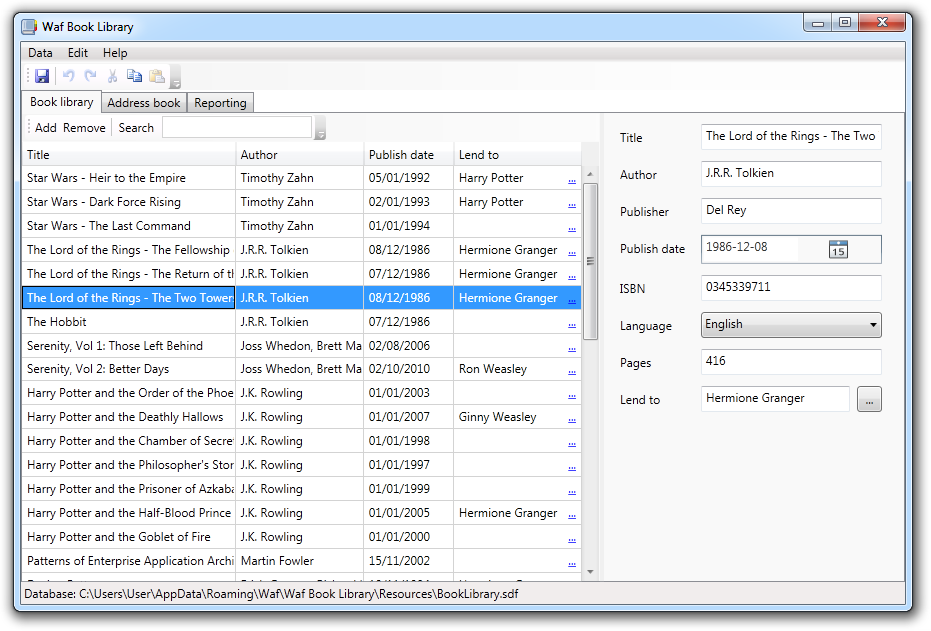
WAF Book Library

WPF Application Framework (WAF)

# Introduction

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

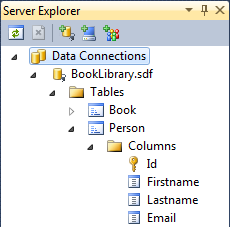
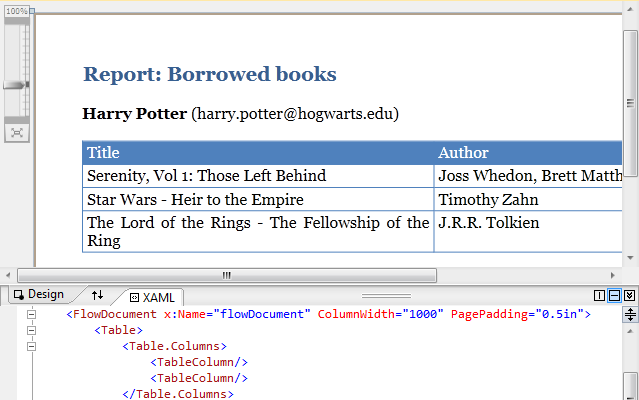
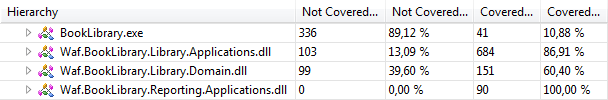


This sample application is part of the WPF Application Framework (WAF) [download](http://waf.codeplex.com/).

# 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).
* Usage of an Extension. This Extension is optional and can be deactivated via configuration file.  
    
  
* 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.
* Create reports via the WPF FlowDocument infrastructure.  
    
  
* All layers are unit tested. The Domain and Application layers are completely tested except the entity framework related code. The Presentation layer is partly tested.  
  Unit test code coverage:  
  

# Project Structure

BookLibrary.Library The main application.

BookLibrary.Reporting An extension which provides the Reporting feature.

BookLibrary.Library.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.Library.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.Library.Domain

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

BookLibrary.Reporting.Presentation

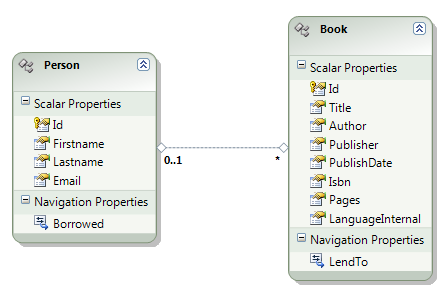
1. Controls Custom WPF Controls
2. DesignData Design time support
3. Reports WPF Reports (FlowDocuments)
4. Views WPF Views (Windows, UserControls)

