

What is This Cabana Thing?

The “Cabana Reference Application” takes twenty-six projects divided over two solutions and almost 1,000 files. You’ve got to be kidding, right?

Yeah, yeah, I know ... and I’m not kidding.

But before you close it up and walk away, give me a chance to explain.

Most Projects are “Below the Waterline”

CabanaLib.sln, the “Cabana External Infrastructure” solution, holds eleven source code projects for the **external infrastructure** that IdeaBlade and third parties support. The architect in you will want to investigate them – when you are ready; “Joe application developer” (perhaps you wearing a different hat) may never go there.

The **Cabana.xxx.sln**, the “Cabana Application” solution, corresponds to what your application will be. Only twelve of its projects concern the application proper. The other projects were developed and managed by someone else..

Most of the files in these twelve projects were generated by Visual Studio (e.g., project files, assembly.info, resx, and designer). The Object Mapper emits a bunch (datarow, .orm, EntityRelations). There are fifty-plus images in the Resources directory.

I counted **under 120 files** that might have been touched by human hands. They are all small – less than a screen’s worth of code on average. Many of them repeat patterns that will become familiar to the Cabana-style developer.

Theme and variations

This application strives to illustrate a variety of development options. You’ll want some right away; some you’ll want later; and some you will never need. You certainly don’t have to learn it all at once.

There is (or will be shortly) a companion solution, the “Cabana Baseline”, that strips it “bare” so you can start with a relatively clean slate. However, you’ll still want to refer to the full Cabana to see the techniques *in vivo*.

Consequences of Loose Coupling

Cabana strives for the de-coupled architecture that folks look for in a CAB application. “De-coupling” always translates to “lots of projects, lots of files.” That’s the price. Is it worth it?

It is not worth it if you’re just jamming out a single form, throw away “application”. I’ll assume you’re here because you have a different intent. You want a robust, loosely-coupled architecture ... you just don’t want to be lost in it.

Finding Your Way

Landmarks and conventions are vital to finding your way. Given a decent plan, consistent application of the conventions, and a clear statement of what goes where and why, the initially bewildering landscape settles down into something that “makes sense”.

We all know what it’s like to land in an unfamiliar city. One of my first steps it to buy a map. If the city is arranged according to an old medieval plan – a warren of twisting curving streets – it’s going to be an adventure. If the city is laid out as a grid, I can quickly navigate a thousand streets.

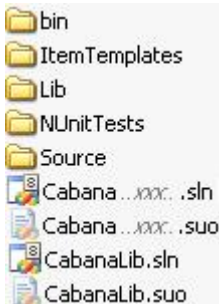
This document is that map. Cabana started as a medieval village. It’s moving toward something more like mid-town Manhattan although there are a few “old quarters” that are more annoying than quaint. Our crack “urban renewal” team has targeted a number of such slums.

Meanwhile ... here’s the map.

Directory Structure

Let's look at the highest level of the Cabana directory before we dive into the Visual Studio Cabana solution.

After unzipping the distribution zip file and building it for the first time, we get something like the following:



Top Folders	Description
bin	The CAB application convention arranges for application binaries to arrive here rather than in the individual project files.
ItemTemplates	<p>We recommend Visual Studio templates as the means for quickly generating new class and project files.</p> <p>VS templates are lighter weight than Software Factories. They are far less capable than software factories but they're easier to code, debug, and deploy.</p>
Lib	<p>CAB applications rely upon a considerable number of CAB, "Enterprise Library", and SCSF assemblies.</p> <p>While these are all available in source, we rarely need to see that source and we don't want to pay the price of recompiling them each time we build the solution. Accordingly, we store these assemblies here and refer to them from our project files.</p> <p>You may decide that certain of the projects included in Cabana should also be referred to in their compiled form rather than re-built each time; we recommend you follow the pattern and store such assemblies here as well.</p> <p>There are some projects which are built directly into Lib even though their source files are in the project (to make it easy to view and breakpoint them). <i>See CabanaLib.sln.</i></p>
NUnitTests	<p>Unit tests are "never" shipped with the application and belong in this separate code tree. By default they are not compiled.</p> <p>N.B.: They require NUnit v.2.4.1.0. The build will fail if you try to compile them and do not have NUnit installed. You can download it from http://www.nunit.org/.</p>
Source	The source code for the application
Cabanasln	The Visual Studio Cabana Application solution files (the .suo is the "user" file) that particular to the Cabana application.
CabanaLib.sln	<p>The Visual Studio Cabana External Infrastructure solution files that <i>manage external infrastructure (IdeaBlade and ThirdParty) projects. These build directly into the Lib</i> directory.</p> <p>The other Cabana projects have references to these (and other) dlls in Lib. They do not have references to the projects in CabanaLib.sln! This will make it easier to work with Cabana (your application ultimately) independently of developments in the external infrastructure projects.</p>

