

Book of FarCry

FarCry Developer Jump Start

Try the FarCry 6.x Jump Start Course - its absolutely the best place to start learning how to tailor your own FarCry solution.



The developer course is always under review. If something seems a little quirky, or you have a great idea that should be included please let modius-AT-daemon.com.au know.

Table of Contents

- [UNIT 01 - Introducing the Course](#) — FarCry Core is a web application framework based on the ColdFusion language. FarCry CMS is a popular content management solution built with FarCry Core. As a community, we provide support for the framework, CMS, and a host of community plugins including libraries for Google Maps, free text searching, and many more.
- [UNIT 02 - Installation](#) — After completing this unit you should be familiar with the specific system requirements and the various options for a typical FarCry installation. Completion of this unit is necessary to set up your local development environment for the rest of the course.
- [UNIT 03 - FarCry Overview](#) — FarCry provides a variety of out-of-the-box services including a comprehensive array of content management options. However, FarCry is in fact a web application framework that can be used to build sophisticated solutions that go way beyond a typical content management platform. One of the first steps to understanding what's possible is getting to grips with what installs by default.
- [UNIT 04 - Webskins I](#) — In this unit we learn about webskins: the "view" for FarCry Framework. Webskins form the presentation tier of any FarCry application. By the end of this unit you should have a basic understanding of how webskins are managed, their relationship to content types and how to create your own.
- [UNIT 05 - Content Types](#) — This unit covers the fundamental building blocks of any FarCry application: the Content Type. By the end of this unit you should be able to create and deploy your own content types.
- [UNIT 06 - Content Relationships](#) — By the end of this unit you will have learnt how to relate content types to one another, using one-to-many and many-to-many relationships. You will be able to create user interfaces to allow editors to select and relate objects from libraries.
- [UNIT 07 - Webskins II](#) — This unit is a workshop to discuss how to hook up different content types in the presentation tier - linking from one view to the next by the content's relationships.
- [UNIT 08 - ORM, Object Broker, View Caching](#) — By the end of this unit you will be able to apply caching to various aspects of your applications to dramatically increase performance.
- [UNIT 09 - The Webtop](#) — By the end of this unit you will be able to modify the webtop tabs, sub-sections and menus.
- [UNIT 10 - Building Forms](#) — After completing this unit you will be able to build your own FarCry forms to edit and save content objects.
- [UNIT 11 - Plugins, Skeletons & Extension](#) — By the end of this unit, we should have an understanding of how FarCry navigates our project and plugins to find both content type metadata and relevant webskin locations.
- [UNIT XX - Containers & Publishing Rules](#) — Adding containers to templates and creating publishing rules to populate them.

Download Courseware

There are currently no attachments on this page.



Instructor Lead Training

Daemon provides face-to-face and remote, online training for all FarCry courseware. Support the FarCry community: please consider investing in instructor lead training, mentoring and development support for your team.

Contact: training@daemon.com.au

URL: <http://www.daemon.com.au/>

Copyright (c) 2008-2011 Daemon Pty Limited

UNIT 01 - Introducing the Course

Overview

FarCry Core is a web application framework based on the ColdFusion language. FarCry CMS is a popular content management solution built with FarCry Core. As a community, we provide support for the framework, CMS, and a host of community plugins including libraries for Google Maps, free text searching, and many more.

FarCry 6.x Jump Start is designed to bring developers up to speed with the FarCry Core Web Application Framework. Although FarCry is a huge topic, this course distills key components of the development environment in a bid to enable students to start building their own solutions immediately.

Prerequisites

In order to get the most from the course students should be:

- familiar with basic web development concepts such as HTML, CSS and JavaScript
- comfortable with the ColdFusion mark-up language

Course Format

Concepts

The course is divided into units, each of which presents new information and contains demonstrations, walkthroughs, and a lab. At the end of each unit, you will find a summary and a short review to test your knowledge of the unit's content.

The following concepts are used throughout the course:

- Demonstrations illustrate new concepts
- Walkthroughs guide you, with the instructor's assistance, through procedures in a hands-on context.
- Labs let you practice new skills on your own.
- Summaries provide a brief synopsis of the unit's content.
- Reviews test how well you remember the concepts from the unit.

Author Highlights



Additional Information or Cross Reference

A cross reference or additional information about something that is not specifically course related. For example, we might refer to Blueprint CSS Framework in the courseware and need to define what it is and provide a link to the site.



Something Important

A call out to something important in the course material.



Alternative Approaches

Additional information about a topic being covered. For example, this might be a popular alternative approach or an advanced option.



Warning. Be Careful!

An alert highlighting an activity that might cause a problem for the student.

Great Learning Resources



FarCry Cheat Sheet

Get the FarCry developer cheat sheet, print it out and set it beside your keyboard. It will be a great help in getting to grips with FarCry Core: <http://farcry.posterous.com/farcry-developer-cheat-sheet-v01>

In addition to this course there are a variety of great online learning resources that you should explore:

- [FarCry Developer Forums](#); google group forum for the FarCry developer community
- [FarCry Developer WIKI](#); the home of this course, references, tutorials, sample code and the FarCry Contributor Guide
- [FarCry Core Auto-Docs](#); automatically generated docs from the FarCry code base (great for tag, formtool and FAPI options)
- [FarCry Committers Blog](#); tips and tricks from the FarCry Core development team

**FarCry Custom Google Index**

We have a custom search index for all the sites, blogs, and docs from the FarCry community. Use this search to focus your results: <http://www.farcrycore.org/search?q=help>

UNIT 02 - Installation

Objectives

After completing this unit you should be familiar with the specific system requirements and the various options for a typical FarCry installation. Completion of this unit is necessary to set up your local development environment for the rest of the course.

System Requirements

ColdFusion Application Server

FarCry runs on a variety of different servers that can interpret the ColdFusion mark-up language.

CFML Server	Version	Download
Adobe ColdFusion	8.0+	http://www.adobe.com/products/coldfusion/
Railo	3.3.0+	http://www.railo-technologies.com/



OpenBD support is probably no far off – give the community a shout if you are interested in this platform

Database Platforms

FarCry supports a number of different relational database platforms.

Database	Version	Download
MS SQL	2000+	http://www.microsoft.com/Sqlserver/2005/en/us/express.aspx
MySQL	4.1+	http://dev.mysql.com/downloads/mysql/
H2	Railo Express	http://www.h2database.com/



Contact Daemon for Support Options

Postgres	8.0+	http://www.postgresql.org/download/
Oracle	10g+	http://www.oracle.com/technology/software/products/database/xe/index.html



Other Databases

You may have varying degrees of success with other database platforms or version numbers. Adding additional platform support is certainly possible, but talk to the community first before embarking on such an ambitious project.

Web Server & Operating System

FarCry will run on just about anywhere you can get a supported ColdFusion implementation working. As a consequence there are installations running on various versions of IIS and Apache web servers and on a variety of operating systems including Windows, OSX and Linux.



URL Rewrites

Your web server implementation will require some form of URL re-write engine in order to activate the FarCry Friendly URL sub-system. For example, mod_rewrite on Apache or an ISAPI Rewrite filter for IIS (several open and commercial options are available. <https://farcry.jira.com/wiki/display/FCDEV50/Friendly+URLs>)

Installation Options



Deployment Configurations

Detailed information on deployment options is available on the developer WIKI
<http://farcry.jira.com/wiki/display/FCDEV50/Deployment+Configurations>

Standalone

Default installer configuration option.

Specifically aimed at one application per website. For standalone application deployment and/or shared hosting deployment that allows for a single project with a dedicated core framework and dedicated library of plugins.

Sub-Directory

Recommended for local development only.

For multiple application deployment under a single webroot. Specifically aimed at multiple applications per website.

Advanced Configuration (ColdFusion Mapping)

This installation configuration is not suitable for projects running in a shared hosting environment.

An enterprise configuration that allows for an unlimited number of projects to share a single core framework and library of plugins. Sharing is done through common reference to specific ColdFusion mapping of `/farcry`.

Advanced Configuration (Webserver Mapping)

New for Fortress. Considered an advanced deployment option similar to ColdFusion mapping.

An enterprise configuration that allows for an unlimited number of projects to share a single core framework and library of plugins. Sharing is done through common reference to specific web server mapping (aka web virtual directory) of `/farcry`.



Daemon Commercial Support

Daemon provides commercial priority support offerings for installation, and ongoing operational maintenance.
<http://www.daemon.com.au/>

Course Development Environment

This course and its associated walkthroughs, labs and sample code are based on the following local development environment:

- Railo 3.3 (Express Version)
- H2
- Resin

The courseware is based on a FarCry Express demo installation, which has its own database and web server built-in.



FarCry Express

Download the latest FarCry Express installation or a dedicated training installation provided by your instructor:
<http://www.farcrycore.org/builds/>

Walkthrough: Installing Your Local Development Environment

The Jump Start course is suitable for **any** valid FarCry installation. The walkthrough details the FarCry Express demo installation as this is arguably the easiest to install (ie. you do not need to know how to configure the web server). Database references throughout the course assume the built-in H2 server, however, any supported database would be suitable for the course.



Check Port 8888

When starting the FarCry Express edition, remember that the server runs on <http://localhost:8888> and so you will need to ensure nothing else is running on this port when you start the server.

In this walkthrough you will install and configure your local development environment:

1. Copy the FarCry Express installation onto your local development workstation (<http://www.farcrycore.org/download>)
2. Locate the local Demo Media and Docs folders (they should be located within the FarCry Express distribution)
3. Create a development project in your IDE
 - a. Open the IDE installed on your desktop
 - b. Create a Project from the webroot of your FarCry Express installation; for example,
`INSTALL-ROOT/farcry-express-install/webroot`
 - c. Browse the project and make sure you can see all the relevant FarCry code base directories
4. Your H2 database is already in place – there is no need for a database installation
5. Start your FarCry Express installation (review the README.txt notes for specific instructions for your OS)
6. Browse to <http://localhost:8888/webtop>

**Double Check That Everything is Working**

Please ensure that your development environment is full functional before proceeding. This is critical for the success of the rest of the course. Friendly URLs will not work with the FarCry Express installation. This is a limitation of the Express demo only.

