

Case Study A: "WPF and Threads"



TEKNOLOGISK
INSTITUT

Agenda

- ▶ **UI and Threads**
- ▶ The WPF Dispatcher
- ▶ Data Binding vs. Threading in WPF

UI and Threads

- ▶ Windows UI Context
 - Notion of “Main” thread
- ▶ Message Pump
- ▶ WinForms ~ ISynchronizeInvoke
- ▶ WPF ~ Dispatcher
- ▶ **Mantra:**
 - “Keep Working Threads Away From UI”



Image by victor408 is licensed under [CC BY-NC 2.0](#)

Agenda

- ▶ UI and Threads
- ▶ **The WPF Dispatcher**
- ▶ Data Binding vs. Threading in WPF

WPF Class Hierarchy

▶ object

- **DispatcherObject**

Access only on creating thread

- DependencyObject

- Freezable

- Visual

- UIElement

- FrameworkElement

- Control

Routed events, layout, focus, ...

Styling, data binding, ...

Foreground, Background, ...

- Visual3D

- UIElement3D

- ContentElement

- FrameworkContentElement

The Dispatcher

- ▶ Any operation on **DispatcherObject** must happen on the UI thread
 - **InvalidOperationException**
- ▶ Use **DispatcherObject.Dispatcher** property
 - **Invoke()** – Synchronous
 - **BeginInvoke()** – Asynchronous
- ▶ WPF “emulates” two built-in main threads
 - Main thread
 - Render thread

DispatcherPriority

- ▶ Priority is captured by **DispatcherPriority** enumeration
 - **Send** Highest (= immediately)
 - **Normal**
 - **DataBind**
 - **Render**
 - ...
 - **Background**
 - ...
 - **ApplicationIdle**
 - **SystemIdle** Lowest
- ▶ Best practice
 - Always make this Normal (unless you have a very good reason not to!)

DispatcherTimer

- ▶ We have previously covered two threading timers:
 - `System.Timers.Timer` ~ Thread Pool
 - `System.Threading.Timer` ~ Thread Pool
- ▶ But... Perfectly suited for WPF UI:
 - `System.Windows.Threading.DispatcherTimer` ~ Dispatcher
 - Tick event
 - Interval
 - Start()
 - Stop()

Module 05

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

- ▶ UI and Threads
- ▶ The WPF Dispatcher
- ▶ **Data Binding vs. Threading in WPF**

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.DefaultView()`

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

- ▶

<code>ICollectionView</code>	<code>CollectionView</code>
• <code>IList</code>	<code>ListCollectionView</code>
• <code>IBindingList</code>	<code>BindingListCollectionView</code>

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

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 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

Summary

- ▶ UI and Threads
- ▶ The WPF Dispatcher
- ▶ Data Binding vs. Threading in WPF



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