Cabana Structure

The Lib Directory

The “Lib” is a “library” of assemblies referred to by Cabana-style applications.

Before you build, as shipped, it comes with assemblies for which the source code is **not** provided.

Assembly prefix	Description
Microsoft.Practices.CompositeUI	The two CAB UI assemblies.
Microsoft.Practices.EnterpriseLibrary	The Enterprise Library v.3 assemblies. Only a few of these are referenced by Cabana and these few are referenced only by the Smart Client Software Factory (SCSF) projects; they could be pruned back if desired.
Microsoft.Practices.ObjectBuilder	The CAB ObjectBuilder, also used by Enterprise Library
Microsoft.Practices.SmartClient	More Enterprise Libraries, dedicated to Smart Client apps
SampleVisualizations	The example “Visualizer” shipped with CAB with bug fixes.

The CabanaLib solution holds infrastructure source code that (in principle) you should not modify. It includes a combination of material from MS Patterns and Practices (e.g., parts of the Smart Client Software Factory), contributions from third parties, and infrastructure we have found useful in building CAB applications with DevForce.

The CabanaLib solution adds all of its assemblies to the “Lib” directory and the Cabana application projects have references to them here in the “Lib” directory.

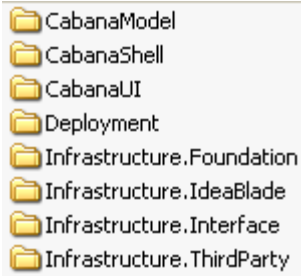
Remember you must build CabanaLib before building the Cabana application (Cabana_{xxx}.sln) or the application will not be able to find these library assemblies.

After you build the CabanaLib solution, you will see:

Assembly prefix	Description
DevEx.CompositeUI.Extensions	Only if you build the Developer Express version of Cabana will you see this assembly dedicated to CAB wrappers of DevEx controls.
DotNet.CompositeUI.Extensions	Third party contributions to building CAB apps with DotNet controls.
IdeaBlade.Cab	The CAB extensions provided by IdeaBlade
IdeaBlade.Common.EntityModel	Extensions to IdeaBlade DevForce business object classes
Scsf.Infrastructure	SCSF assemblies for the classes generated by the SCSF application start recipe (2006 version) that were not subsequently modified. This recipe delivers a mix of classes one shouldn't touch and classes you <i>must</i> modify. We moved the “must modify” classes into the application projects, Infrastructure.Foundation and Infrastructure.Interface, leaving the unmodified classes in the two assemblies prefixed “Scsf.Infrastructure”.

The Source Directory

Let's drill in to the Source directory.



