M-V-VM: How to keep collections of ViewModel and Model in sync

By jmix90, 2 Mar 2010

 5.00 (1 vote)

Source: http://www.codeproject.com/Articles/62537/M-V-VM-How-to-keep-collections-of-ViewModel-and-Mo

---

As pointed out in this post, **collections of the ViewModels and the models are not in sync**. This is because we do not directly access the model but an `ObservableCollection` (in the `viewModel`) which contains the object of the original collection (in the model) and these two collections are not the same...

As pointed out in the comments on CodeProject, there is a workaround. Here, I try to present two of them!

A First Solution: Register to the Wrapping Collection Changes

The first solution is to register to the events of the `ObservableCollection` in your `ViewModel` and to translate the changes to the wrapped collection.
It is very straighforward, but it becomes very fastidious if you have a lot of collections to deal with.

Here is the code:

⊟Collapse | Copy Code

```
//Wrap the business object collection
_friendsName = new
ObservableCollection<string>(myBusinessObject.FriendsName);
//Register to the wrapping CollectionChanged event
_friendsName.CollectionChanged += new NotifyCollectionChangedEventHandler
        (_friendsName_CollectionChanged);

...

//Translate the changes to the underlying collection
void _friendsName_CollectionChanged(object sender,
NotifyCollectionChangedEventArgs e)
{
 switch (e.Action)
 {
  case NotifyCollectionChangedAction.Add:
   _myBusinessObject.FriendsName.AddRange(
```

```
    e.NewItems.OfType<String>()
    );
   break;
  case NotifyCollectionChangedAction.Remove:
   _myBusinessObject.FriendsName.RemoveAll(
    friendName => e.OldItems.Contains(friendName)
    );
   break;
  //Reset = Clear
  case NotifyCollectionChangedAction.Reset:
   _myBusinessObject.FriendsName.Clear();
   break;
 }
}
```

Another Solution: Create a Proxy

You also can create a class which will act as a Proxy to the `businessObject`. Its only function will be to leverage the `INotifyCollectionChanged` events when necessary. I called it `MVMCollectionSyncher` for `ModelViewModelCollectionSyncher` and here is the code (which is very straightforward) :

⊟Collapse | Copy Code

```
public class MVMCollectionSyncher<T> : ICollection<T>,
        IDisposable, INotifyCollectionChanged
{
 #region fields
 private ICollection<T> _wrappedCollection;
 #endregion

 public MVMCollectionSyncher(ICollection<T> wrappedCollection)
 {
  if (wrappedCollection == null)
   throw new ArgumentNullException(
    "wrappedCollection",
    "wrappedCollection must not be null.");
  _wrappedCollection = wrappedCollection;
 }

 #region ICollection<T> Members
 public void Add(T item)
 {
  _wrappedCollection.Add(item);
  FireCollectionChanged(
   new NotifyCollectionChangedEventArgs(NotifyCollectionChangedAction.Add,
item));
 }

 public void Clear()
 {
  FireCollectionChanged(
   new
NotifyCollectionChangedEventArgs(NotifyCollectionChangedAction.Reset));
  _wrappedCollection.Clear();
```

```
 }

 public bool Contains(T item)
 {
  return _wrappedCollection.Contains(item);
 }

 public void CopyTo(T[] array, int arrayIndex)
 {
  _wrappedCollection.CopyTo(array, arrayIndex);
 }

 public int Count
 {
  get { return _wrappedCollection.Count; }
 }

 public bool IsReadOnly
 {
  get { return _wrappedCollection.IsReadOnly; }
 }

 public bool Remove(T item)
 {
  if (_wrappedCollection.Remove(item)) {
   FireCollectionChanged(
     new
NotifyCollectionChangedEventArgs(NotifyCollectionChangedAction.Remove,
item));
   return true;
  }
  return false;
 }

 #endregion

 #region IEnumerable<T> Members
 public IEnumerator<T> GetEnumerator()
 {
  return _wrappedCollection.GetEnumerator();
 }
 #endregion

 #region IEnumerable Members
 System.Collections.IEnumerator
System.Collections.IEnumerable.GetEnumerator()
 {
  return _wrappedCollection.GetEnumerator();
 }
 #endregion

 #region INotifyCollectionChanged Members
 public event NotifyCollectionChangedEventHandler CollectionChanged;
 private void FireCollectionChanged(NotifyCollectionChangedEventArgs
eventArg)
 {
  NotifyCollectionChangedEventHandler handler = CollectionChanged;
```

```
  if (handler != null) handler.Invoke(this, eventArg);
 }
 #endregion

 #region IDisposable Members
 public void Dispose() { _wrappedCollection = null; }
 #endregion
}
```

Then in your `ViewModel`, instead of presenting an `ObservableCollection<>`, you offer an `MVMCollectionSyncher` with this code.

⊟Collapse | Copy Code

```
//Creation
MVMCollectionSyncher _friendsName = new MVMCollectionSyncher
                                <string>(myBusinessObject.FriendsName);
...
//Property
public MVMCollectionSyncher<String> FriendsName
{
 get { return _friendsName; }
 set
 {
  if (value != null){
   _friendsName.Dispose();
   _friendsName = value;
   FirePropertyChanged("FriendsName");
  }
 }
}
```

Here are some links dealing with the same subject:

- A stackOverflow thread
- Another stackOverflow thread
- MVVM: To Wrap or Not to Wrap? BLINQ and CLINQ to the Rescue! (Part 3)

License

About the Author

jmix90
Software Developer http://wpf-france.fr

France (Metropolitan)
Member

Follow on Twitter

Jonathan creates software, mostly with C#,WPF and XAML.

He really likes to works on every Natural User Interfaces(NUI : multitouch, touchless, etc...) issues.

He is awarded Microsoft MVP in the "Client Application Development" section since 2011.

You can check out his WPF/C#/NUI/3D blog http://www.jonathanantoine.com.