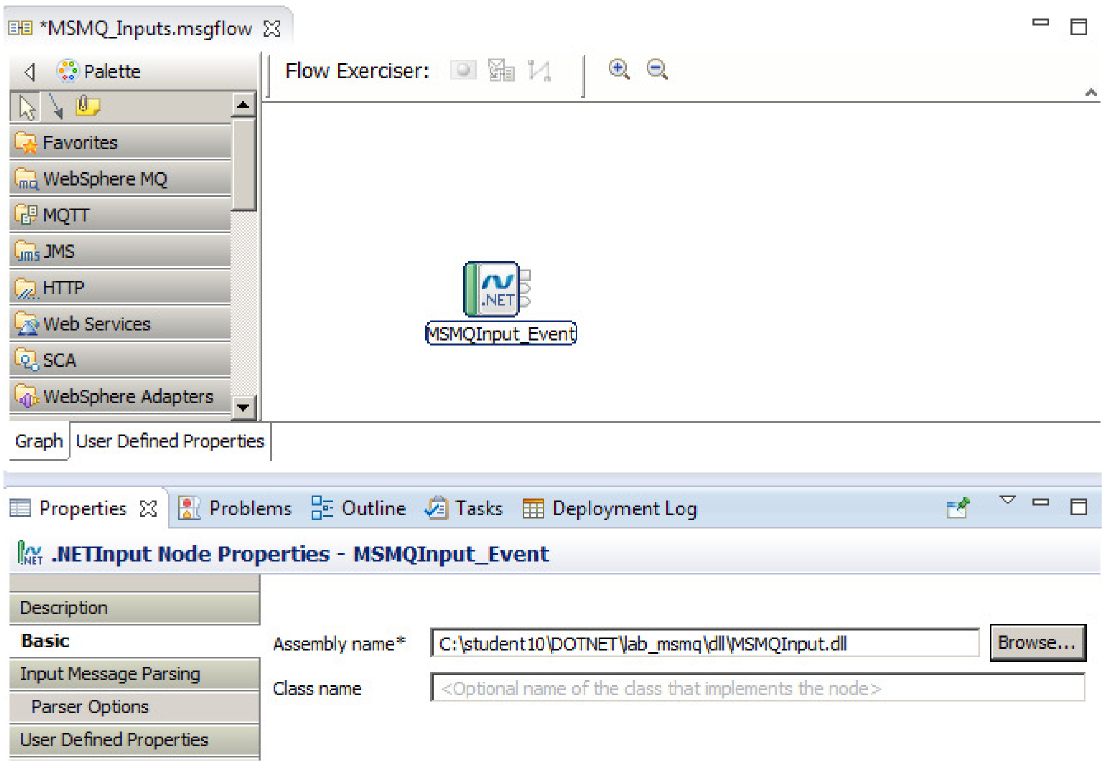
☑ Use default broker schema

Schema: (default broker schema)

4. Drop a .NET Input node onto the flow editor. Name it MSMQInput_Event.



5. On the Node Properties Basic tab, set the Assembly name to c:\student10\DOTNET\lab_msmq\dll\MSMQInput.dll.

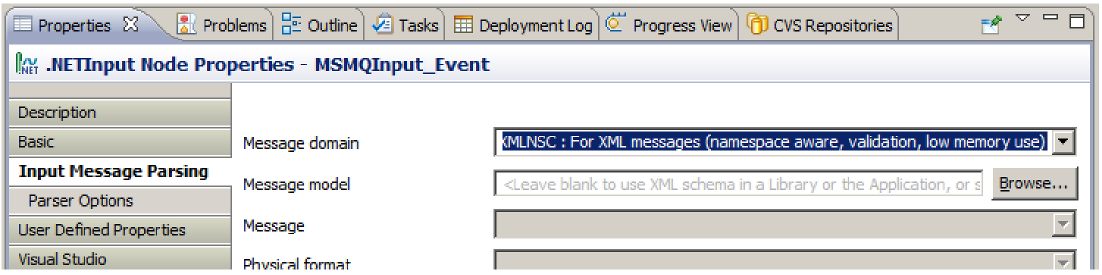


.NETInput Node Properties - MSMQInput_Event

Assembly name* C:\student10\DOTNET\lab_msmq\dll\MSMQInput.dll

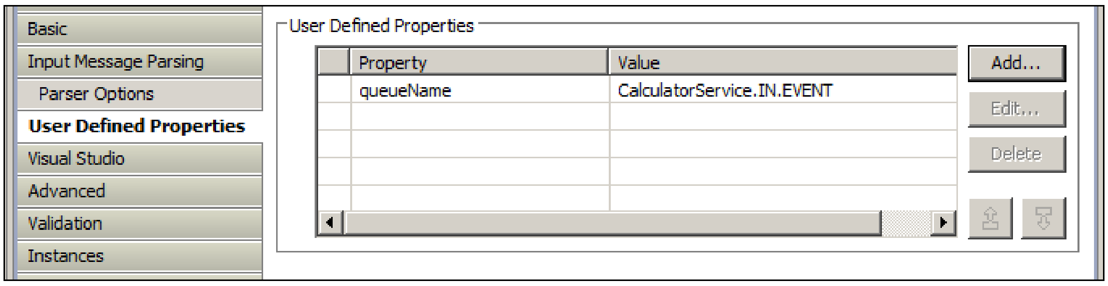
Class name <Optional name of the class that implements the node>

6. On the Input Message Parsing tab, set the Input Message Parsing to XMLNSC.



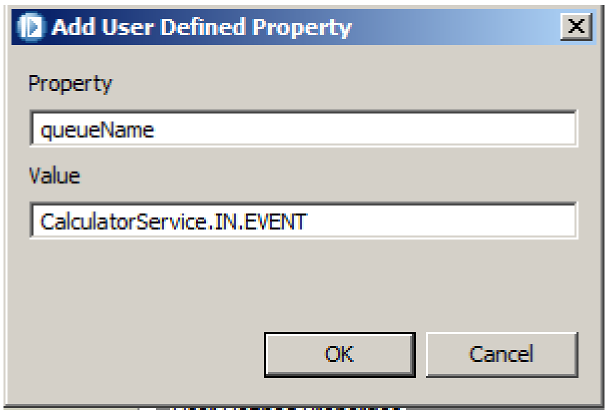
Message domain: XMLNSC : For XML messages (namespace aware, validation, low memory use)
Message model: <Leave blank to use XML schema in a Library or the Application, or s
Message
Physical format

7. On the User Defined Properties tab, create a new UDP as follows:
   - Property = queueName
   - Value = CalculatorService.IN.EVENT



Property	Value
queueName	CalculatorService.IN.EVENT

Use the Add button and the UDP dialogue to achieve this.

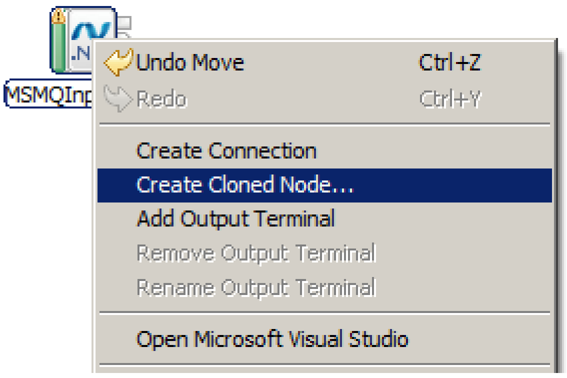


Property: queueName
Value: CalculatorService.IN.EVENT

8. Save the message flow.

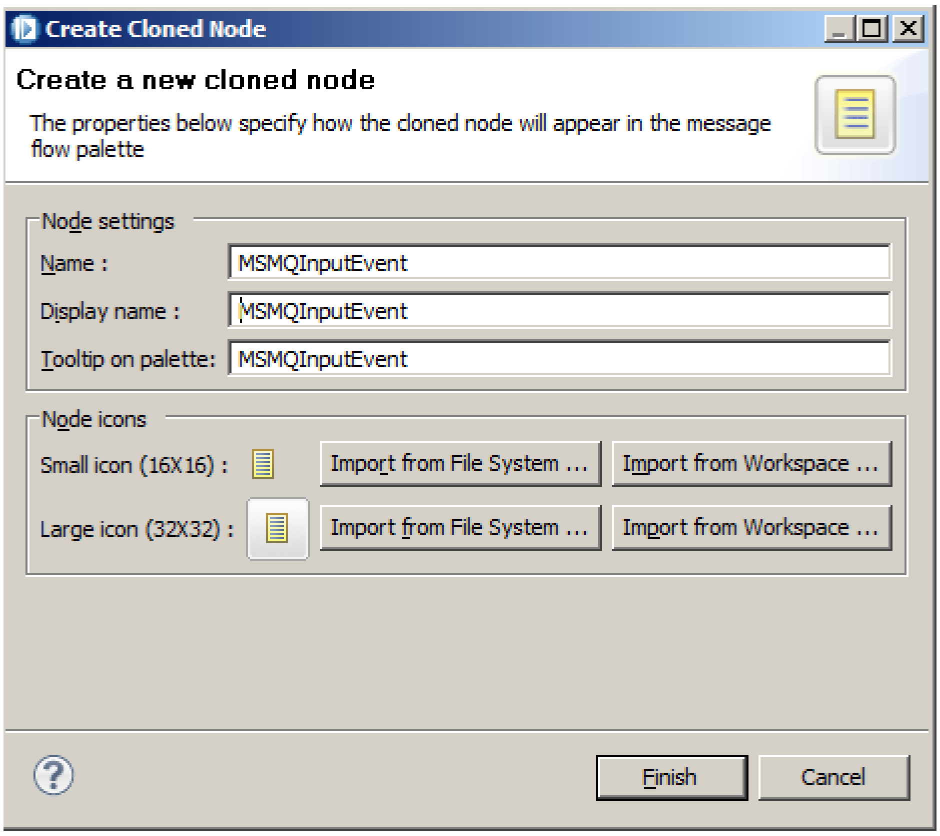
   Although the new node shows a warning, this is only because it is not connected to any other node. However, we can now use this node to create a Cloned Node.

   Right-click the new node, and select Create Cloned Node.



9. Name the cloned node "MSMQInputEvent, and choose appropriate names for the display name and tooltips. (Note, the underscore character is not permitted in the cloned node name).

   Click Finish.


Create Cloned Node

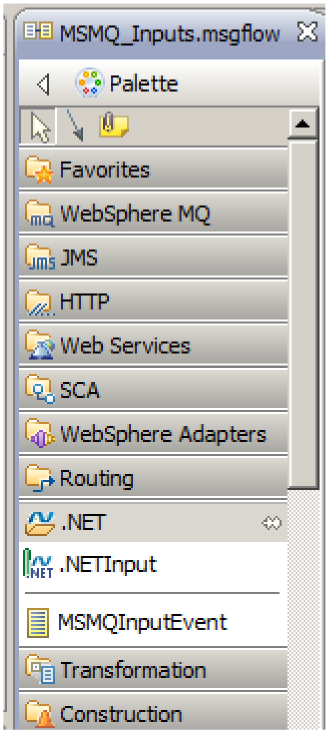
Create a new cloned node

The properties below specify how the cloned node will appear in the message flow palette

Node settings

Name :	MSMQInputEvent
Display name :	MSMQInputEvent
Tooltip on palette:	MSMQInputEvent

10. When the process completes, open the .NET category in the node palette. You will see a new node called MSMQInput_Event.