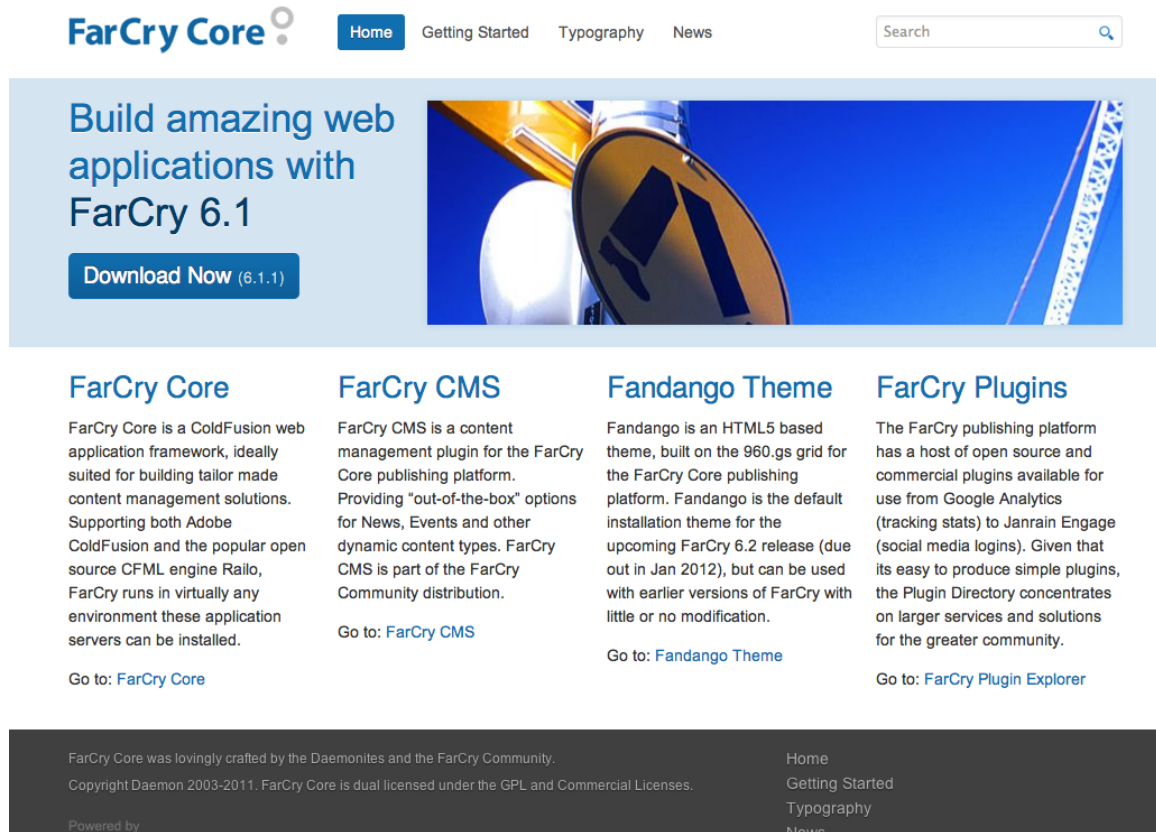
**Other Environments**

The course sample code should work in any operational environment that supports FarCry. However, the specific references in the walkthrough may need to be adjusted to reflect your actual development environment.

UNIT 03 - FarCry Overview

Objectives

FarCry provides a variety of out-of-the-box services including a comprehensive array of content management options. However, FarCry is in fact a web application framework that can be used to build sophisticated solutions that go way beyond a typical content management platform. One of the first steps to understanding what's possible is getting to grips with what installs by default.



The screenshot shows the FarCry Core website homepage. At the top, there is a navigation bar with links for Home, Getting Started, Typography, and News, along with a search bar. The main banner features the text "Build amazing web applications with FarCry 6.1" and a "Download Now (6.1.1)" button, accompanied by an image of a satellite dish. Below the banner, there are four columns of content: FarCry Core, FarCry CMS, Fandango Theme, and FarCry Plugins. Each column provides a brief description of the service and a link to go to the respective page. At the bottom, there is a footer with copyright information and a navigation bar.

FarCry Core
FarCry Core is a ColdFusion web application framework, ideally suited for building tailor made content management solutions. Supporting both Adobe ColdFusion and the popular open source CFML engine Railo, FarCry runs in virtually any environment these application servers can be installed.
Go to: [FarCry Core](#)

FarCry CMS
FarCry CMS is a content management plugin for the FarCry Core publishing platform. Providing "out-of-the-box" options for News, Events and other dynamic content types. FarCry CMS is part of the FarCry Community distribution.
Go to: [FarCry CMS](#)

Fandango Theme
Fandango is an HTML5 based theme, built on the 960.gs grid for the FarCry Core publishing platform. Fandango is the default installation theme for the upcoming FarCry 6.2 release (due out in Jan 2012), but can be used with earlier versions of FarCry with little or no modification.
Go to: [Fandango Theme](#)

FarCry Plugins
The FarCry publishing platform has a host of open source and commercial plugins available for use from Google Analytics (tracking stats) to Janrain Engage (social media logins). Given that its easy to produce simple plugins, the Plugin Directory concentrates on larger services and solutions for the greater community.
Go to: [FarCry Plugin Explorer](#)

FarCry Core was lovingly crafted by the Daemonites and the FarCry Community.
Copyright Daemon 2003-2011. FarCry Core is dual licensed under the GPL and Commercial Licenses.
Powered by

Home
Getting Started
Typography
News

Out-Of-The-Box

FarCry is a framework for building web applications. It comes complete with a huge number of services including data modelling, views, controllers, ORM, object caching, nested tree model, classification engine, plugin architecture and a host of other odd sounding but extremely useful acronyms and jargon.

In addition to the framework, FarCry developers assumed that nearly every application has some basic requirements. As a consequence, the standard install ships with some very helpful features to kick start your project.

Sample Application

The installation leaves you with a basic website based on the Fandango theme. We call this the Fandango Project Skeleton or just plain Fandango for short.

This set of designs, templates and sample content is completely optional and can be replaced with whatever takes your fancy. However, its a great place to start your development especially if you want to concentrate on the code. You can always implement new templates later and reskin your application when needed.



Fandango Theme

Fandango is based on the very popular 960.gs CSS grid framework and is a standards based template design released to open source by Daemon.
<https://farcry.jira.com/wiki/display/FAN/Home>

Webtop Administration

FarCry Core has a built in administration area, called the "webtop". The **webtop** is completely configurable in terms of tabs, menus and options. Different configurations of the webtop can be secured to specific roles through the FarCry security model.

Every application, no matter how small, invariably needs a secured administration area for managing the application. Consequently the webtop is available for all installations and forms an integral part of the FarCry Core Framework.

Information Hierarchy

An information hierarchy or, perhaps more simply put, "a set of menus", is critical for any application. FarCry provides a built in nested tree service for modelling hierarchical data sets. Although this is most commonly visualised in the webtop as the "Site" tab, the tree model can be leveraged anywhere within your application.



The FarCry nested tree is based on ideas for graphing hierarchical data by database guru Joe Celko.
<http://www.celko.com/>

Commodity Content Management

Despite the stigma attached to being seen as "just a CMS", the FarCry community believe that content management is a commodity requirement. Every application needs it - that's why content management options are a core feature of the FarCry Framework. Depending on your application these services can be ignored or utilised as often as required.

Walkthrough: Out-Of-The-Box

Walkthrough the default Fandango project installation and get a handle on what goes where.

Site Overview

1. Open up your web browser and view the sample "Fandango" sample site. Review with your instructor.
2. Login to the webtop, using the credentials you selected on installation.
3. Browse to the "Site" tab and review the Site Overview Tree.
4. OK so now we want to add a bit of content of our own. Browse to the Content Tab, select Content Publishing from the sub-section drop down menu and then choose the Quick Site Builder utility.
5. Build your own web site content from HTML pages, the standard template and approving all content up-front. Use the following as a guide:
 - Select Home from the Site Tree
 - Do not select Navigation Aliases at this time.
 - Copy the following short form syntax or make up your own (In FarCry 6.x this is the edit box labeled Structure)

Sample Quickbuilder Notation

```
Hero Hotline
-Here to Help
-Lurid Spandex
Photos
About
-Contact Us
-History
```

- Select HTML Content (In FarCry 6.x - From Auto Create Children dropdown, select HTML Page)
 - Choose a simple layout template for your content, perhaps a 2 column layout
 - Build your content! (Click Build Site Structure)
6. Now switch back to the Site Overview Tree to see what wonders the quick builder hath wrought.
 7. View the website itself, and note how the menu's should have updated to reflect your newly added content.

Media Library

1. Select the Content Tab, select Content Publishing from the sub-section drop down menu and then open the Bulk Image Upload utility.
2. Select the "DemoMedia" folder from your file system and upload all the images there in one fell swoop.
3. Go to the Media Library of images (by selecting the menu option for Image Library) and review the image library with your instructor. Notice how the images have been auto-generated for both mid and thumbnail sizes.
4. Edit an image, and re-crop one of the thumbnails... awesome, eh!

Rich Text Editing

1. Open the webtop Site Tab and select an HTML page to edit.
2. You will need to create an Editable Draft.
3. Edit HTML Page by running through the wizard with your instructor.
4. On the BODY step, add some images to the content item from the media library you recently imported. Do this by selecting "Open Library" on the related media section and drag/drop some images from the media library.
5. Use the "FarCry Button" on the rich text editor (its the one that looks like a little FarCry logo) to insert your images into the body content directly.
 - Select an image from the drop down – you should see a choice of those images you associated earlier in the related media section.
 - Select a size of image and preview.
 - Insert the image when you are happy with your choice.
6. Save your masterpiece and preview your work.
7. Go back to the webtop overview for your content item and approve the content to go live.

Publishing Rules

Publishing rules are a more advanced concept to develop but are nevertheless easy for non-technical editors to use. Think of them as Widgets (Yahoo) or Gadgets (Google).

