



Introduction

Welcome to the Funhouse Source Code!

These notes are divided into two sections:

- [Installation Notes](#)
- [Release Information](#)

Funhouse requires DevForce

You must have installed a DevForce edition such as DevForce Express, available on our website for *free* at www.ideablade.com/downloads.html.

Installation of the Funhouse is nearly painless but there are a few places that can give you momentary trouble. For example, you may need to tweak the connection string to find your copy of the IdeaBlade Tutorial database.

Please read the Installation Notes carefully !

What Funhouse Is and Is Not

Funhouse is supposed to be ... well, fun. It's intended to show you some of the things you can do as a DevForce enterprise application developer.

Funhouse is not intended to be a sophisticated application

Funhouse demonstrates intermediate and advanced techniques in the context of an application. We want the application itself to be as simple as possible without obscuring the key points. Our customers build large, sophisticated applications that draw on these techniques.

Funhouse is not the proper model for a small, simple CRUD application.

Funhouse is far more complicated than is necessary or appropriate for a beginning application. Funhouse techniques make sense only in a more sophisticated application.

For example, there is no point to separating view and controller so rigorously in an application with a single working page. There are far too many projects in this solution for such a simple application. The [DataSourceKeyResolver](#) is an advanced trick that most applications never need. The list goes on.

Funhouse source code is not for beginning DevForce application developers

Funhouse is a challenging place to start if you have just installed DevForce. Please try the tutorials first as they establish a foundation for the ideas and suggestions here. Please feel free to explore the Funhouse for interesting details and for clues about solutions to advanced issues – just keep in mind that it is a tutorial, not a real application.

Install the Funhouse Code (for DF v.3.3.0.0)

You are almost ready to get started after you've downloaded the Funhouse code and unzipped it.

You will have to **rebuild** the entire solution. You may want to run the Object Mapper and regenerate entities in both business object projects, `Model` and `Server` before you rebuild.

You should confirm that the connection string(s) shipped with Funhouse are correct for the IdeaBladeTutorial database in your environment.

There are a few likely “gotchas”:

1. DevForce version incompatibility
2. Can't find IdeaBladeTutorial database
3. Missing `[DataSourceKeyInfo]` table
4. Can't login
5. Cannot deploy as an n-tier application (missing `ServerConsole.exe`)
6. Test projects don't compile or run; missing NUnit.

DevForce Version Compatibility

Your version of DevForce may not be the same as the version we used to build Funhouse (currently 3.3.0.0).

If so, some lines may not compile, especially in `Model.Common`; typically there are alternative lines next to the line that fails and you need only comment out the offending line and uncomment the working alternative. You will probably have to fire up the Object Mapper and regenerate the `DataRow` classes as well.

Can't Find IdeaBladeTutorial Database

Funhouse relies upon a running copy of the IdeaBladeTutorial database.

There are connection string specifications in several places, all of them reading as follows:

```
Provider=SQLOLEDB.1;Integrated Security=SSPI;  
Persist Security Info=False;Initial Catalog=IdeaBladeTutorial;  
Data Source=localhost
```

You won't be able to connect (and login) if any aspect of this string is wrong, such as:

- You're not using MS SQL Server (2000 or 2005)
- You have a named instance of MS SQL Server (rather than a default instance)
- You can't use the SQL OLEDB provider
- You can't use integrated security (Windows Authentication)
- The database isn't named “IdeaBladeTutorial”
- You can't refer to the database server as “localhost” (e.g., named pipes not enabled for your server).

Launch the Object Mapper while examining the `Model` project to discover if this is a problem for you. If the Object Mapper can't find the database, you'll have to find the connection string that does. The DevForce Installation Guide and Developer Guide offer some useful discovery tips.

Once you have a working connection string, you'll have to put that string in the following places:

- The Object Mapping file for the Model project (via the Object Mapper)
- The Object Mapping file for the Server project (via the Object Mapper)

Funhouse Source Code Notes

- The two RdbKeys in the AppHelper.IdeaBlade.ibconfig; remember to rebuild - not just build – this project after you make these changes.
- The connection strings in the [DataSourceKeyInfo] table in IdeaBladeTutorial; see the next topic if you don't have such a table.

When you decide to explore the n-tier deployment options, you'll also have to correct the strings in:

- The "security" RdbKey in the Server.IIS. IdeaBlade.ibconfig (but not in the "default" key).
- The "security" RdbKey in the Server.WinSvc. IdeaBlade.ibconfig (but not in the "default" key).

