Rachel Lim's Blog

Simplifying programming into something I can understand

Communication between ViewModels with MVVM

Communication between ViewModels can be tricky at first glance, but there are some easy frameworks and patterns that can help you out.

For example, lets say you had a ShellViewModel that controlled your entire application, and it has an Exit command. Now lets say one of the sub-view-models in the application would like to call this exit command.

The easiest way is through a Relative Binding to the Window's DataContext.ExitCommand.

```
<Button Command="{Binding
    RelativeSource={RelativeSource AncestorType={x:Type Window},
    Path=DataContext.ExitCommand}}
```

Of course, this only works well if you can find the control containing the command you want through a binding.

Enter the Event System

Another way is through a loosely coupled event system. I would recommend either MVVM Light's Messenger (http://www.galasoft.ch/mvvm/) or Microsoft Prism's EventAggregator (http://msdn.microsoft.com/en-us/library/ff921122(v=pandp.20).aspx). You could always build your own event system as well if you really wanted to.

Both event systems makes me think of a paging system. Any ViewModel can broadcast a message, and any ViewModel can subscribe to receive broadcasted messages. Messages are often packaged into different types, so a ViewModel can subscribe to only listen for specific message types such as CloseApplicationMessages.

This kind of loosely coupled event system is ideal for MVVM because the ViewModels don't need to know about each other to be able to do their job.

Using either of these systems, your ViewModel that handles the event would subscribe to receive a specific message type. It would tell the event system that if a message of type X is broadcasted, send it to a specified delegate method. The ViewModel that wishes to raise the event simply has to broadcast the

message and include any parameters needed in the body of the message.

Example using Prism's EventAggregator

```
// Subscribe
eventAggregator.GetEvent<CloseAppliactionMessage>().Subscribe(ExitMethod);

// Broadcast
eventAggregator.GetEvent<CloseAppliactionMessage>().Publish();
```

(I also have some code posted here (https://rachel53461.wordpress.com/2011/10/09/simplifying-prisms-eventaggregator/) which simplifies PRISM's EventAggregator for smaller applications)

Example using MVVM Light's Messenger

```
//Subscribe
Messenger.Default.Register<CloseAppliactionMessage>(ExitMethod);

// Broadcast
Messenger.Default.Send<CloseAppliactionMessage>()
```

This entry was posted on Sunday, June 5th, 2011 at 9:03 pm and is filed under MVVM. You can follow any responses to this entry through the RSS 2.0 feed. You can leave a response, or trackback from your own site.

10 Responses to *Communication between ViewModels with MVVM*

Juan pablo says:
October 21, 2015 at 9:57 pm
Hi I'm starting with Messenger and and really lost with them. Do you have a complete simple of how to use it ?

Reply

Rachel says:
October 22, 2015 at 3:38 pm
I don't have anything written up on the Messenger, however I think the code listed at the end of this article was all that was needed.

Messenger.Default should reference a singleton instance of the Messenger class in MVVM light. You can use Messenger.Default.Register to hook a method up to it that would get run anytime a message is broadcasted to your application, and Messenger.Default.Send to broadcast a message.

The values in the angle brackets are optional, and can be used to broadcast or subscribe a specific message type, which can be useful if you have many types of messages being broadcasted in your system, or if you want parameters passed around.

The post I have about simplifying Prism's EventAggregator actually simplifies the EveentAggregator code to use roughly the same syntax as MVVM Light's Messenger, so perhaps you may want to take a look at that for an example. Otherwise you could probably find plenty of examples with Google.

Good luck with it!

Reply

Paul Gibson says:
March 16, 2015 at 4:48 pm
Hi Rachel, Thanks for sharing! I have a question relative to your answer above to Donnie, where you suggest passing a specific viewmodel to each of the other viewmodels from a parent viewmodel. I have several view models in my application. The Main view model gets created (instantiated I guess) in the constructor of the main window. Then, when I create a window for another view that Windows constructor instantiates it's own view model. So the constructor for each child window looks like:

```
InitializeCompotent();
w = new ChildViewModel();
DataContext = w;
```

So there is no way that I can think of to have my child view models access the main view model. I tried passing a reference to the mainviewmodel (as this) when it creates a child window, but the "this" reference is private and the window constructor is public and so I get an exception for inconsistent access. Can you clarify how you were thinking that this might be done?

Reply

Donnie K says:
July 19, 2013 at 5:41 pm
Hello, I am just trying to make the transition from WinForms to WPF and I am a vb.net developer. Do you have any examples of communicating between ViewModels without the use of a framework? I am working on my first app and would like to have a StatusBar that is "globally" accessible from all ViewModels. This is posing quite a problem! My plan is to create a ViewModel for each View and have the StatusBar updated from each View. Any help would be greatly appreciated!

Reply

Rachel says:
July 26, 2013 at 11:55 am
Hi Donnie,

I'm sorry, don't have any example messaging systems that don't use a framework. You could use a single StatusBarViewModel somewhere, such as a singleton, or inject/pass a StatusBarViewModel to each of your viewmodels from your parent ViewModel.

Don't forget that with MVVM, your ViewModels are your application, not your Views, so your shared StatusBar object should be a ViewModel or Model, not a UserControl.

I do have an example of a simplified EventAggregator that works with the PRISM library that you might find useful, or if you're new to WPF you may want to check out my article about transitioning from Winforms to WPF.

Good Luck,
Rachel

Reply

Keith Nicholson says:
June 12, 2013 at 3:41 pm
In what methods are you placing the MVVM Light Messenger calls? My sender has it in a private method responding to a RelayCommand and the receiver is in the constructor of another view model.

Reply

Rachel says:
June 15, 2013 at 4:04 pm
Hi Keith,

Typically hooking up the receiver occurs in the Constructor or initialization code so it's easy for me to find and keep track of, while broadcasting a message happens anywhere in the code that it needs to happen.

Reply

Aaron B says:
February 6, 2012 at 4:41 pm
Excellent post. Well written and very informative. Thank you for taking the time to share. I really like the idea of using a messenger class. I may have to explore that avenue a little more. ☺

Reply

Musaab says:
June 30, 2011 at 3:43 pm
I'm using MVVM Light Messenger , I don't know about the others but this one can send and receive anything , you can't imagine how much I was happy when I was able to send and receive a message of type ObservableCollection

Reply

Rachel says:
June 30, 2011 at 5:55 pm
I used to send objects such as collections or integers directly through the event system too, but as my applications got bigger, I found it was easier to understand my code when I packaged the event parameters into a Message class and used that class with the event system. Not only did it make my application easier to manage and understand, it also allowed me to pass around multiple parameters with my events.

Reply