Values

1. Make it easy and intuitive to build WPF applications that have a clean architecture which values a clear distribution of concerns, layering and separation of concerns.
2. Prefer compile-time checking over runtime-checking by using static typing, generics and designs that are checked by the compiler.
3. Minimize the code required to be written by the developer but prefer a little more code that is straight-forward and hard to get wrong over a little less code that is more complex and harder to maintain.
4. Make the API intuitive and easy to discover without resorting to the documentation (optimize for IntelliSense for example).
5. Design the library so that it is
   1. Easily understandable and maintainable.
   2. Extensible from outside at very few selected places.
   3. Extensible by modifying the source code of the library.
6. Only add essential features that are needed by most applications to the library. Maximize the value/lines of code ratio. Provide feature that are not used by many applications as source files that can be included in the target application.
7. Strive for a clean design and excellent quality

Documentation

|  |  |
| --- | --- |
|  |  |
| Model | The model implements the business logic. It is usually implemented as a Domain Model as described by Martin Fowler. |
| ViewModel | The view model reshapes a part of the model to a form that can be easily consumed by the view. It provides the data of the model as easily bindable properties and provides actions that can be triggered by the view as commands. The view model does not load/manage data itself but is passed all model objects it needs.  The view model is also responsible to validate its data and to implement change notification (slimming down the model). |
| Screen | A screen is the logical, UI independent representation of a user screen. It has its own lifecycle: it is created, activated/deactivated multiple times and eventually closed.  A screen manages the domain objects that are presented by the screen and their lifecycle (load/save/discard). It will probably use the model to load the objects and may use the Unit of Work pattern to track and save them.  A screen creates and provides a root view model that is used by the UI to actually display the screen. The screen itself does not provide any data or behavior intended for the UI.  The screen can be passed to the root view model (and its child view models) if they need some behavior of the screen to carry out the user actions they implement (like opening new screens, or loading some data). It is recommended that the view models do only know a small interface and not the concrete type of the screen.  In other words the screen implements the behavior that concerns all view models whereas the view model only implements the behavior related to the domain object(s) it presents. |
| ScreenConductor | A screen conductor manages a collection of screens and their lifecycle. The view that displays a screen conductor may be some form of tabbed interface control (e.g. a simple TabControl), a docking system (like in Visual Studio) or a simple control that only shows the currently active screen (e.g. a ContentControl). |
| ScreenSubject | Holds information (usually some sort of a key) a screen needs to determine what it should display. It also decides if a screen with the same subject is already open or if a new screen should be created. |
| ScreenFactory | A factory that can create an instance of a certain screen configured with a certain subject. A screen factory also provides a way to check whether a given screen presents the same information as a screen that would be created by this factory.  Only screen factories and not screens can generally be shown by screen conductors and other services. The reason for this is that a screen conductor first checks if there is already a screen open that presents the same information. If so, it does not need to create the screen described by the screen factory. The screen subjects of two screens are used to determine whether they present the same information or not. |
|  |  |
|  |  |

## Screens

### Screen

A screen is the logical, UI independent representation of a user screen.

[Class diagram]

A screen manages the domain objects that are presented by the screen and their lifecycle (load/save/discard). It will probably use the model to load the objects and may use the Unit of Work pattern to track and save them.

A screen creates and provides a root view model that is used by the UI to actually display the screen. The screen itself does not provide any data or behavior intended for the UI.

The screen can be passed to the root view model (and its child view models) if they need some behavior of the screen to carry out the user actions they implement (like opening new screens, or loading some data). It is recommended that the view models do only know a small interface and not the concrete type of the screen.

In other words the screen implements the behavior that concerns all view models whereas the view model only implements the behavior related to the domain object(s) it presents.

### Screen subject

The subject of a screen is the set of information a screen needs to determine what information it should load and present. The subject of a PersonScreen may be the personnel number, the subject of a SkillAssignmentScreen may be a Person object.

The subject of the screen is specified when a ScreenFactory is created for it. It is passed to a screen instance by a call to its Initialize(TSubject) method made by the framework after the screen is instantiated.

It is also used by a ScreenCreationBehavior to decide if a potential screen is already open or if a new one must be created.