1. Go to a page on your website, and look for the "Tray Menu" (bottom of your website when logged in as an administrator). When you find it you should see an option to "Show Rules".
2. In "Show Rules" mode you should notice some special utilities have appeared on the page. These are containers. Containers can be associated with any template - more on this later. Containers are for publishing dynamic content using publishing rules!
3. Click on the container "Add Rule" button.
4. Select Image Gallery from the list of available rules.
5. Select some images from the media library and publish the rule.
6. Hide Rules, using the option in the tray menu.
7. You should have a great little image gallery available on the website - Cool!

Lab: Sophisticated Web Applications

Review with the instructor a cross section of applications built in FarCry that go beyond the typical content management solution.



Bluescope Steel Australia

One of the worlds largest steel manufacturers. Rich cross related content, membership services, geo mapping, and subsites (colorbond.com).

- <http://www.bluescopesteel.com.au/>
- <http://www.colorbond.com/>



Australian Olympic Committee

The home of the Summer and Winter Olympic Teams:

- <http://www.olympics.com.au/>



webDU developer Conference

Like a rock concert for geeks. Video multimedia (plugin), complex agenda, cross relation of content (agenda, session, speaker, sessions in previous years).

- <http://www.webdu.com.au/>

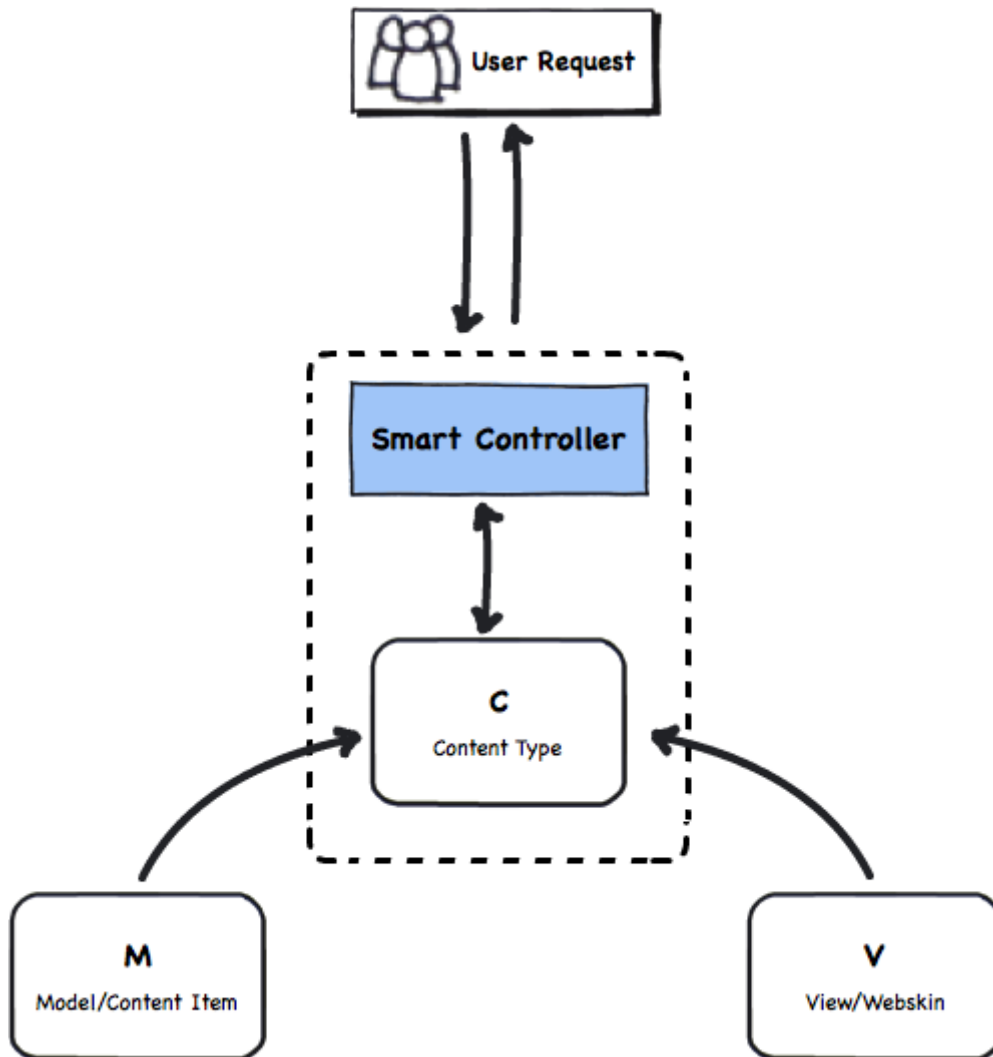
UNIT 04 - Webskins I

Objectives

In this unit we learn about webskins: the "view" for FarCry Framework. Webskins form the presentation tier of any FarCry application. By the end of this unit you should have a basic understanding of how webskins are managed, their relationship to content types and how to create your own.

The COAPI

The FarCry framework is based on a variation of the classic Model-View-Controller pattern. The COAPI (or Content Object API) is a high level name for the MVC engine inside FarCry. Without getting bogged down in the details of the framework, its good to have a high level or "helicopter" view of how things work under the hood. We'll expand on these themes gradually as you go through the course.



Requests from users are processed by FarCry's built-in "smart controller" which automatically determines wiring based on the URL convention or the friendly URL sub-system. The COAPI always ends up calling a principal content type (a special type of class) that manages how we interact with the `model` and `view`. When thinking of `views` (or webskins) its important to realise that they are always executed in the context of a specific content type.

It's not at all important for developers to understand how the COAPI works, but hey, for those that want the details we've added some advanced info boxes where relevant.

The Webskin

The webskin is a templating layer that effectively translates your data into a view. Typically this is a complete HTML page or fragment of HTML representing a content teaser or some other element used in assembling a page. But a webskin could just as easily be outputting a form, RSS, XML or other format.

In its basic form we can run a webskin on an object by using the following URL Syntax:

URL Syntax
<code>http://superheroes.local?objectId=E689D66F-96FD-E9F6-B1AF64B8DAE78A69&view=displayPageStandard</code>

The url above is running the webskin "displayPageStandard" on the content item with the objectId of **E689D66F-96FD-E9F6-B1AF64B8DAE78A69**

Pretty ugly, eh. But not to worry. In practice the Friendly URL (FU) engine dynamically changes these links to beautiful looking URLs. For example, /hero/batman

Where to Find Webskins

Webskins are all located in a single folder under your project root called "webskin" funnily enough. Inside that directory branch you should find a single folder for each content type you want a webskin for.

Webskin Directory Structure
<pre>[install-root] \farcry-express-install \webroot \farcry \projects \myproject \webskin \dmHTML \dmNews \dmEvent \myContentType</pre>



Webskin Inheritance

Webskins can be stored in the core framework, and any plugin as well as your current project. However, any webskin placed in your project with the same name as an existing webskin will always be overridden by your project's webskin.

Walkthrough: Call a Webskin on a HTML Object

In this walkthrough we're going to call the standard dmHTML webskins on the Home Page HTML object.

1. Login to the Webtop and click on the Home Page dmHTML object in the site tree (the white page icon... not the blue dmNavigation object)
2. Copy the objectId of this object to the Clipboard (located at the bottom of the summary for the object)
3. Now, locate the ./webskin/dmHTML folder in your project
4. Take note of the current webskins in that folder.
5. Now enter a url to call each of the webskins in that folder using the following format

Example URLs Only
<code>http://localhost:8888/index.cfm?objectId=E689D66F-96FD-E9F6-B1AF64B8DAE78A69&view=displayPageStandard</code>

6. Discuss how this forms the basis of almost ALL web applications

Creating Webskins

The "smart controller" wires up the content type, the content item and webskin (view) by convention based on the parameters in the URL.

Webskins are literally just ColdFusion templates. Any file with a .cfm extension in a directory under ./myproject/webskin will be automatically registered as a webskin by the FarCry Framework. Consequently only files that you intend to be webskins should ever be stored here. To create a webskin simply create the template in the correct directory and RELOAD/RESTART the application.

**Reloading/Restarting The Application To Recognise Webskin Changes**

When a FarCry application starts it works out all the available webskins for every content type and stores them in memory. Every time you add a new webskin you need to reload the application in order for the system to recognise it. If you are logged in you can reload the application from the "Tray Menu" or by simply running a page with &updateapp=1 at the end of the URL.

Alternatively, if you only want to reload part of the application, go into the [webtop / admin / developer utilities / reload application] and simply select the options you wish to reload. This is a handy tab to have open at all times while your developing and constantly adding/updating webskins and metadata.



Changing webskins does **NOT** require an application restart. You only need to restart if you are adding, removing or renaming webskin files.

Naming Webskin Templates

Although you can give your webskin template any file name, in practise it makes sense to follow the naming standards used by the rest of the community. The following table outlines common naming prefixes for templates and explains their uses within the FarCry framework. An asterisk (*) denotes a wildcard, where you would use your own unique name to differentiate templates of similar purpose.

Template Name	Purpose	Examples
display*	General prefix for all display templates. By default FarCry allows anonymous users to view anything prefixed with display*	
displayPage*	A full page template display, typically incorporating header and footer HTML, and designating the entire page layout. These templates are available by default in the content editing wizards for contributors in community plugins, such as FarCry CMS.	./dmHTML/displayPageHome.cfm
displayPageStandard.cfm	In the absence of any additional criteria, FarCry will attempt to display a content item with this template, assuming it is available. It is in effect the "standard" full page template view.	
displayTeaser*	A teaser view such as a title and short copy with a link to the full page view. Often used for listing other content objects on a page. Automatically recognised by many publishing rules.	./dmHTML/displayTeaserFeature.cfm
displayTeaserStandard.cfm	Similar to displayPageStandard, in that if no other criteria is given the framework will attempt to render a given content object with this template, assuming it is available and a teaser view is required.	
displaySearchResult.cfm	Used in search plugins to provide a universal search result teaser. As individual content types might be better suited to different teasers you can provide your own as needed.	
edit*	General prefix for editing a content type. By default, these views are secured to content publishers only and are not accessible by anonymous users.	./dmProfile/editOwnProfile.cfm
edit.cfm	Default edit handler. Typically this is not present, at least in simple content types, as the framework will automatically build edit handlers from the formtool metadata. However, like most things in FarCry you can override this as needed.	

