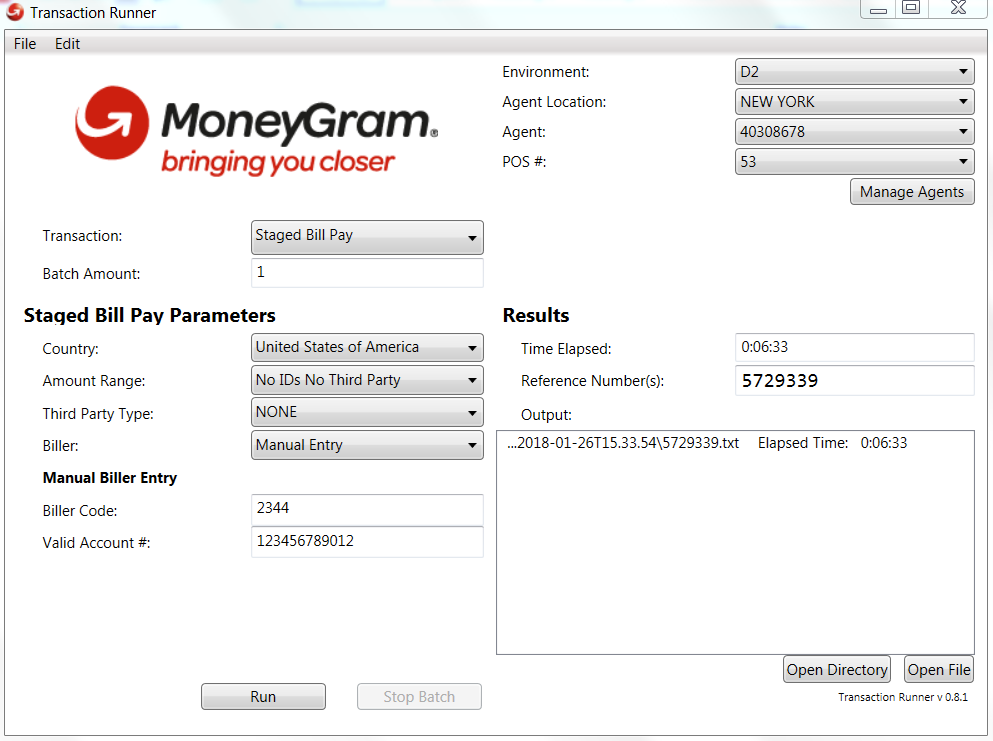
**Transaction Runner Quick Start Developers Document**

What is it? Transaction runner is a utility created to run transactions outside of the Agentworks .Nxt web app. Primarily this is used to assist in testing the functionality of the application without taking a lot of time. It features the ability to change, add and edit agents quickly, change parameters and to run transactions in batch.



**Technology:** Transaction runner is a C# WPF application using .net 4.7.1. It has a dependency on our AWAPI project for the operations layer to send and receive the data we need for transactions.

**Source Code:** Transaction runner exists inside of the AgentWorksApi repository. It can be found here: **Source\AgentWorksApi\Tools\TransactionRunner**.

To use the Transaction runner, you must first BUILD the AgentWorksApi solution, then build transaction runner.

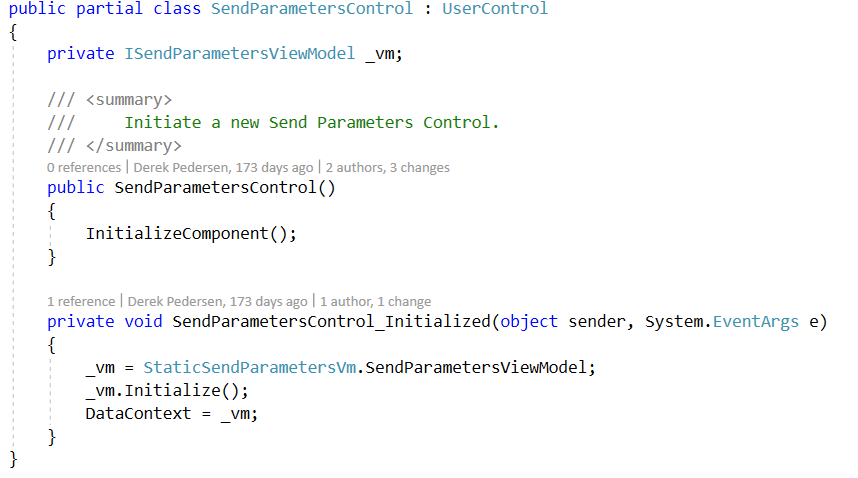
**Overview:** Transaction runner is broken into separate controls for each ui element. They are usually grouped by the transaction. So, send has its own ui control, its own view model and its own handler. This pattern is followed throughout this application.