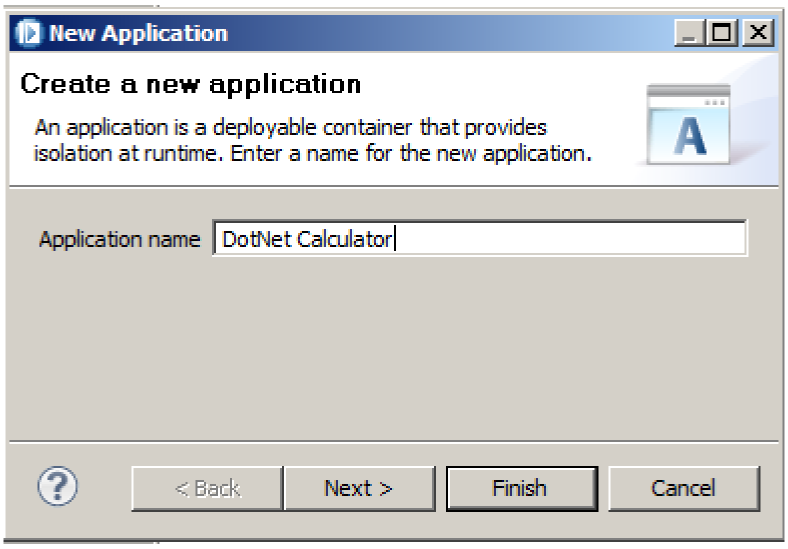
The MSMQ Input Node is now ready for use. You will now proceed to create a new application which will use this new node.

5. Create the Application and Message Flow

In the previous step you implemented the .NET code that receives an MSMQ Event Input, and used this to create a Cloned Node. In this step you will create a new Message Flow, and use the Cloned Node in a specific example.

5.1 Implement the Flow

1. Create a new application and name it "**DotNet Calculator**". Click Finish.



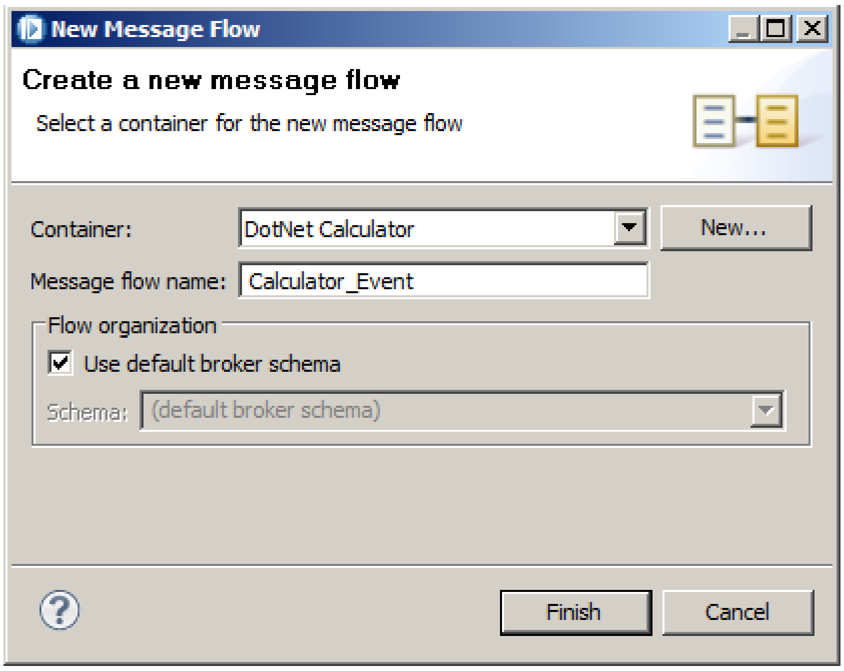
New Application

Create a new application

An application is a deployable container that provides isolation at runtime. Enter a name for the new application.

Application name: DotNet Calculator

2. Create a new Message Flow in this application. Name it "**Calculator_Event**" and click Finish.



New Message Flow

Create a new message flow

Select a container for the new message flow

Container: DotNet Calculator

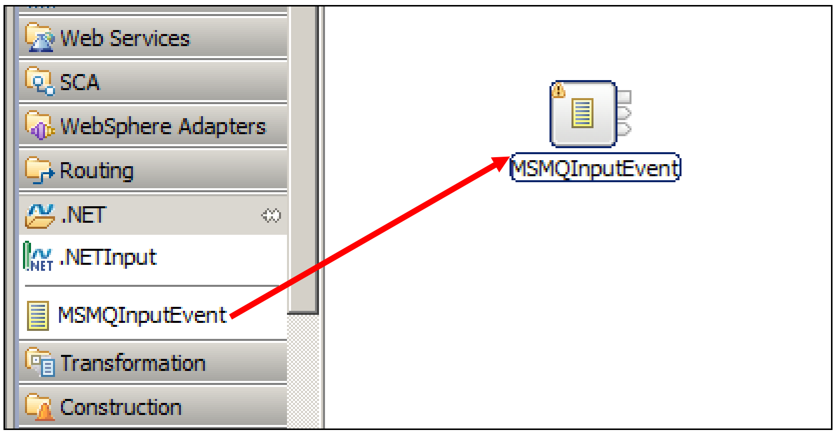
Message flow name: Calculator_Event

Flow organization

[✓] Use default broker schema

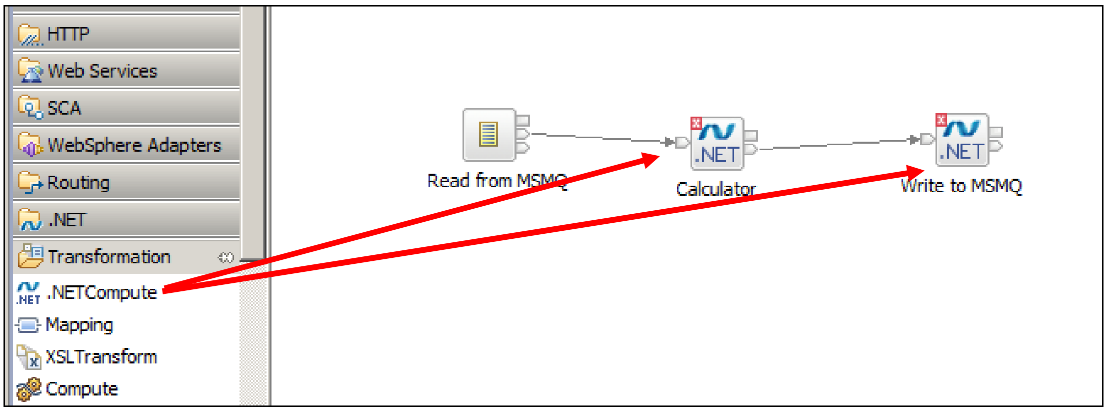
Schema: (default broker schema)

3. In the flow editor expand the “**.NET**” group, and drag the **MSMQInputEvent** node to the flow editor. Name the new node “**Read from MSMQ**”:



4. Expand the “**Transformation**” group, and drag two instances of the “**.NETCompute**” node to the flow, naming them “**Calculator**” and “**Write to MSMQ**”.

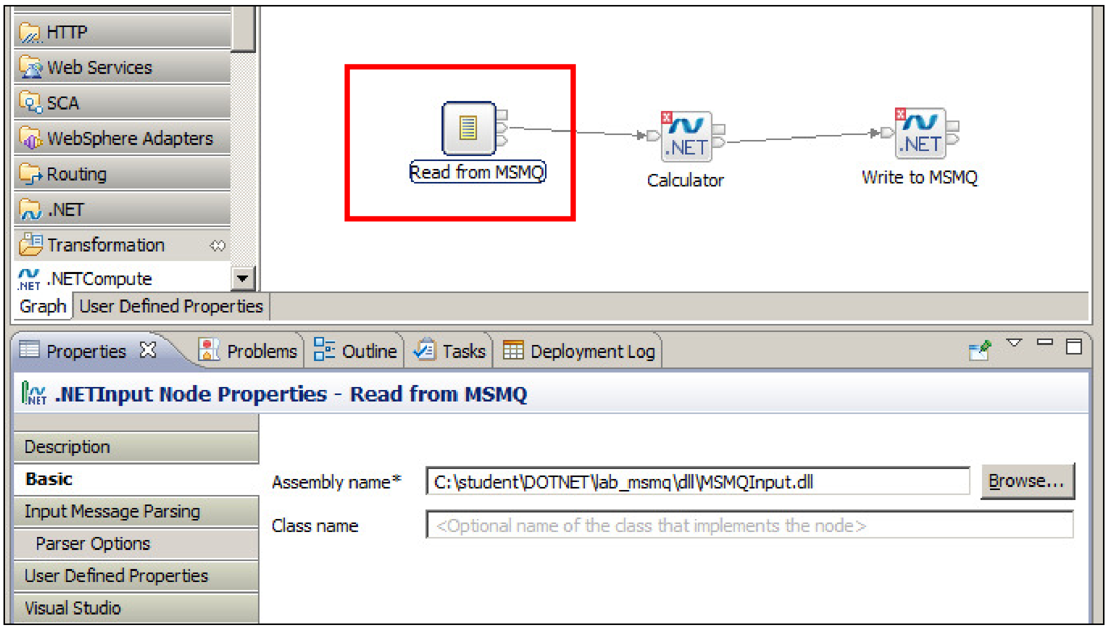
   Connect the nodes as shown, and save your work (Ctrl+S).



You'll notice that the Integration Toolkit has marked the new .NET nodes with an error. That's because there is no implementation code associated to them. You’ll fix this error next, by creating those associations.

5. First, click the “**Read from MSMQ**” node, and switch to the “**Basic**” tab.

   The node properties have been inherited from the Cloned Node. You can change these if you wish, but do not make any changes this time.