These are just a few of the regularly used webskin names within the FarCry framework. For a complete list of reserved webskin template names review: <https://farcry.jira.com/wiki/display/FCDEV50/Webskin+Templates>

Hooking Up The Webskin To Data

Every time a webskin is invoked it is done so in the context of a specific content type.

For example, viewing a particular news article with a displayPageStandard, or a product item with displayPageProduct.cfm, or whatever. FarCry always provides the entire content item record to the webskin (or view) as a structure called **stobj**.

stobj contains the typename, and all the property keys and values for the object in question, including array properties.

farcry - open source - struct	
AOBJECTIDS	farcry - open source - array [empty]
ARELATEDIDS	farcry - open source - array [empty]
BODY	<P>Built from the ground up on the revolutionary ColdFusion MX server platform, farcry is the affordable, powerful site management solution that is quick to implement, and intuitive to use.</P><P>farcry offers the advanced features of high-end site-management solutions, including sophisticated container management, publishing rules, version control and integrated search, at a small fraction of the cost.</P>
CATHTML	[empty string]
CREATEDBY	farcry
DATETIMECREATED	2008-06-03 10:21:07.0
DATETIMELASTUPDATED	2008-06-08 06:53:39.0
DISPLAYMETHOD	displayPageStandard
EXTENDEDMETADATA	[empty string]
LABEL	farcry - open source
LASTUPDATEDBY	farcry_CLIENTUD
LOCKED	0
LOCKEDBY	[empty string]
METAKEYWORDS	[empty string]
OBJECTID	4BCC751B-C373-8C48-73FB36FE3DD7E87D
OWNEDBY	CDD3B33E-A463-9B75-DC0150D316830765
REVIEWDATE	2050-12-17 00:00:00.0
STATUS	draft
TEASER	Built from the ground up on the revolutionary ColdFusion MX server platform, farcry is the affordable, powerful site management solution that is quick to implement, and intuitive to use.
TEASERIMAGE	[empty string]
TITLE	farcry - open source
TYPENAME	dmHTML
VERSIONID	E689D66F-96FD-E9F6-B1AF64B8DAE78A69



Array Properties

Array properties are automatically provided as a CF array in the structure value field. The array contains all the related object references as UUID values. More on this later.

The data contained in the **stobj** structure can be referenced as a simple ColdFusion variable, and then combined with mark-up to produce the desired output. The similarity between this and any normal procedural ColdFusion template is deliberate - the framework authors have tried hard to make the creation of webskins or views very similar to building a simple ColdFusion template.

Simple Example of a News Template

```
<cfsetting enablecfoutputonly="true">
<!--- // Dead Simple Example Webskin Template --->
<cfoutput>
  <h1>#stobj.title#</h1>
  #stobj.body#
</cfoutput>
<cfsetting enablecfoutputonly="false">
```



Whitespace Management

FarCry best practice sets `<cfsetting enablecfoutputonly="true" />` at the top of your webskin and `<cfsetting enablecfoutputonly="false" />` at the bottom. This means simply that any content you want displayed must appear between `<cfoutput></cfoutput>` tags. Try to avoid putting custom tags within `<cfoutput>` - without special treatment (eg. using `cfsilent` internally) the custom tag will output all its contents as whitespace.

Webskin Decorators

Every template can be supplemented with additional metadata, called a "decorator". You should make a habit of adding relevant decorators as it makes the whole system read much better, helping to provide human readable template names, inline documentation and so on. Metadata is incorporated by including a series of specific comments at the top of each template.

Attribute	Description
@@displayname:	Human readable display name for the template. Otherwise the framework will display the filename instead.
@@description:	Longer description about the template's purpose. This can run to any length but is typically a paragraph only.

For a complete list of decorators and their uses, see: <https://farcry.jira.com/wiki/display/FCDEV60/Summary+of+View+Decorators>

Template metadata is always stored in a ColdFusion comment.

On initialisation, FarCry scans the registered webskins for their additional metadata and stores it in memory. If you make a change to the metadata you may need to re-start the application in order to see the change come into effect.

Sample Metadata Attached With Decorators

```
<!--- @@displayname: Core Home Page --->
<!--- @@description: Home page for the FarCry Core developer portal. --->
<!--- @@Cachestatus: 1 --->
<!--- @@Cachetimeout: 60 --->
<!--- @@Fualias: home --->
```

Walkthrough: Create displayPageSuper.cfm

In this walkthrough we're going to create a simple dmHTML webskin template to play with some of the ideas we've just considered.

1. Locate the ./webskin/dmHTML folder in your project
2. Create a file called displayPageSuper.cfm
3. Write up some basic HTML in a <cfoutput>

./webroot/farcry/projects/myproject/webskin/dmHTML/displayPageSuper.cfm

```
<!--- @@displayname: Demo Template --->
<cfoutput>
<h1>Hello Cruel World</h1>
</cfoutput>
```

4. Reload the application to pick up the template change. Use the [Reload Application] tool in the webtop.
5. Select the Site Overview Tab. Edit the FarCry Support HTML page and change its template to the one you just created.
6. Save and Preview the page. Check the HTML source and discuss with your instructor.
7. Edit your template and add a cfdump to the page for **stobj**

```
<cfdump var="#stobj#" label="Content Object" />
```

8. Save and preview the page.

Static Media Assets

In a FarCry project we normally make a distinction between media assets (such as files, images, video, etc) that are managed by the application (ie. Content Managed) and assets which are hard-coded and fixed in the code base (ie. Application Chrome).

When you are dealing with the application's chrome, the fixed static images and so on that make up the graphic theme of the site, in most instances you do not want to have these mixed with assets that are content managed by users. The standard for managing media assets of this nature is to store them under the webroot (for obvious reasons) in a directory called **.wsimages/** or under the **.css/images** directory if they are relative to the style sheet.

Sample Project Directory Structure

```
c:\farcry
  \projects
    \myproject
      \webskin
      \www (standard project webroot)
        \wsimages
        \css\images
```



Version Control

If you version control your code base you want to make sure that content managed images are not included in your repository but be absolutely sure that images, etc pertaining to the design of the application are in the repository. Hence the clear split in where to store them.

Webskin Tag Libraries

Remember a webskin is just like a ColdFusion template - you can do all sorts of things. The webskin is strictly speaking a VIEW and following good programming practise you should refrain from doing business logic in this area. However, you can reference ColdFusion tags, FarCry service components and custom tag libraries as needed.

FarCry has a special custom tag library dedicated to making life easier when building webskins. It includes all sorts of goodies from building navigation, to breadcrumbs, to rendering other views and more. You can find this library in the core framework at: **./core/tags/webskin**. To make use of these tags you will need to import them first.

For a complete list of webskin tags see: <http://docs.farcrycore.org/p600/>

Importing Custom Tag Libraries

```
<!-- @@displayname: Home Page --->
<!-- @@author: Matthew Bryant (mbryant@daemon.com.au)--->

<!-- import tag libraries --->
<cfimport taglib="/farcry/core/tags/webskin" prefix="skin" />

<!-- breadcrumb; detects position in the site tree and builds a breadcrumb --->
<skin:breadcrumb />
```



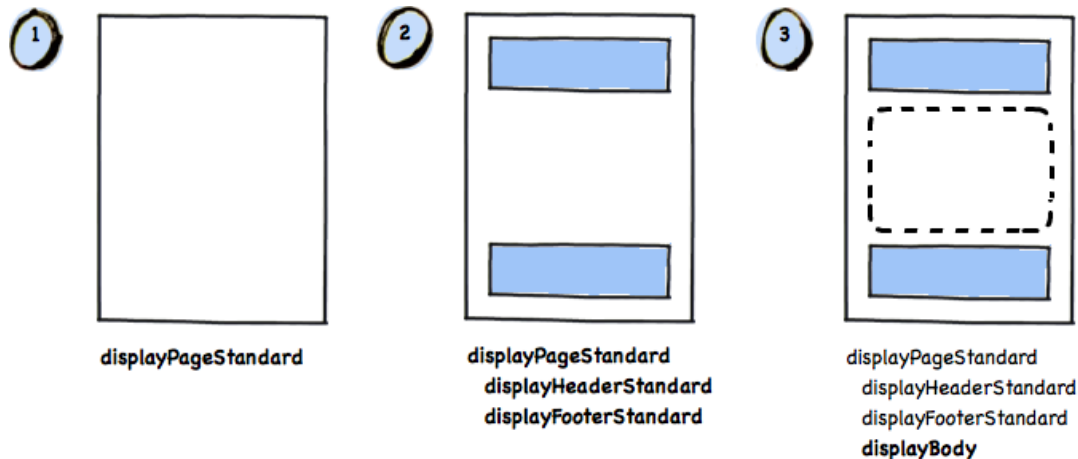
Project Tag Libraries

We recommend creating your own custom tag libraries under the project folder and importing them in the same way:
./myproject/tags/mytaglibrary

Composite Views

In practice, web pages in FarCry are assembled from multiple webskins. While you can do a lot with custom tags, the best way to build web pages is by compositing multiple webskins together.

Consider the following real world examples of composite views from the Fandango theme.



FarCry doesn't prescribe a specific templating architecture - you can get pretty sophisticated in your approach or keep it dead simple. Fandango is a good best practice starting point.

Building a Composite View

A typical starting point for any composite view is working with headers and footers. It's very common to have a similar header across all web pages, albeit with slight variations such as page title, highlighted navigation and so on. Likewise the footer structure.

You can call another view directly using the `skin:view` tag. `skin:view` is a real workhorse - we use it **all the time**.

Example from ./dmHTML/displayPageHome.cfm

```

<div id="main">
  <div class="container_12">
    <div class="grid_12 content">
      <skin:view typename="#stobj.typename#" objectid="#stobj.objectid#" webskin="#url.bodyView#" />
    </div>
  </div>
</div>

```

View Inheritance

FarCry is an object oriented framework. All content types extend the core `types` component (an abstract class). Any views associated with `types` are automatically inherited by other content types, such as our dmHTML content type.

