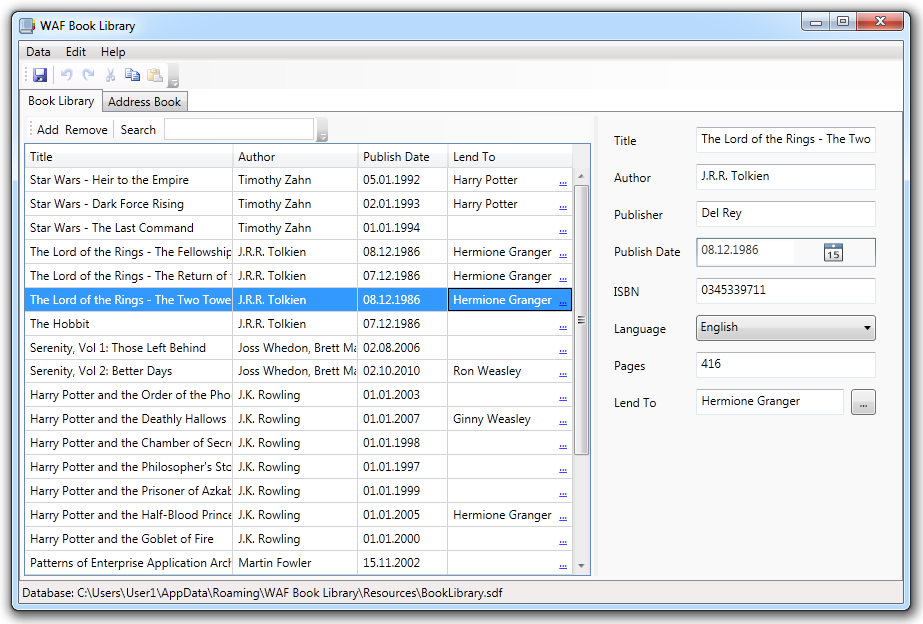
WAF BookLibrary

WPF Application Framework (WAF)

# Note: This document is outdated!

# Introduction

The BookLibrary sample application shows how to use the WPF Application Framework (WAF) in a data oriented application.

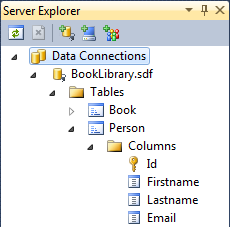
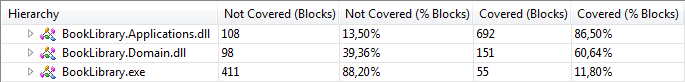


This sample application is part of the WPF Application Framework (WAF) [download](http://waf.codeplex.com/).

# Run the sample

1. Open the WpfApplicationFramework solution.
2. Set the BookLibrary.Library.Presentation project as StartUp project and start it in Debug mode.

# Highlights

* Layered Architecture and usage of the Model-View-ViewModel pattern (MVVM).
* The Entity Framework is used in combination with a SQL Server Compact Edition database.  
  
* The entities provide validation rules which are reflected on the user interface.
* The data can be sorted and filtered in the DataGrid.
* Create a new Email when the user clicks on an email address in the BookLibrary application.
* All layers are unit tested. The Domain and Application layer are completely tested except the entity framework related code. The Presentation layer is partly tested.  
  Unit test code coverage:  
  

# Project Structure

BookLibrary.Presentation

1. Converters Value converters
2. DesignData Design time support
3. Resources ResourceDictionaries, Images, Icons
4. Services UI service implementations
5. Views WPF Views (Windows, UserControls)

BookLibrary.Applications

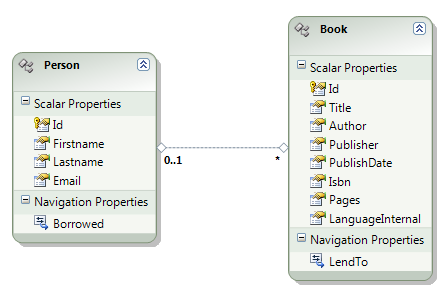
1. Controllers Use case controllers
2. DataModels DataModels for DataTemplates
3. Services Interfaces and implementation of services
4. ViewModels ViewModels for the Views
5. Views Interfaces for the Views

BookLibrary.Domain

1. (Root) Entity Data Model and partial classes for the Entities
2. Resources Database file