.NETInput Node Properties - Read from MSMQ

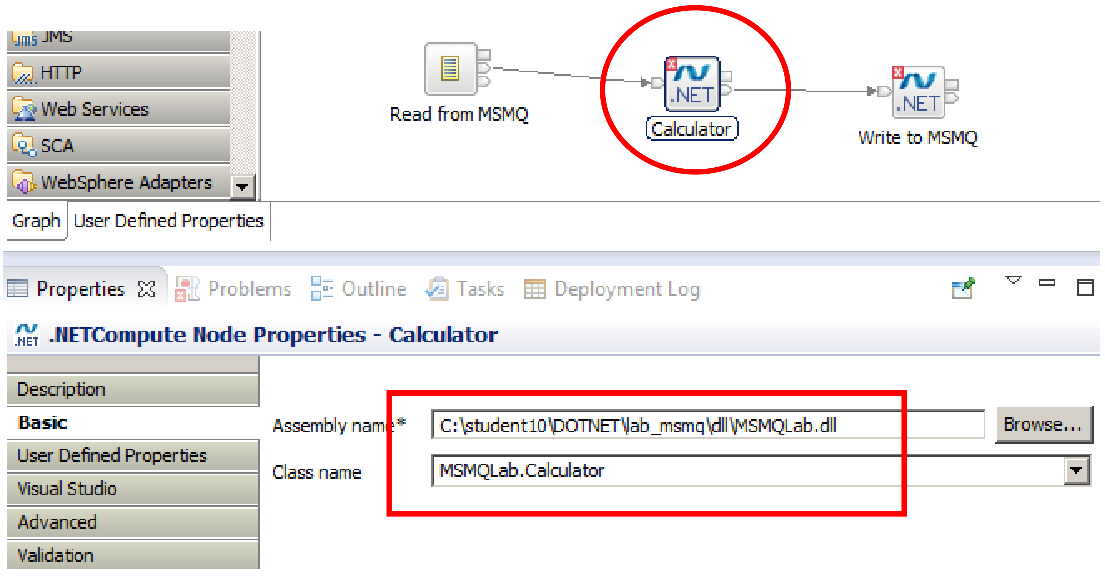
Assembly name* C:\student\DOTNET\lab_msmq\dll\MSMQInput.dll

Class name <Optional name of the class that implements the node>

6. Highlight the “**Calculator**” node and select the Basic Properties tab.

   Set the Assembly name = c:\ student10 \ DOTNET \ lab_msmq \ dll \ **MSMQLab.dll**

   Set the Class name = MSMQLab.Calculator



.NETCompute Node Properties - Calculator

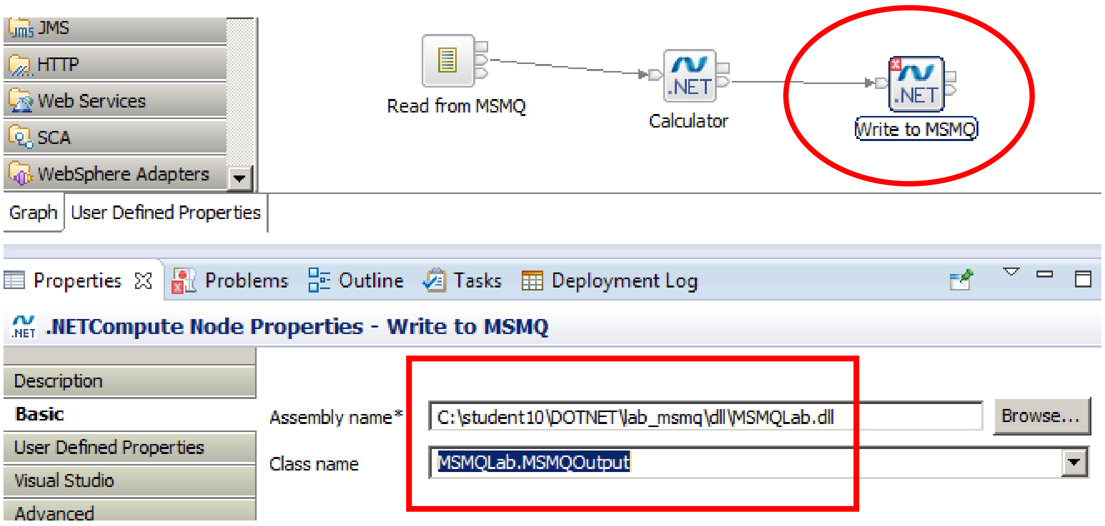
Assembly name* C:\student10\DOTNET\lab_msmq\dll\MSMQLab.dll

Class name MSMQLab.Calculator

7. Highlight the "**Write to MSMQ**" node select the Basic Properties tab.

   Set the Assembly name = c:\ student \ DOTNET \ lab_msmq \ dll \ MSMQLab.dll

   Set the Class name = MSMQLab.MSMQOutput.



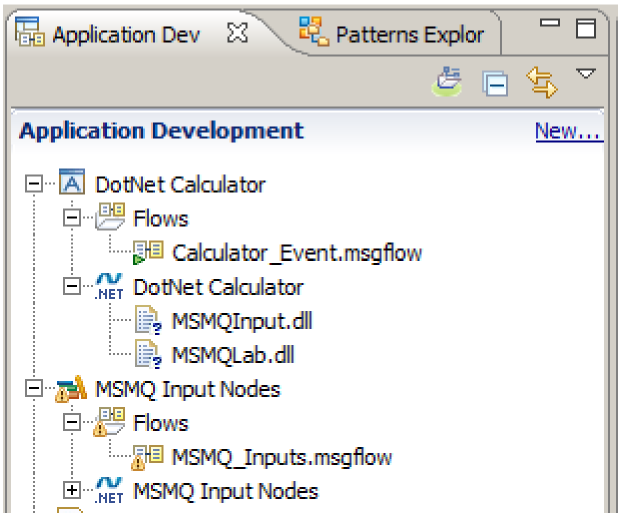
.NETCompute Node Properties - Write to MSMQ

Assembly name* C:\student10\DOTNET\lab_msmq\dll\MSMQLab.dll

Class name MSMQLab.MSMQOutput

8. Save your work by pressing "Ctrl+S". All errors should now have been resolved.

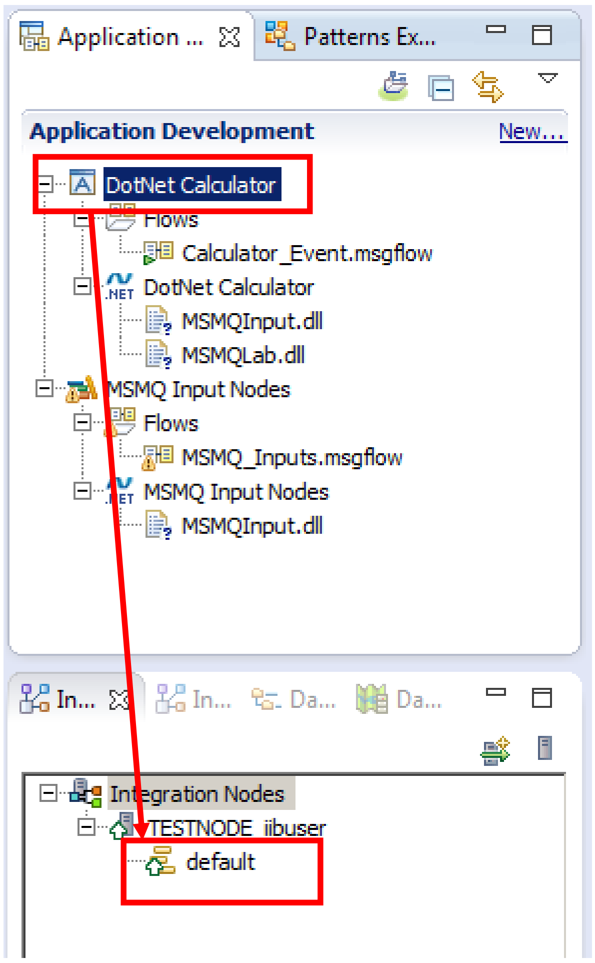
   Note that the .NET applications that are used by the Integration Bus nodes are shown in the Application Development project navigator. They are shown under the generated AppDomain, referenced by the same name as the primary project.



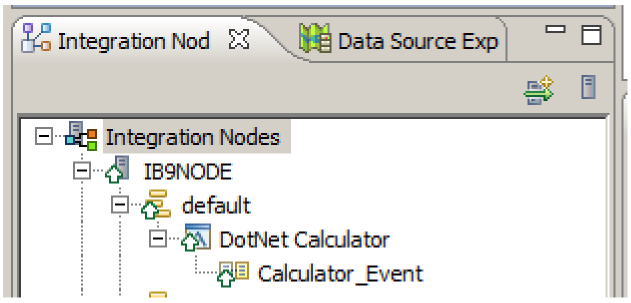
```
DotNet Calculator
  Flows
    Calculator_Event.msgflow
  DotNet Calculator
    MSMQInput.dll
    MSMQLab.dll
MSMQ Input Nodes
  Flows
    MSMQ_Inputs.msgflow
  MSMQ Input Nodes
```

5.2 Deployment to the Integration Node

1. In the Application Development navigator, drag the “**DotNet Calculator**” application onto the “**default**” Integration Server:



2. In the “Integration Nodes” view, expand the “**default**” Integration Server and the “DotNet Calculator” application to see the deployed artifacts.



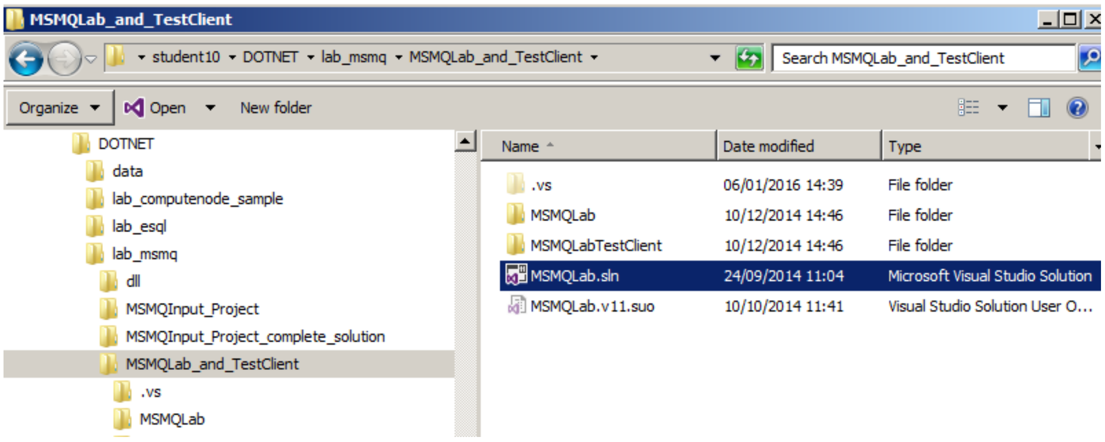
6. Testing the Event-Driven Scenario

The .NET code for the .NET Test application and the code that is used by the .NET Output Node are contained in a pre-built Visual Studio project, called MSMQLab. (Note – you should not make any changes to this project).