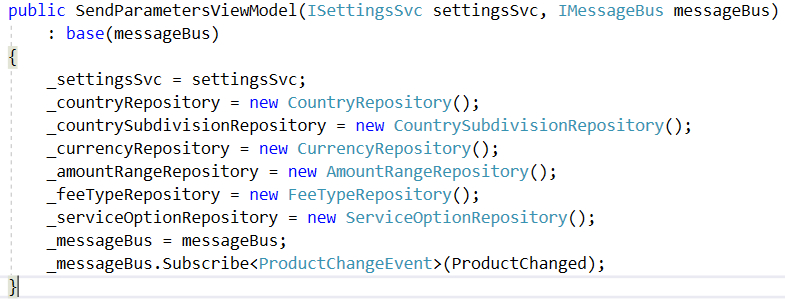


The UI control usually doesn’t do much in terms of logic. It grabs a singleton instance of the view model and assigns it to the UI controls data context so that the properties can be bound.

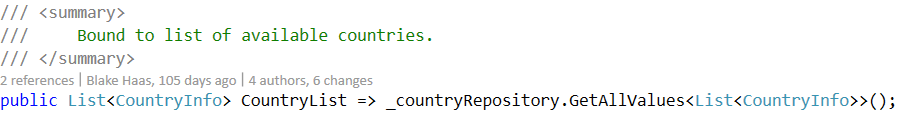


The view model initializes any repositories and fetches the data that is required for the UI element to display, this is where the dependency on AwApi project comes into play. The repository layers exist within that solution. Here is a snippet of the SendParametersViewModel:

Constructor:



**Populating property data from repo:**





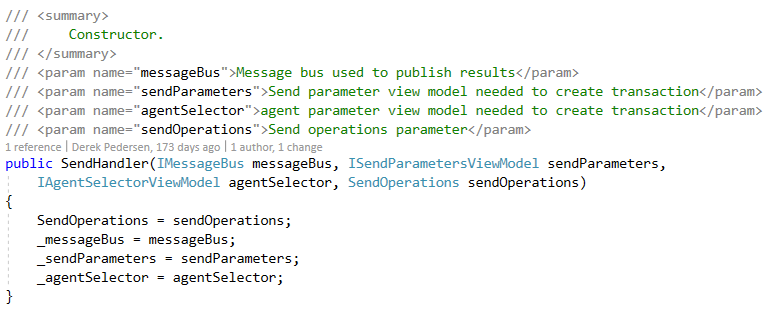
The UI binds this information like so:

<ComboBox Grid.Row="1" Grid.Column="1" x:Name="CbxCountry" HorizontalAlignment="Stretch" Margin="0,1,0,1" DisplayMemberPath="CountryName" ItemsSource="{Binding CountryList}" SelectedItem="{Binding SelectedCountry}" IsEnabled="{Binding IsAgentValid}" />

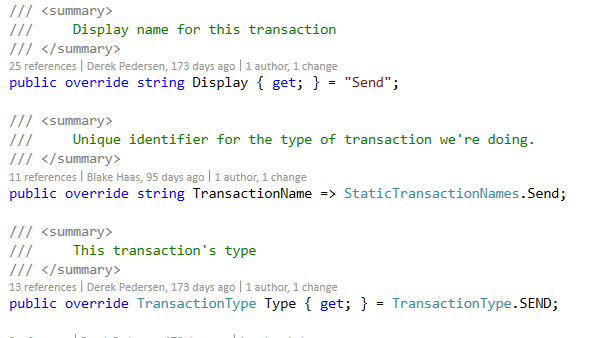
For more information on WPF and what databinding is, look here: <https://docs.microsoft.com/en-us/visualstudio/designers/introduction-to-wpf>

Once all the data has been gathered for the transaction, the last thing that is done is the final calls to the AwApi operations layer that allows us to hit AgentConnect (the soap service). This is done through the Handler classes.