More on content types later. Long story short, you can assume that **any** view located in `./webskins/types` is available to **all** content types.

Looking at our dmHTML content type in Fandango, we have at least the following views available:

- displayPageStandard
- displayTeaserStandard
- displayHeaderStandard
- displayFooterStandard
- displayBody
- displaySimpleSearchResult

Webskin Tracer

The **tray** that appears at the bottom of the website when you are logged in, contains a marvellous developer utility called the "Webskin Tracer". You can activate it for any page by simply selecting the webskin tracer option.

The screenshot shows the Fandango website with the Webskin Tracer tray visible at the bottom. The tray contains a list of navigation links on the left and a list of developer utilities on the right. The Webskin Tracer utility is highlighted in the tray.

Fandango Theme

Fandango is an HTML5 based theme, built on the 960.gs grid for the FarCry Core publishing platform. Fandango is the default installation theme for the upcoming FarCry 6.2 release (due out in Jan 2012), but can be used with earlier versions of FarCry with little or no modification.

Go to: [Fandango Theme](#)

FarCry Plugins

The FarCry publishing platform has a host of open source and commercial plugins available for use from Google Analytics (tracking stats) to Janrain Engage (social media logins). Given that its easy to produce simple plugins, the Plugin Directory concentrates on larger services and solutions for the greater community.

Go to: [FarCry Plugin Explorer](#)

Webskin Tracer Tray:

- Switch Tray Position
- Hide Tray
- Rebuild Page
- Rebuild Site
- Update Application
- Debug Mode
- Profiler
- Webskin Tracer**

Navigation Links:

- Home
- Getting Started
- Typography
- News

Footer:

- Edit
- Rules
- Drafts
- Caching

The entire pages composite of views is laid bare; highlighting each webskin executed for the page. Select a specific view and the HTML fragment is highlighted, with a tooltip detailing exactly where the webskin template can be found.

The screenshot shows the FarCry Core website with a 'Webskin Tracer' window open. The tracer displays a list of webskins and their execution times:

- dmNavigation: displayPageStandard
- dmHTML: displayPageHome (0.26s)
- dmHTML: displayHeaderStandard (0.04s)
- container: displayContainer (0.05s)
- ruleText: execute (0.001s)
- container: displayContainer (0.043s)
- ruleCarousel: execute (0.037s)
- dmHTML: displayBody (0.017s)
- container: displayContainer (0.01s)
- ruleColumns: execute (0.004s)
- dmHTML: displayFooterStandard (0.009s)

A 'FarCry Profiler' window is also open, showing a list of views and their execution times:

- 1 Request initialisation - Server specific request scope
- 0 Request initialisation - Parse URL
- 3 Request initialisation - Request modes
- 6 Display - Object
- 9 Display - Check permission
- 3 View - displayPageStandard [dmNavigation:E689D721-B6C9-605B-DE1D813E4CDA3339]
- 2 Display - Object
- 50 Display - Check permission
- 4 View - displayPageHome [dmHTML:E689D66F-96FD-E9F6-B1AF64B8DAE78A69]
- 18 View - displayHeaderStandard [dmHTML:E689D66F-96FD-E9F6-B1AF64B8DAE78A69]
- 4 View - displayContainer [container:2774ED70-1009-11E1-803400241D85B4DF]
- 3 View - execute [ruleText:2772E50-1009-11E1-803400241D85B4DF]
- 4 View - displayContainer [container:277B5610-1009-11E1-803400241D85B4DF]
- 13 View - execute [ruleCarousel:27775E70-1009-11E1-803400241D85B4DF]
- 4 View - displayBody [dmHTML:E689D66F-96FD-E9F6-B1AF64B8DAE78A69]
- 7 View - displayContainer [container:2780AD40-1009-11E1-803400241D85B4DF]
- 4 View - execute [ruleColumns:277DA000-1009-11E1-803400241D85B4DF]
- 16 View - displayFooterStandard [dmHTML:E689D66F-96FD-E9F6-B1AF64B8DAE78A69]

FarCry Profiler

The FarCry profiler details how long each part of your request is taking to execute on the server. This is a great tool for identifying slow running pages and candidates for caching.

The screenshot shows the FarCry Core website with a 'FarCry Profiler' window open. The profiler displays a list of views and their execution times:

- 1 Request initialisation - Server specific request scope
- 0 Request initialisation - Parse URL
- 3 Request initialisation - Request modes
- 6 Display - Object
- 9 Display - Check permission
- 3 View - displayPageStandard [dmNavigation:E689D721-B6C9-605B-DE1D813E4CDA3339]
- 2 Display - Object
- 50 Display - Check permission
- 4 View - displayPageHome [dmHTML:E689D66F-96FD-E9F6-B1AF64B8DAE78A69]
- 18 View - displayHeaderStandard [dmHTML:E689D66F-96FD-E9F6-B1AF64B8DAE78A69]
- 4 View - displayContainer [container:2774ED70-1009-11E1-803400241D85B4DF]
- 3 View - execute [ruleText:2772E50-1009-11E1-803400241D85B4DF]
- 4 View - displayContainer [container:277B5610-1009-11E1-803400241D85B4DF]
- 13 View - execute [ruleCarousel:27775E70-1009-11E1-803400241D85B4DF]
- 4 View - displayBody [dmHTML:E689D66F-96FD-E9F6-B1AF64B8DAE78A69]
- 7 View - displayContainer [container:2780AD40-1009-11E1-803400241D85B4DF]
- 4 View - execute [ruleColumns:277DA000-1009-11E1-803400241D85B4DF]
- 16 View - displayFooterStandard [dmHTML:E689D66F-96FD-E9F6-B1AF64B8DAE78A69]



You can turn on the webskin tracer and profile tools without being logged in by using your projects "secret key" on the URL: <http://localhost:8888/index.cfm?objectid=E689D721-B6C9-605B-DE1D813E4CDA3339&tracewebskins=farcry>

Walkthrough: Creating a Composite View

1. turn on the webskin tracer
2. review various pages with your instructor
3. Add Header & Footer views to `./dmHTML/displayPageSuper.cfm`

Walkthrough: Putting It All Together

Get comfortable with how webskins are assembled in the real world. You should be able to dissect the sample application webskins and understand how they have been put together.

1. Review the other templates in the `./webskin/dmHTML` directory with your instructor
2. Consider the use of custom tags, headers, footers and other layout mechanisms

UNIT 05 - Content Types

Objectives

This unit covers the fundamental building blocks of any FarCry application: the Content Type. By the end of this unit you should be able to create and deploy your own content types.

Super Hero Handbook

In a bid to test the theory that FarCry is indeed a framework, we're building the Super Hero Handbook application. It's a somewhat frivolous application that has the benefit of not tying us down to a specific set of functionality. Plus it should be a lot of fun.

We can invent things as we need them to showcase aspects of the technology without being constrained by the more traditional, user focused goals of a typical application. It also has the potential of being a lot more complex than your average "build a blog in 10 minutes" exercise which appears to be the benchmark for determining the viability of many frameworks.

FarCry is accused of being an extensible, cutting edge, content management system (CMS). Ok so that's true. But the CMS functionality is actually built on the FarCry Framework. It's a framework that just happens to be suitable for building many other sorts of web based applications. In fact, we're firm believers that content management is really a commodity service that just about every application needs, so its great to be able to build an application that has immediate access to such rich CMS tools.

Content Types

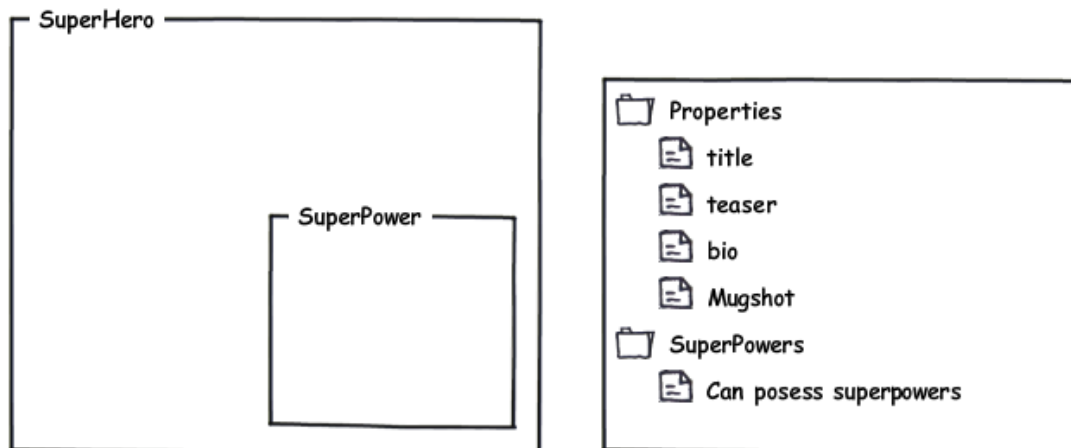
The central building block of the farcry framework is the content type. Think of a content type as the definition of a table in a relational database as this is exactly what they are.

A content type is defined by a ColdFusion component (CFC), and depending on its complexity a variety of auxiliary files. A content type can be just about any sort of persistent data; for example, a news item, product, log table, transaction record, user profile and so on. For example, the pages currently on your site are stored in the content type called dmHTML.

More about the mechanics of content types later - for now we need to think about the basic content types that constitute the Super Hero Handbook, what "content objects" will model the data we have in our application and how they fit together. Lets start relatively simple, and evolve from there.