Missing Table

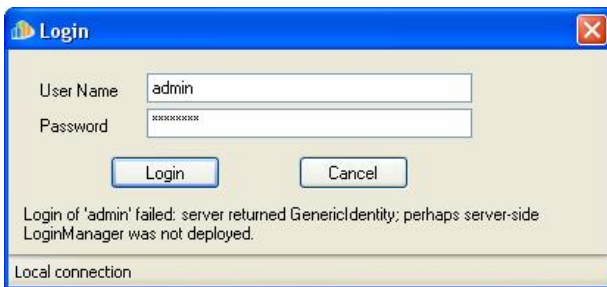
The Funhouse uses the IdeaBladeTutorial database in which it expects to find a table called [DataSourceKeyInfo]. This table is in the currently shipped database but may not be in your version of the database if your database pre-dates DevForce 3.1.4.

We've included scripts to create and populate it: "DataSourceKeyInfo_Create.sql" for SQL Server 2005 and "DataSourceKeyInfo_Create.sql2000.sql" for SQL Server 2000. Please play them before running the Funhouse.

Can't Login

You've left the username as "admin" and the password as "password". You press the [Login] button. You get the error message:

"Server returned GenericIdentity; perhaps server-side LoginManager was not deployed".



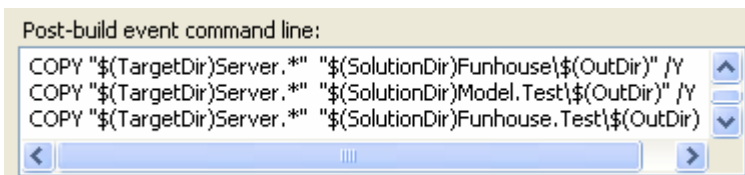
The most likely cause is that the Funhouse "Server" assembly, `Server.dll`, was not deployed to the directory that holds the Funhouse executing assembly, `Funhouse.exe`.

This would be in the directory `...\Funhouse\bin\Debug` if you're using the default `Debug` build configuration.

Building (or re-building) the server project is supposed to copy the `dll` into that (and the test directories) for you.

⇒ Look of the "Properties" page for the Server project

⇒ Turn to the "Build Events" sub-tab. It should show Post-build events that look like this:



Perhaps these commands should be different on your machine.

Before you decide to edit the Post-build events, confirm that this is really the cause:

⇒ Go to the directory for the `Server` project's `dll` (typically `...\Server\bin\Debug`)

Funhouse Source Code Notes

- ⇒ Copy both `Server.dll` and `Server.pdb` to the clipboard
- ⇒ Go to the directory that houses the `Funhouse.exe` executable (typically `...\Funhouse\bin\Debug`)
- ⇒ Scroll through looking for `Server.dll`.

If you don't find it, that's the cause. You can now

- ⇒ Paste `Server.dll` and `Server.pdb` from the clipboard
- ⇒ Try running again.

Cannot deploy as an n-tier application

The Funhouse ships with Setup projects in support of an n-tier deployment. These presuppose that you are licensed to use the **Business Object Server** (BOS).

The BOS is not included in the **Express** edition of DevForce. Therefore, while you can see how one might deploy as an n-tier application, you cannot actually do so unless you purchase the **Enterprise** version of our product.

You get the **n-tier experience** when you run the application as deployed via ClickOnce. In that case, we at IdeaBlade have deployed the middle tier on our web site and you are interacting with it as a client.

Feel free to remove these Setup projects from your solution and build again. The Funhouse is shipped for deployment as a 2-tier application and will work just fine for **Express** customers.

Deployment as an n-tier application is not covered in these Release Notes. It will be covered in one of the on-line videos.

Test Projects don't build or run; missing NUnit

The test projects (all ending in ".Test") rely upon NUnit, the popular and free unit testing tool.

This installation package sets the solution build configuration to exclude these projects as of Funhouse 3.1.5. Many of you will not have NUnit installed and these projects did not build successfully for want of the NUnit libraries.

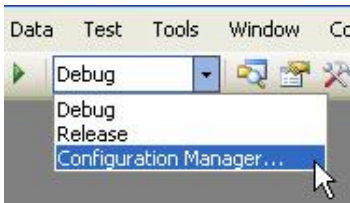
This means you can't test properly until you add NUnit and include the test projects back into the build – a practice we strongly recommend.

You should use unit testing in your application and the Funhouse offers clues as to how to proceed. Here's how to re-engage unit testing with the Funhouse.

