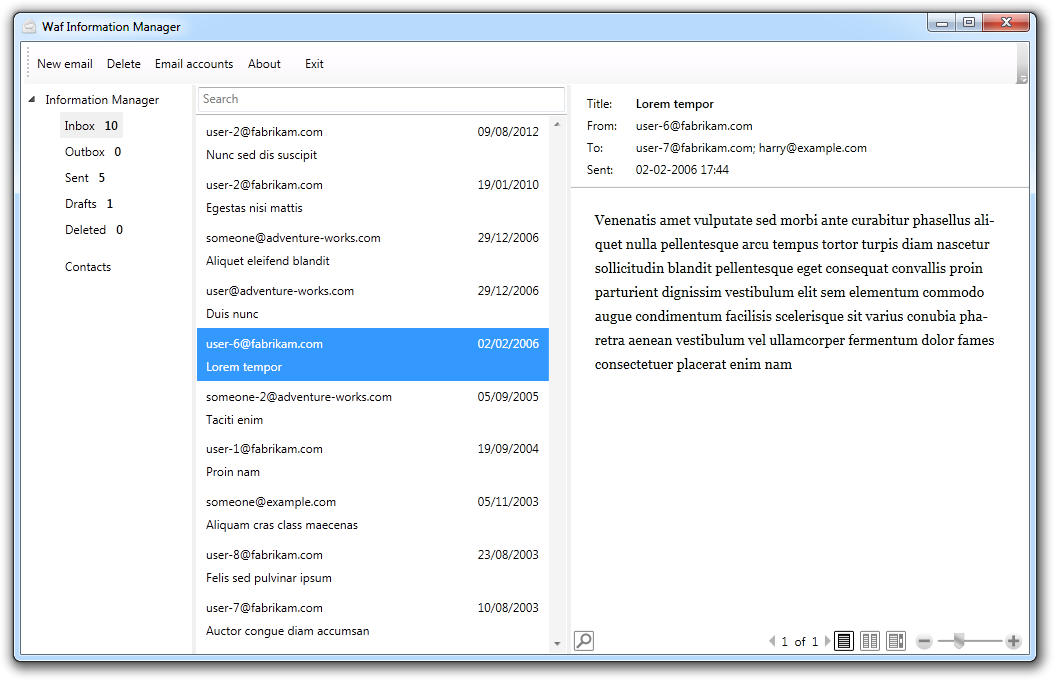
Waf Information Manager

WPF Application Framework (WAF)

# Introduction

The Information Manager sample application shows how to create a modular application with the WPF Application Framework (WAF).

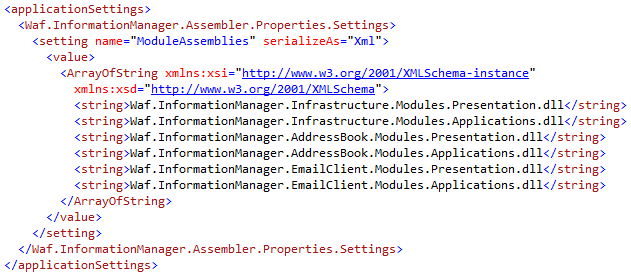
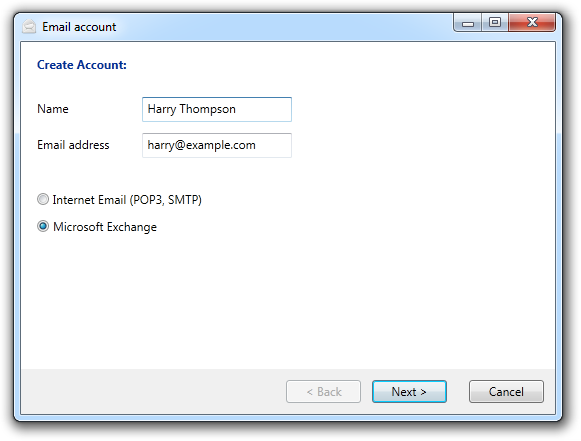
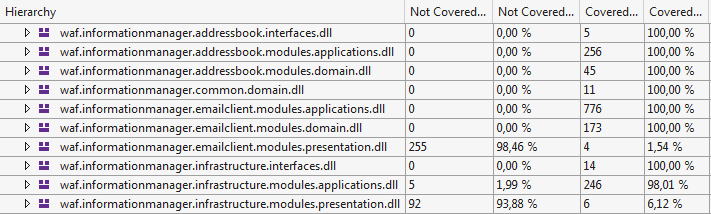


This sample application is part of the WPF Application Framework (WAF) [download](http://waf.codeplex.com/).

# Run the sample

1. Open the WpfApplicationFramework solution.
2. Build the solution.
3. Set the Assembler project as StartUp project and start it in Debug mode.

# Highlights

* Layered Architecture and usage of the Model-View-ViewModel pattern (MVVM).
* Modular software architecture which can be controlled via configuration file.  
  
* Wizard for the configuration of an email account.  
  
* All modules save their data in a single file. The file is a ZIP container with the same format than the new Microsoft Office document format (2007 and newer). The sample application uses the Package API and the DataContractSerializer from the .NET Framework.  
    
  Path: %AppData%\Waf\Waf Information Manager\InformationManager.datx  
    
  The file can be renamed to a .zip extension for further investigation. Every module saves its own data as XML file into the file container.
* The domain objects provide validation rules which are reflected on the user interface.
* All layers are unit tested. The Domain and Application layers are completely tested. The Presentation layer is partly tested.  
  Unit test code coverage:  
  

# Architecture



## Assembler

Responsible to initialize the application and load all modules.

## Infrastructure Module

Responsible for the infrastructure. This contains the ShellView and the toolbar. Furthermore, it handles the file persistency of the modules.

Infrastructure.Interfaces: It provides services for other modules to use the infrastructure.

## AddressBook Module

Responsible for the address book features. This includes the master / detail views of all contacts and the SelectContactView.

AddressBook.Interfaces: It provides services for other modules to let the user select a contact and return it.

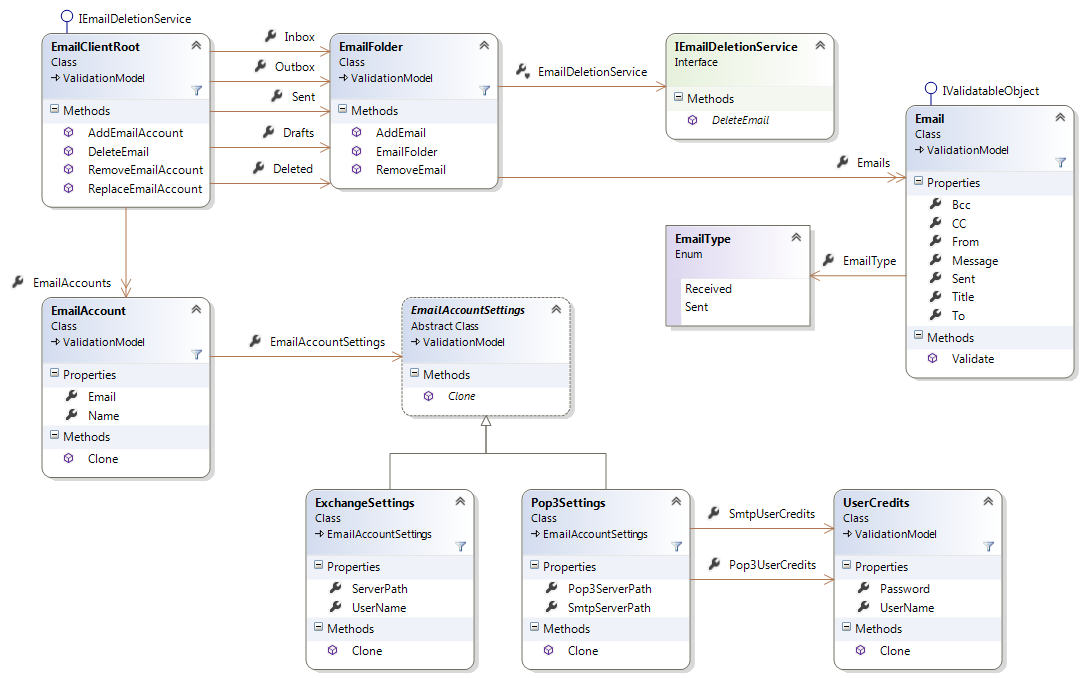
## EmailClient Module

Responsible for the email client features. This includes the master / detail views of all email folders and the email account management. Creating and editing of an email account is done via a wizard.

## Common

Provides reusable classes. It includes the base class for a Model, the SearchBox WPF Control and the common WPF styles. These types are shared by all modules.

# EmailClient Domain



# Features

|  |  |
| --- | --- |
| Modularization This sample application uses a modular software architecture. It consists of three modules: Infrastructure, AddressBook and EmailClient. The application configuration file defines which modules should be loaded at runtime.  Assembler/App.config (see ModuleAssemblies)  It is possible to exclude the EmailClient module because it does not require services from other modules. If we want to exclude one of the other modules then we need to provide a replacement that provides the services defined in the interface assemblies.  Infrastructure.Interfaces/Applications (see the interfaces)  AddressBook.Interfaces/Applications (see the interfaces)  The assembler project is responsible for the startup sequence. It reads the application configuration file and loads the defined modules.  Assembler/App.xaml.cs (see OnStartup method)  Important: All projects are configured so that the build results come together in the same directory. See the “InformationManager/Output” directory. This is necessary so that the module assemblies are found. |  |
| File Persistency for all modules The Information Manager comes with an extensible file persistency. It saves all data in a single file:  %AppData%\Waf\Waf Information Manager\InformationManager.datx  The file is a ZIP container which is managed by the DocumentController via the Package API. This is the same document format as Microsoft Office uses to save its documents since the 2007 version.  Infrastructure.Modules.Applications/Controllers/DocumentController.cs  The modules can create a Stream to save their own data within the ZIP container.  AddressBook.Modules.Applications/Controllers/ModuleController.cs (see Initialize and Shutdown methods)  The data is serialized with the DataContractSerializer into an XML Stream. And this results in a XML file within the ZIP container.  Advantage: The modules can persist their data independent of each other. If a new module is developed for the Information Manager then it just creates a new Stream to persist its own data. But the user has all the data in a single file. |  |
| Extensible navigation view The infrastructure module provides an extensible navigation view. Other modules can add new navigation nodes into this view.  Infrastructure.Interfaces/Applications/INavigationService.cs  The EmailClient module does this within the ModuleController.  EmailClient.Modules.Applications/Controllers/ModuleController.cs (see Initialize method)  The INavigationNode interface allows a module to update the ItemCount of a navigation node. The EmailClient does this in the nested class ItemCountSynchronizer within the ModuleController. |  |
| Context sensitive toolbar The application comes with a context sensitive toolbar. The toolbar is implemented in the Infrastructure module. This module allows the manipulation of the toolbar buttons via the shell service.  Infrastructure.Interfaces/Applications/IShellService.cs  The ModuleController of the EmailClient module is responsible to update the toolbar. It changes the toolbar when the email view is shown or when something else is shown.  EmailClient.Modules.Applications/Controllers/ModuleController.cs (see ShowEmails and CloseCurrentView methods) |  |
| Select an email address from the address book The new email dialog allows us to select a contact from the address book. The email dialog is provided by the EmailClient module and the select contact dialog is provided by the AddressBook module. The AddressBook module exposes this dialog via a service.  AddressBook.Interfaces/Applications/IAddressBookService.cs  The NewEmailController consumes this service and calls it when the user wants to select a contact.  EmailClient.Modules.Applications/Controllers/NewEmailControllers.cs (see SelectContact method) |  |
| Validation with DataAnnotations The .NET Framework ships a validation framework in the System.ComponentModel.DataAnnotations namespace.  But WPF Binding does not support DataAnnotations directly. One way to activate the Validation in WPF Binding is by using the IDataErrorInfo interface.  {Binding ... ValidatesOnDataErrors=true}  WAF provides the DataErrorInfoSupport class which connects the DataAnnotations validation with the IDataErrorInfo interface.  Common.Domain/ValidationModel.cs (see Error and this[] property)  For simple validation rules the validation attributes can be used.  EmailClient.Modules.Domain/Emails/Email.cs (see Title property)  More advanced validation scenarios can be solved by implementing the IValidatableObject interface.  EmailClient.Modules.Domain/Emails/Email.cs (see Validate method) |  |
| Disable commands when an input error exists The email accounts wizard disables the Next 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.  EmailClient.Modules.Applications/ViewModels/ EditEmailAccountViewModel.cs (see IsValid property)  These attached properties are used on the Window element.  waf:ValidationHelper.IsEnabled="true"  waf:ValidationHelper.IsValid=  "{Binding IsValid, Mode=OneWayToSource}"  EmailClient.Modules.Presentation/Views/EditEmailAccountWindow.xaml  The IsValid flag is used to enable/disable the Next command.  EmailClient.Modules.Applications/Controllers/EditEmailAccountController.cs (see CanNext method) |  |
| Wizard for editing an email account The EmailClient module includes a wizard to create and edit an email account. The EditEmailAccountController is responsible for the wizard workflow.  EmailClient.Modules.Applications/Controllers/EditEmailAccountController.cs  This controller implements the back and next command. It shows the correct view when the user navigates through the wizard.  The Back and Next button has triggers so that an appropriate tooltip is shown which explains why the button is disabled.  EmailClient.Modules.Presentation/Views/EditEmailAccountWindow.xaml (see Back and Next button) |  |
| Reusable SearchBox Control The AddressBook module and the EmailClient module show a search box to filter for contacts or emails.  The sample application comes with a SearchBox custom control that encapsulates the behavior of a search box. It is designed for MVVM so that it is easy to use Bindings with this custom control.  Common.Presentation/Controls/SearchBox.cs  Common.Presentation/Themes/Generic/SearchBox.xaml |  |
| 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.  Infrastructure.Modules.Applications/ViewModels/ShellViewModel.cs  (see ViewClosed method)  The ModuleController is responsible to save the settings.  Infrastructure.Modules.Applications/Controllers/ModuleController.cs  (see Shutdown method)  The next time the application starts it tries to restore the saved window location and size.  Infrastructure.Modules.Applications/ViewModels/ShellViewModel.cs  (see Constructor) |  |