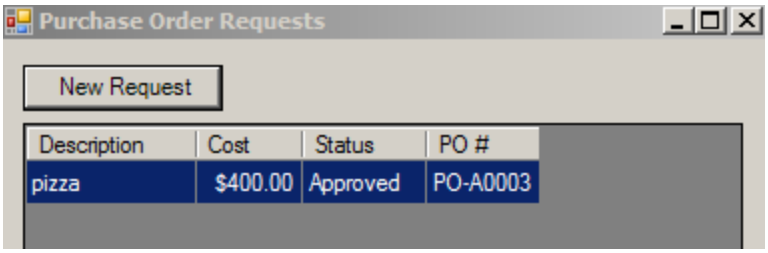
Chapter 2, *NServiceBus Saga Architecture*, expands on the uses of Sagas for persistence, timeouts, message durability, and message handling. We will discuss various Message Exchange patterns through examples to include Gateway and cluster managing. These are important concepts as it drives the high-availability and high performance that NSB brings to the table.

These projects were built VS2012 with Windows Server 2008, with MSMQ, DTC, NServiceBus references, and SQL Server 2012. If running in Windows 8.1, please run as "administrator", and there may be warnings if running the first time that some of the queues do not exist and it is creating the queues.

The Source code in this section:

In this section, we will be using the **BasicSagas-ServiceControl** directory. **This project requires NServiceBus ServiceControl to be installed.** It contains the following projects :

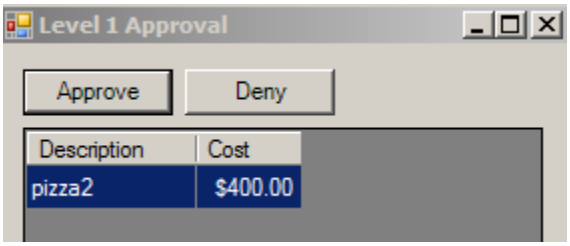
- AppForSubmittingRequests – Sends the initial SubmitRequestCommand to the Saga, and will receive a response from MySaga . It will look like the following:



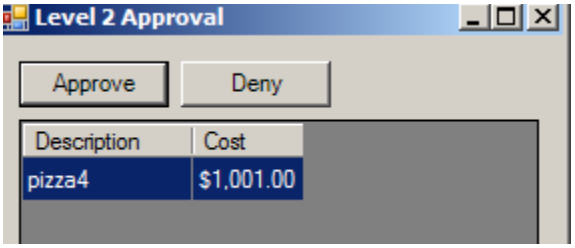
- MySaga – Will send approval requests based on above $100 and above $1000 for the purchases from AppForSubmittingRequests .

1. MySaga receive SubmitRequestCommand data, creates a SolicitApprovalFromlevel1Command and forwards it to Approver1 if above $100.
2. If MySaga receives ApproveRequestCommand, and the cost is above $1000, sends SolicitApprovalFromLevel2Command to Approver2.
3. If MySaga receives ApproveRequestCommand from the required approvers, sends SubmitRequestReplyMessage and RecordEncumbranceCommand to Accounting.
4. If MySaga receives DenyRequestCommand that the approvers denied the purchase, it sends SubmitRequestReplyMessage that it was denied.

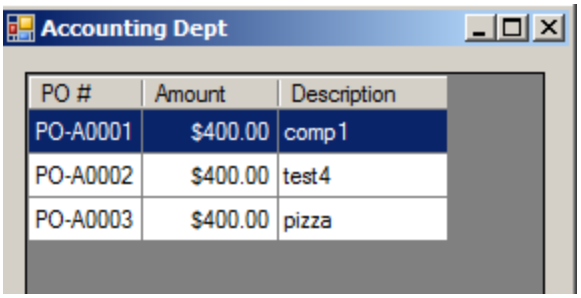
- AppForApproversLevel1 – receives SolicitApprovalFromlevel1Command, if approved, responds with ApproveRequestCommand, else it responds with DenyRequestCommand . This is for approval of purchases above $100. The process will look as follows:



- AppForApproversLevel2 – receives SolicitApprovalFromLevel2Command, if approved, responds with ApproveRequestCommand, else it responds with DenyRequestCommand . This is for the approval of purchases above $1000. The process will look as follows:



- AppForAccountingDept – receives RecordEncumbranceCommand from MySaga when approved. This receives the purchase order as an accouting department would. The process will look as follows:



The Source code in this section:

In this section, we will be using the **TimeoutManager** solution with the following projects:

- TimeoutManager – This project will perform several timeout functions through NServiceBus.