Source Folders	Description
CabanaModel	<p>The business object model as it emerges from the Object Mapper, supplemented by additional model support classes.</p> <p>You would replace this with your application Model folder. In some scenarios, developers prefer to keep the Cabana reference classes around for awhile as a convenient reference point; they remove these “training wheels” later when they are ready to work completely independently.</p> <p>These developers simply add a new folder to hold a second Model.</p>
CabanaShell	Holds the start-up project.
CabanaUI	<p>Holds the Business Modules of the application UI.</p> <p>The application developer works here.</p> <p>The Admin, SalesRep, and SaleOrder projects live here.</p> <p>Business Modules correspond to the “units of application work” as the end user experiences them. They rely upon the Foundation Modules located in the Infrastructure directory.</p>
Deployment	<p>Projects that support deployment of the application in a variety of scenarios.</p> <p>Cabana can be deployed as both a 2-tier and n-tier application. Most of the tiny projects here concern n-tier deployment.</p>
Infrastructure.Foundation	<p>The application’s Foundation Modules.</p> <p>Foundation Modules and the other classes in this directory provide application-specific infrastructure. The Business Modules (and other application code) refer directly to this infrastructure and typically derive from classes and interfaces here.</p>
Infrastructure.Interface	<p>Public interfaces for the application’s Foundation.</p> <p>This starts virtually empty except for application constants. Experience suggests that you will be adding Foundation modules and you will need a separate interface project the moment you do.</p>
External Infrastructure	
Infrastructure.IdeaBlade	IdeaBlade.Cab projects with infrastructure to bridge your application infrastructure (Infrastructure.Foundation) to CAB assemblies and your business object model.

Cabana Structure

Source Folders	Description
Infrastructure.ThirdParty	<p>A CAB application typically relies upon “third party” libraries. These don’t change often (if at all). To spare constant recompiling, we have pre-compiled them and put <i>the dlls in the Lib directory</i>. The debug build excludes these projects.</p> <p>The CabanaLib.sln solution does this trick should you need to do it again for some reason (it does the same for IdeaBlade.Cab.EntityManager.Inspector as well).</p>

It’s time to turn to the Cabana Visual Studio Solution.

Overall Solution Structures

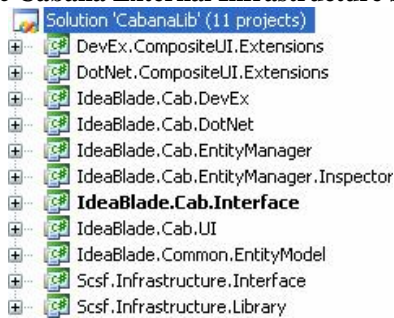
The **Cabana Application Solution** structure mirrors the directory structure almost exactly (as it should):



Here are the parallels

VS Solution Folders / Projects	Source Directories
Deployment	Deployment
Infrastructure	Infrastructure.Foundation Infrastructure.Interface
Model <i>project</i>	CabanaModel
UI	CabanaUI
CabanaShell <i>project</i>	CabanaShell

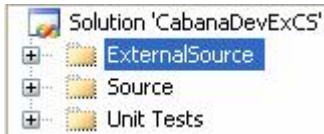
The **Cabana External Infrastructure Solution** is a flat structure:



Cabana Structure

The ExternalSource Visual Studio Solution Folder

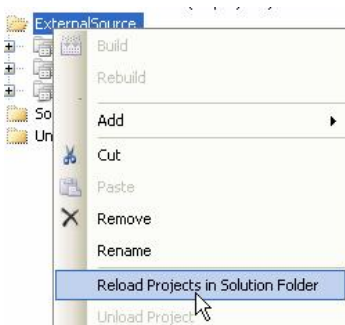
The solution from your distribution of the Cabana source code may include an “ExternalSource” Visual Studio solution folder as in this illustration:



It contains a number of *unloaded* projects from the CabanaLib solution – the IdeaBlade projects that help CAB and DevForce work well together.



They are inaccessible in this form but you can make them visible to the solution by right-clicking on the solution folder and selecting “Reload ...” from the context menu.



This step includes the projects into the solution.

These projects are excluded from the solution build intentionally.

Reloading projects confuses third-party tools that try to keep track of your solution such as ReSharper; you can clear up the confusion by closing and re-opening the solution.

The Purpose of ExternalSource

Developers should be able to focus exclusively on the elements of the application that they should change.

While tracing or debugging our application, we inevitably run into code that we did not write. We want to be able to read this code and enable breakpoints. That is very hard to do when the source code is not part of the solution.

On the other hand, including that source code is an invitation to change it. Changing that code could be disastrous for the application and certainly interferes with our ability to integrate updates from the source code providers (e.g., IdeaBlade).