1. In Windows Explorer, navigate to

   `c:\ student10 \ DOTNET \ lab_msmq \ MSMQLab_andTestClient`

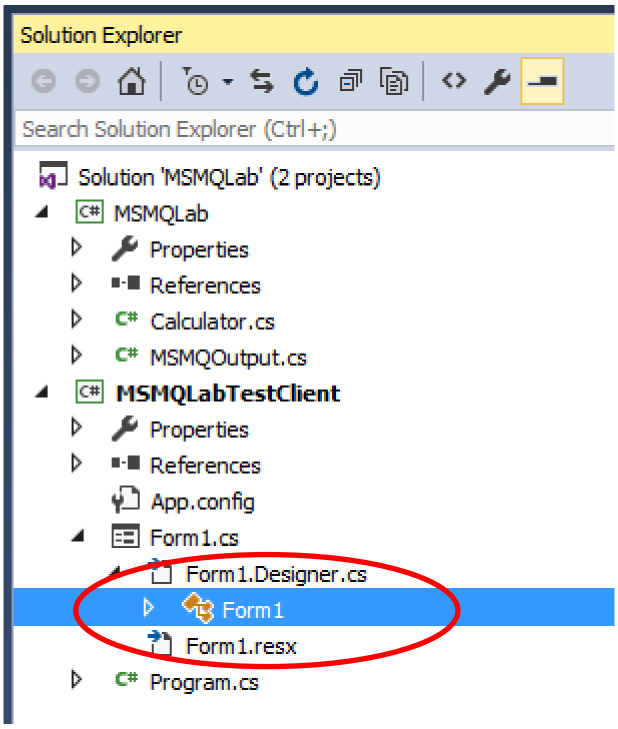
   Open (double-click) the file **MSMQLab.sln**, which will open a new instance of Visual Studio.



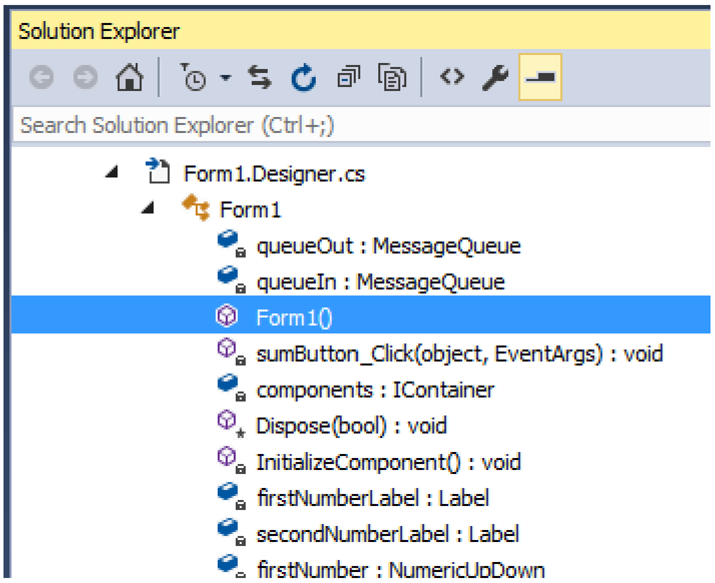
Name	Date modified	Type
.vs	06/01/2016 14:39	File folder
MSMQLab	10/12/2014 14:46	File folder
MSMQLabTestClient	10/12/2014 14:46	File folder
MSMQLab.sln	24/09/2014 11:04	Microsoft Visual Studio Solution
MSMQLab.v11.suo	10/10/2014 11:41	Visual Studio Solution User O…

2. In the “Solution Explorer” view, expand the “**MSMQLabTestClient**” project, then the “**Form1.cs**” file.

   Expand Form1.Designer.cs.

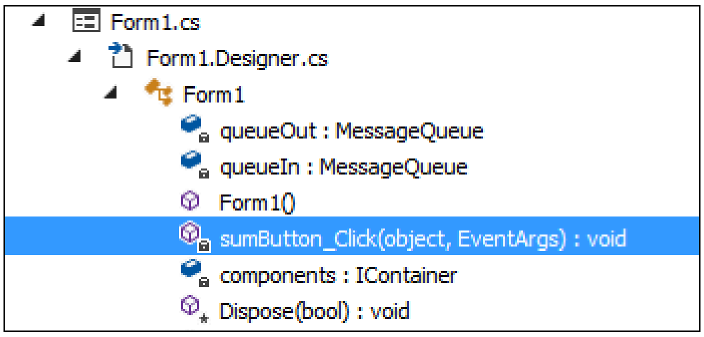


3. Click the Form1() method. This will show the basic logic for the design of the simple window that will drive this application.



Form1.Designer.cs
Form1
queueOut : MessageQueue
queueIn : MessageQueue
Form1()
sumButton_Click(object, EventArgs) : void
components : IContainer
Dispose(bool) : void
InitializeComponent() : void
firstNumberLabel : Label
secondNumberLabel : Label
firstNumber : NumericUpDown

4. Click the sumButton_Click method.

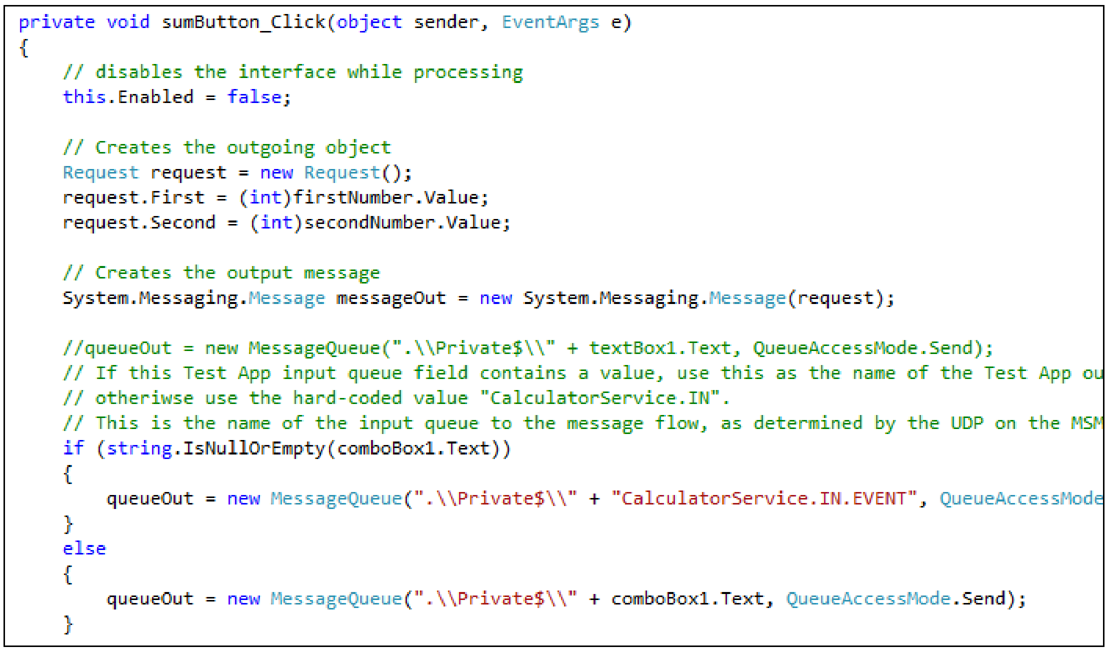


```
◢ Form1.cs
    ◢ Form1.Designer.cs
        ◢ Form1
            queueOut : MessageQueue
            queueIn : MessageQueue
            Form1()
            sumButton_Click(object, EventArgs) : void
            components : IContainer
            Dispose(bool) : void
```

The "**sumButton_Click**" method is called every time the "Sum!" button is clicked. It reads the input numbers, assembles the "Request" object, then sends it to a MSMQ queue, using .NET's default XML serializer (Formatter, in .NET's terminology). When a response is received, it uses the default serializer again to convert it to the "Response" object and presents the result to the user. Take a moment to examine its code in Visual Studio.

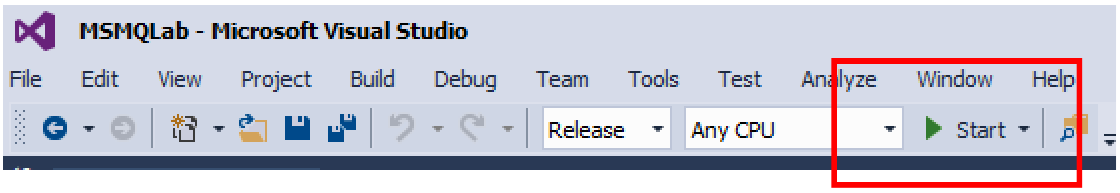
Below is a subset of the code. This section retrieves the queue name from the Test Application form, and uses it to send a message to MSMQ.

If a value is not entered into the Test Application input field for the queue name, the Test Application generates a default of CalculatorService.IN.EVENT.

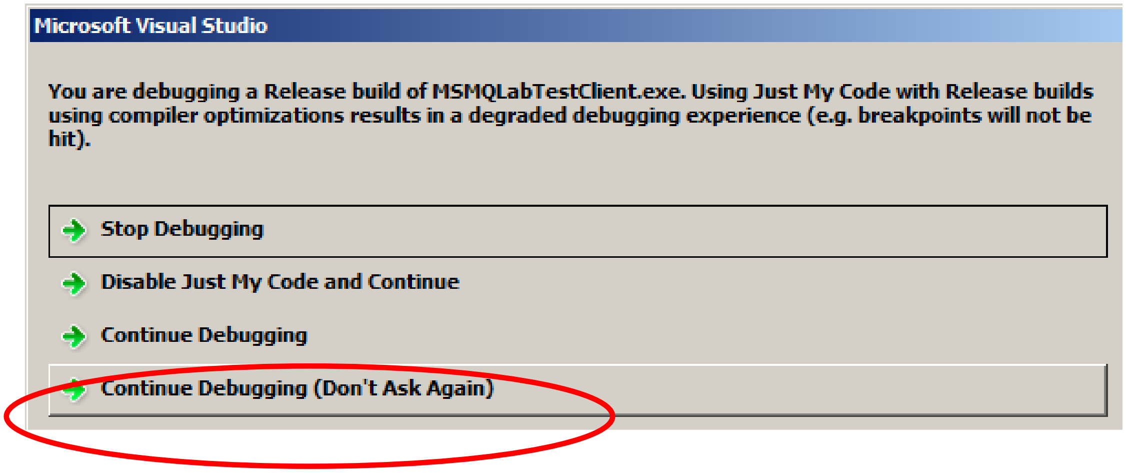


```
private void sumButton_Click(object sender, EventArgs e)
{
    // disables the interface while processing
    this.Enabled = false;

    // Creates the outgoing object
    Request request = new Request();
    request.First = (int)firstNumber.Value;
    request.Second = (int)secondNumber.Value;

    // Creates the output message
    System.Messaging.Message messageOut = new System.Messaging.Message(request);

    //queueOut = new MessageQueue(".\\Private$\\" + textBox1.Text, QueueAccessMode.Send);
    // If this Test App input queue field contains a value, use this as the name of the Test App ou
    // otherwise use the hard-coded value "CalculatorService.IN".
    // This is the name of the input queue to the message flow, as determined by the UDP on the MSM
    if (string.IsNullOrEmpty(comboBox1.Text))
    {
        queueOut = new MessageQueue(".\\Private$\\" + "CalculatorService.IN.EVENT", QueueAccessMode
    }
    else
    {
        queueOut = new MessageQueue(".\\Private$\\" + comboBox1.Text, QueueAccessMode.Send);
    }
```