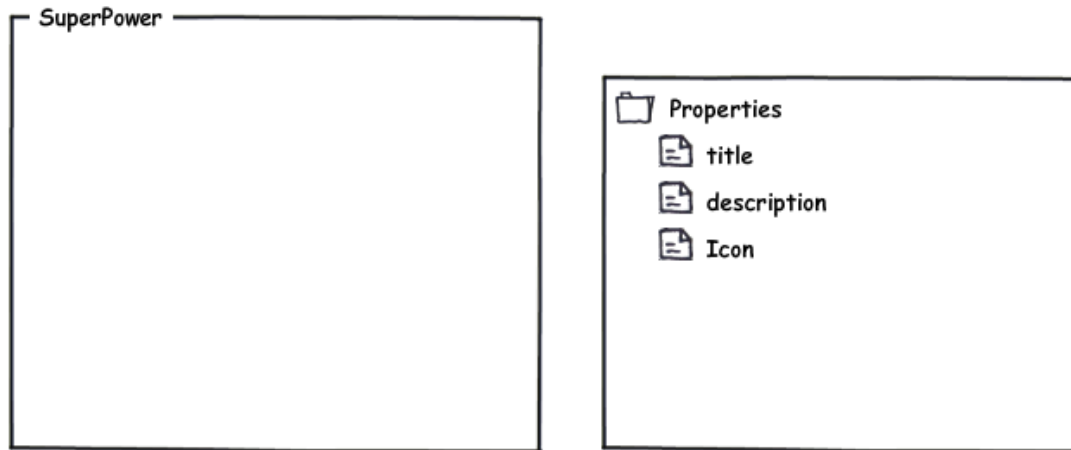
The Super Hero

We need a central profile to capture our catalogue of super powered beings, their properties and relationships.



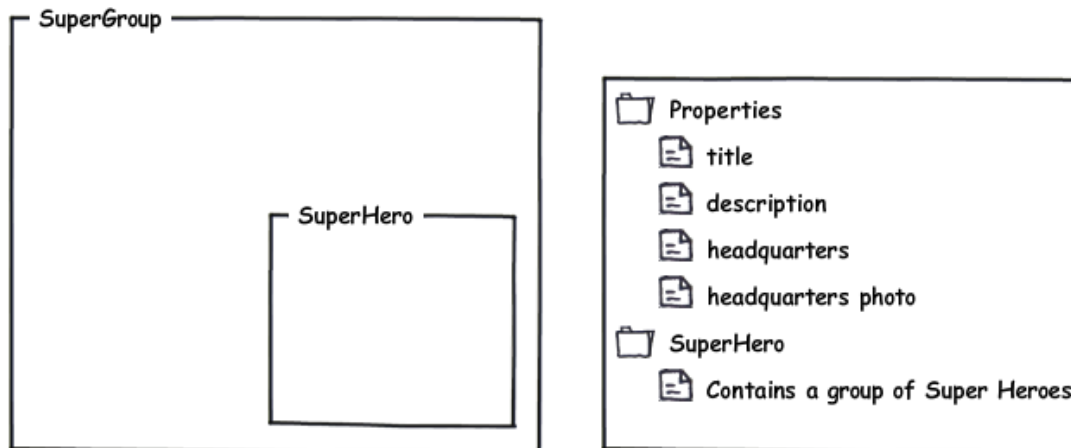
The Super Power

Super powers are generally not unique (with some exceptions). It would be nice to have a library of super powers we can assign to super heroes as we add them to the system.



The Super Group

We're not talking ABBA here. The super group is a league or association of super heroes; for example, the Sovereign 7, Fantastic Four, and the League of Justice. A super hero could potentially belong to several groups over their lifetime.



Building a Content Type

The FarCry content type is a central building block for the FarCry Framework. Developers can extend and change the behaviour of existing content types, create their own content types and group related content types into libraries.

We start by creating a content type component and defining the properties that exist for the content objects it manages. Content types are saved in your projects `./packages/types` sub-directory.

```
<!-- ./packages/types/superHero.cfc -->
<cfcomponent name="superHero" extends="farcry.core.packages.types.types" output="false"
displayname="Super Hero">

    <cfproperty
        name="title" type="string" default="" hint="Super hero title." />
    <cfproperty
        name="secretHideout" type="string" hint="The secret hideout location of the super hero" />
    <cfproperty
        name="teaser" type="longchar" default="" hint="Mini intro for super hero biography." />
    <cfproperty
        name="biography" type="longchar" default="" hint="Super hero biography." />
    <cfproperty
        name="imgHero" type="string" hint="The image of the hero" />

</cfcomponent>
```

Hooking Into the Framework

The component must extend FarCry's core abstract class "types" in order to engage with the FarCry Framework. This abstract class provides both additional system properties and methods necessary for the content type to work within the framework.



Advanced developers might find it useful to build their own abstract classes for collections of content types. However, the component must ultimately extend "types" in order to work.

Content Object API (COAPI)

The cfproperty tag doesn't really do a great deal other than define meta data for the component in ColdFusion. FarCry leverages this fact to allow developers to define the default behaviour of the component within the framework.

In the superHero.cfc example we're using the cfproperty tags to define the name and FarCry data type for each property of our Super Hero content type.

Next we use the FarCry type deployment interface to build a data persistence model for the content items or records associated with the component.

FarCry detects that a component exists but hasn't got a corresponding persistence model in the database - the "deploy" options creates all the tables required for this content type. The columns are mapped to their relevant data types, depending on the FarCry data type nominated and the specific relational database you are using.



The integrated FarCry ORM differs from other frameworks that might ordinarily rely on the database schema for metadata, for example Reactor and Transfer. Typically these frameworks will generate components to manage interactions with the database. In contrast, FarCry generates a data schema to manage the data represented by its components.

FarCry relies on the component to define the required data model definition and has tools to keep the database model in sync with changes to the component property set. Not only will the framework deploy tables it will also alter columns and data-types to match changes in the underlying component as required. The COAPI is even clever enough to set precision on table columns and build indices for better performance.

All properties in the component map to a specific column in the content type table, with the exception of array properties (discussed later on).



types is an abstract class that contains a number of system attributes/properties that are inherited by any custom content type.

Walkthrough: Creating Super Hero

We'll kick off our sample application by creating a content type for the Super Hero.

1. Create a new file called superHero.cfc and save this into your project's ./packages/types directory.
2. Copy the following code into this file, save and review with the instructor

./myproject/packages/types/superHero.cfc

```
<cfcomponent name="superHero" extends="farcry.core.packages.types.types" output="false"
displayname="Super Hero">

  <cfproperty
    name="title" type="string" default="" hint="Super hero title." />
  <cfproperty
    name="secretHideout" type="string" hint="The secret hideout location of the super hero"
  />
  <cfproperty
    name="teaser" type="longchar" default="" hint="Mini intro for super hero biography." />
  <cfproperty
    name="biography" type="longchar" default="" hint="Super hero biography." />
  <cfproperty
    name="imgHero" type="string" hint="The image of the hero" />

</cfcomponent>
```

3. Go to the FarCry Webtop and deploy the content type: ADMIN > Developer Utilities > COAPI Tools > Types
4. Locate your new content type, "superHero" and select DEPLOY.

5. Click on the [Scaffold] button now to create an Administration scaffold
6. Select the checkbox next to "Create type admin interface",
 - a. put in a title for your administration list
 - b. select label and imgHero as the fields to appear in your list



Basic Scaffolding

The Scaffold option for deployed content types generates some sample code you can use to get started right away.

"Create type admin interface" creates a small XML file that generates a menu item for you in the FarCry webtop.

The framework provides a default edit handler and display view so no need to add anything else here just yet.

7. Reload your Application. This time you will need to update the [Webtop] option
8. Go to the Content Tab. Locate your Super Hero admin and create some Heroes (or Villains!)



Editing Content Types

When you created your first superHero object you may have noticed a rudimentary edit handler allowing you to key in details, save and update. That's not actually part of the scaffold code that was generated at all. The edit handler is dynamically generated at run time based on the metadata associated with the superHero component's cfproperty tags. Ok. So its pretty minimalist now but wait.. it's only the beginning.



Wait Up!

Make sure you understand the concept of extends, displayname and property type before the instructor starts babbling on about something else.

Formtools

Formtools is essentially a library of clever UI controls that react dynamically to the metadata encapsulated in your content type definition.

There's an extensive number of configuration options for every UI control. You can bypass FarCry's automated layout engine and hand code your forms using the controls if required. You can also override the behaviour of the default controls or create your own controls entirely.

So without going into the detail behind how formtools works - what's in it for the FarCry developer? Well most of the time you will never have to build an administration interface for your content types. Seriously.

Next we will extend our simple superHero content type to leverage these "formtool" features. We're focusing on a selection of the individual property tags in superHero.cfc for the sake of brevity so be sure to fill in the gaps.

Check out the code sample on this page, and pay particular attention to the attributes starting with "ft".

./myproject/packages/types/superHero.cfc

```
<cfcomponent name="superHero" extends="farcry.core.packages.types.types" output="false"
displayname="Super Hero">

  <cfproperty
    name="title" type="string" default="" hint="Super hero title."
    ftSeq="1" ftFieldset="General Details" ftLabel="Title" />
  <cfproperty
    name="secretHideout" type="string" hint="The secret hideout location of the super hero"
    ftSeq="2" ftFieldset="General Details" ftLabel="Secret Hideout" />
  <cfproperty
    name="teaser" type="longchar" default="" hint="Mini intro for super hero biography."
    ftSeq="3" ftFieldset="General Details" ftLabel="Teaser" />
  <cfproperty
    name="biography" type="longchar" default="" hint="Super hero biography."
    ftSeq="4" ftFieldset="General Details" ftLabel="Biography"
    ftType="richtext" />
  <cfproperty
    name="imgHero" type="string" hint="The source image to upload"
    ftSeq="10" ftFieldset="Imagery" ftLabel="Hero Image"
    ftType="image" ftDestination="/images/superHero/imgHero" ftImageWidth="120"
    ftImageHeight="120" ftAutoGenerateType="fitInside" />
</cfcomponent>
```

Form Layout

By modifying the `ftSeq` and `ftFieldset` attributes we can order form fields and group them into specific field sets. Every field sharing the same fieldset value will appear in a correctly formatted field grouping in the edit handler.



ftHelpTitle and ftHelpSection

If you are feeling really adventurous you could add `ftHelpTitle` and `ftHelpSection` to the very first `cfproperty` of each fieldset and you'll get a nicely formatted inline help message allowing you to describe what the fieldset is all about.

Form UI Controls

There are several `ft` attributes that are common across all formtools, such as `ftLabel` which provides a text label for the form field.

