Chapter 4, *Saga Development*, is the focusing on various useful constructions of Sagas and message handlers. The purpose of Sagas will be discussed as the discussion goes into the need to extending and coordinating transactional integrity by using Sagas. The chapter then morphs into a discussion of NServiceBus some using integrated pre-built WCF bridges, and while some might consider that this may be unusual to discuss WCF in a Saga chapter, Sagas become an intermediate for coordinating WCF and NServiceBus work. We can decouple the workflow from the front-end for interaction to back-end processes through message handling. Sagas provide the means to persist the state information of the messages. This discussion will also handle other queuing sources as well to include RabbitMQ and ActiveMQ.

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 is as follows:

The directory for the code is under Payment_WCFService directory.

The WCFService is used to test the client by being the WCF service.

The Source code in this section is as follows:

In this section, we will create a MVCApp Solution that will be under the BasicPaymentClient directory . It will contain several projects:

- MVCApp project – contains MVCs , Kendo grids, and the WCF client, for a browser user interface to send messages to the WCF service. It will read 5 XML files to load as messages.
- MyMessages project – contains NServiceBus' IMessages for the building of messages.
- WriteXMLFiles project – a utility to write 5 XML files to a C:\temp\ directory for the MVCApp project to load. This application saves messages in the form of XML files, which in turn are loaded through the MVCApp to be sent from the WCF client to the WCF service. These are for testing purposes only, but using files in the form of messages, makes it quick to change the data for various tests in the communication and endpoints, The messages are read from a C:\temp\ directory as 5 XML files saved in a format to work with the WCF messaging. The files can be created in the C:\temp\ directory by running the C:\temp WriteXMLFiles project. These files are simply test messages. These are just test messages that are saved to disk so that they can be modified and tested easily.

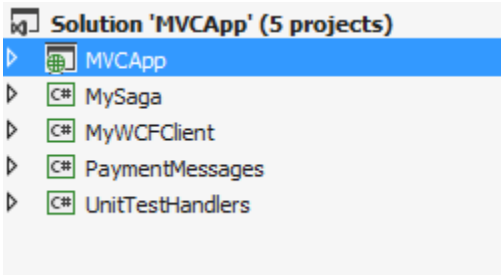
he Source code in this section is as follows:

The directory for the code is under "**SagaPaymentClient**" directory. This will be an addition of a Saga project added on to the BasicPaymentClient directory.

The "MVCApp – WCF" is used to send WCF messages as a client using a Saga.

The solution was built using VS2012 in Windows Server 2012, and also tested with VS 2012 running in Windows 8.1, with MSMQ, DTC, RavenDB, NServiceBus version 4.0 references, and SQL Server 2012 Express LocalDB installed.

Here's what the new project will look like with the Saga:

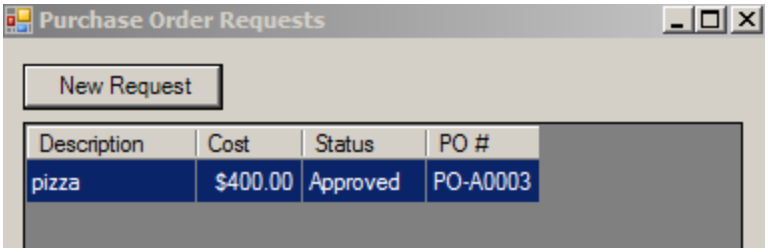


- MySaga project – contains the Saga code to decouple the front-end code from the WCF client code.

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 :

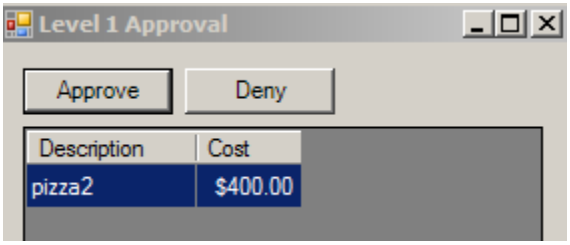
- AppForSubmittingRequests – Sends the initial SubmitRequestCommand to the Saga, and will receive a response from MySaga . It will look like the following:



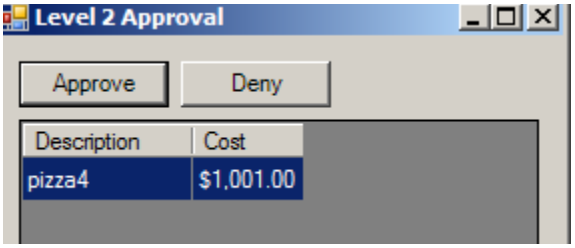
- MySaga – Will send approval requests based on above $100 and above $1000 for the purchases from AppForSubmittingRequests .

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.

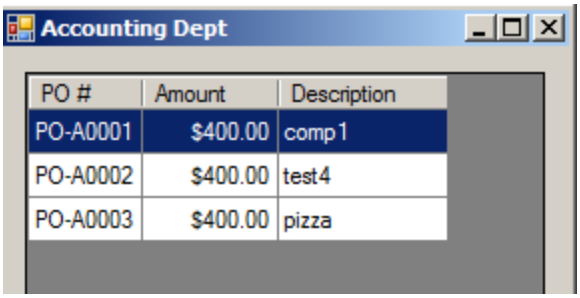
- AppForApproversLevel1 – receives SolicitApprovalFromlevel1Command, if approved, responds with ApproveRequestCommand, else it responds with DenyRequestCommand . This is for approval of purchases above $100. The process will look as follows:



- 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 **RabbitMQ** solution. **RabbitMQ must be installed.**

There will be three basic steps:

1. Add the NServiceBus.RabbitMQ reference.
2. Change the NServiceBus transport mechanism from <MSMQ> to <RabbitMQ>.
3. Set the RabbitMQ transport configuration in the App.config file.

The Source code in this section:

In this section, we will be using the **ActiveMQ** solution, this solution is similar to **RabbitMQ**, except **ActiveMQ** is used instead of **RabbitMQ.** **ActiveMQ must be installed.**

There will be three basic steps:

1. Add the NServiceBus.ActiveMQ reference.
2. Change the NServiceBus transport mechanism from <MSMQ> to <ActiveMQ>.
3. Set the ActiveMQ transport configuration in the App.config file.