5. Click the "**Start**" button.


MSMQLab - Microsoft Visual Studio
File Edit View Project Build Debug Team Tools Test Analyze Window Help
Release
Any CPU
Start

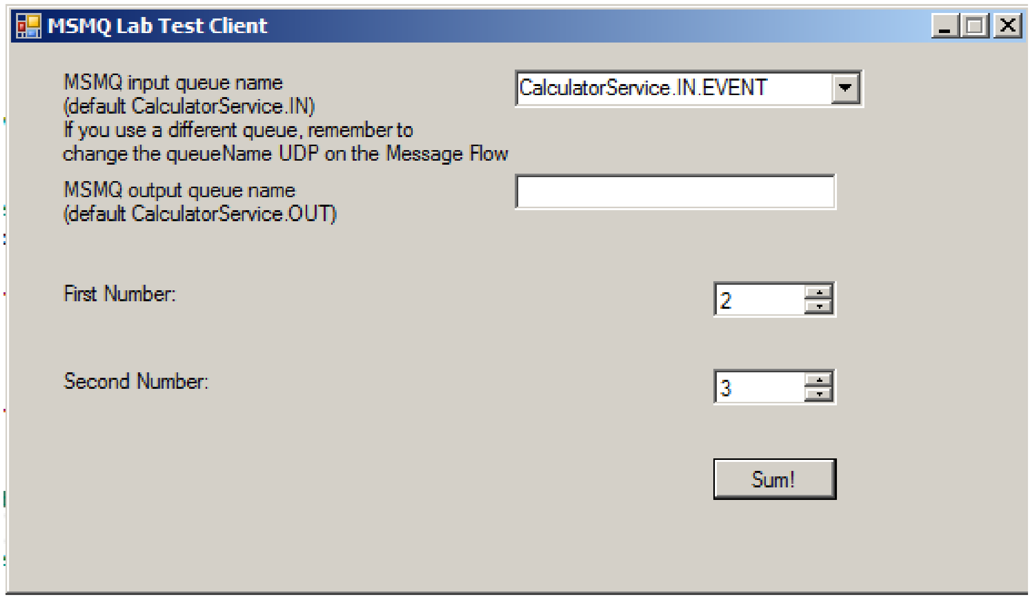
6. You may see a warning message from Visual Studio. If you do, click Continue Debugging (Don't Ask Again).


Microsoft Visual Studio

You are debugging a Release build of MSMQLabTestClient.exe. Using Just My Code with Release builds using compiler optimizations results in a degraded debugging experience (e.g. breakpoints will not be hit).

Stop Debugging
Disable Just My Code and Continue
Continue Debugging
Continue Debugging (Don't Ask Again)

7. The Test Client Application will open. In the MSMQ input queue name field, specify CalculatorService.IN.EVENT (or leave blank to default), provide two numbers for addition, and click the "**Sum!**" button:


MSMQ Lab Test Client

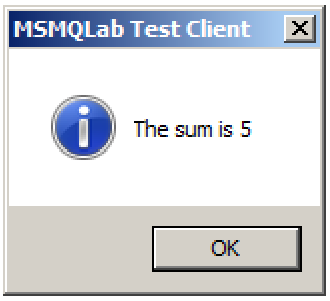
MSMQ input queue name (default CalculatorService.IN) If you use a different queue, remember to change the queueName UDP on the Message Flow: CalculatorService.IN.EVENT

MSMQ output queue name (default CalculatorService.OUT):

First Number: 2

Second Number: 3

8. When you see the Dialog below, it means the message was sent to the named MSMQ queue, was processed by the Integration Bus application, the calculation was done, and the result has been returned to the MSMQ queue CalculatorService.OUT queue, and received by the .NET Test Application.



MSMQLab Test Client

The sum is 5

The .NET Test Application provides some very basic examples of how to handle screen input and output, and how to specify MSMQ queue names. It’s not the intention of this lab to provide a NET primer, but you may be able to extend this example as required for simple scenarios.

9. Close the Test Application.

This concludes the first part of the .NETInput Node lab.

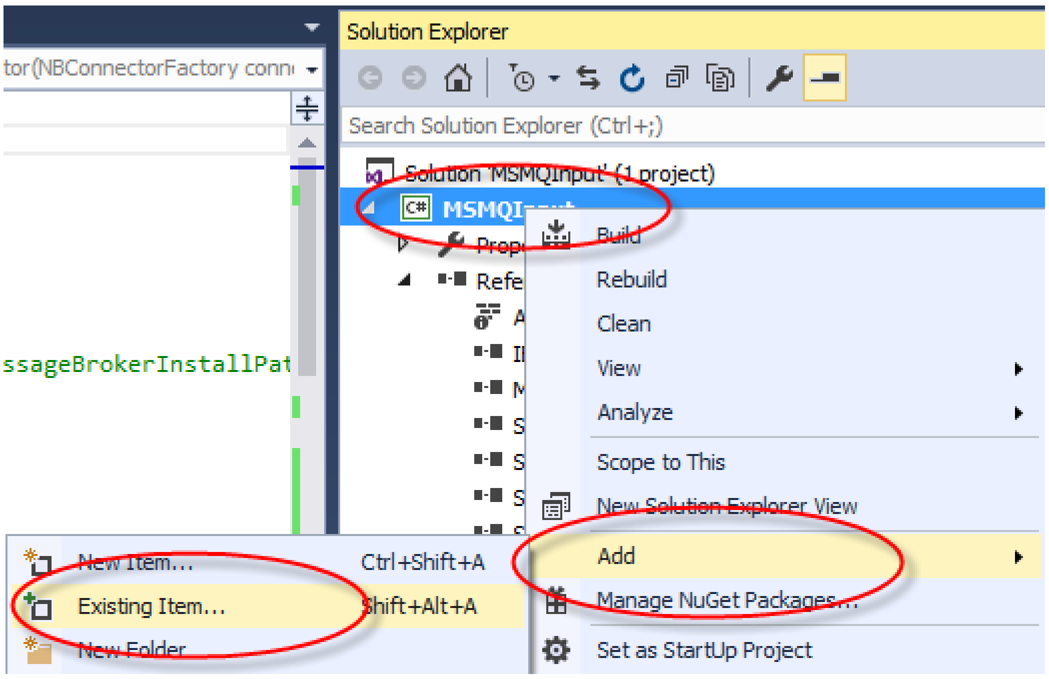
7. Modifying the flow to use Polling

In this part of the Lab, you will change the way the flow is started when a message arrives in the MSMQ input queue. In the first part, we used an "Event-Driven" input node, which registers an UserDelegate method on its "Start()" method. In this part, you will use a "Polling" approach, where the Integration Bus will poll a method in your code, in order to check if there is a message available to be processed.

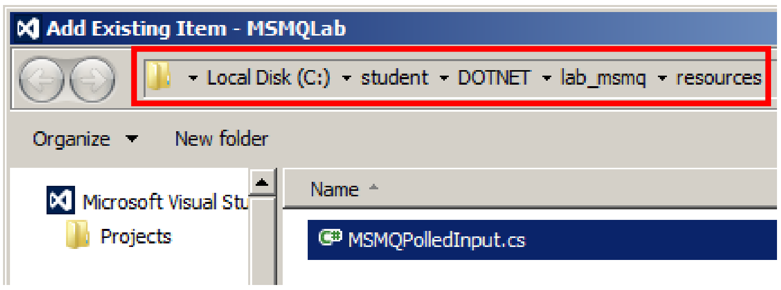
7.1 Implement the changes to the MSMQInput .NET code

1. Return to the Visual Studio editor with the MSMQInput project (not the MSMQLab project).

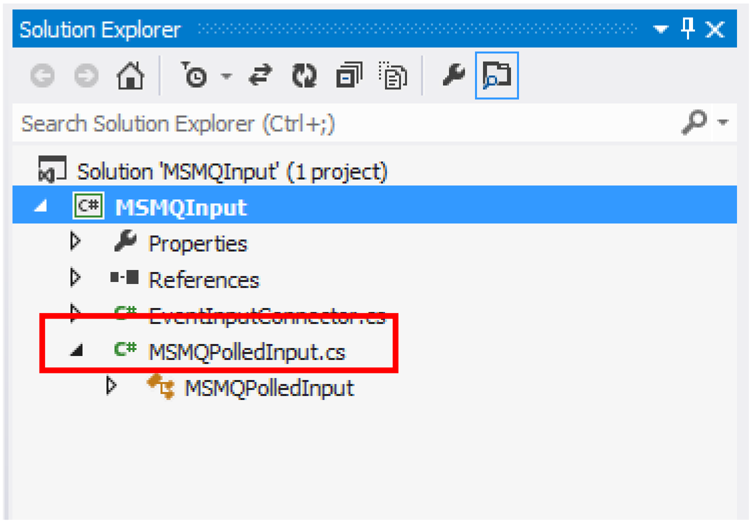
   In the Solution Explorer view, right-click on the "MSMQInput" project and click Add ➔ Existing Item.



2. Navigate to the "**C:\student10\DOTNET\lab_msmq\resources**" folder, and select the "**MSMQPolledInput.cs**" file.


Add Existing Item - MSMQLab
Local Disk (C:) ▸ student ▸ DOTNET ▸ lab_msmq ▸ resources
Organize
New folder
Microsoft Visual Stu
Projects
Name
MSMQPolledInput.cs

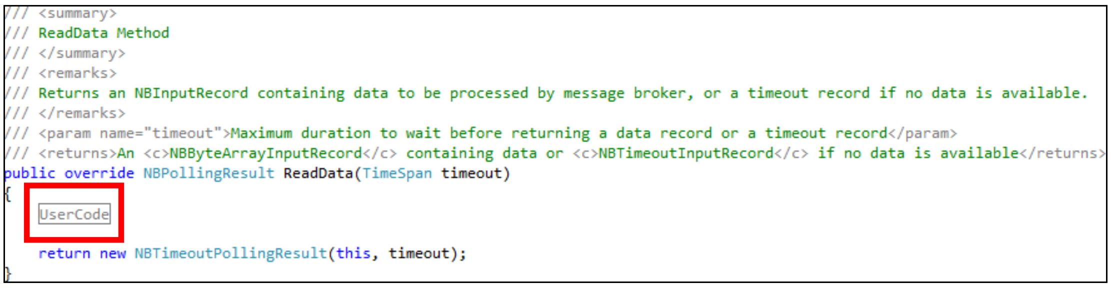
3. After importing, open it by double-clicking MSMQPolledInput.cs in the “Solution Explorer” view.



4. Take a minute to examine the imported class; You'll notice that it has the same basic methods of the event-driven class, but instead of having the “**UserDelegate**” region, it has a ReadData() method.