Cabana Structure

This folder is both a blessing and a curse. I find it essential to have these projects available because I'm always interested in what's happening just below the application waterline. But then I'm easily confused about which projects belong to the open application solution and which projects really, really do not (that is, these projects). I've changed this library code accidentally many times while thinking I was updating it through the CabanaLib solution.

This should be less of a temptation for you because you would rarely consider changing the library code. I strongly recommend the following.

- Consider dropping the ExternalSource directory from your solution.
- Definitely drop it for developers who do not need to delve into the architecture – or should not.
- Unload the directory when you don't need it.
- Never build these projects here; if you feel you must modify the IdeaBlade.Cab source (e.g. to explore what would happen if we fixed a flaw), do so only through the CabanaLib solution.
- Keep the original distribution zip file handy so you can restore these projects if you make a mistake.

Comparing Cabana to SCSF-generated Solution

Anyone familiar with the Smart Client Software Factory (SCSF) will notice that this solution structure is quite different than the one generated by the 2006 application generation recipe.

That recipe mingled “invariant” SCSF files with files you must alter to make your application work. This was a terrible idea because lose track of what is “theirs” and what is “yours.” You would have an unnecessarily difficult time updating your application with a new release of SCSF (should you decide to do so).

The structure wasn't clear anyway.

Cabana Solution Structure Rationale

We have rationalized the solution structure in a manner intended to support the way we imagine you will work.

The most important separation is between “Application Developer Modules” and “Application Architect Modules”. This distinction aligns with a similar division of labor within many shops.

Even if you are a lone developer, you will find yourself alternating between “Developer Mode” and “Architect Mode.” I do this myself all of the time and have even established two personas for myself, imagining this to be a kind of constructive schizophrenia.

Developer Mode and the UI Folder

In “Developer Mode” I want to write end-user facing application code as quickly as possible, following guidelines and recipes set down for me by the “application architect.”

It's not that I can't think or am too unreliable and stupid to proceed creatively. It's that I shouldn't have to wonder what I'm doing when I'm in this part of the code base. Why? Because this is where the majority of the application code will live. This is where changing business requirements will strike first.

Which means that the burden of voluminous and routine maintenance should land here. Accordingly, I want this material to be as simple and straightforward as possible. If I have to get fancy once, ok. If I get fancy twice, it's time to put on my architect hat and figure out how to abstract the complexity “up” into the Infrastructure folders.

Cabana Structure

Architect Mode and the Infrastructure Folders

In “Architect Mode” I think about the commonalities that run through the application.

VS Folders	Purpose
Infrastructure	The application’s own infrastructure. The infrastructure specific to Cabana today and your application tomorrow.
External Infrastructure	
Infrastructure.IdeaBlade	Infrastructure provided by IdeaBlade. This will change “regularly” as we evolve our support for building Cabana application and respond to your suggestions.
Infrastructure.ThirdParty	Infrastructure provided someone other than you or IdeaBlade. We assume this changes infrequently if at all.

The objective is to separate the work you do from the work anyone else does so you can take advantage of their improvements without fear of a serious collision.

Architect Mode and the Other Folders / Projects

In “Architect Mode” I think about the commonalities that run through the application.

VS Folders / Projects	Purpose
CabanaShell	<p>This is the start-up project. It holds ShellApplication – the equivalent of Program.cs – which hardly changes.</p> <p>ShellApplication handles everything from the application launch to the loading of the Foundation Module and other modules – a process that includes the user authentication. This doesn’t change much but you’ll have to get it right.</p> <p>The project owns the all-important ProfileCatalog, the XML file that tells CAB which modules to load. You modify it as you add (and remove) modules and module dependencies.</p> <p>There are a lot of options relating to this catalog – see the CAB documentation. The Cabana version is in the “stock format” from SCSF.</p> <p>Finally, it is worth calling out the AppConfig which contains CAB directives, menu items, and assembly redirects to handle versioning problems with 3rd party UI control suites.</p>
Deployment	<p>Venture here only in “Architect Mode”. Deployment is not hard but it’s not cookbook either. You shouldn’t need to visit this arena often.</p>
Model	<p>The business object model is critical component of your application. After an initial flurry, most models settle down.</p> <p>We recommend that you manage this model in “Architect Mode”. If a “developer” needs something that isn’t there, you probably want an “architect” to vet the suggestion and handle the implementation.</p> <p>Some tasks are simple and uncontroversial such as adding Verifications. Some shops split the “final” class (which is a partial class) into two files. Thus the Employee class might be split between “Employee.cs” and “Employee.dev.cs”.</p> <p>The “architect” retains strict control over “Employee” while the developer is free to modify “Employee.dev”.</p> <p>A simple script or Visual Studio template can do the work of generating “Employee.dev”.</p>

UI Control Suite Support

DevForce offers direct support for several UI control suites including those in DotNet, Developer Express, and Infragistics.