# Domain Layer

BookLibrary.Domain\BookLibraryModel.edmx



# Features

|  |  |
| --- | --- |
| Validation with DataAnnotations The .NET Framework ships a validation framework in the System.ComponentModel.DataAnnotations namespace.  But WPF Binding supports only validation with the IDataErrorInfo interface.  {Binding ... ValidatesOnDataErrors=true}  WAF provides the DataErrorInfoSupport class which connects the DataAnnotations validation with the IDataErrorInfo interface.  BookLibrary.Domain/Person.cs (see Error and this[] property)  Because entity classes are partial classes generated by the Entity Framework, it's not possible to write the validation attributes directly to the properties.  A workaround is to introduce an interface with these properties and specify the attributes there.  BookLibrary.Domain/Person.cs (see IPerson interface)  The MetadataType attribute defines the association between the entity class and the interface.  BookLibrary.Domain/Person.cs (see Person class)  Enable the DataAnnotations infrastructure to use the MetadataType information.  BookLibrary.Domain/BookLibraryEntities.cs (see static constructor) |  |
| Visualize parsing errors and exceptions When the user enters "abc" in a TextBox which is bound to an integer property then WPF isn't able to parse the user input. Such errors should be shown similar to validation errors.  {Binding ... ValidatesOnExceptions=true}  BookLibrary.Presentation/Views/BookView.xaml (see Pages text box) |  |
| Disable commands when an input error exists The BookLibrary disables the Save and Add (Book) command when an input error exists.  {Binding ... NotifyOnValidationError=true}  The Validation.AddErrorHandler event is used to track errors in the UI. These errors are synchronized with the IsValid flag of the ViewModel.  BookLibrary.Presentation/Views/ShellWindow.xaml.cs  (see ErrorChangedHandler method)  The IsValid flag is used to enable/disable the Save command.  BookLibrary.Applications/Controllers/EntityController.cs  (see CanSave method) |  |
| Use a DataModel to support commands In simple scenarios we can bind an ItemsControl (e.g. DataGrid) directly to a collection of Model objects.  BookLibrary.Applications/ViewModels/PersonListViewModel.cs  (see Persons property)  But when we need to provide additional functionality for the user interface then we can wrap the Model object with a DataModel object. The sample application defines the LendToCommand in the BookDataModel class.  BookLibrary.Applications/DataModels/BookDataModel.cs  (see LendToCommand property)  The DataGrid binds to the collection of BookDataModels. Thus, the LendTo column is able to bind the Hyperlink directly to the LendToCommand property of the DataModel.  BookLibrary.Presentation/Views/BookListView.xaml (see DataGrid element)  The BookController is responsible to keep the collection of Books synchronized with the collection of BookDataModels. It uses the WAF ConverterCollection for this task.  BookLibrary.Applications/Controllers/BookController.cs  (see Initialize method) |  |
| Filter entities in the DataGrid The BookLibrary application allows the user to filter the books or persons.  The filter text is bound to the underlying ViewModel. Furthermore, we listen to the TextChanged event.  BookLibrary.Presentation/Views/BookListView.xaml  (see x:Name="searchBox" element)  In the code behind file the CollectionView of the Books collection is retrieved. Here the Filter delegate is set which calls the Filter method on the ViewModel.  BookLibrary.Presentation/Views/BookListView.xaml.cs  (see FirstTimeLoadedHandler method)  The TextChanged event handler calls the Refresh method on the CollectionView.  BookLibrary.Presentation/Views/BookListView.xaml.cs  (see FilterBoxTextChanged method) |  |
| Select the next element after removing DataGrid items The BookLibrary application supports multi-selection inside the DataGrid. This can be used to remove multiple items at once. The synchronization of the selected items between the View and the ViewModel is done in the code-behind file.  BookLibrary.Presentation/Views/PersonListView.xaml.cs  (see DataGridSelectionChanged method)  After the user removes some Persons the next Person should be selected automatically. This is a bit tricky because the user might have sorted or filtered the DataGrid. The PersonController is responsible for the remove operation. It uses the CollectionView which represents the sorted or filtered collection to get the correct next Person to select.  BookLibrary.Applications/Controllers/PersonController.cs  (see RemovePerson method)  The CollectionView is passed from the code-behind file to the ViewModel.  BookLibrary.Presentation/Views/PersonListView.xaml.cs  (see FirstTimeLoadedHandler) |  |
| Check for unsaved changes The application checks for unsaved changes before it closes. If there are unsaved changes then the user is asked how to proceed. This logic is in the ApplicationController.  BookLibrary.Applications/Controllers/ApplicationController.cs  (see ShellViewModelClosing method) |  |
| Save and restore the window location and size The sample application remembers its window location and the size when it is closed. This is done via an application setting.  BookLibrary.Applications/ViewModels/ShellViewModel.cs  (see ViewClosed method)  The ApplicationController is responsible to save the settings.  BookLibrary.Applications/Controllers/ApplicationController.cs  (see Shutdown method)  The next time the application starts it tries to restore the saved window location and size.  BookLibrary.Applications/ViewModels/ShellViewModel.cs  (see constructor) |  |
| Deploy the database file The BookLibrary application uses a SQL Compact database file to store its data. This database file resides in the sub-directory "Resources" of the application. A user doesn't have write permissions in the application directory by default. Therefore, we need to copy the database file into the user data directory first and load the database from there.  BookLibrary.Applications/Controllers/EntityController.cs (see Initialize) |  |