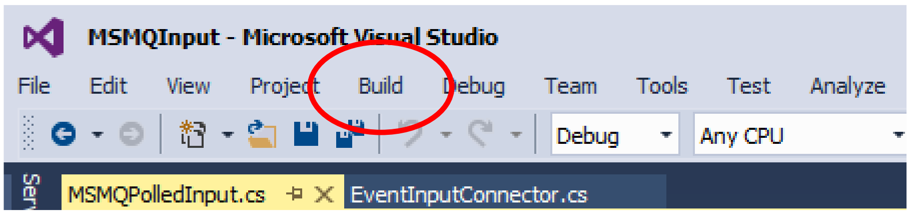
   This method is repeatedly called by the Integration Bus at a specific interval; That interval can be specified in the method’s return value, if the return value is an instance of the NBTimeoutPollingResult class; Returning an instance of that class will inform the Integration Bus that no data was available for processing at that time.

   Conversely, if any data is found, the method must return an instance of the “NBByteArrayPollingResult” class, which will cause the Integration Bus to immediately call your method again, to establish if more data is available. You can expand the #**UserCode** region below to understand this logic:



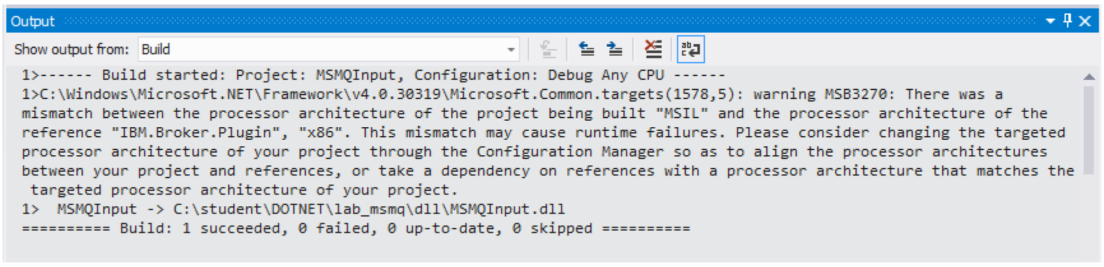
```
/// <summary>
/// ReadData Method
/// </summary>
/// <remarks>
/// Returns an NBInputRecord containing data to be processed by message broker, or a timeout record if no data is available.
/// </remarks>
/// <param name="timeout">Maximum duration to wait before returning a data record or a timeout record</param>
/// <returns>An <c>NBByteArrayInputRecord</c> containing data or <c>NBTimeoutInputRecord</c> if no data is available</returns>
public override NBPollingResult ReadData(TimeSpan timeout)
{
    UserCode

    return new NBTimeoutPollingResult(this, timeout);
}
```

5. Build the solution, either by pressing “F7” or using the “Build” menu.



WARNING!
If the Test Application is still running this step will fail! Close it before trying to build the solution again.