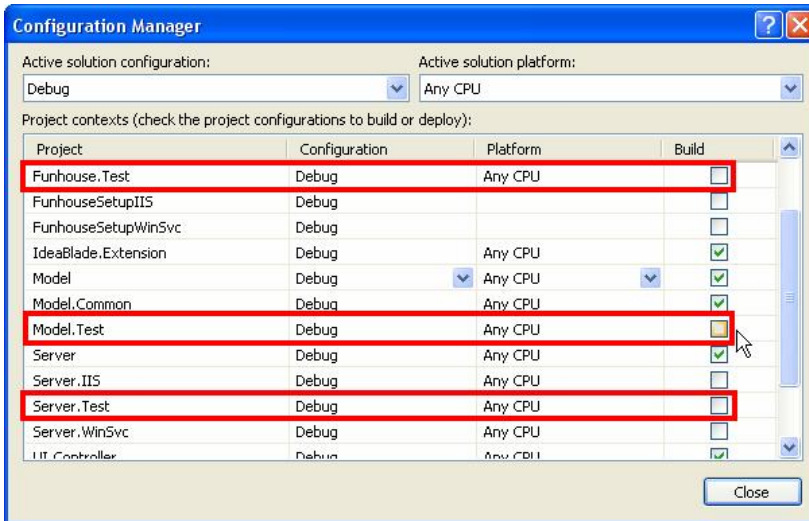
- ⇒ Shut down Visual Studio.
- ⇒ [Download NUnit v.2.2.8 for .NET 2.0](#) and install it.
- ⇒ Launch Visual Studio.

Funhouse Source Code Notes

⇒ Open the “Configuration Manager” from the tool strip.



⇒ Find the test projects and check their checkboxes so that they are included in future builds.



⇒ Rebuild the solution.

Release Information

Version 3.3.0.0

This version is compatible with DevForce v. 3.3.0.x. There were no other changes.

Version 3.2.1.1

Very little change this release – but one important one. DevForce version 3.2.1.1 introduced the `EntityInfo` class which clashed with the class of the same name in `Model.Common`. References to `Model.Common.EntityInfo` have been made namespace-specific in the few places that was necessary.

Version 3.2.0.0

There are minor changes but one of them – `Csla.ReflectionFns` replaced – could break existing code. This is easily fixed; see the relevant note.

- The `Csla.ReflectionFns` class in `Csla.validation` has been removed. We were only using a single method of it, `Csla.ReflectionFns.GetPropertyValue`, and that was a method we wrote that was never in CSLA.

The method name remains but the class is replaced by a new class added to the `IdeaBlade.Extension` project, `IdeaBlade.Util.ExtendedReflectionFns`. We are hopeful that the methods of this class may some day become part of the official `IdeaBlade.Util.ReflectionFns`.

The change affects three class of the 3.2.0.0 version all of which have been updated for this release.

1. `Csla.validation.CommonRules`
2. `Csla.validation.PropertyRequiredRule`
3. `Model.Employee`

Version 3.2.0.0

This Funhouse release is incompatible with DevForce releases prior to DF 3.2.0.

There are a few minor changes, mostly to accommodate migration from DF 3.1.5 to 3.2.0:

- `OrderControllerHelper`, `ValueColorProperty`, and `RowStateImageProperty` each depend upon `AdaptedPropertyDescriptor` and `PropertyDescriptorFns.BuildPropertyDescriptor()` whose signatures were changed to repair a defect (#299); see the DevForce Release Notes.
- The DevForce Object Mapper can now find project and `DataRow` class files in Solution Folders; the Funhouse projects are re-arranged in Solution Folders to better present the Funhouse organizing principles.
- The business object entities in the `Model` and `Server` projects were re-generated with the latest Object Mapper. Their `DataRow` classes are defined using the new “Property Interceptors” in preparation for a forthcoming demonstration of data-driven, property-level authorization.

Version 3.1.5.0c

This version contains tiny changes, mostly those needed to make automated conversion to VB easier.

The Employee Page appears in a tab, indicating a “tab” approach to adding modules. Not a serious foray into a shell.

Funhouse Source Code Notes

`EmployeePageController` sets a new employee's photo with the "DefaultEmployee" photo of "Razzle, the Wonder Dog" at 4 months old. This illustrates one way to update a business object photo. The implementation relies upon the image manipulation functions in the new `ImageFns` class in VUtility project.