If running in Windows 8.1, please run as "administrator", and there may be warnings if running the first time that some of the queues do not exist and it is creating the queues.

The Source code in this section:

In this section, we will be using the **MessageMutators** solution with the following projects:

- Client – The client will send messages to the server.

- Server – Will receive the mutated message.
- Messages – The message format being passed between client and server.
- MessageMutators – This project will contain the mutation code to compress and uncompress the messages in "TransportMessageCompressionMutator.cs" and validate the message annotation in "ValidateMessageMutator.cs".

The Server is setup to run by default in Visual Studio 2012, please run the Client in a separate instance of Visual Studio 2012.

The Source code in this section:

In this section, we will be using the **Encryption** solution with the following projects:

- Client – The client will send an encrypted credit card messages to the server.
- Server – The server will receive the credit card message and decrypt it.
- Messages – The message format being passed between client and server.

The Server is setup to run by default in Visual Studio 2012, please run the Client in a separate instance of Visual Studio 2012.

The Source code in this section:

In this section, we will be using the **ScaleOut** solution with the following projects:

- Orders.Messages – The common messages for the sender and handlers.
- Orders.Sender – Will send messages to Orders.Handler to be handled across the workers, worker1and worker2.
- Orders.Handler.Worker1 – One of the worker services that is using a worker profile to send a response back to the sender. This will be an additional worker copy of Orders.Handler.
- Orders.Handler.Worker2 – One of the worker services that is using a worker profile to send a response back to the sender. This will be an additional worker copy of Orders.Handler.
- Orders.Handler– an endpoint which processes the message and configured to the distributor. This will be the master profile that the sender will send the place order command to in the "orders.handler" MSMQ. In the Visual Studio 2012 debugger, the "NServiceBus.Integration NserviceBus.Master" is set in the command line to be used instead of "Configure.Instance.RunDistributor()".

You may need to run Visual Studio as an administrator.

- If one does not start up 'Orders.Handler' first and wait for it to be up-and-running the

workers fail trying to access the distributor queue.

The Source code in this section:

In this section, we will be using the **ScaleOut-Performance** solution with the following projects:

- Orders.Messages – The common messages for the sender and handlers.
- Orders.Sender – Will send messages to Orders.Handler to be handled across the workers, worker1and worker2.
- Orders.Handler.Worker1 – One of the worker services that is using a worker profile to send a response back to the sender. This will be an additional worker copy of Orders.Handler.
- Orders.Handler.Worker2 – One of the worker services that is using a worker profile to send a response back to the sender. This will be an additional worker copy of Orders.Handler.
- Orders.Handler– an endpoint which processes the message and configured to the distributor. This will be the master profile that the sender will send the place order command to in the "orders.handler" MSMQ. In the Visual Studio 2012 debugger, the "NServiceBus.Integration NserviceBus.Master" is set in the command line to be used instead of "Configure.Instance.RunDistributor()".

You may need to run Visual Studio as an administrator.

- If one does not start up 'Orders.Handler' first and wait for it to be up-and-running the workers fail trying to access the distributor queue.

This is the same as the ScaleOut example, except that performance information will be set in the Worker projects, such as EndpointSLA in the EndpointConfig.cs.

The Source code in this section:

In this section, we will be using the **Gateway** solution,

1) Headquarter.Messages – The common messages for the Headquarters, SiteA, and SiteB.
2) Headquarter – Will receive messages from http://localhost:25899/Headquarter/ and http://localhost:25899/Headquarter2/, and send messages to http://localhost:25899/SiteA/ and http://localhost:25899/SiteB/ .

3) SiteA – A project that will receive update price information from Headquarters across http://localhost:25899/SiteA/ and respond that it was successful back to the Headquarters across http://localhost:25899/Headquarter2/.

4) SiteB – A project that will receive update price information from Headquarters across http://localhost:25899/SiteB/ .

5) WebClient – Will have a Index.htm page to send a JSON script to http://localhost:25899/Headquarter/ .

A "nservicebus" database must be present on the local SQLExpress.

Please click on the "index.htm" to run the web client.