Cabana supports the DotNet and DevEx suites at the moment. We will add Infragistics support soon and hope to have an example of WPF controls in the next few months.

We took on multi-vendor support as a design challenge. We wanted Cabana to be as UI control suite agnostic as possible. In theory, the decoupled, controller-driven, Model-View-Presenter pattern should make this possible (if not exactly easy).

We've worked the MVP pattern pretty hard and to pretty good success. Almost all presenters can be use either for views built with either control suite. We use a **ViewFactoryService** to deliver the suite-specific views. With just a little tweaking and throwing a few compiler switches, Cabana can swing from one suite to another.

Evidence of this intent is clearest in the **IdeaBlade.Cab.DevEx** and **IdeaBlade.Cab.DotNet** projects which isolate the material particular to the vendor suites.

You probably don't care about this particular exercise. Once you've picked a suite, you expect to stay with it.

Perhaps you are right (although some day WPF is going to seem awfully tempting and you'll want to switch to it with a minimum of fuss and bother).

You may discover that the ViewFactoryService has other potential uses. For example, you might want to craft "customer-specific" views that change depending upon which user logged in. You could try to build metadata-driven customization into the view themselves – and I think I'd try that first. But some day the machinery for automated self-customization may seem too baroque and fragile. It's a life-safer to be able to craft an alternate view and simply load it at runtime without changing any other logic. Perhaps a combination of self-customizing and factory approaches will prove most effective.

Project Directory

The following two tables enumerate the Cabana projects in each of the two solutions.

Each table is sorted by project name within Visual Studio folder.

Cabana Application Solution

Folder	Projects	Description
	CabanaShell	<p>This is the start-up project. It holds ShellApplication – the equivalent of Program.cs – which hardly changes.</p> <p>ShellApplication handles everything from the application launch to the loading of the Foundation Module and other modules – a process that includes the user authentication. This doesn't change much but you'll have to get it right.</p> <p>The project owns the all-important ProfileCatalog, the XML file that tells CAB which modules to load. You modify it as you add (and remove) modules and module dependencies.</p> <p>There are a lot of options relating to this catalog – see the CAB documentation. The Cabana version is in the “stock format” from SCSF.</p>
Deployment	AppHelper	Application deployment configuration.
Deployment	ClickOnceShim	<p>The Visual Studio ClickOnce manifest builder will only include assemblies that are referenced. CAB loose coupling means that many assemblies (e.g., all Business Modules) are not referenced by any other assembly and thus are excluded from the manifest.</p> <p>There is a workaround tool – Mage – but it is painful to use where the VS ClickOnce tool is quick and easy. Most of us don't want to waste time with dev builds that are “architecturally pure”; we don't have time for that.</p> <p>This shim holds references to all of the assemblies that would otherwise be missed. It's temporary expedient; a production release should exclude the shim and build the ClickOnce manifest with Mage.</p>
Deployment	SetupIIS	<p>The setup project for an IIS deployment. This is used by IdeaBlade to deploy Cabana on its commercial website.</p> <p>[The version in this distribution may not be correct because updating the website Cabana is not a priority. Check with us before relying on it.]</p>
Infrastructure	Infrastructure.Foundation	<p>The application's Foundation Modules.</p> <p>Foundation Modules and the other classes in this directory provide application-specific infrastructure. The Business Modules (and other application code) refer directly to this infrastructure and typically derive from classes and interfaces here.</p>
Infrastructure	Infrastructure.Interface	Public interfaces for the application's Foundation.
Model	Cabana.Model	The business object model for Cabana includes Employee, Customer, Order, etc.