### The lifecycle of a screen

The following lifecycle events are called in the listed order:

|  |  |
| --- | --- |
| Method | Description |
| Constructor | If you use a DI container you should declare every external dependency as a constructor parameter so that is automatically injected into your screen. You should not declare business objects or other objects that are related to information that is presented by the screen as constructor arguments – use the screen subject instead! |
| OnInitialize() | This method is called immediately after basic properties like the Parent screen are set. This method is only called once in the lifetime of a screen. |
| OnInitialize(TSubject) | This only exists in the Screen<TSubject> class. It is called right after the OnInitialize() method. Use this method to create/load all objects you need, setup the screen and to create the root ViewModel. This method is only called once in the lifetime of a screen. |
| OnActivated() | This method is called every time the user uses a screen conductor to select the current certain screen (e.g. if the user selects the “John Smith” tab in an tabbed user interface OnActivated is called for the screen that presents “John Smith”). |
| OnDeactivated() | This method is called every time the user uses a screen conductor to select another screen hiding the current one. |
| bool RequestClose() | This method is called if the screen should be closed (e.g. the user clicked the close button). Return true if it is ok to close the screen and false to cancel the close process. This method can be used for example to tell the user that he or she has unsaved changes and provide them a way to cancel the close process. |
| Close() | If RequestClose returned true, Close may be called to actually close the screen. RequestClose may be called multiple times before Close is finally called, but it is guaranteed that RequestClose is always called before Close and that Close is the last method that is called on the screen. |

## Screen management

### Screen conductors

A ScreenConductor manages a collection of screens and their lifecycle.

[Class diagram]

It provides all its screens as an ObservableCollection and raises the PropertyChanged event if the active screen changes. The currently active screen can be changed by setting the ActiveScreen property.

The ScreenConductor is responsible for calling the Activate, Deactivate, RequestClose and Close methods of its screens.

A ScreenConductor is also a Screen and therefore has a corresponding view. The view is responsible to synchronize the ScreenConductor with an UI control. The UI control could be a TabControl (where every tab is one screen), a simple ContentControl (which displays only the currently ActiveScreen) or a more advanced solutions like a docking library (as in Visual Studio).

### Identifying screen conductors

Because an application can have multiple screen conductors at a time each one has an ID which can be used to get a reference to a screen conductor. The following diagram shows the screens of a sample application:



The search for a screen conductor always starts at a concrete screen of the screen hierarchy. Starting with that screen, it and all ancestor screens are checked for the conductor ID. If the root screen is reached without finding the ID a breadth-first search of all screens is performed. The above diagram outlines a search starting at the leftmost PersonInfoScreen. If the searched ID is DockingConductor the search would step at step 6.

### Screen Factories

A ScreenFactory is a factory that can create Screen and Screen<TSubject> instances that are configured with a certain subject.

[Class diagram]

Screen conductors, child screen collections and other services can in general only open a ScreenFactory and not a Screen itself. There are two reasons for this: (1) a screen conductor does not always create a new screen if it is directed to open a ScreenFactory (this behavior is described in the next sections) and (2) the ScreenFactory initializes the screen right after creating/resolving it (e.g. setting the parent screen and calling Initialize with the configured subject).

### Screen creation behaviors

The ScreenCreationBehavior determines how a ScreenConductor behaves if it is directed to show a ScreenFactory that would create a screen that is of the same type as one already contained by the ScreenConductor. The following options are available:

|  |  |
| --- | --- |
| Behavior |  |
| SingleInstance | The ScreenConductor checks if it already contains a Screen of the same type and activates it if it contains one. If not it uses the ScreenFactory to create a new Screen and opens/activates it. |
| MultipleInstances | The ScreenConductor always creates a new screen. |
| UseLocator | The ScreenConductor calls the Matches(TLocatorSubject) method for all screens that it contains and that implement the ILocatableScreen<TLocatorSubject> interface.  The ScreenConductor passes the subject of the factory to the Matches method which should return true the screen presents the same information that is described by the given subject.  It activates the first screen whose Matches method returns true. If no screen returns true, the ScreenConductor creates a new Screen and opens/activates it.  This behavior can only be specified on screens with a subject (Screen<TSubject>). |

The ScreenCreationBehavior is specified by applying the ScreenCreationBehaviorAttribute attribute on the screen.

### Opening/closing a screen

Open a new screen involves two steps:

1. Call ScreenFactory.Create<TScreen>() to create a new ScreenFactory for a screen without a subject or ScreenFactory.With(subject).Create<TScreen>() for a screen that has a subject.
2. Call OpenScreen(conductorId, screenFactory) on the current Screen instance to open a new screen in the specified conductor. The algorithm described in the section “Identifying screen conductors” is used to locate the ScreenCondcutor in which the new screen is opened (the screen on which this method is called is used as the starting point for the conductor search).

# View models

Features & concepts

Declaring a ViewModel

Creating a descriptor

### Developing custom behaviors

1. Derive your new behavior from VMPropertyBehavior.
2. Register your behavior type by calling one of the VMBehaviors.Register methods.
3. Enable your behaviors by using the WithBehaviors method of the VMDescriptorBuilder fluent interface.

# Views

Location and creation of views

Binding ViewModels