General Formtool Property Options

Attribute	Description	Value
<code>ftType</code>	Type of form control	Defaults to the type value
<code>ftLabel</code>	Text label for the form element	Defaults to property name if absent
<code>ftShowLabel</code>	Flag to show/hide text label for the form element	Defaults to true
<code>ftSeq</code>	Numeric value for form field display order	No default value
<code>ftFieldset</code>	Text used as a title to group a set of fields	No default value
<code>ftWizardStep</code>	Allows you to create multi-step forms; set to the title of the step as you would <code>ftFieldset</code>	No default value
<code>ftStyle</code>	Inline style to apply to form element	No default value
<code>ftClass</code>	Class to apply to form element	No default value
<code>ftDisplayOnly</code>	Boolean that prevents the field from being editable	Defaults to false
<code>ftDefault</code>	Default form field value	No default value
<code>ftDefaultType</code>	The type of default value: "value", "expression" or "evaluate"	Defaults to "value"
<code>ftHelpTitle</code>	Title of a fieldset-related inline help section; only works on the first property in a fieldset	No default value
<code>ftHelpSection</code>	Section text for inline help related to fieldset; only works on the first property in a fieldset	No default value
<code>ftValidation</code>	A comma-separated list of validation requirements	No default value
<code>ftHint</code>	A small inline hint that appears below the field to assist users	No default value

Formtool Validation



Client Side Validation Uses jQuery Validation Plugin

Client side validation options reflect the move to jquery/jqueryui as the standard UI for the webtop

- <http://docs.jquery.com/Plugins/validation>
- <http://bassistance.de/jquery-plugins/jquery-plugin-validation/>

ftValidation with a comma separated list of validation requirements.

- required (not blank)
- validate-number (a valid number)
- validate-digits (digits only)
- validate-alpha (letters only)
- validate-alphanum (only letters and numbers)
- validate-date (a valid date value)
- validate-email (a valid email address)

- validate-date-au (a date formatted as; dd/mm/yyyy)
- validate-currency-dollar (a valid dollar value)
- validate-selection (first option e.g. 'Select one...' is not selected option)
- validate-one-required (At least one textbox/radio element must be selected in a group)

ftType: Specialised Form Controls

The specific type of UI control is determined by the ftType attribute. If you specify nothing it defaults to the same value as the type attribute, that is the underlying data type. On the other hand if you nominate something more exciting like ftType="richtext" formtools will automatically load all the relevant JavaScript libraries to render the tinyMCE rich text editor to capture input for your property.

Example ftType="richtext" Formtool Control

```
<cfproperty
name="Body" type="longchar" hint="Main body of content." required="no" default=""
ftSeq="12" ftFieldset="Body" ftWizardStep="Body" ftLabel="Body"
ftType="richtext"
ftImageArrayField="aMedia" ftImageTypeName="dmImage" ftImageField="StandardImage"
ftTemplateTypeList="dmImage,dmFile" ftTemplateWebSkinPrefixList="insertHTML"
ftTemplateSnippetWebSkinPrefix="insertSnippet" />
<cfproperty
name="aMedia" type="array" hint="" required="no" default=""
ftSeq="13" ftFieldset="Relationships" ftWizardStep="Body" ftLabel="Associated Media"
ftJoin="dmImage,dmFile" bSyncStatus="true" />
```

It won't surprise you to learn that the ftType attribute options begin to describe a whole library form controls that you can leverage to do your bidding including:

- date time pickers
- list boxes
- tree controls
- checkboxes
- radio buttons
- And more

A complete compendium of formtool controls can be found on the FarCry Developer WIKI at:

- <http://docs.farcrycore.org/p600/> – select Formtools
- <http://farcry.jira.com/wiki/display/FCDEV50/Formtools>
- <http://farcry.jira.com/wiki/display/FCDEV40/Form+Tool+Property+Metadata>

Anatomy of a Formtool

Each ftType corresponds to a specific formtool component that comprises of at least an edit, display and validation method. The edit method defines the relevant form element, any additional semantic markup that might be relevant (such as a label) and any associated JavaScript or UI cleverness needed to render the control.

Standard formtool methods:

- **Edit** builds the form control with associated smarts.
- **Display** describes the semantic markup required to render a display for the property. For example, an ftType="url" will activate the URL as a link by default.
- **Validation** provides some server side validation, and in the case of complex controls mangles the data into the format the underlying database model expects.

Best of all if you don't like the core library of widgets then you can always make your own. The core formtools can be overridden, and/or supplemented with brand new controls by deploying them through both plugins and your own project. But lets just stick to the basics for now.

Walkthrough: Using Formtools Metadata

1. Update your superHero component metadata using the example code above as a guide.



Restarting Your Application

Every time you update component metadata you will need to reinitialise the application for FarCry to recognise the change. This can be done by adding updateapp=1 to the end of your projects URL or using the [Reload Application] utility in the webtop or the admin "tray" option.

2. Reload the edit form for your "Super Hero" and make sure things are going according to plan.
3. Try uploading a few images and discuss with your instructor.

**Image Quality Awesome**

FarCry 6.x leverages the underlying ColdFusion 8 CFIMAGE tools.

Wizards

But what about a multi-step wizard you say? So far the content types have been simple, but what if we have a component with properties up the wazoo or just plain like this wizard caper? Wizards can be defined with simple formtool (aka "ft") metadata as well.

Walkthrough: Formtool Wizards

Lets separate the image upload onto a step of its own.

1. Open the ./packages/superHero.cfc
2. Add Wizard Steps using the ftWizardStep formtool metadata

```
<!-- ./packages/types/superHero.cfc -->
<cfcomponent name="superHero" extends="farcry.core.packages.types.types" output="false"
displayname="Super Hero">

    <cfproperty
        name="title" type="string" default="" hint="Super hero title."
        ftSeq="1" ftFieldset="General Details" ftWizardStep="Teaser Information"
ftLabel="Title" ftValidation="required" />
    <cfproperty
        name="secretHideout" type="string" hint="The secret hideout location of the super hero"
        ftSeq="2" ftFieldset="General Details" ftWizardStep="Teaser Information"
ftLabel="Secret Hideout" />
    <cfproperty
        name="teaser" type="longchar" default="" hint="Mini intro for super hero biography."
        ftSeq="3" ftFieldset="General Details" ftWizardStep="Teaser Information"
ftLabel="Teaser" />
    <cfproperty
        name="biography" type="longchar" default="" hint="Super hero biography."
        ftSeq="4" ftFieldset="General Details" ftWizardStep="Biography Details"
ftLabel="Biography"
        ftType="richtext" />
    <cfproperty
        name="imgHero" type="string" hint="The source image to upload"
        ftSeq="10" ftFieldset="Imagery" ftWizardStep="Imagery" ftLabel="Hero Image"
        ftType="image" ftDestination="/images/superHero/imgHero" ftImageWidth="120"
ftImageHeight="120" ftAutoGenerateType="center" />

</cfcomponent>
```

3. Make sure that each <cfproperty> has a relevant ftWizardStep entry
4. Reload the formtool metadata and test that your wizard works.

**"Wizard Overkill"**

I think you'll agree that this example is slight overkill for the sake of demonstrating the wizard functionality. However, if you had a content type with 30 fields grouped into 10 fieldsets and maybe even editable by different security levels, wizards come into their own.

Lab: Build A Super Power Content Type

Put into practice everything you've just learnt and build a super power content type. We'll hook this up with the Super Hero later in the course.

1. Create a new ColdFusion component called ./packages/types/SuperPower.cfc
2. Create the following properties
 - title (string; required... think ftValidation="required")
 - description (longchar; try limiting the total characters to 512)
 - imgPower (image; 100x100 dimensions, destination: /images/superPower/imgPower)
3. Deploy the content type into the database
4. Use the Scaffold tool in the COAPI area to create an administration page
5. Add a bunch of super powers to your application (ColdFusion, Flex, Flash, Photoshop, etc.)

Bonus Lab: Image Sourced from Library

You can easily source images from the global image library. If you have some time on your hands, try implementing the an image sourced from the library with custom crops for your local content item.

Sample Code from ./myproject/packages/dmCarouselItem.cfc

```
<cfproperty name="imageSourceID" type="string"
  ftSeq="2" ftFieldset="General Details" ftLabel="Source Image"
  ftType="uuid" ftJoin="dmImage"
  ftHint="Select an image from the image library or create a new image.">

<cfproperty name="imageCarousel" type="string"
  ftSeq="3" ftFieldset="General Details" ftLabel="Carousel Image"
  ftType="image" ftDestination="/images/dmCarouselItem/imageCarousel"
  ftAllowUpload="false" ftSourceField="imageSourceID:SourceImage"
  ftAutoGenerateType="center" ftImageWidth="620" ftImageHeight="200"
  ftQuality="0.8" ftInterpolation="blackman">

<cfproperty name="imageThumbnail" type="string"
  ftSeq="4" ftFieldset="General Details" ftLabel="Thumbnail Image"
  ftType="image" ftDestination="/images/dmCarouselItem/imageThumbnail"
  ftAllowUpload="false" ftSourceField="imageSourceID:SourceImage"
  ftAutoGenerateType="center" ftImageWidth="95" ftImageHeight="30"
  ftQuality="0.8" ftInterpolation="blackman"
  ftHint="">
```

UNIT 06 - Content Relationships

Objectives

By the end of this unit you will have learnt how to relate content types to one another, using one-to-many and many-to-many relationships. You will be able to create user interfaces to allow editors to select and relate objects from libraries.

Array Properties

An array property can contain a series of object references to other objects in the COAPI; a many-to-many relationship. When it's presented to the system view or webskin it's represented as an actual array property, where each index contains the objectid (or primary key) of the related object. This objectid can then be used to reference the related object as needed. In addition, the order (or sequence) of the relationship is preserved so that programmers can rely on the order to be exactly as the user nominated.

Array properties in terms of how they are defined in a content type really don't differ all that much to normal properties. You add a `<cfproperty>` and simply define `type="array"` with a list of content types you would like to be associated using `ftJoin`; for example `ftJoin="superPower"`.

Example Array Property

```
<cfcomponent ..>
...
<cfproperty
  name="aPowers" type="array" hint="Array of superhuman powers."
  ftSeq="12" ftFieldset="Related Content" ftWizardStep="Relationships" ftLabel="Powers"
  ftType="array" ftJoin="superPower" />
...
</cfcomponent>
```



Array Properties Must Start With "a"

Array properties must be prefixed with the letter "a". If you don't do that then the deployment window will not be able to correctly detect that you have deployed the array property.