Looking at the SendParametersHandler.cs, we can see that the operations layer is passed as a parameter and the request object is constructed to pass into the operations layer for processing the transaction.

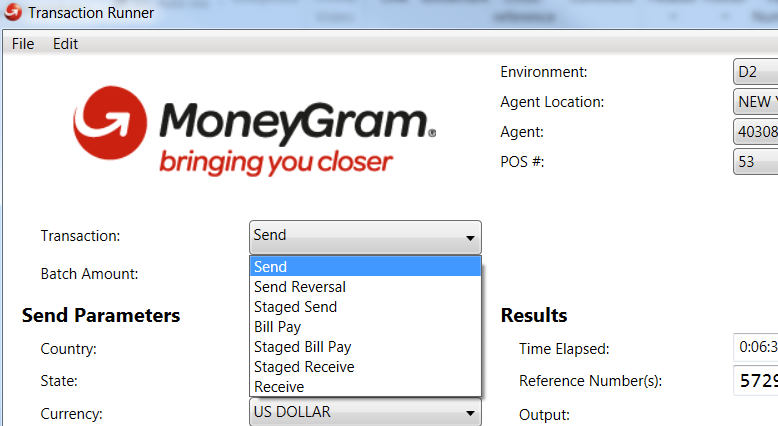


Each handler injects the appropriate operations layer, in this case it is the SendOperations layer. There are some properties defined inside of each handler to identify which transaction it handles:



Some of these properties end up on the UI, specifically the Display property.

You can see “Send” displayed in the dropdown here: (look further down this document to see how this dropdown is populated in the Extras section)



The remaining parts of the handler are where most of the important logic happens.

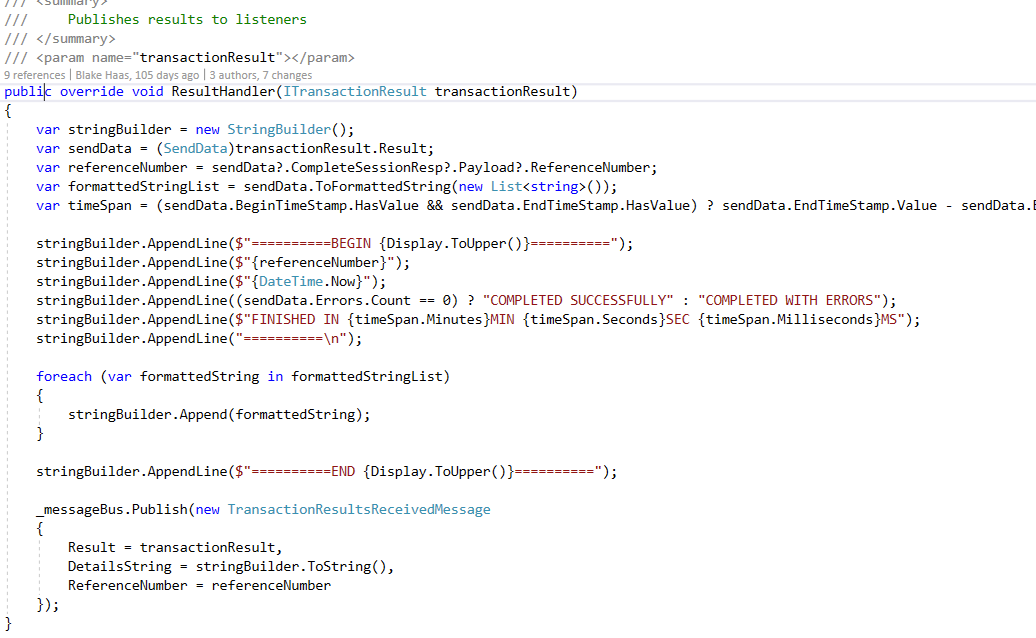
The BuildParams object builds up the Selected UI choices into a model which will be used later to make the final request object:



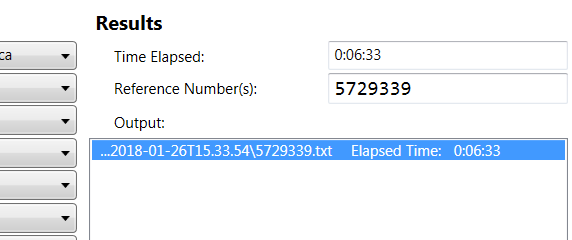
The Transaction method takes in the BuildParams object created previously as a parameter and starts to build the final request object:



The request is passed into an AwApi layer object as a parameter to append extra information to the request and then is finally sent to the SendOperations layer to the SendCompleteForNewCustomer method. The final result is returned and processed by the next important section of code, the ResultHandler.



The result handler will take the results and iterate through the data writing out logging info and the serialized request and result objects to a file that is written to disk for the user to view. The Result object, Details and Reference number all published to be shown on the UI:



**Extra things:** You may have noticed that there is this thing being called “MessageBus” throughout the application.

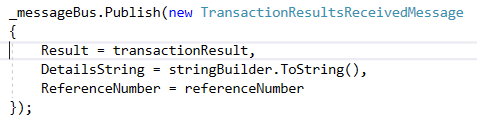


The MessageBus is a way to communicate within the application. It is implemented to do so in two ways:

1. Publishing
2. Subscribing

This operates like a basic event handler and a trigger for the handler.

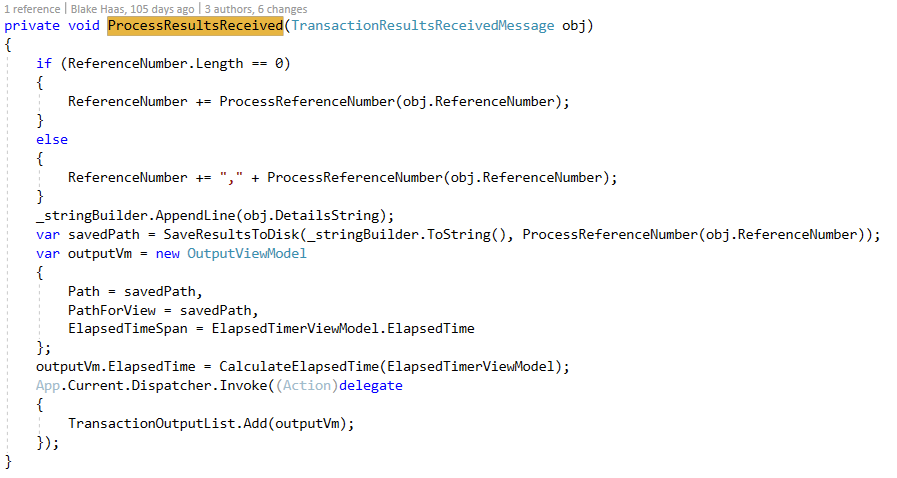
So you Publish a message using a model to make your message have a unique fingerprint.



Elsewhere in the application, you Subscribe to that model. Meaning any time that model is published on the message bus, it will end up hitting this method you define in the subscribe parameters.

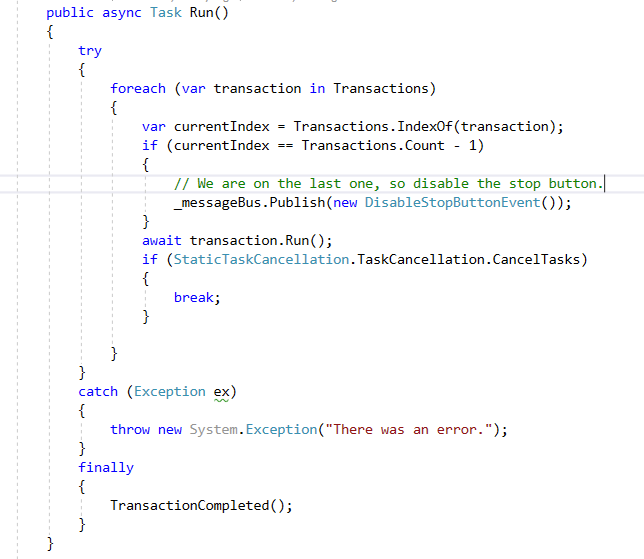


In this example, we are publishing TransactionResultsReceivedMessage, passing our transaction results. Elsewhere in the application, we have a subscriber for TransactionResultsReceivedMessage that has a method to call with the parameters sent called “ProcessResultsReceived”:

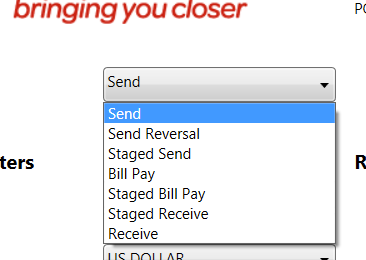


To see more on MessageBus, look here: <http://brentedwards.net/tag/message-bus/>

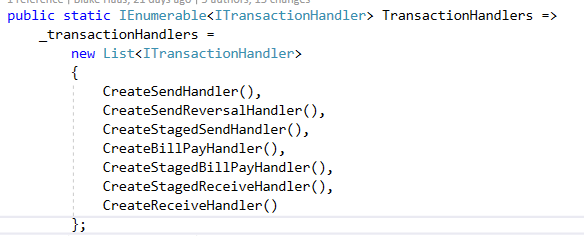
When you hit the start button, the beginning path of the transaction starts in the TransactionQue.cs file:



On the UI, the dropdown list for the transactions combines everything together.

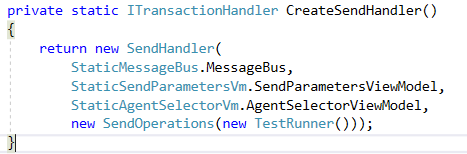


In the file TransactionProvider.cs, this transaction list is constructed:



Anything not put in this list will not appear in the UI Dropdown.

These helper methods (CreateSendHandler()) look like this:



Here you can see that this Transaction (Send) initializes everything it needs for that single transaction.

The same pattern is followed for the other transactional types.

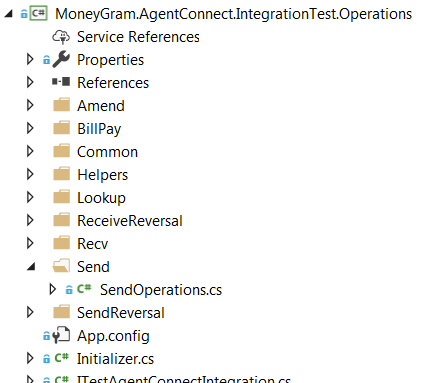
**AwApi Dependency:** The AwApi Solution contains many projects, there are a few that are important to the transaction runner:

MoneyGram.AgentConnect.IntegrationTest.Operations,

MoneyGram.AgentConnect.IntegrationTest.Data

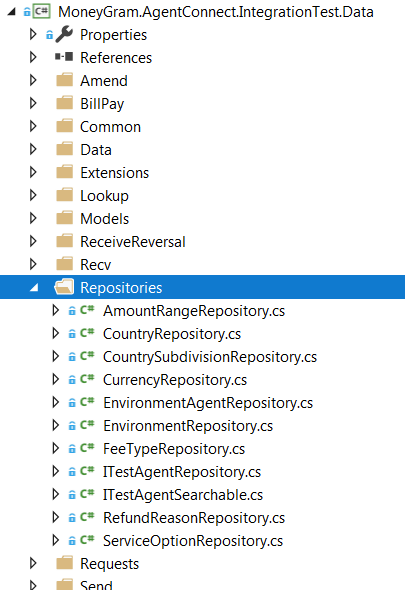
The repo for the AwApi is called “AgentWorksApi”

The IntegrationTest.Operations project allows access to make transactions, there is an operations layer defined for each transaction type.



The IntegrationTest.Data project contains the repositories that we use to populate data for the UI and for some requests.

You can find the Repositories here:



The data that the repositories access is JSON data, you can find this raw data here:

