Smart Client Software Factory Demo Application

Demo Script



The goal of this demo script is to help presenters give a presentation that illustrates the main aspects of SC-SF such as WorkItems, Commands, EventBroker, Services, Workspaces and the Dependency Injection pattern. This demo stript provides step-by-step instructions to create a SC-SF application. The application consists of two **Business Modules**:

- Notifications module: this module populates the Main Menu Strip of the Shell with two items and adds them as invokers of the *DumpWorkItem* and *ShowNews* commands respectively. It also adds two views to the Shell.
- Stocks module: this module shows BuyStock and Reports SmartParts in the the Shell.
 The Shell is made up of two Workspaces: n
 OutlookBarWorkspace (the one in the right) and a
 DockPanelWorkspace (the one in the left). Both
 Workspaces are available in <u>SCSF Contrib</u> web site.

Key Technologies:

The following technologies are utilized within this demo script:

Technology / Product	Version
1. Visual Studio 2008	RTM
2NET Framework	3.5
3. Smart Client Software Factory	April 2008
4. SCSFContrib.CompositeUI.WinForms extensions	1.5

Before starting

Create a new folder named **temp** in the root directory "C:\". In that folder, the DemoApp will place its log file.

Step-by-step Walkthrough

Estimated time to complete the demo script: **30 minutes**.

Use the guidance package to create a new Smart Client Solution

Act	ion	Script	Screenshot
 1. 2. 3. 4. 5. 	In Visual Studio, point to New on the File menu, and then click Project. In the New Project dialog box, expand the Guidance Packages node. Click the Smart Client Development - April 2008 project type. In the Templates window, click Smart Client Application (C#). Change the Name to DemoApp. (Optional) Change the location for the solution to C:\Projects\DemoApp (this	 Use the guidance package to create a new Smart Client Solution 	New Project Image: Intelligence Projects Business Intelligence Projects Visual Studio installed templates Business Intelligence Projects Visual Studio installed templates Business Intelligence Projects Similar Clerr Application (Visual Basic) Guidance Packages Application met - April 2008 Guidance Packages Development Similar Clerr Application that uses CAB and Enterprise Library Umm: DemoApp Uccabon: C(Projects) Solution flags: DemoApp Uccabon: C(Projects) Solution flags: DemoApp Uccabon: C(Projects) DemoApp Monte Uccabon: C(Projects) DemoApp Monte Uccabon: C(Projects) DemoApp Monte
6.	c:\projects\DemoApp (this path will be used throughout the whole script). Click OK .		
8.	Enter the location of the Composite UI Application Block, Enterprise Library, and the offline application blocks assemblies. (The wizard sets the default location to the Lib subfolder of the folder where you installed the software factory.) Enter DemoWorkshop as the Root namespace for your application. This value appears as the first part of every	• The Smart Client Application template references the CreateSolution recipe. The Guidance Automation Extensions calls the CreateSolution recipe when you unfold the template. The CreateSolution recipe starts a wizard to gather information that it uses to customize the generated source code	Contract
9. 10.	namespace in the generated solution. Unselect the option Create a separate module to define the layout for the shell . In this application, you will not use a separate module to define the layout for the shell. Instead, you will define the layout in a view within the Shell project. Unselect the Allow solution to host WPF SmartParts check box. In this application you		

SmartParts; therefore you do not need support for WPF SmartParts.

11. Select the **Show** documentation after recipe completes check box. You will see after the recipe completes a summary of the recipe actions and suggested next steps.

12. Click **Finish**. The recipe unfolds the Smart Client Solution template.

Add SCSFContrib binaries

Act	ion	Script	Screenshot		
1.	Go to the SCSFContrib project page: http://www.codeplex.com/sc sfcontrib. In the Source Code tab	 SCSFContrib is a community- developed library of extensions to the patterns & practices Smart Client Software Factory. We are going to use the 	C (C VProjects) DemoApp) Lib Ele Edt Yew Pavortes Icols Heb C Back - O P (Pointes Icols Heb Address C (Projects) DemoApp) Lib Name - Marcosoft: Practices Composite UI, WriFrms. dl Marcosoft: Practices Composite UI, WriFr. dl	Size Type Date Modified 156 KB Application Extension 16/05/2007 05:05 p 74 KB Application Extension 16/05/2007 05:05 p 70 KB Application Extension 16/05/2007 05:05 p	Co Attributes m. A m. A
3.	download the last Change Set that contains the source code of the project. Extract the content from the	e code extensions for WinForms in the application.	Microsoft Practices Entreprises Unray Oceanian, all Microsoft Practices Entreprises Unray Obta and Microsoft Practices Entreprises Unray Obta and Microsoft Practices Entreprises Unray Obta and Microsoft Practices Entreprises Unray Charge State Microsoft Practices Entreprises Microsoft Practices Entreprises Microsoft Practices Entreprises Microsoft Practices Microsoft Mi	136 KB Application Extension 10/05/20/01 05:20 p 90 KB Application Extension 10/05/20/07 05:20 p 94 KB Application Extension 10/05/20/07 05:20 p 94 KB Application Extension 10/05/20/07 05:20 p 95 KB Application Extension 10/05/20/07 05:20 p 93 KB Application Extension 10/05/20/07 05:20 p 94 KB Application Extension 10/05/20/07 05:20 p 95 KB Application Extension 10/05/20/07 05:20 p 95 KB Application Extension 10/05/20/07 05:20 p 95 KB Application Extension 10/05/20/07 05:05 p	m. A m. A m. A m. A m. A m. A m. A m. A
	.zip and compile the project Trunk\src\Extensions\WinFor ms\SCSFContrib.CompositeUI .WinForms\SCSFContrib.Com positeUI.WinForms.csproj.		Microsoft Predices Smart Clent Enterpriset brary, dl 46 KB SCS-Control Companieul Winforms, dl 44 KB Welfer Luo, WinformsUl Looding, dl 408 KB	4648 Application Extension 16(/5)(2007 06/05 p.m. 4448 Application Extension 10/(0207 020-04 p.m. 408 K8 Application Extension 04/(11/2007 10:23 a.m.)	л. А л. А л. А л. А
4.	Copy the SCSFContrib.CompositeUI.Wi nForms.dll and WeifenLuo.WinFormsUI.Dock ing.dll assemblies to the Lib folder of your application				

- In Solution Explorer, right-click the Shell project and select
 Add Reference.... In the
 Browse tab, go to the Lib
 folder of your application
 (C:\Projects\DemoApp\Lib)
 and select
 SCSFContrib.CompositeUI.Wi
 nForms.dll,
 WeifenLuo.WinFormsUI.Dock
 ing.dll.
- 6. Click OK.

Add references to the SCSFContrib.CompositeUI.Win Forms and WeifenLuo.WinFormsUI.Docki ng.dll assemblies in the Shell project to be able to use the DockPanelWorkspace and the OutlookBarWorkspace.

Look in: 🗀 L	.ib 🔽 🗿 🗊 🖽 -
Microsoft.Pra	actices.EnterpriseLibrary.Logging.dll
Microsoft.Pra	actices.ObjectBuilder.dll
Microsoft.Pra	actices.SmartClient.ConnectionMonitor.dll
Microsoft.Pr	actices.SmartClient.DisconnectedAgent.dll
Microsoft.Pra	actices.SmartClient.EndpointCatalog.dll
SCSEContrib	Compositel II WinForms di
WeifenLuo.V	WinFormsUI.Docking.dll
•	1
File <u>n</u> ame:	"SCSFContrib.CompositeUI.WinForms.dll" "WeifenLuo.WinForms 🗾
iles of <u>type</u> :	Component Files (*.dll;*.tlb;*.olb;*.ocx;*.exe;*.manifest)
	· · · · · · · · · · · · · · · · · · ·

Customizing the Shell

Action	Script	Screenshot
 Double-click in ShellForm.cs file on the Shell project to open the View Designer. Open the Toolbox. Right-click the Toolbox and select Choose Items. In the .NET Framework Components tab click on Browse and navigate to the Lib folder of your application. Select the SCSFContrib.CompositeUI.Wi nForms.dll assembly. Click OK. 	 Add the DockPanelWorkspace and the OutlookBarWorkspace to the Toolbox. This allows you to drag and drop these controls. 	Choose Toolbox Items Y Maintenance Tasks SSIS Data Flow Items SSIS Control Flow Items .NET Framework Components COM Components Activities Mantenance Tasks SSIS Data Flow Items SSIS Control Flow Items AccessDataSource System, Web, UI, Web/Controls System, Web (2,0,0,0) Global Ass Account Microsoft, AnalysisServices Microsoft, AnalysisService, Global Ass Activity System, Workflow, ComponentM, System, Workflow, ComponentM, Global Ass AddAttributeAction Microsoft, Practices, RecipeFram., Microsoft, Practices, RecipeFram., Microsoft, Practices, RecipeFram., AddConfigurationSect., Microsoft, Practices, RecipeFram., Microsoft, Practices, RecipeFram., Microsoft, Practices, RecipeFram., AddCustonAction Microsoft, Practices, RecipeFram., Microsoft, Practices, Reci, Ci/Program Enter: Clear Clear Clear Circuit Circuit Language: Invariant Language (Invariant Country) Clear Circuit Contextices Version: 2.0.0.0 Cot Cancel Beset Circuit
 Select the Left and Right DeckWorkspaces and delete them. Drag an OutlookBarWorkspace to the <i>left</i> panel of the SplitContainer. Set its Dock property to <i>Fill</i> and change its Name to _<i>leftWorkspace</i>. Drag a DockPanelWorkspace to the <i>right</i> panel of the SplitContainer. Set its Dock and DocumentStyle properties to 	 Change the Shell layout. Put an OutlookBarWorkspace on the left and a DockPanelWorkspace on the right. 	Shelform.cs (Design)

Fill and DockingWindow respectively and change its **Name** to _*rightWorkspace*.

Add the LoggingService global service

Create the ILoggingService interface

Action	Script	Screenshot	
 Right-click in the Services folder of Infrastructure.Interface project and point to Add -> New Item In the Add New Item dialog box, select Interface and enter ILoggingService.cs as the Name of the file. 	 Create an interface for the logging service. Locate the interface in the Infrastructure.Interface project so that it is available for all modules. 	Add New Rem - Infrastructure Interface Jengiates: Visual Studio installed templates: Plow Document (WPF) Page (WPF) Class Static installed templates: Class Static installed templates: Class Static installed templates: Class Static installed templates: Class Web Configuration File Class Web Configuration File Component Class SQL Distabase MRL Schema XSL File Mindow Storph File Corsor File Markeler Class Storph File Thistaller Class Storph File Thistaler Class Storph File Thistaler Class Storph File Thistaler Class Storph File Thistaler Class Storph File Instaler Class Storph File Instaler Class Storph File Instaler Class Storph File Instaler Class Storph File Annemic Information File Application Configuration File Application Class Storph File Instaler Class Storph File Instal	2 X

3. Open the ILoggingService.cs file created in the previous step.

4. Replace the interface definition with the following:

C#

```
public interface ILoggingService
{
    void Log(string message);
}
```

Implement the service

Action	Script	Screenshot

1.	Create a Services folder in the Infrastructure.Module	Create the class that represents the logging service.	Add New Item - Infrastructure.Lib Iempletes: Visual Studio installed templa	rary tes	
2. 3.	Right-click the Services folder of the Infrastructure.Module project and point to Add -> Class In the Add New Item dialog		Flow Document (WPF) ResourceDictionary (WPF) Custom Control (WPF) El User Control El Interted Porm El Control Market Porm Market Porm Market Por Market P	Dege (WPF) Construction (WPF) Construction (WPF) Configuration File Configuration File Configuration File Construction Co	PageFunction (WPF) Mndow (WPF) Mndow (WPF) Mndows Form Custom Control Wdb Custom Control Wdb Custom Control DataSet XSIT File Report Mndows Service MDModws Service MDModws Service MDMOdws Control Control Mndows Service MDMOdws Service MDMOdws Control Control Control MDMOdws Control MD
L	box, select Class and enter LoggingService.cs as the Name of the file.		An empty class definition Name: LoggingService.	4	dd

? ×

4. Open the LogginService.cs file created in the previous step.

5. Add the following using statements at the top of the file:

C#

```
using DemoWorkshop.Infrastructure.Interface.Services;
using Microsoft.Practices.CompositeUI;
using System.IO;
```

6. Replace the class definition with the following:

C#

```
[Service(typeof(ILoggingService))]
public class LoggingService : ILoggingService
{
    #region ILoggingService Members
    public void Log(string message)
    {
        File.AppendAllText("C:\\temp\\log.txt", message);
    }
    #endregion
}
```

The [Service] attribute indicates to ObjectBuilder that it has to register the logging service in the **RootWorkItem**. This service will be global and available to all modules.

Add Notifications module

Action

Script

Screenshot

 In Solution Explorer, right-click the Source solution folder, point to Smart Client Software Factory, and then click Add Business Module (C#). The Add New Project dialog box appears with the Add Business Module (C#) template selected. Enter Notifications as the Name and set the Location to the Source folder of the solution. Click OK. 	 Add the Notifications Business Module. Modules are distinct deployment units of a Composite UI Application Block application. You use modules to encapsulate a set of concerns of your application and deploy them to different users or applications. A Business Module has at least one WorkItem (specifically, a ControlledWorkItem) and contains business logic elements. Typically, it includes some combination of services, views, presenters, and business entities. 	Add New Project 2 × Project types: Implates: MET Framework 3.5 10 Business Intelligence Projects Visual Studio installed templates Add Business Module (V8) Usual Date Add Foundational Module (C2) Add Business Module (V8) Cother Projects Add Foundational Module (C2) Add Foundational Module (V8) Usual Cather Projects Add Foundational Module (C2) Add Foundational Module (V8) Usual Cather Projects Add Foundational Module (C2) Add Foundational Module (V8) Usual Date Image: Cather Cat
 The guidance package displays the Add Business Module wizard. Unselect the option Create an interface library for this module. If you select this option, the recipe will create an additional project to contain the elements that provide the public interface to the assembly. Unselect the option Create a unit test project for this module. If you select this option, the recipe will create a test project for the module with test classes for your module components. Select the option Show documentation after recipe completes to see a summary of the recipe actions and suggested next steps after the recipe completes. Click Finish. 	 The guidance package will generate a new class library project named Notifications. The Module class derives from the CAB class ModuleInit. CAB calls the Load method of this class on startup. The Load method contains code to create and run a new WorkItem. This WorkItem is the module's main WorkItem. The ModuleController class contains methods that you can modify to customize the behavior of the module on startup. You can add services or display user-interface items. The project also contains the following folders: The Constants folder contains four classes named CommandNames, EventTopicNames, UIExtensionSiteNames, and WorkspaceNames. You can modify these classes to define module-specific identifiers for your commands, event topics, UIExtensionSites, and Workspaces. The Services folder, where you 	Ad Business Module Verset Module growter Ottoos
	of business services.The Views folder, where you	



Add News view to Notifications module

Using Add View (with presenter)... recipe

Action

Script

Screenshot

1.	In Solution Explorer, right-click the Views folder of the Notifications project, point to	•	The recipe generates: A view interface. The presenter class uses this interface to	Add View (with presenter).	resenter	াম
2.	Smart Client SoftwareFactory, and then click AddView (with presenter)In the wizard, enter News inthe View Name field andselect the Showdocumentation after recipecompletes option to see asummary of the recipe actions	•	communicate with the view. You will modify this interface. A view implementation user control. This class derives from UserControl and has the [SmartPart] attribute. The user control also implements the view interface and contains a reference to its presenter. You	Add Wew with Presenter properties	Vew name: Vervs: └ create a folder for the vew ✓ Show documentation after recipe completes	Solution Preview Source Provinciations Ververs Ververs Network, CS Network
3.	and suggested next steps after the recipe completes. If Create a folder for the view is selected, the recipe will create a folder and place the new items in this folder. Click Finish .	•	 will modify this class to call the presenter for user-interface actions that affect other views or business logic. A presenter class for the view. This class extends the Presenter base class defined in Infrastructure.Interface project and contains the business logic for the view. You will modify this class to update the view for your business logic. 		< Previous	Igent > Enish Cancel

Customizing News view

- 1. In the Views folder of the Notifications project, open the INews.cs file.
- 2. Paste the following method declaration inside the interface definition:

C#

void ShowNews(string n);

This method will be called from the presenter whenever news has to be displayed to the user.

- 3. In the Views folder of the Notifications project, open the NewsPresenter.cs file.
- 4. Add the following using statements at the top of the file.

C#

using DemoWorkshop.Infrastructure.Interface.Services; using Microsoft.Practices.CompositeUI.SmartParts;

5. Replace the **OnViewReady** method with the following code.

C#

```
public override void OnViewReady()
{
    string[] news = { "Some text, some text, some text", "Some text, some text, some text" };
    foreach (string n in news)
```

```
{
    View.ShowNews(n);
}
base.OnViewReady();
```

This method will be called when the view is initialized and will populate the view with news.

6. Add the following two methods to the body of the **NewsPresenter** class.

```
C#
private void DisposeView(object smartpart, WorkItem workItem)
{
    if (smartpart is IDisposable) ((IDisposable)smartpart).Dispose();
    workItem.SmartParts.Remove(smartpart);
}
public void ChangeTitle()
{
    IWorkspaceLocatorService locator = WorkItem.Services.Get<IWorkspaceLocatorService>();
    IWorkspace wks = locator.FindContainingWorkspace(WorkItem, View);
    wks.ApplySmartPartInfo(View, new SmartPartInfo("New Title", ""));
}
```

The **ChangeTitle** method locates the workspace where the view is showed and applies a new **SmartPartInfo** with a new title. The **DisposeView** method disposes the current view if it is disposable.

7. Add the following line of code at the bottom of the **OnCloseView** method:

C#

}

DisposeView(View, WorkItem);

- 8. Double-click in the News.cs file in the Views folder of the Notifications project. This will open the Designer.
- 9. Set the Size property of control to 349, 200.
- 10. Drag a Label to the top of the view. Set its Name to NewsLabel and erase the text in the Text property.
- 11. Drag two **Buttons** to the view surface. Set their **Text** properties to "*Change SmartPartInfo*" and "*Close View Programatically*" respectively. Adjust the size of the buttons to see the text on them.
- 12. Double-click on the "Change SmartPartInfo" button to auto-generate the handler of Click event.
- 13. Add the following code into the body of the handler.

```
C#
```

```
_presenter.ChangeTitle();
```

14. Go back to the **Design** of the **News** view and double-click on the "*Close View Programatically*" button to auto-generate the handler of **Click** event.

15. Add the following code into the body of the method.

C#

```
_presenter.OnCloseView();
```

16. Replace the head of the **News** class with the following:

C#

```
public partial class News : UserControl, INews, ISmartPartInfoProvider
```

In this way, the **News** class implements **ISmartPartInfoProvider**.

17. Implement the interfaces **INews** and **ISmartPartInfoProvider**. To do this past the following code in the **News** class body.

```
C#
#region INews Members
public void ShowNews(string n)
{
    NewsLabel.Text += n + Environment.NewLine;
}
#endregion
#region ISmartPartInfoProvider Members
public ISmartPartInfo GetSmartPartInfo(Type smartPartInfoType)
{
    ISmartPartInfo spi = (ISmartPartInfo)Activator.CreateInstance(smartPartInfoType);
    spi.Title = "Today News";
    return spi;
}
#endregion
```

Add Alerts view to Notifications module

Using hud view (with presencer). recipe	Using Add View	(with	presenter) recipe
---	-----------------------	-------	-----------	----------

Action	Script	Screenshot

1.	In Solution Explorer, right-click •	IDEM News view	Add Yiew (with presenter)		? ×
	the Views folder of the				
	Notifications project, point to		Add View with P	resenter	
	Smart Client Software		Add View with Presenter	View name:	
	Factory, and then click Add			Alerts	- Solution Preview
	View (with presenter)			Show documentation after recipe completes	Source
2.	In the wizard launched, enter				E TAlerts.cs
	Alerts in the View Name field				Alerts.cs
	and select the Show				
	documentation after recipe				
	completes option to see a				
	summary of the recipe actions				
	and suggested next steps after			< <u>P</u> revious	Mext > Einish Cancel
	the recipe completes. If				
	Create a folder for the view is				
	selected, the recipe will create				
	a folder and place the new				
	items in this folder.				
3.	Click Finish .				

Customizing Alerts view

- 1. Open the EventTopicNames.cs file located in the Constants folder of Infrastructure.Interface project.
- 2. Paste the following code in the body of the EventTopicNames class:

C#

public const string NewStockBuy = "NewStockBuy";

This event topic name will be used in the Notifications and Stocks modules to notify when a new stock is bought.

- 3. In the Views folder of the Notifications project, open the IAlerts.cs file.
- 4. Paste the following code inside the interface definition:

C#

void ShowAlerts(string p);

This method will be called from the presenter whenever a new alert has to be displayed to the user.

- 5. In the **Views** folder of the **Notifications** project, open the **AlertsPresenter.cs** file.
- 6. Add the following using statements at the top of the file:

C#

```
using Microsoft.Practices.CompositeUI.EventBroker;
using DemoWorkshop.Notifications.Constants;
```

7. Paste the following code in the body of AlertsPresenter class.

C#

[EventSubscription(EventTopicNames.NewStockBuy, ThreadOption.UserInterface)]

```
public void OnNewStockBuy(object sender, EventArgs<string> eventArgs)
{
    View.ShowAlerts("Alert for " + eventArgs.Data);
}
```

This method is an event handler for the **NewStockBuy** event. The **[EventSubscription]** attribute allows you subscribe to an event in a loosely coupled way. In next tasks, you will publish the **NewStockBuy** event.

- 8. Double-click the Alerts.cs file in the Views folder of the Notifications project. This will open the Designer.
- 9. Drag two Labels to the view's surface and put them at the top-left corner of the view (one below the other).
- 10. Set the **Text** property of the first label to *Alerts* and set also the **Bold** property of the **Font** to *true*.
- 11. Change the Name of the second label to AlertsLabel and erase the text in its Text property.
- 12. Right-click onto the view surface and click on View Code.
- 13. Add the following using statements at the top of the file:

C#

using SCSFContrib.CompositeUI.WinForms.SmartPartInfos;

14. Replace the head of the Alerts class with the following:

C#

```
public partial class Alerts : UserControl, IAlerts, ISmartPartInfoProvider
```

In this way, the Alerts class implements ISmartPartInfoProvider.

15. Implement the IAlerts and ISmartPartInfoProvider interfaces. To do this, paste the following code in the Alerts class body.

C#

```
#region IAlerts Members
public void ShowAlerts(string p)
{
    AlertsLabel.Text += p + Environment.NewLine;
}
#endregion
#region ISmartPartInfoProvider Members
public ISmartPartInfo GetSmartPartInfo(Type smartPartInfoType)
{
    ISmartPartInfo spi = (ISmartPartInfo)Activator.CreateInstance(smartPartInfoType);
    spi.Title = "Alerts";
    if (spi is DockPanelSmartPartInfo)
    {
        ((DockPanelSmartPartInfo)spi).DockingType = DockingType.TaskView;
    }
}
```

```
}
return spi;
}
```

#endregion

Showing News and Alerts views in the DockPanelWorkspace

- 1. Open the ModuleController.cs file located in the root of the Notifications project.
- 2. Add the following using statements at the top of the file:

C# using DemoWorkshop.Notifications.Constants; using System.Diagnostics; using Microsoft.Practices.CompositeUI.SmartParts;

3. Replace the ExtendMenu method in the ModuleController class with the following one:

C#

```
private void ExtendMenu()
{
    ToolStripMenuItem menuItem = new ToolStripMenuItem();
    menuItem.Text = "Dump WorkItem";
    WorkItem.UIExtensionSites[UIExtensionSiteNames.MainMenu].Add<ToolStripMenuItem>(menuItem);
    WorkItem.Commands["DumpWorkItem"].AddInvoker(menuItem, "Click");
    ToolStripMenuItem showNewsMenuItem = new ToolStripMenuItem();
    showNewsMenuItem.Text = "Show News";
    WorkItem.UIExtensionSites[UIExtensionSiteNames.MainMenu].Add<ToolStripMenuItem>(showNewsMenuItem);
    WorkItem.UIExtensionSites[UIExtensionSiteNames.MainMenu].Add<ToolStripMenuItem>(showNewsMenuItem);
    WorkItem.UIExtensionSites[UIExtensionSiteNames.MainMenu].Add<ToolStripMenuItem>(showNewsMenuItem);
    WorkItem.UIExtensionSites[UIExtensionSiteNames.MainMenu].Add<ToolStripMenuItem>(showNewsMenuItem);
    WorkItem.Commands["ShowNews"].AddInvoker(showNewsMenuItem, "Click");
}
```

In the previous code, you've added two button as invokers of the commands "DumpWokItem" and "ShowNews".

4. Paste the following three methods inside the body of ModuleController class:

C#

```
[CommandHandler("DumpWorkItem")]
public void DumpWorkItem(object sender, EventArgs e)
{
    Debug.WriteLine("SmartParts Count : " + WorkItem.SmartParts.Count);
}
[CommandHandler("ShowNews")]
public void ShowNews(object sender, EventArgs e)
{
    ShowViewInWorkspace<News>(WorkspaceNames.RightWorkspace);
}
```

```
private void DisposeView(object smartpart, WorkItem workItem)
{
    if (smartpart is IDisposable) ((IDisposable)smartpart).Dispose();
        workItem.SmartParts.Remove(smartpart);
}
```

When the "*DumpWorkItem*" Command is raised, the **DumpWorkItem** method will be executed. The same occurs for the "ShowNews" command and the **ShowNews** method.

5. Replace the AddViews method in the ModuleController class with the following one:

C#

}

```
private void AddViews()
{
    ShowViewInWorkspace<News>(WorkspaceNames.RightWorkspace);
    ShowViewInWorkspace<Alerts>(WorkspaceNames.RightWorkspace);
    WorkItem.Workspaces[WorkspaceNames.RightWorkspace].SmartPartClosing += new
EventHandler<Microsoft.Practices.CompositeUI.SmartParts.WorkspaceCancelEventArgs>(delegate(obje
ct workspace, WorkspaceCancelEventArgs e)
    {
        DisposeView(e.SmartPart, WorkItem);
    });
```

Add Stocks module

Action	Script	Screenshot
 In Solution Explorer, right-click the Source solution folder, point to Smart Client Software Factory, and then click Add Business Module (C#). The Add New Project dialog box appears with the Add Business Module (C#) template selected. Enter Stocks as the Name and set the Location to the Source folder of the solution. 	IDEM Notifications module.	Add New Project ?x Broisect types: Ienplates: .MET Franework 3.5 If
3. Click OK.		

4. The guidance package displays the **Add Business Module** wizard.

IDEM Notifications module.

٠

- 5. Unselect the **option Create an interface library for this module**. If you select this option, the recipe will create an additional project to contain the elements that provide the public interface to the assembly.
- Unselect the option Create a unit test project for this module. If you select this option, the recipe will create a test project for the module with test classes for your module components.
- Select the option Show documentation after recipe completes to see a summary of the recipe actions and suggested next steps after the recipe completes.
- 8. Click Finish.

Add Business Module ? × Add Business Module Module namespace: DemoWorkshop.Stocks olution Pre burce Options Stocks Constants Services Views Module.cs ModuleController.cs Create an interface library for this module Create a unit test project for this module Show documentation after recipe completes Einish Cancel Solution Explorer - Infrastruc... 👻 🗜 🗙 Solution 'DemoApp' (6 projects) 🖻 🔛 Source 🕀 📄 Infrastructure 🗄 🔯 Notifications 🖻 - 🔯 Stocks 🕀 🚾 Properties 😟 🔤 References 🗄 📄 Constants - 📄 Services Views Module.cs ModuleController.cs

 9. Right-click the Stocks project and point to Add Reference In the Browse tab, go to the Lib folder of your application (C:\Projects\DemoApp\Lib) and select SCSFContrib.CompositeUI.Wi nForms.dll. 10. Click OK. 	 Add a reference to the SCSFContrib.CompositeUI.Win Forms.dll assembly. This allows you to use the DockPanelSmartPartInfo and the OutlookBarSmartPartInfo and change some features of your views. 	Solution Explorer - Solution 'DemoApp' (6 projects) Solution 'DemoApp' (6 projects) References Stocks Stocks Microsoft.Practices. CompositeUI Solution 'DemoApp' (6 projects) System.Data System.Data System.Web.Services System.Xml Constants Services Wews Module.cs Module.cs ModuleController.cs 		
11.	Right click onto the Stocks project and point to Add -> New Item In the Add New Item dialog box, select the Resources File template and change the Name of the file to Resources.resx, and then drag it to the Properties folder of the Stocks project.	 Add a resources file where you can place the view icons showed by the OutlookBarWorkspace. 	Add New Item - Stock Image: Stock Image: Stoc	

13.	Double click onto the Resourses.resx file to open it.	 Add two icons that should be representative of each view. 	P Resource.ress ☐ Itons + ① Add Becorce - × Regione Resource □ +	• x
14.	Select Icons in the first dropdown lists.	rou can use the following icons:	V Roontidi and Stods	
15.	Click in the Add Existing File in the second dropdown list.	BuyStock view		
16.	In the Add existing file to resources dialog box, navigate	Reports view		
	to the folder where you have			
	the icons, one for each view,			
	and select them. Click in			
	Open.			
17.	Rename the resources added			
	previously with the names			
	ReportEdit and Stocks			

Add BuyStock view to Stocks module

Using Add View (with presenter)... recipe

respectively.

Action		Script	Screenshot		
1.	In Solution Explorer, right-click the Views folder of the Stocks project, point to Smart Client Software Factory, and then	IDEM News view	Add Yiew (with presenter).)?_X	
2.	click Add View (with presenter) In the wizard launched, enter		properties	BuyStock Create a folder for the view Show documentation after recipe completes Show Source Sourc	
	BuyStock in the View Name field and select the Show documentation after recipe			Bay SoudPresenter	
	completes option to see a summary of the recipe actions and suggested next steps after			<pre>Carcel</pre>	
	the recipe completes. If Create a folder for the view is selected, the recipe will create				
3.	a folder and place the new items in this folder. Click Finish.				

Customizing the BuyStock view

- 1. In the Views folder of the Stocks project, open the IBuyStock.cs file.
- 2. Paste the declaration of the **ShowMessage** method inside the interface body:

<pre>void ShowMessage(string p);</pre>	C#
	<pre>void ShowMessage(string p);</pre>

This method will be called from the presenter when a message has to be shown to the user.

- 3. In the Views folder of the Stocks project, double-click on the BuyStockPresenter.cs file.
- 4. Add the following using statements at the top of the file:

C#

```
using Microsoft.Practices.CompositeUI.EventBroker;
using DemoWorkshop.Stocks.Constants;
using DemoWorkshop.Infrastructure.Interface.Services;
```

5. Paste the following code inside the body of **BuyStockPresenter** class.

C#

```
[EventPublication(EventTopicNames.NewStockBuy, PublicationScope.Global)]
public event EventHandler<EventArgs<string>> NewStockBuy;
```

```
private ILoggingService _logger;
[ServiceDependency]
public ILoggingService Logger
{
   get { return _logger; }
   set { _logger = value; }
}
```

The following code publishes an event using the **[EventPublication]** attribute of the EventBroker system. It also injects the logging service using the **[ServiceDependency]** attribute thanks to the dependency injection pattern implemented by **ObjectBuilder** and **CAB**.

6. Paste the following methods in the **BuyStockPresenter** class.

```
C#
public void BuyStock(string stock)
{
    OnNewStockBuy(new EventArgs<string>(stock));
    Logger.Log("A new stock was bought " + stock + " - ");
    View.ShowMessage("The stock was succesfully bought");
}
protected virtual void OnNewStockBuy(EventArgs<string> eventArgs)
{
    if (NewStockBuy != null)
    {
        NewStockBuy(this, eventArgs);
    }
}
```

The **BuyStock** method is called by the view every time the user decides to buy. This method raises the **NewStockBuy** event, log the transaction using the logging service and show a message to the user in a **MessageBox**.

7. Double-click the BuyStocks.cs file in the Views folder of the Stock project. This will open the Designer.

- 8. Change the Size of the user control to 265, 40 from the Properties view.
- 9. From left to right, drag a Label, a ComboBox and a Button to the view surface.
- 10. Set the **Text** property of the label to *Select Stock*.
- 11. Set the **Anchor** property of combo box to *Top, Left, Right* and add to its **Items** collection the strings *MSFT, UFIDA* and *etc* (one per line) as you can see in the following image:

Takan bisa shuisas in bisa sallastina (an	e nu line).
ncer the strings in the collection (on MSFT UFIDA etc	e per iine):
۲	▼. ▶

- 12. Set the **Text** and **Anchor** properties of the button to *Buy* and *Top*, *Right*. Double-click on the button surface to auto-generate the handler for **Click** event.
- 13. Paste the following code inside the body of the auto-generated method in the previous step:

C#

_presenter.BuyStock(comboBox1.SelectedItem as string);

14. Add the following using statements at the top of the **BuyStock.cs** file:

C#

using SCSFContrib.CompositeUI.WinForms.Workspaces;

15. Replace the head of the **BuyStock** class with the following:

C#

public partial class BuyStock : UserControl, IBuyStock, ISmartPartInfoProvider

In this way, the **BuyStock** class implements **ISmartPartInfoProvider**.

14. Implement the **IBuyStock** and **ISmartPartInfoProvider** interfaces. To do this, paste the following code in the body of the **BuyStock** class:

```
C#
#region IBuyStock Members
public void ShowMessage(string p)
{
   MessageBox.Show(p);
}
#endregion
#region ISmartPartInfoProvider Members
public ISmartPartInfo GetSmartPartInfo(Type smartPartInfoType)
{
   ISmartPartInfo spi = (ISmartPartInfo)Activator.CreateInstance(smartPartInfoType);
   spi.Title = "Stocks";
   if (spi is OutlookBarSmartPartInfo)
   {
          ((OutlookBarSmartPartInfo)spi).Icon = Properties.Resources.Stocks.ToBitmap();
   }
   return spi;
}
#endregion
```

Add Reports view to Stocks module

Using Add View (with presenter)... recipe

Action		Script	Screenshot
Act 1. 2.	ion In Solution Explorer, right-click the Views folder of the Stocks project, point to Smart Client Software Factory, and then click Add View (with presenter) In the wizard launched, enter Reports in the View Name field and select the Show documentation after recipe completes option to see a summary of the recipe actions	Script • IDEM News view	Screenshot
	and suggested next steps after the recipe completes. If Create a folder for the view is selected, the recipe will create a folder and place the new		< Previous Blext > Enish Cancel

items in this folder.

3. Click Finish.

Customizing Reports view

- 1. Right-click onto the **Reports.cs** file and click on **View Code**.
- 2. Add the following using statements at the top of the file:

C#

using SCSFContrib.CompositeUI.WinForms.Workspaces;

3. Replace the head of the **Reports** class with the following:

C#

```
public partial class Reports : UserControl, IReports, ISmartPartInfoProvider
```

In this way, the Reports class implements ISmartPartInfoProvider.

4. Implement the ISmartPartInfoProvider interface. To do this, paste the following methods in the Reports class.

C#

```
#region ISmartPartInfoProvider Members
```

```
public ISmartPartInfo GetSmartPartInfo(Type smartPartInfoType)
{
    ISmartPartInfo spi = (ISmartPartInfo)Activator.CreateInstance(smartPartInfoType);
    spi.Title = "Reports";
    if (spi is OutlookBarSmartPartInfo)
    {
        ((OutlookBarSmartPartInfo)spi).Icon = Properties.Resources.ReportEdit.ToBitmap();
    }
    return spi;
}
```

#endregion

Showing BuyStock and Reports views in the OutlookBarWorkspace

- 1. Open the ModuleController.cs file located in the root of the Stocks project.
- 2. Add the following using statements at the top of the file:

C#

using DemoWorkshop.Stocks.Constants;

3. Replace the AddViews method in the ModuleController class with the following one:

C#

```
private void AddViews()
{
```

ShowViewInWorkspace<BuyStock>(WorkspaceNames.LeftWorkspace);

ShowViewInWorkspace<Reports>(WorkspaceNames.LeftWorkspace);

}

The previous method shows the Stocks module's views in the **OutlookBarWorkspace** (the left one, in the **Shell**).

Compile, run and show the application

Act	ion	Scri	pt	Screenshot
1. 2.	Set the Shell project as StartUp Project . Compile and Run the Application (F5).	•	Run the application.	Debug Tools Test Window Community Help Windows • * Start Debugging F5 • Start Striberts • Soliton: Depreser-Source • Start Striberts • Start Striberts • Start Striberts • Startsubstrate Interface • Startsubstrate Interface • Startsubstrate Interface • Debeter All Breakpoint • P • Debeter All Breakpoints' Coll+Suft+F9 • Startsubstrate
3.	Show the application.	•	The application consists of two Business Modules. Business modules are distinct deployment units of a Composite UI Application Block application that contain business logic elements. SCSF allows loading modules specified in a Profile Catalog file. In this file, you can add different roles for each module. You can see the "Dump Workitem" and "Show News" buttons in the Main Menu Strip. These items are added when CAB loads the Notification module. You can also see two workspaces. An OutlookBarWorkspace on left side and a DockPanelWorkspace on right side. Each workspace on right side. Each workspace on right side. Each workspace shows views in different ways. Workspaces are components that encapsulate a particular visual layout of controls and SmartParts. The Notification module loads its two views in the right workspace and the Stocks module in the left one.	Self semi fine fung Worklaw Work Have Fine fung Worklaw Work Have Fine fine fung (some test, some test; Work Have Charge SmartPartine Charge Work Have Fine facts Work Have Fine facts Work Have

4. Show the right workspace and	٠	Smartparts are data views such	
its SmartParts.		as a control, a Windows Form,	Shell Form File Dump Workfrem Show News
		or a wizard page. In the right	
		workspace you can see two	Reports Alerts 4 X Alerts
		SmartParts: News on the left	Some text, some text, some text Some text, some text, some text
		and Alerts on the right. The	
		DockPanelWorkspace can	Change SmartPartinfo Close View Programatically
		show SmartParts in two	
		different Docking Types :	
		• TaskView . like the	
		Alerts view	
		• Document , like the	
		News view	Siccis
	•	A SmartBartInfo is a niece of	
	•	information about a SmartDart	Ready
		that a workspace can use such	
		as the title of the Smort Port of	
		as the title of the Sindripart. If	
		SmartDartinfe" button the title	
		SmartPartinio button, the title	
		of the view is changed. That is	
		because when you press that	
		button, the presenter of the	
		view tells the	
		DockPanelWorkspace to apply	
		a new SmartPartInfo.	
	•	Click in the "Close View	
		Programatically" button in the	
		News smartpart. See how the	
		SmartPart is closed by its	
		presenter.	
5. Show the left workspace and	٠	In the left workspace you can	
its SmartParts.		see an OutlookBarWorkspace.	Shell Form File Dump Workfrem Show News
		This workspace allows you	
		switch the views by clicking in	Reports Today News X Alerts Alerts
		the button bellow.	Some text, some text, some text Some text, some text, some text
	٠	Also you can click in the little	
		arrow bellow and select the	Change SmartPartInfo Close View Programatically
		"Show More Buttons" or	
		"Show Fewer Buttons" options	
		if you have to many buttons	
		and you want to hide them	
		and you want to mue them.	
			Show More Buttons Show Farmer Buttons
			Ready

6.	Show the BuyStock view.	•	Now you can see the Reports view but if I click in the Stocks button you can see the BuyStocks view In the Buy Stock view select one option in the combo box (for example MSFT) and then click in " Buy " button. You can see a Message Box and the text " <i>Alert for MSFT</i> " in the Alerts view. This is achieved by EventBroker . This system allows you publish and subscribe to events in a loosely coupling way. The BuyStock and Alert views are in different modules and they doesn't have reference each other.	Stell Form Image: Stell Form File Dump WorkRem Show News Stocks Image: Stock Stock MSFI - Buy Some text, some te
7. 8. 9.	Go to the "C:\temp" directory. Open the "log.txt" file. Show the application log.	•	Every time that the Buy button of the BuyStock view is clicked, the Logging Service is called, which logs the operation in a log file. A Service is a supporting class that provides functionality to other components in a loosely coupled way. Services are singletons that can be injected using the Dependency Injection pattern and live in the Service collection of WorkItem . A WorkItem is a run-time container of the components and services that are collaborating to fulfill a use case.	Iog bt - Notepad File Edit Format View Help A new stock was bought UFIDA -

10. Show the "Show News " button in the Main Menu Strip.	 If the "Show News" button in the Main Menu Strip is clicked, a new News view appears in the right Workspace. This is achieved by Commands. You can use Command to bind an UIElement event to more than one command handler and a single command handler to multiple UIElements in a loosely coupling way. 	*** Shell form File Dump Worklern Show News Stocks *** Alerti ** Some tox some tox, some tox *** Alerti or MSFT Some tox, some tox, some tox, some tox Some tox, some tox, some tox Alerti or MSFT Change SmartPartInfo Close View Programatically If for MSFT *** Stacks * * **** ************************************
 Maximize Visual Studio. Restore the DemoApp. Make sure that the Output view of Visual Studio can be seen. 	 When the other button is clicked (the "Dump WorkItem" button), you can see in the Output view of Visual Studio the text "SmartParts Count: 3". This represents the count of Smartparts that the module's WorkItem has (views in the left workspace). This also executes a Command that can be used to debug our application. 	Socks Some text, som

Summary

Now you have minimum knowledge about the main features of SCSF. You can deepen your knowledge by reading the documentation, by doing the Hand-On-Labs and by reviewing the Quickstars and the Reference Implementation.

Useful Links

- Hand-On-Labs
 - o <u>http://www.codeplex.com/smartclient/Release/ProjectReleases.aspx?ReleaseId=6357</u>
- SC SF Knowledge Base
 - <u>http://www.codeplex.com/smartclient/Wiki/View.aspx?title=SCSF%20Knowledge%20Base&referringTitle=H</u> <u>ome</u>

- SCSF Community Site
 - o <u>http://www.codeplex.com/smartclient</u>
- SCSF Contrib
 - <u>http://www.codeplex.com/scsfcontrib</u>