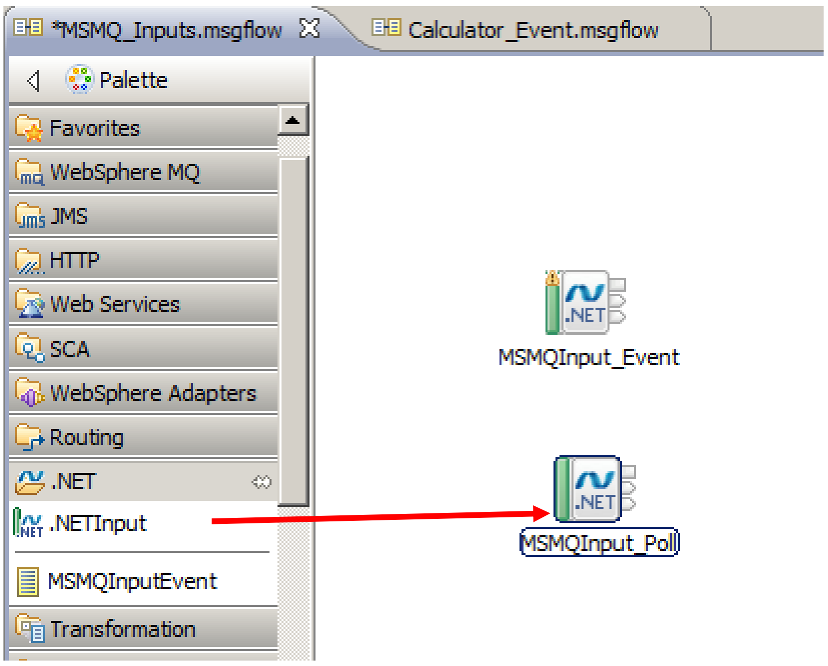
6. The Output view is shown, with the following result. Ensure, as before, that the updated dll is written to the \ lab_msmq \ dll folder.



```
1>------ Build started: Project: MSMQInput, Configuration: Debug Any CPU ------
1>C:\Windows\Microsoft.NET\Framework\v4.0.30319\Microsoft.Common.targets(1578,5): warning MSB3270: There was a
mismatch between the processor architecture of the project being built "MSIL" and the processor architecture of the
reference "IBM.Broker.Plugin", "x86". This mismatch may cause runtime failures. Please consider changing the targeted
processor architecture of your project through the Configuration Manager so as to align the processor architectures
between your project and references, or take a dependency on references with a processor architecture that matches the
 targeted processor architecture of your project.
1>  MSMQInput -> C:\student\DOTNET\lab_msmq\dll\MSMQInput.dll
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```

7.2 Create a new MSMQ Input Node for Polling

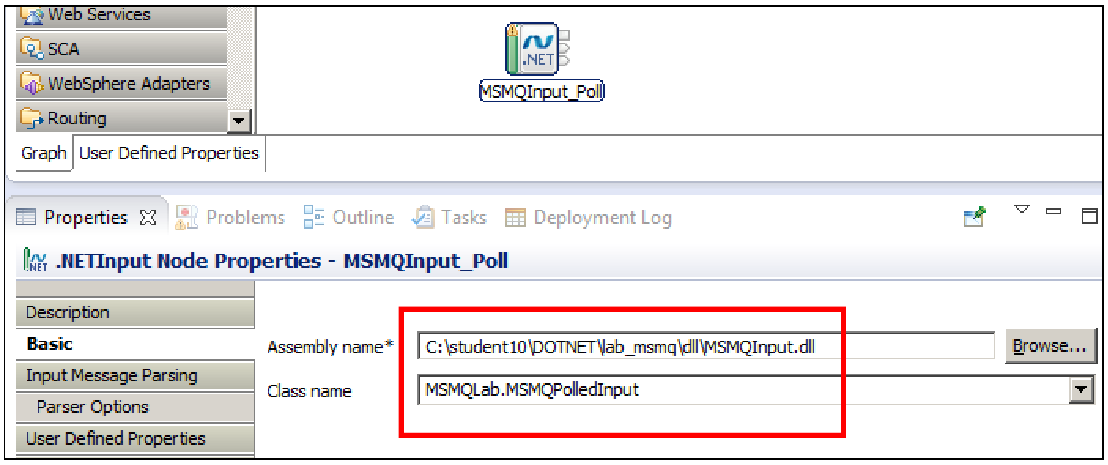
1. Back to the Integration Toolkit, in the MSMQ_Inputs message flow, add a new .NETInput node to the flow, and name it MSMQInput_Poll.



2. On the Basic Properties tab, set the Assembly name to c:\student10\DOTNET\lab_msmq\dll\MSMQInput.dll.

   You will notice that the Class name property now shows a drop-down option. This is because the MSMQInput dll now has two classes, so the node has to be able to distinguish between them. Select the Polled Input class for this node.

   Save the flow.

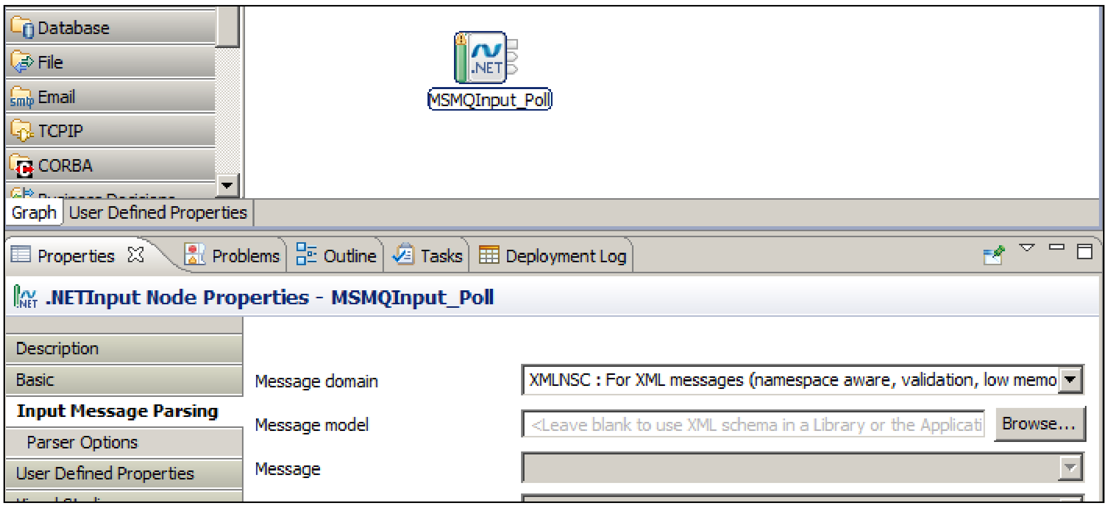


.NETInput Node Properties - MSMQInput_Poll

Assembly name* C:\student10\DOTNET\lab_msmq\dll\MSMQInput.dll

Class name MSMQLab.MSMQPolledInput

3. Set the Input Message Parsing to XMLNSC.



.NETInput Node Properties - MSMQInput_Poll

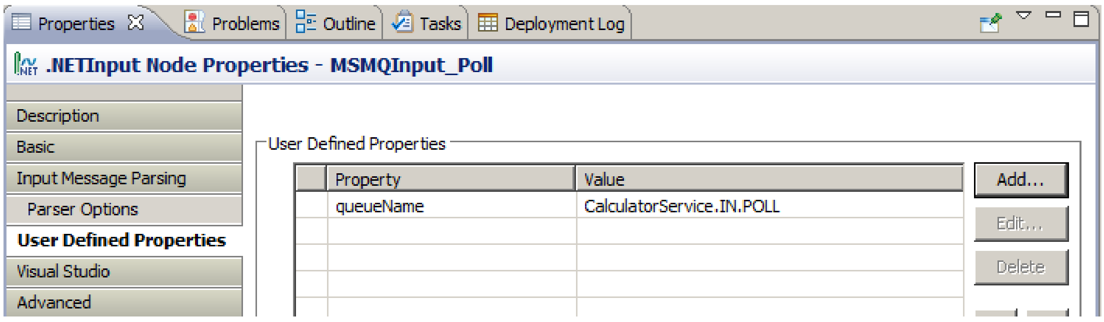
Message domain: XMLNSC : For XML messages (namespace aware, validation, low memo

Message model: <Leave blank to use XML schema in a Library or the Applicati

Message:

4. Create a new User Defined Property **queueName**, and set the value to **CalculatorService.IN.POLL**.

Save the flow.

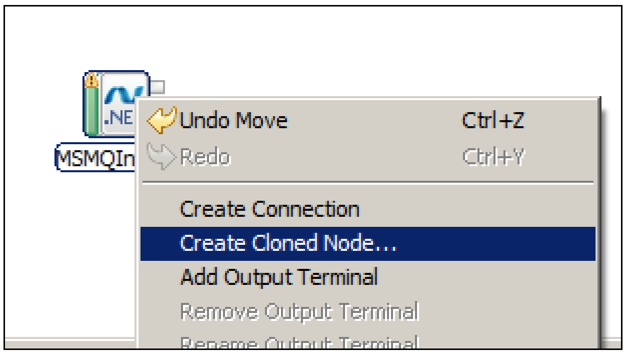


.NETInput Node Properties - MSMQInput_Poll

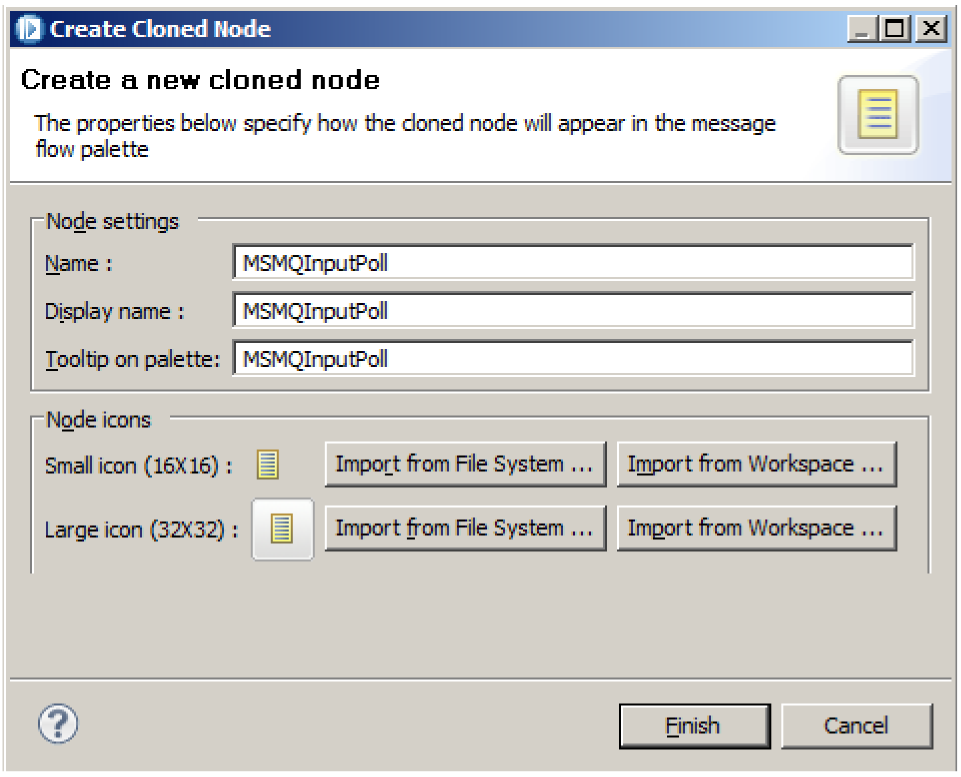
User Defined Properties

Property	Value
queueName	CalculatorService.IN.POLL

5. Create a second Cloned Node. Right-click the MSMQInput_Poll node and select Create Cloned Node.



6. Name it **MSMQInputPoll**. Set appropriate values for the Display Name and Tooltip values, and click Finish.



Create Cloned Node

Create a new cloned node

The properties below specify how the cloned node will appear in the message flow palette

Node settings

Name : MSMQInputPoll

Display name : MSMQInputPoll

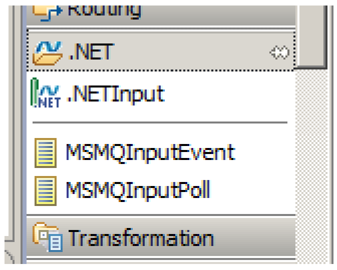
Tooltip on palette: MSMQInputPoll

Node icons

Small icon (16X16) :

Large icon (32X32) :

7. You will now have a new node in the node palette called MSMQInputPoll.



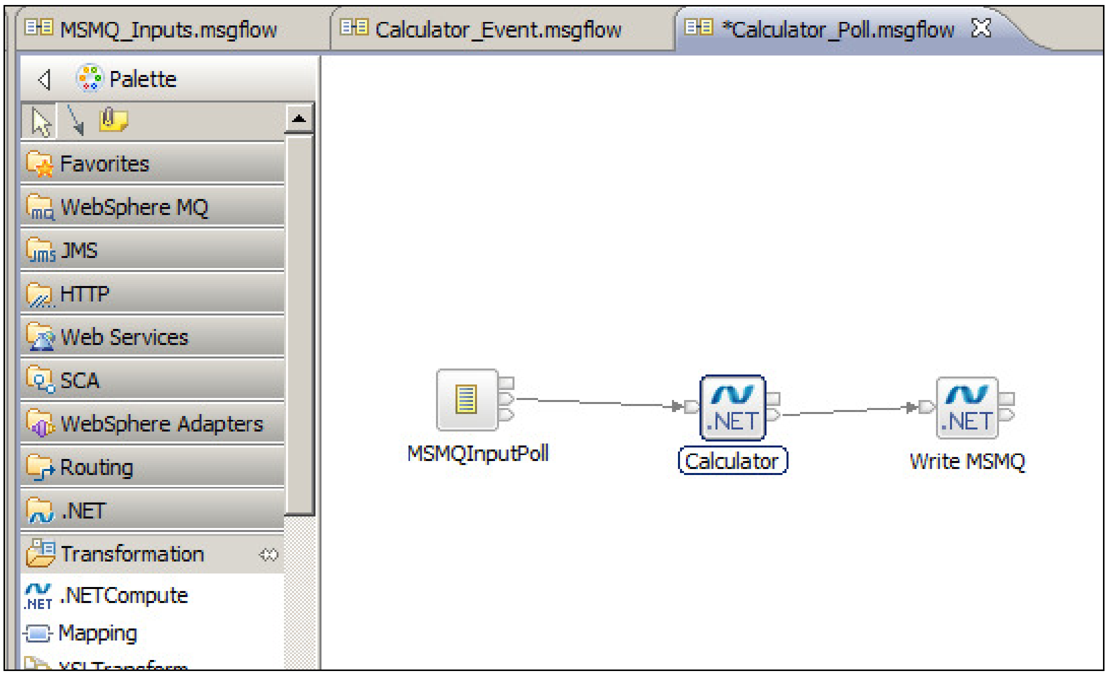
8. In the DotNet Calculator application, create a new message flow called Calculator_Poll.

   In the Calculator_Poll flow drop an instance of the MSMQInputPoll onto the flow editor. The node properties will be derived from the Cloned Node, so no changes are necessary.

   Replicate the other nodes of the flow as before, by using two .NET Compute nodes. Set the .NET assemblies and classes as in the previous flow:

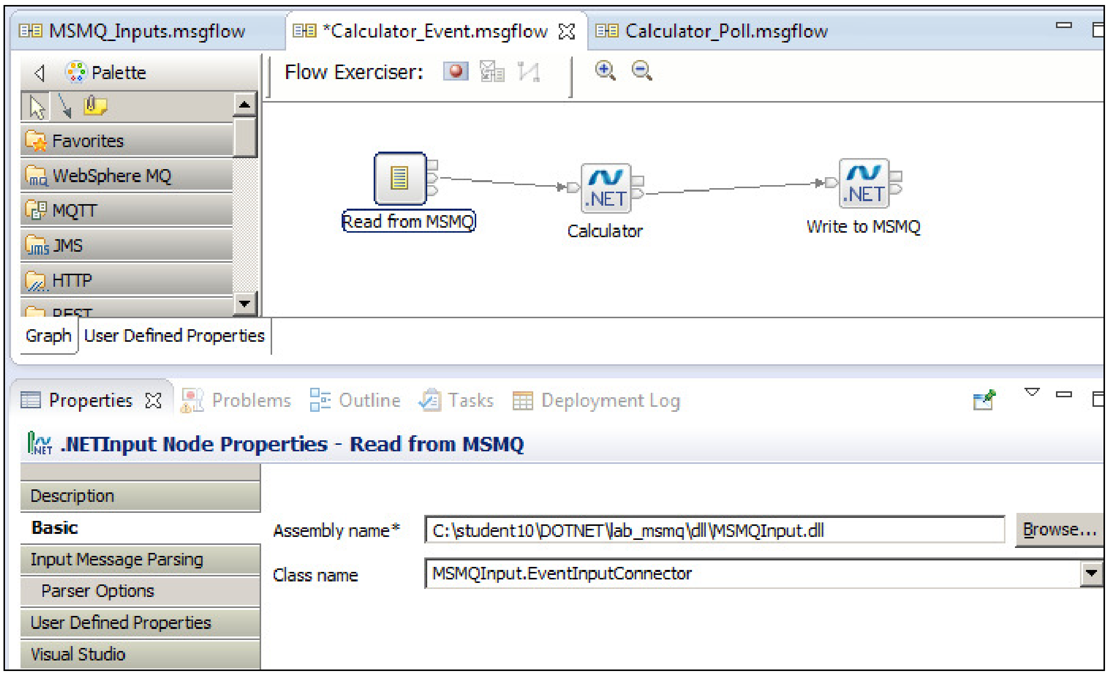
   - Assembly name (both nodes) = MSMQLab.dll.
   - Calculator node class name = MSMQLab.Calculator
   - Write MSMQ class name = MSMQLab.MSMQOutput

   Connect the nodes as normal, and save the flow.



9. One final change must be made to the previous message flow, Calculator_Event. Because you are now using the same .NET project in two input nodes in the same application, you must also define the specific class for the existing MSMQ Event node. (If you don't do this, the deploy will fail, because the application cannot resolve the correct .NET class).

   In the Calculator_Event flow, highlight the "Read from MSMQ" node, and on the Basic tab, set the Class Name to EventInputConnector.



.NETInput Node Properties - Read from MSMQ

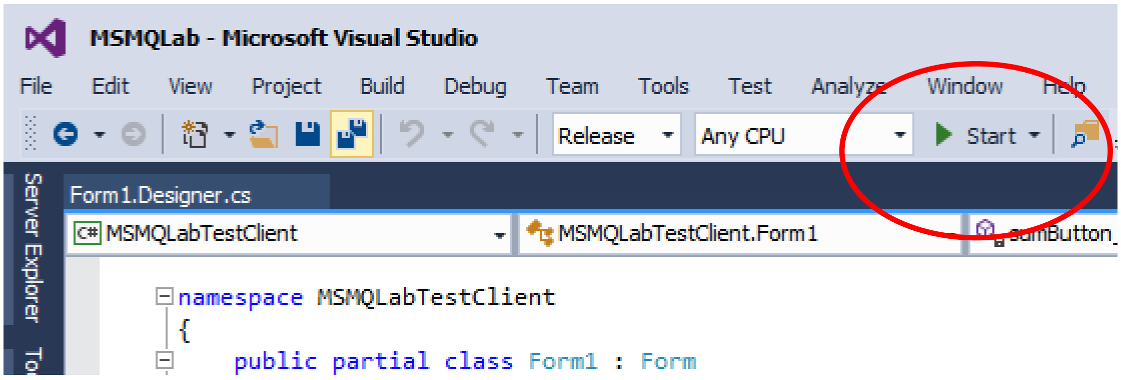
Assembly name* C:\student10\DOTNET\lab_msmq\dll\MSMQInput.dll

Class name MSMQInput.EventInputConnector

10. Save the message flow, and deploy the DotNet Calculator application again.

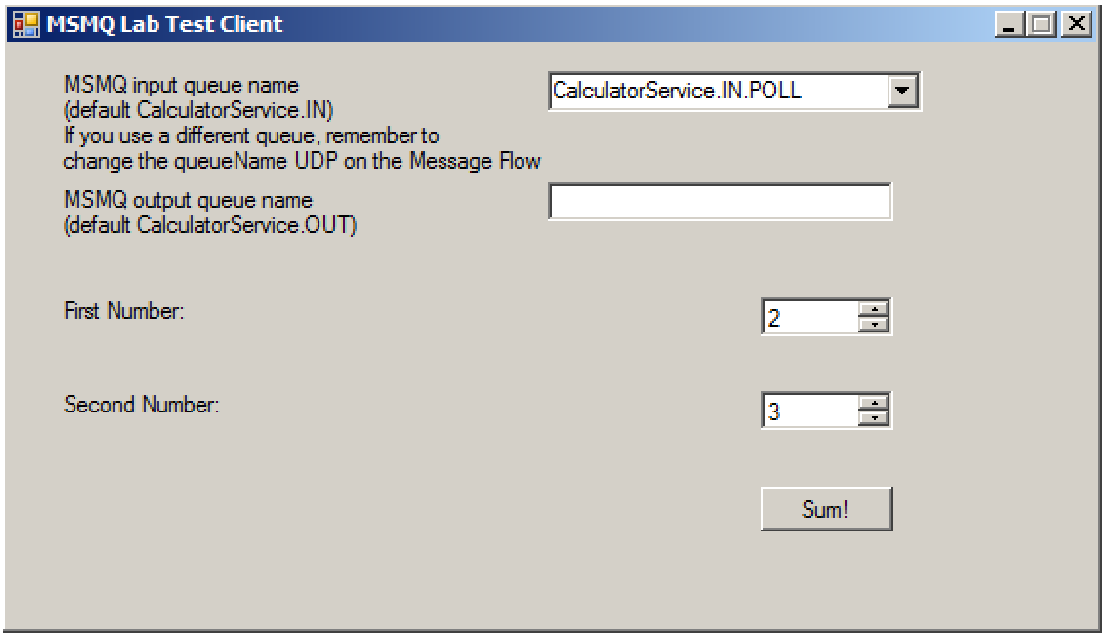
11. Retest the application in the same way as before.

    Switch to the MSMQLab project, and in the Test Application (Form1), click Start.