`EmployeeFilter.ApplyFilter` has a paged query version illustrating techniques for retrieving subsets of query results by "paging". The paged query invocation is commented out.

The "simplest possible Dynamic Property" has been added to `OrderControllerHelper`. We expose it as a Freight column on the Employee order grid (see `AddFreightColumn`). The `Order.Freight` property actually exists but we pretend that it doesn't and that the data for the mapped column is accessible only through a "user defined field".

Version 3.1.5.0b

`EntityTypeInfo.PrimaryKeyDataColumns` and `EntityTypeInfo.GetPrimaryKeyDataColumns` are gone (except for the overload of `EntityTypeInfo.GetPrimaryKeyDataColumns` that takes an `EntityTable` argument).

There is an important, if subtle, difference between `EntityColumn` and `DataColumn` – the former is general for the class; the latter is specific to an `EntityTable` of a particular `PersistenceManager`. In general we only need the `EntityColumn`. It is best if `EntityTypeInfo` operates without reference to any particular `EntityTable` instance. Therefore, we expunged these `DataColumn` –oriented methods.

The change affected the implementation of `EntityAdapter` but not its API.

The core of `ListConverters` is refactored into `EntityListConverter<T>` to abstract out the duplicated code.

Version 3.1.5.0a

A customer suggested that the essence of the Funhouse `BaseEntity` class be abstracted into `Model.Common` because it would generalize far beyond the example in Funhouse application.

We've taken this to heart and we're pleased with the results so far.

The two files in the `Model` project, `BaseEntity` and `BaseEntity.Validation`, are now combined in `CommonEntity` file which defines a `CommonEntity` class in `Model.Common`.

There remains a bare-bones `BaseEntity` in `Model`. It serves as the root, abstract class for all `Model` entity classes and it's a great place to put your application-specific business object behavior ... without colliding with the more general members of `CommonEntity`.

This Funhouse version is more of a beta than 3.1.5.0 because we haven't had much time to lean on it. We may have to make more of the `CommonEntity` methods virtual (overrideable), for example. Feedback is welcome.

Using *CommonEntity* in your application

When you adapt Funhouse to your purpose and start building your own business model, **please note this essential step**. You must have a base entity in your model and it must inherit from `Model.Common.CommonEntity`. The Object Mapper can't jump that gap for you but it is easy to do yourself.

When the Object Mapper first generates your `BaseEntity` class, it inherits from `BaseEntityDataRow` like so:

```
public abstract class BaseEntity : BaseEntityDataRow {...  
Public MustInherit Class BaseEntity: Inherits BaseEntityDataRow
```

You will change that so it inherits from `Model.Common.CommonEntity` like so:

```
public abstract class BaseEntity : Model.Common.CommonEntity {...
```

Funhouse Source Code Notes

```
Public MustInherit Class BaseEntity: Inherits Model.Common.CommonEntity
```

You only do this once per business model; the Object Mapper won't change this file again.

Version 3.1.5.0

There are many changes, many of them breaking – especially the move to [EntityAdapter](#).

We would never do this in a mature framework because backward compatibility is crucial. We can let it slide at this stage because (a) Funhouse is a tutorial and (b) not many of you are basing your code on Funhouse components. We will be more cautious once Funhouse components gain traction.

Here are the key points.

- [EntityHelper](#) has been renamed [EntityAdapter](#) which more accurately expresses its purpose and aligns it, philosophically, with similar .NET “adapters” such as the [TableAdapters](#) generated for strongly typed data sets.
- [AuditHelper](#) is now [AuditColumnManager](#) and there is a new class, [AuditColumnManagerCollection](#) for wiring update of audit columns when the entity changes.

The management of audit columns ([CreatedTs](#), [CreatedById](#), [ModifiedTs](#), [ModifiedById](#)) has been separated from [EntityAdapter](#) which no longer knows anything about auditing columns. Updates audit columns whenever the entity changes (see [MainPm](#)) and upon save (see [BaseEntity.Validation](#)).

The complexity would be needless if all editable entities were mapped to tables with identical audit columns. You would put the logic into the [BaseEntity](#) class and be done with it (the approach taken in our tutorials). This ideal world is rarely ours (not even in [IdeaBladeTutorial](#)). [AuditColumnManager](#) is a general approach to a world of heterogeneous auditing.

