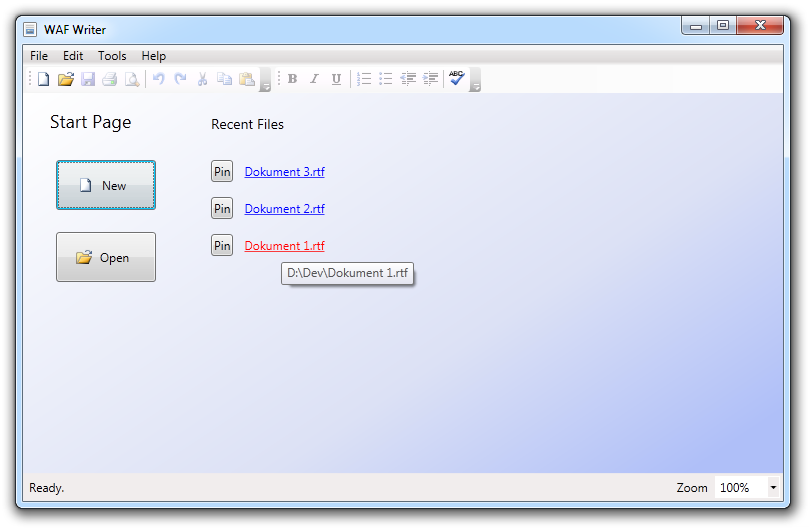
WAF Writer

WPF Application Framework (WAF)

# Note: This document is outdated!

# Introduction

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

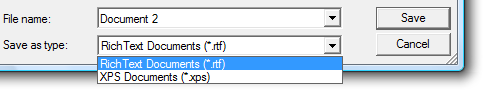
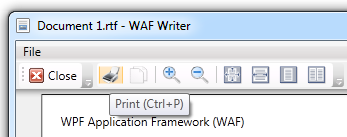
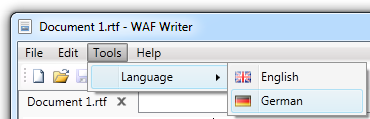
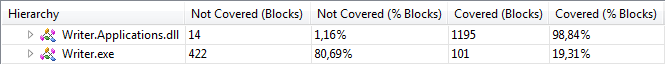


This sample application is part of the WPF Application Framework (WAF) [download](http://waf.codeplex.com/).

# Run the sample

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

# Highlights

* Layered Architecture and usage of the Model-View-ViewModel pattern (MVVM).
* Tabbed MDI (Multiple Document Interface).  
  
* Animated transition between the start page and the document views.
* Usage of the WAF RecentFileList which is integrated on the start page and the file menu.
* Document management with support for multiple file types. The application supports saving a text in the “.rtf” or “.xps” format.  
  
* Usage of the WAF message service and the open / save file dialog service.
* The application shows how to write an own dialog service for the print dialog.
* Print preview in the same window (Similar to Microsoft Word).  
  
* Complete localized application (English and German).  
  
* Full unit tested application layer and partly tested presentation layer (regarding code coverage).  
  

# Project Structure

Writer.Presentation

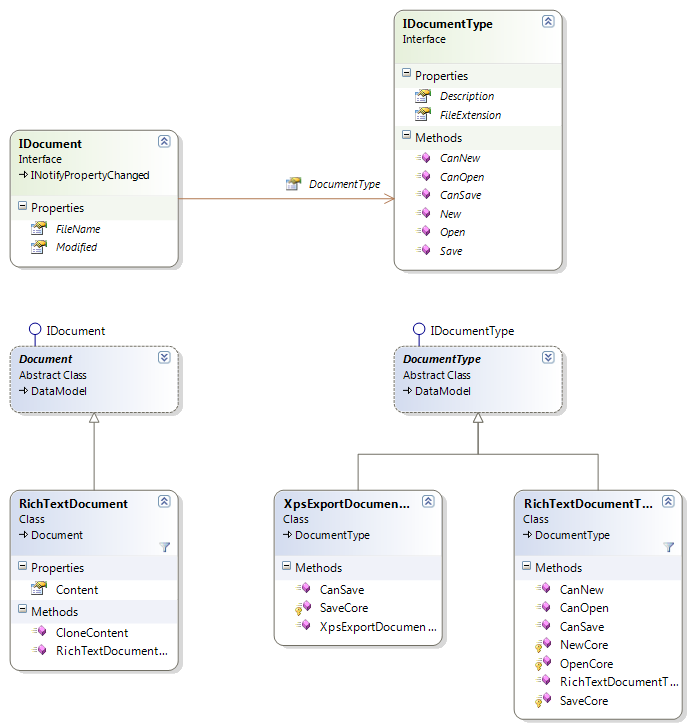
1. Converters Value converters
2. DesignData Design time support
3. Resources ResourceDictionaries, Images, Icons
4. Services UI service implementations
5. Views WPF Views (Windows, UserControls)

Writer.Applications

1. Controllers Use case controllers
2. Documents Implementation of document types and documents
3. Services Interfaces and implementation of services
4. ViewModels ViewModels for the Views
5. Views Interfaces for the Views

# Document Namespace

Writer.Applications\Documents\Overview.cd



# Features

|  |  |
| --- | --- |
| Open and save a document The MainView shows the open and save document commands.  Writer.Presentation/Views/MainView.xaml  They are bound to the FileService command properties. The FileService also exposes the current list of opened documents.  Writer.Applications/Services/FileService.cs  The FileService is only a mediator between the ViewModels and the FileController. The specific command implementation is found in the FileController.  Writer.Applications/Controllers/FileController.cs  The Open method looks for registered DocumentTypes which are able to open files. These DocumentTypes are shown in the OpenFileDialog as filter.  When the user selects a file to open then the OpenCore method delegates the Open operation to the DocumentType.  Writer.Applications/Documents/DocumentType.cs (see Open method)  This abstract class performs some checks and delegates the Open operation to the specific sub type.  Writer.Applications/Documents/RichTextDocumentType.cs  (see OpenCore method)  The OpenCore method reads the file stream and creates a RichTextDocument.  Writer.Applications/Documents/RichTextDocument.cs  This RichTextDocument is added to the FileService document list by the FileController.  The Save operation works very similar to the Open operation. |  |
| Open a file with the Windows Explorer When a user opens a file with the Windows Explorer then the Explorer starts the application and passes the file name via the command line arguments.  Writer.Applications/Controllers/ApplicationController.cs (see Run method)  Reading of the command line information is done in the EnvironmentService.  Writer.Applications/Services/EnvironmentService.cs  The ApplicationController is decoupled from the concrete service implementation via an interface. Therefore, it is possible to mock the service in unit tests.  Writer.Applications.Test/Controllers/ApplicationControllerTest.cs (see OpenFileViaCommandLine method)  Remark: To see this feature you have to register the .rtf file extension with the Writer.exe file. You might do this with the Windows Explorer's Open With context menu and then select Choose Default Program. |  |
| Show recent files on the start page and in the file menu Showing the recent files in the Start Page is done with a simple binding on the RecentFiles collection.  Writer.Presentation/Views/StartView.xaml (see ItemsControl)  To see the recent files in the File menu we need to add files dynamically to the menu.  Writer.Presentation/Views/MainView.xaml.cs  (see FileMenuItemSubmenuOpened method)  The RecentFileList is exposed by the FileService.  Writer.Applications/Services/FileService.cs (see RecentFileList property)  The FileController creates or loads the RecentFileList and it adds new items to the list.  Writer.Applications/Controllers/FileController.cs (see OpenCore method)  When the application shuts down the FileController sets the new RecentFileList into the application settings property which is saved in an xml file.  Writer.Applications/Controllers/FileController.cs (see Shutdown method)  Remark: The Visual Studio UI for Settings files doesn't show you the RecentFileList. But you can select Browse for the type and enter "System.Waf.Applications.RecentFileList". |  |
| Tabbed MDI (Multiple Document Interface) The documents are shown in a TabControl which works similar to the Visual Studio document tabs.  Writer.Presentation/Views/MainView.xaml (see x:Name="documentView")  The TabControl is bound to the DocumentViews and ActiveDocumentView property.  Writer.Applications/ViewModels/MainViewModel.cs  The Document Views are managed by the RichTextDocumentController. This controller synchronizes the documents with the document views.  Writer.Applications/Controllers/RichTextDocumentController.cs |  |
| Save changes before a document is closed Before a document is closed (Close command) or the application is closed the Writer application asks the user if he wants to save his documents first.  The application tracks if a document needs to be saved. This is done by setting the Document.Modified property to true.  Writer.Presentation/Views/RichTextView.xaml.cs  (see RichTextBoxChanged method)  When the user closes a document the CanDocumentsClose method is called.  Writer.Applications/Controllers/FileController.cs  (see CanDocumentsClose method)  This method checks for modified documents which should be saved first. When a document is found it shows the SaveChangesWindow.  The same happens when the user closes the application. In this case the ApplicationController calls the CloseAll method on the FileController.  Writer.Applications/Controllers/ApplicationController.cs  (see ShellViewModelClosing method)  In this scenario it is possible that more than one document needs to be saved. The SaveChangesWindow shows a list of all affected documents. |  |
| Print preview and print The PrintController is responsible for the PrintPreview and the Print operation. It implements the print commands and sets them into the MainViewModel.  Pressing the PrintPreview button creates a XPS document of the current text. The XPS document is shown in the PrintPreviewView.  Writer.Applications/Controllers/PrintController.cs  (see ShowPrintPreview method)  The Print command shows the Print Dialog.  Wirter.Presentation/Services/PrintDialogService.cs  After the user has chosen his printer the Print Dialog object can be used to print the document. |  |
| Formatting toolbar The formatting toolbar uses the WPF EditingCommands.  <ToggleButton Command="EditingCommands.ToggleBold" .../>  Unfortunately, these commands do not support the toggle behavior. Thus, the IsChecked property of the ToggleButton must be controlled manually.  Writer.Presentation/Views/MainView.xaml (see ToggleButton in ToolBar)  The RichTextViewModel implements the state properties for the ToggleButtons.  Writer.Applications/ViewModels/RichTextViewModel.cs  These properties are set in the code-behind file when the selection or text in the RichTextBox changes.  Writer.Presentation/Views/RichTextView.xaml.cs  (see UpdateFormattingProperties method) |  |
| Spell checking and dynamic context menu The Writer comes with the WPF spell checking functionality. It is enabled with the SpellCheck.IsEnabled property.  Writer.Presentation/Views/RichTextView.xaml (see RichTextBox)  We define a ContextMenu for the RichTextBox. It contains static menu items (always visible) and dynamic ones (suggestions from the spell checker). The dynamic menu items are added in the code-behind file.  Writer.Presentation/Views/RichTextView.xaml.cs  (see RichTextBoxContextMenuOpening method)  Remark: If you are not using English as input language and the spell checker doesn't work then you might have to install a .NET Framework language pack first. |  |
| Animate the transition of a view change An animation is used when the application replaces the Start Page view with a document view.  The storyboards for the animations are defined in the VisualStateManager of the root container.  Writer.Presentation/Views/MainView.xaml (see x:Name="rootContainer")  The Visual States are activated in the code-behind file when the ContentViewState property is changed.  Writer.Presentation/Views/MainView.xaml.cs  (see ContentViewState property)  This property is controlled by the MainViewModel class.  Writer.Applications/ViewModels/MainViewModel.cs  (see DocumentViewsCollectionChanged method) |  |
| Document zoom The user can set a zoom level for every document.  Writer.Presentation/Views/ShellWindow.xaml (see x:Name="zoomBox")  The update behavior of the zoom ComboBox is improved by listening to events in the code-behind file. They are used to manually update the Binding.  Writer.Presentation/Views/ShellWindow.xaml.cs  (see ZoomBoxDropDownClosedHandler method)  The zoom level is bound to the ShellService.Zoom property.  Writer.Applications/Services/ShellService.cs (see Zoom property)  Because every document view has its own zoom level the ViewModels need to remember this information. The ZoomViewModel base class is responsible to synchronize the zoom level with the ShellService when the associated View is visible.  Writer.Applications/ViewModels/ZoomViewModel  The RichTextViewModel and the PrintPreviewViewModel inherit from the ZoomViewModel.  The Views need to synchronize their IsVisible state with the associated ViewModel.  Writer.Presentation/Views/PrintPreviewView.xaml.cs  (see IsVisibleChangedHandler method) |  |
| Localization and language selector The Writer application is localized in English and German. The user can select the language in the menu.  The language commands call the SelectLanguage method.  Writer.Applications/ViewModels/MainViewModel.cs  (see SelectLanguage method)  The user is asked to restart the application. The ApplicationController is responsible to save the language setting.  Writer.Applications/Controllers/ApplicationController.cs  (see Shutdown method)  The next time the application starts the ApplicationController initializes the saved language settings.  Writer.Applications/Controllers/ApplicationController.cs  (see InitializeCultures method)  and  Writer.Presentation/Services/PresentationService.cs  (see InitializeCultures 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.  Writer.Applications/ViewModels/ShellViewModel.cs  (see ViewClosed method)  The ApplicationController is responsible to save the settings.  Writer.Applications/Controllers/ApplicationController.cs  (see Shutdown method)  The next time the application starts it tries to restore the saved window location and size.  Writer.Applications/ViewModels/ShellViewModel.cs  (see constructor) |  |