```
namespace MSMQLabTestClient
{
    public partial class Form1 : Form
```

12. In the Test Application, set the name of the queue to CalculatorService.IN.POLL, and click Sum.



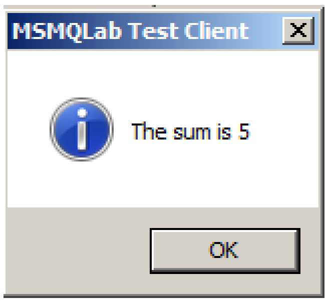
MSMQ input queue name
(default CalculatorService.IN)
If you use a different queue, remember to
change the queueName UDP on the Message Flow: CalculatorService.IN.POLL

MSMQ output queue name
(default CalculatorService.OUT):

First Number: 2

Second Number: 3

The application should run as before.



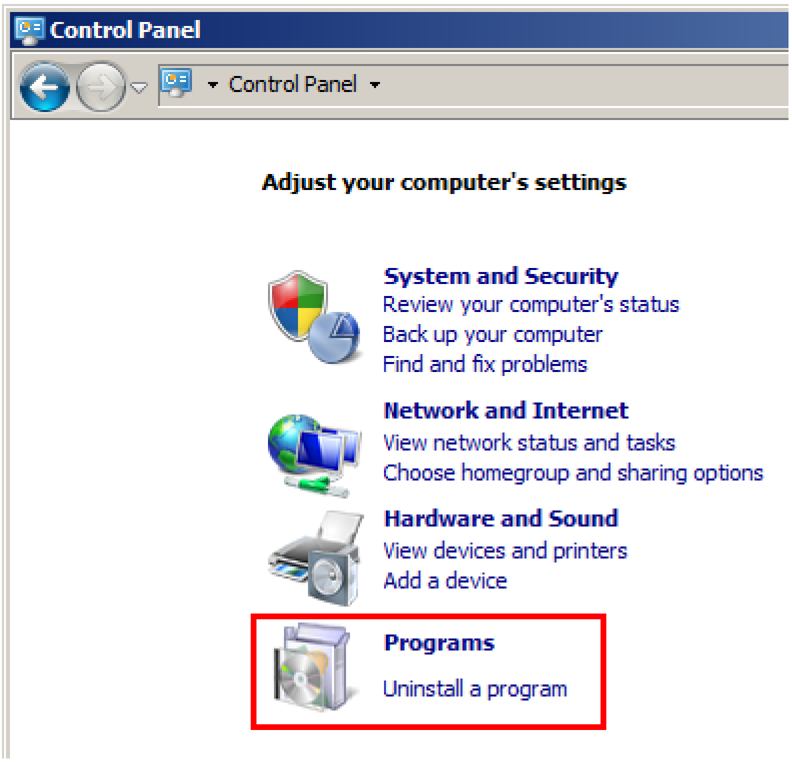
The sum is 5

This concludes the second part of the .NETInput Node lab.

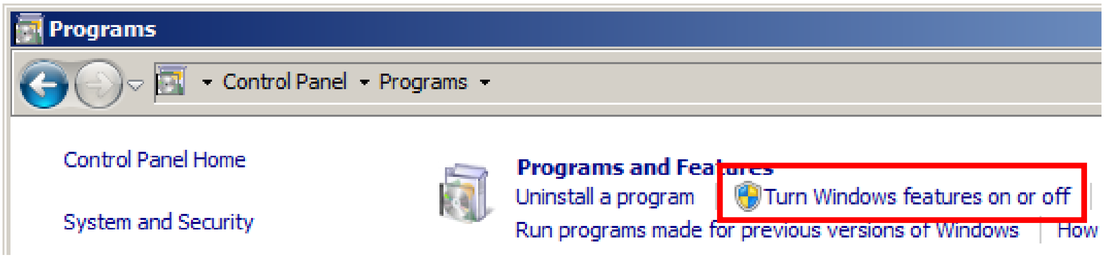
8. Appendix A: Enabling the MSMQ Component

If you're not using the supplied VMWare image, probably the Microsoft Windows installation that you're using doesn't have the MSMQ component installed. In order to install/enable it, follow the steps bellow.

1. From the Windows Start menu, click "Control Panel".
2. Click "**Programs**":



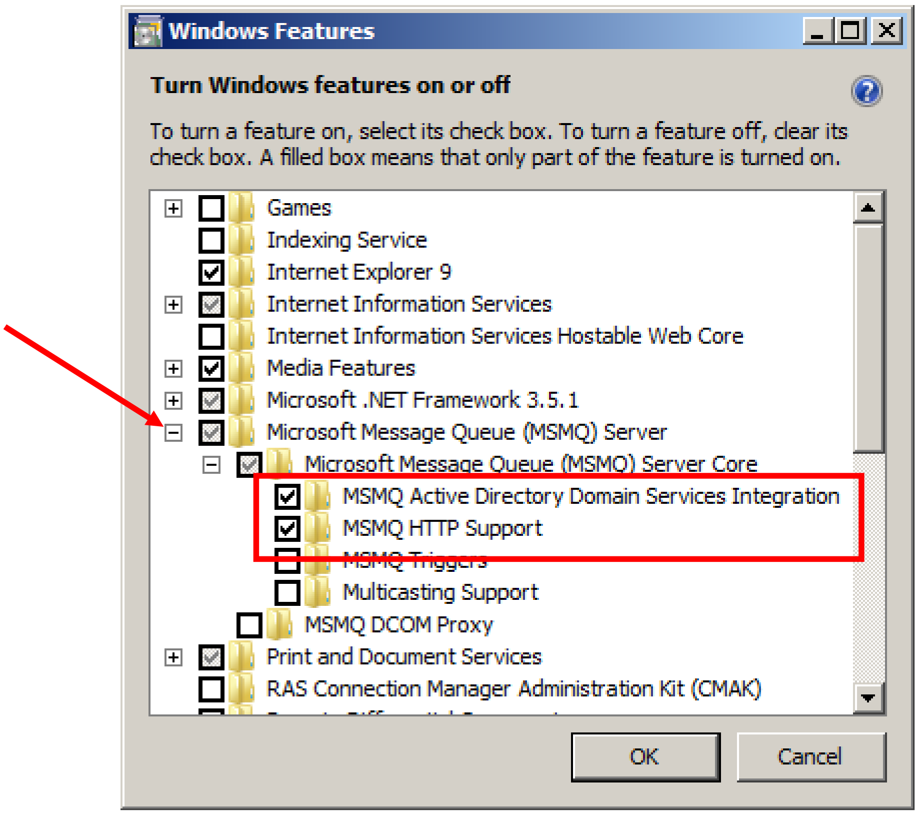
3. Click "**Turn Windows features on or off**":



4. In the “Windows Features” dialog, expand the “**Microsoft Message Queue (MSMQ) Server**” section and its “**Microsoft Message Queue (MSMQ) Server Core**” subsection.

   Mark the “**MSMQ Active Directory Domain Services Integration**” and “**MSMQ HTTP Support**”.

   Click OK.



To turn a feature on, select its check box. To turn a feature off, clear its check box. A filled box means that only part of the feature is turned on.

- [ ] Games
- [ ] Indexing Service
- [x] Internet Explorer 9
- [▨] Internet Information Services
- [ ] Internet Information Services Hostable Web Core
- [x] Media Features
- [▨] Microsoft .NET Framework 3.5.1
- [▨] Microsoft Message Queue (MSMQ) Server
  - [▨] Microsoft Message Queue (MSMQ) Server Core
    - [x] MSMQ Active Directory Domain Services Integration
    - [x] MSMQ HTTP Support
    - [ ] MSMQ Triggers
    - [ ] Multicasting Support
  - [ ] MSMQ DCOM Proxy
- [▨] Print and Document Services
- [ ] RAS Connection Manager Administration Kit (CMAK)

5. After Windows finishes updating its configuration, you can close the “Control Panel” window, and the MSMQ is ready to be used in this lab.

END OF LAB GUIDE