- The [EntityHelperDef.snippet](#) is now the [EntityAdapterDef.snippet](#). It resides in [Model.Common](#) and is included as Funhouse “solution item.” Please install this in your snippet library (found in C:\Documents and Settings\[YourName]\My Documents\Visual Studio 2005\Code Snippets\ under the Visual C# or Visual Basic subdirectories) and discard the previous version.
- The test projects are excluded from the Debug build configuration by default. You may include them once you have installed NUnit (see pertinent installation note).
- The code snippet for dropping [EntityAdapter](#) boilerplate into a final class is now part of the Funhouse solution and resides in [Model.Common](#).
- Added “Most Recently Used” (MRU) to Search Textbox; the code is in [EmployeePageController](#).
- Added one level of employee Undo – for all employees or just the current employee. Demonstrates use of the [SplitButton](#) which is new in .NET 2.0. The code is in [EmployeePageController](#).
- AppControllers and AppViews now have a Close() method which calls Dispose(). These classes now implement IDisposable by way of. IAppController which now inherits from IDisposable. See AppControllerBase and AppViewBase for implementations.

This became necessary because, under many conditions, .NET did not properly dispose of the [BindingNavigator](#) (the tool strip) when the [MainForm](#) closed. Something like this was bound to happen anyway, given the separation of controller and view. Controllers maintain references into views and should close them. The [MainForm](#) had direct references to the views and was closing/disposing them before their controllers had a shot at them. Now the page controllers (see [EmployeePageController](#)) can respond to the MainForm's [Closing](#) event and take charge through their own [Close](#) methods.

N.B.: [Close](#) and [Dispose](#) mean the same thing. You should only call Close. I could have hidden Dispose behind an explicit interface implementation but decided to follow .NET's lead and leave them both exposed.

Funhouse Source Code Notes

- The client-side `DataSourceKeyResolver` in `AppHelper` no longer looks for a `ServerDataSourceKeyResolver` in 2-tier deployments. The `ServerDataSourceKeyResolver` only made sense in an n-tier deployment and, while it was convenient to test in 2-tier, it seemed to get in the way for folks just starting out with the Funhouse.
- Broken Rules validation has been substantially cleaned up.

Most important is the addition of support for rules that involve multiple properties; see `MultiPropertyRuleMethod` and an example of it in `Employee.AddBornBeforeHiredRule`.

Formerly (and in the original CSLA), all rules were independent. But rules that involve comparisons among properties are mutually dependent. The enables rule `MultiPropertyRuleMethod` interdependence.

To see the problem, observe the (former) misbehavior in the following scenario involving a “Before date must be earlier than After date” rule:

- Set the Before date later than the After date. We see the `BrokenRule` for the Before date but not for the After date even though this property is broken too.
 - There is only one `BrokenRule` in `BrokenRulesCollection`; there should be two.
 - Run `CheckRules()`; it processes all rules so now we see both `BrokenRules`.
 - The `BrokenRulesCollection.ToString()` displays both rules which have identical messages; we only want to show one of them.
 - Set the After date beyond the Before date; the rule should pass but the `BrokenRule` for Before date is not cleared.
 - Still have the Before date `BrokenRule` in `BrokenRulesCollection`; there should be none.
 - `CheckRules()` clears the decks.
- `PropertyRequiredRule.AddToList()` is now just `Add()`.
 - There is now a `Csla.Validation.Test` project and the test class, `CslaValidationTests`, was moved there from `Model.Test`.
 - The `BaseEntity.Validate()` method no longer performs “Broken Rules” validation when the entity is scheduled for deletion. We usually don’t care if the data are invalid for an entity we’re going to delete anyway. See the XML comments for that method.
 - The `BaseEntity.Validate()` method has been refactored for readability and to make sure that entities are actually allowed to be saved from their current `RowStates`, regardless of their ability to pass Broken Rules validation. Thus, for example, we can block deletion of an entity that should not be deleted (per the object) but somehow evaded the UI checks.
 - Added `IPermitPersistence` to help here and in future releases where knowledge of an instance’s persistence permissions (whether it can be inserted, updated, or deleted) is needed for processing.
 - `EntityInstanceDisplayName` and `EntityTypeDisplayName` are now `EntityInstanceName` and `EntityTypeName`. `ToString()` override returns result of `EntityInstanceName`.
 - Save processing logic used to be distributed among several, unexpected classes. Such logic has been refactored into the new `SavingControllerBase` class in the `UI.Controller` project. This affords room for future development both of the base class and derived classes that need special behavior. Look for the addition of Concurrency Conflict Resolution in upcoming Funhouse releases.

We began tracking release information as of version 3.1.5.0