Module 11

"Threads and Asynchrony in WPF [Deeper Dive]"

TEKNOLOGISK
INSTITUT

TEKNOLOGISK INSTITUT

Agenda

- Multiple Dispatchers
- Data Binding vs. Threading in WPF
- Concluding Discussion

Multiple Dispatchers

- More dispatcher threads can be created for
  - Performance
  - Fault tolerance
  - …
- `Dispatcher.Run()` on separate thread creates new message loop
- Be careful..!
  - **`Application.*`** is now misleading and dangerous!
  - Application.Windows
  - Application.Dispatcher

A Word on ApartmentState…

- COM is the ancestor of .NET
  - Uses *apartments* for threading requirements (.NET does not!)
    - STA = Single-Threaded Apartment
    - MTA = Multi-Threaded Apartment
- Default for .NET Threads is MTA
  - Threads are default MTA, but can be changed
  - Thread pool threads are always MTA and cannot be changed!
- UI threads should always be STA
  - Uses Clipboard, Drag 'n Drop, Shell Dialogs, … which are only available for STA

Agenda

- Multiple Dispatchers
- **Data Binding vs. Threading in WPF**
- Concluding Discussion

`INotifyPropertyChanged`

- Data bindings are the crucial mechanism of WPF
  - Especially in MVVM
  - Can automatically notify and signal updates

- Implement **`INotifyPropertyChanged`** to propagate modifications to a single element through data binding
  - **`PropertyChanged`** event
    - Raise event with CLR property name whenever it is changed

Good News for Properties!

Data Binding automatically converts
`INotifyPropertyChanged` notifications to the
Dispatcher thread

`ObservableCollection<T>`

- Implement collections by inheriting **`ObservableCollection<T>`**
- Automatically propagates adding and removal of elements to collection
- Handling change notifications overall
  - Implement **`INotifyPropertyChanged`** on single elements of type **`T`**
  - Inherit collection storage class from **`ObservableCollection<T>`**

Bad News for Collections!

Data Binding does not automatically convert **`INotifyCollectionChanged`** notifications to the Dispatcher thread

But why…??? ☹

Collection Views

- A collection view manages data currency for collection
  - Is automatically generated behind the scenes
  - Retrieve via `CollectionViewSource.GetDefaultView()`

- `ICollectionView`
  - `CurrentPosition`, `CurrentItem`
  - `MoveCurrentTo`, `MoveCurrentToFirst`, `MoveCurrentToLast`, `MoveCurrentToNext`, `MoveCurrentToPrevious`, `MoveCurrentToPosition`
  - `IsCurrentBeforeFirst`, `IsCurrentAfterLast`

- `ICollectionView`
  - `IList`
    - `IBindingList`

`CollectionView`
`ListCollectionView`
`BindingListCollectionView`

CollectionViewSource

- Collection views can similarly be created in XAML
  - Define a **`CollectionViewSource`** instance bound to data
  - Bind ItemsControl to the **`CollectionViewSource`** instance

```
<Window.Resources>
    <clr:Participants x:Key="participants" />
    <CollectionViewSource x:Key="cvs"
       Source="{Binding Source={StaticResource participants}}" />
</Window.Resources>
```

```
<ListBox ItemsSource="{Binding Source={StaticResource cvs}}"
         DisplayMemberPath="FullName"/>
```

- Sorting can also be applied in XAML

TEKNOLOGISK INSTITUT

Collection Notifications and Threads

- Adding elements to **`ObservableCollection`** by other threads
  - Not directly possible
  - Needed ugly dispatching!

- WPF 4.5 adds easy-to-use Collection Synchronization
  - Provide lock for the collection
  - Enable collection synchronization
  - Update **`IEnumerable`** from any thread

```
BindingOperations.EnableCollectionSynchronization(
   _participants,   // collection
   _syncObject      // lock object
);
```

Better News for Collections!

You can manually enable Data Binding to convert **`INotifyCollectionChanged`** notifications to the Dispatcher thread

Note: This is does however not automatically ensure thread-safety

Asynchronous Data Binding

- Data binding can be evaluated asynchronously on thread pool threads
  - Binding.IsAsync
- Is often combined with PriorityBinding

```
<PriorityBinding FallbackValue="N/A">
    <Binding Path="Slowest" IsAsync="True"/>
    <Binding Path="Slow" IsAsync="True"/>
    <Binding Path="Normal" IsAsync="True"/>
    <Binding Path="Fast" IsAsync="True"/>
    <Binding Path="Fastest" />
</PriorityBinding>
```

- Don’t use asynchronous data binding:
  - Asynchronous bindings is usually a sign of poor design

Agenda

- Multiple Dispatchers
- Data Binding vs. Threading in WPF
- **Concluding Discussion**

TEKNOLOGISK INSTITUT

Discussion: Thread Affinity in MVVM?

- Question:
  - How do we ensure specific operations are executed on Dispatcher in ViewModels?
  - Do we need to…?

WTF? UI Buttons are Not Updating…?!

- In MVVM the RelayCommand (a.k.a. DelegateCommand) uses the WPF CommandManager to reevaluate command enabledness
  - CommandManager.RequerySuggested
- Threading and asynchronous occasionally "confuses" the CommandManager in WPF
- Solution is to manually instruct CommandManager to test

```
// Forcing the CommandManager to raise the RequerySuggested event
CommandManager.InvalidateRequerySuggested();
```

- See e.g.
  - https://github.com/lbugnion/mvvmlight/blob/master/GalaSoft.MvvmLight/GalaSoft.MvvmLight%20(PCL)/Command/RelayCommand.cs

Summary

- The WPF Dispatcher
- Data Binding vs. Threading in WPF
- Concluding Discussion


WINCUBATE
Jesper Gulmann Henriksen
PhD, MCT, MCSD, MCPD
Phone : +45 22 12 36 31
Email : jgh@wincubate.net
WWW : http://www.wincubate.net
Ringgårdsvej 4A
8270 Højbjerg
Denmark