Cabana Structure

Folder	Projects	Description
UI	Cabana.UI.Admin	<p>The “Administration” module for managing user information. The focus is on users and their permissions. The project illustrates a technique for using proxy objects to represent “missing” associations. In this case, there are proxies representing permissions that users do not have.</p> <p>The proxy classes are in a subfolder called “UIModel” because they serve as the “Model” in the MVP triangle for many of the views in this module.</p>
UI	Cabana.UI.SalesOrder	The “Sales Order” module for tracking suppliers, customers and customer orders.
UI	Cabana.UI.SalesRep	The “Sales Representative” management module.
UI	Cabana.UI.SalesRepNav	<p>A gratuitous demonstration of CAB’s ability to break features down into separate, loosely coupled modules.</p> <p>This project provides the “OutlookBar” navigation to the “SalesRep” page. In principle, one could swap out the SalesRep navigation upon launch, either substituting another one or navigating to the page by some other means.</p> <p>All other UI (Business Modules) combine the “OutlookBar” navigation with the management “Page” itself.</p>
UI	Cabana.UI.SharedEditor	<p>Home to the “pop-up Editors” for viewing and modifying a single instance of an entity in a non-modal window.</p> <p>Such an editor might have been defined within the Business Module of a page that could launch it.</p> <p>The OrderEditor is the lone inhabitant of this module. It can be launched from a SalesRep module page or a SalesOrder module page.</p> <p>To demonstrate CAB’s flexibility, we put the OrderEditor in a separate module so that one of the Sales modules could launch it even if the other Sales module was excluded at launch.</p>
UI	Cabana.UI.Splash	<p>Displays either of two “splash” screens in the page content area.</p> <p>This is a simple (silly) example of a module whose views are too trivial to require either a model or PageController.</p>

We did not include the Unit Test modules in this list.

Cabana External Infrastructure Solution

Folder	Projects	Description
Infrastructure. IdeaBlade	IdeaBlade.Cab. EntityManager	<p>A CAB-aware, UI-oriented wrapper of a PersistenceManager that mediates between the UI and the DevForce persistence layer.</p> <p>An application likely maintains several EntityManager’s in-flight at any one time; they are accessible through an EntityManagerService whose base class is defined here.</p>
Infrastructure. IdeaBlade	IdeaBlade.Cab. EntityManager.Inspector	A CAB “Visualizer” that enables inspection of all in-flight EntityManagers as the application is running. This should only run in debug mode (a policy ensured by the ShellApplication class).

Cabana Structure

Folder	Projects	Description
Infrastructure. IdeaBlade	IdeaBlade.Cab.DevEx	Functionality targeting the Developer Express UI control suite.
Infrastructure. IdeaBlade	IdeaBlade.Cab.DotNet	Functionality targeting the DotNet UI control suite.
Infrastructure. IdeaBlade	IdeaBlade.Cab.Interface	Interfaces for members of the IdeaBlade CAB infrastructure.
Infrastructure. IdeaBlade	IdeaBlade.Cab.UI	IdeaBlade-developed infrastructure for CAB application development.
Infrastructure. IdeaBlade	IdeaBlade.Common. EntityModel	IdeaBlade-provided infrastructure for business object models such as support for Verification and auditing. There is no CAB or UI awareness in this project so that it can be deployed to a host machine in a middle tier along with the application model (UI should never be deployed to a middle tier).
Infrastructure. ThirdParty	DevEx.CompositeUI. Extensions	CAB UI extensions to support the Developer Express UI control suite. These completely general facilities were developed by an independent 3 rd party and are community supported.
Infrastructure. ThirdParty	DotNet.CompositeUI. Extensions	CAB UI extensions for DotNet UI controls. The OutlookBarWorkspace for DotNet, written by Matias Woloski, is the featured attraction.
Infrastructure. ThirdParty	Scsf.Infrastructure. Interface	Interfaces for members of the SCSF infrastructure.
Infrastructure. ThirdParty	Scsf.Infrastructure. Library	Smart Client Software Factory (SCSF) infrastructure.