BookLibrary.Reporting.Applications

1. Controllers Use case controllers
2. DataModels DataModels for DataTemplates
3. Reports Interfaces for the Reports
4. ViewModels ViewModels for the Views
5. Views Interfaces for the Views

# Domain Layer

BookLibrary.Library.Domain\BookLibraryModel.edmx



# Features

|  |  |
| --- | --- |
| Reporting Extension The BookLibrary application comes with the Reporting extension. When it is activated then it extends the application by the reporting feature. The activation is done in the BookLibrary.exe.config file or during development in the App.config file.  BookLibrary.Library.Presentation/App.config (see ModuleAssemblies)  In the config file the extension assemblies are added. The bootstrapping code was adapted to load the extension assemblies as well.  BookLibrary.Library.Presentation/App.xaml.cs (see OnStartup method)  The extension integrates its feature via the ShellService into the user interface.  BookLibrary.Reporting.Applications/Controllers/ModuleController.cs (see Initialize method)  The TabItem Reporting is only shown when the Reporting extension is loaded. Furthermore, the Reporting view is loaded on demand when the Tab is shown the first time (Better startup performance).  BookLibrary.Library.Presentation/Views/ShellWindow.xaml  (see TabItem with Header=”…Reporting”)  Note: All projects of the BookLibrary application build into the same Output directory. This way the extension assemblies are available for the application although they are not referenced. | D:\Dev\Codeplex\Waf-Temp\Documentation\BookLibrary\2.5\App Config.png  D:\Dev\Codeplex\Waf-Temp\Documentation\BookLibrary\2.5\Reporting Tab.png |
| Reporting via FlowDocument The Reporting extension uses the WPF FlowDocument infrastructure to create, show and print reports. The FlowDocuments are encapsulated inside a UserControl so that the design time support of Visual Studio and Expression Blend works.  BookLibrary.Reporting.Presentation/Reports/BookListReport.xaml  The FlowDocument is disconnected from the UserControl during runtime. This way the FlowDocument can be reused within the ReportView. Furthermore, the Reports implement an interface which follows the same concept as the IView interfaces.  BookLibrary.Reporting.Presentation/Reports/BookListReport.xaml.cs (see Constructor)  A ReportDataModel contains the data which is shown in the report. This object is set as DataContext of the FlowDocument.  BookLibrary.Reporting.Applications/DataModels/ BookListReportDataModel.cs  The Report and the ReportDataModel are created in the ModuleController.  BookLibrary.Reporting.Applications/Controllers/ModuleController.cs (see CreateBookListReport method)  When the Report is set into the ReportViewModel then it is shown by the FlowDocumentReader.  BookLibrary.Reporting.Presentation/Views/ReportView.xaml (see FlowDocumentReader)  Some Elements that can be used inside a FlowDocument do not support Binding (e.g. Table). The sample application provides enhanced versions of some Elements (BindableTable, ItemsElement, ContentElement) that supports the Binding.  BookLibrary.Reporting.Presentation/Controls/… |  |
| 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.Library.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.Library.Domain/Person.cs (see IPerson interface)  The MetadataType attribute defines the association between the entity class and the interface.  BookLibrary.Library.Domain/Person.cs (see Person class)  Enable the DataAnnotations infrastructure to use the MetadataType information.  BookLibrary.Library.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.Library.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 WAF ValidationHelper class is used to track the errors in the UI. These errors are synchronized with the IsValid flag of the ViewModel.  waf:ValidationHelper.IsEnabled="true"  waf:ValidationHelper.IsValid=  "{Binding IsValid, Mode=OneWayToSource}"  These attached properties are used on the Window element.  BookLibrary.Library.Presentation/Views/ShellWindow.xaml  The IsValid flag is used to enable/disable the Save command.  BookLibrary.Library.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.Library.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.Library.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.Library.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 SynchronizingCollection for this task.  BookLibrary.Library.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.Library.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.Library.Presentation/Views/BookListView.xaml.cs  (see FirstTimeLoadedHandler method)  The TextChanged event handler calls the Refresh method on the CollectionView.  BookLibrary.Library.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.Library.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.Library.Applications/Controllers/PersonController.cs  (see RemovePerson method)  The CollectionView is passed from the code-behind file to the ViewModel.  BookLibrary.Library.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 ModuleController.  BookLibrary.Library.Applications/Controllers/ModuleController.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.Library.Applications/ViewModels/ShellViewModel.cs  (see ViewClosed method)  The ModuleController is responsible to save the settings.  BookLibrary.Library.Applications/Controllers/ModuleController.cs  (see Shutdown method)  The next time the application starts it tries to restore the saved window location and size.  BookLibrary.Library.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.Library.Applications/Controllers/EntityController.cs  (see Initialize) |  |