If you open up the database model after you deploy an array property, you'll notice that the property (for example, `aPowers`) is not represented by a column in the table. Array data is essentially an index of references to other objects and is defined by its own linked table (for example, `superHero_aPowers`). The array table includes the parent object, the referenced objectid, the sequence multiple entries should be represented in, and the typename of the referenced object. Importantly, the data model in the database itself does not need to be understood to work with arrays - the framework handles recording and retrieving this information.



Extended Arrays

Array properties can themselves be extended to include additional attributes - but lets keep it simple for now.

ftJoin is a critical piece of metadata as it designates exactly what content types are allowed to be related to this specific property. Interestingly, FarCry Framework is quite happy to have multiple content types referenced, even though the content types themselves may have very different sets of properties. The default UI controls for libraries are designed to allow users to switch between nominated content types if multiple types have been referenced.

Example Multi-Type Array Property

```
<cfproperty
  name="aMedia" type="array" hint="Local media library." required="no" default=""
  ftSeq="13" ftFieldset="Relationships" ftLabel="Associated Media"
  ftType="array" ftJoin="dmImage,dmFile" />
```

By default the library UI control for an array property will list all content objects of the nominated type, ordered by **datetimelastupdated**.



Library Options

It won't surprise you that the options for the library are **extensive** and include the ability to restrict what content objects are available, the order they are displayed, how they are displayed and so on. Review the formtool metadata dictionary entry on Libraries for more details.

Walkthrough: Super Powers

We're going to add an array property [aPowers] to join our hero to the powers they possess.

1. Open the ./packages/types/superHero.cfc component for editing.
2. Add the following property

```
<cfproperty
  name="aPowers" type="array"
  ftSeq="12" ftFieldset="Related Content" ftWizardStep="Relationships" ftLabel="Powers"
  fttype="array" ftJoin="superPower"
  hint="Array of superhuman powers." />
```

3. Go to the webtop COAPI admin area and deploy your new property. You should see a single conflict for the Super Hero component.
 - Deploy the superPower array property.
4. Reload the COAPI Metadata
5. Go to the Super Hero administration screen, and associate super powers from the library picker that should now be available in the edit handler.

Library Selected Webskin

By default the library will only show a content object's label, and if that's blank the objectid. However, like many aspects of the framework, this behaviour can be modified to suit your specific application. FarCry Framework has a special webskin (or view) for rendering the objects selected for the library: ./webskin/typename/librarySelected.cfm (where typename is represented by the actual typename you are trying to modify).

Walkthrough: Library Selected

Your super powers only shows the label for your content object. Lets make that look prettier.

1. Create a new webskin template ./webskin/superPower/librarySelected.cfm
2. Copy the following code into the webskin:

./myproject/webskin/superPower/librarySelected.cfm

```
<cfsetting enablecfoutputonly="true" />
<!--- @@displayname: Power (Library) --->

<cfoutput>
<div>
 #stobj.title#<br />
#stobj.description#
</div>
</cfoutput>

<cfsetting enablecfoutputonly="false" />
```

3. Reload the COAPI Metadata to register the newly added webskin.
4. Go back to the Super Hero admin and try adding super powers. Make sure your librarySelected webskin is working as expected.

UUID Property

A UUID property is a single object reference or one-to-many relationship. It behaves in very much the same way as an array property in terms of UI and the library options that are available. The object reference is stored in a simple string field in the database.

Example UUID Property

```
<cfproperty
  name="sidekickid" type="uuid" hint="Super hero sidekick."
  ftSeq="11" ftFieldset="Related Content" ftWizardStep="Relationships" ftLabel="Sidekick"
  ftType="uuid" ftJoin="superHero" />
```

Walkthrough: Side Kick (there can be only one)

Let's create an option to select another hero in the system as a sidekick.

1. Open the `./packages/types/superHero.cfc` component for editing.
2. Add a UUID property for sidekickid

```
./myproject/packages/types/superHero.cfc

<cfproperty
  name="sidekickid" type="uuid" hint="Super hero sidekick."
  ftSeq="11" ftFieldset="Related Content" ftWizardStep="Relationships" ftLabel="Sidekick"
  ftType="uuid" ftJoin="superHero" />
```

3. Go to the webtop COAPI admin area. You should see a single conflict for the Super Hero component.
 - Deploy the sidekickid UUID property.
4. Go to the Super Hero administration screen, and associate a side-kick from the library picker that should now be available in the edit handler.

Lab: Super Group

Create your own "super group" content type, complete with an array of super hero objects.

1. Create a new component; `./packages/types/superGroup.cfc`
 - title (string, required)
 - description (longchar, rich text area)
 - headquarters (string)
 - imgHeadquarters (image; 200x200 dimensions, fit inside)
 - aSuperHeroes (joining to superHero)
2. Deploy the new content type under the COAPI area
3. Use the scaffold utility to create an admin area and default webskins
4. Go to the administration area and add "Super Groups" containing your favourite Super Heroes



Bonus Points

If you are feeling super-keen, add a librarySelected webskin for your Super Hero content type.

UNIT 07 - Webskins II

Objectives

This unit is a workshop to discuss how to hook up different content types in the presentation tier - linking from one view to the next by the content's relationships.

Building the Presentation Tier

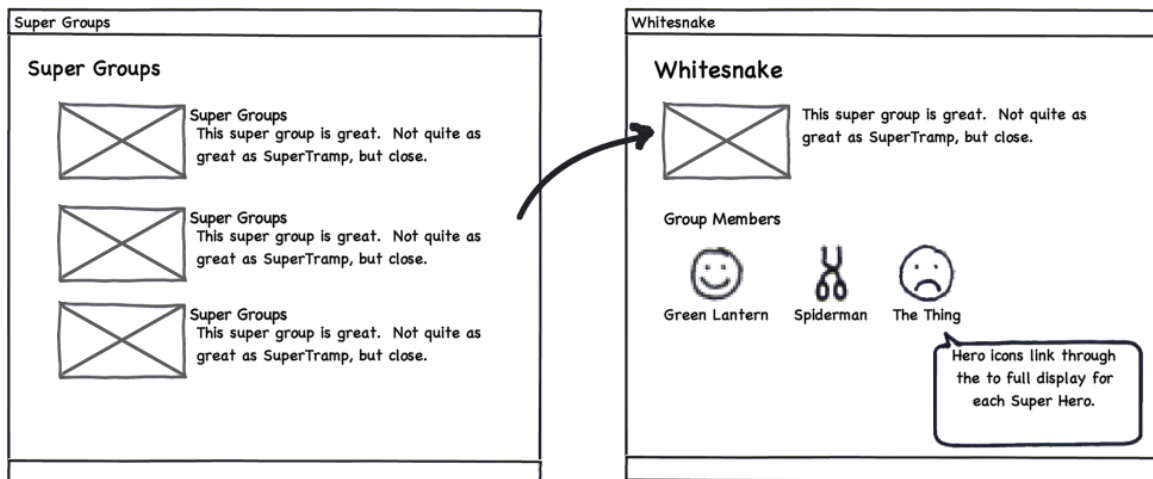
Up until this point we've been dealing with individual content types in isolation, with the exception of library references. What we want to look at now is how to bring these different content types together in the presentation tier for users to effectively interact with them.

Walkthrough: Wireframe Workshop

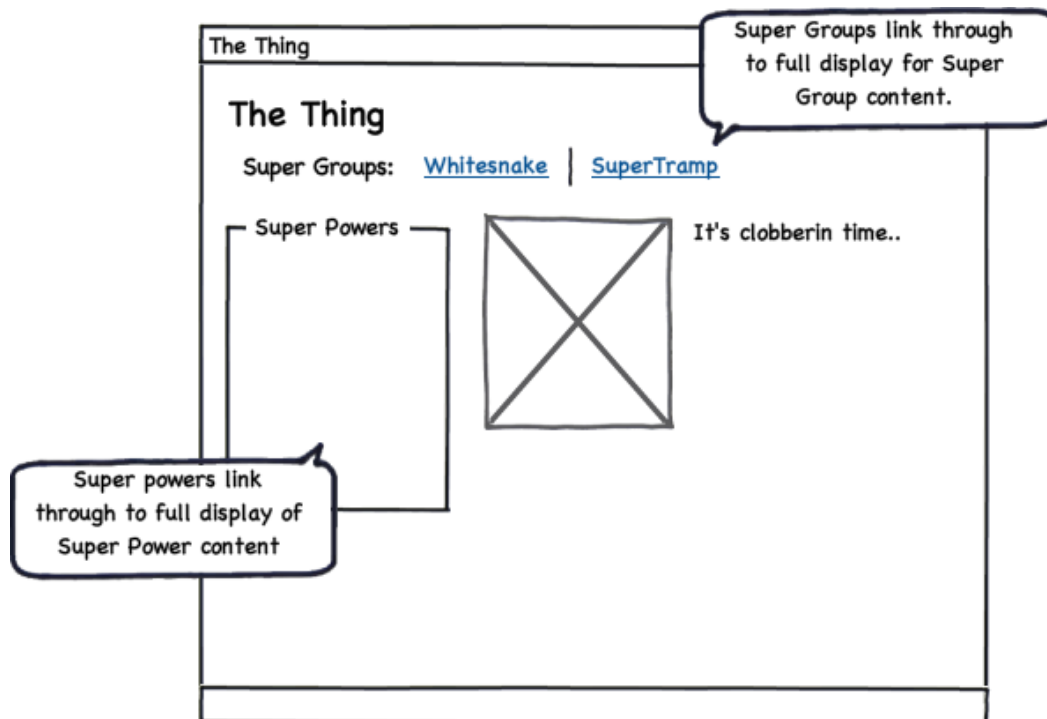
Now for some brainstorming and workshopping ideas of exactly how we're going to present this Super Hero information to the user.

1. Discuss how the different content types will interact
2. Draw out wireframes for:
 - Super Groups
 - Super Heroes
 - Super Powers

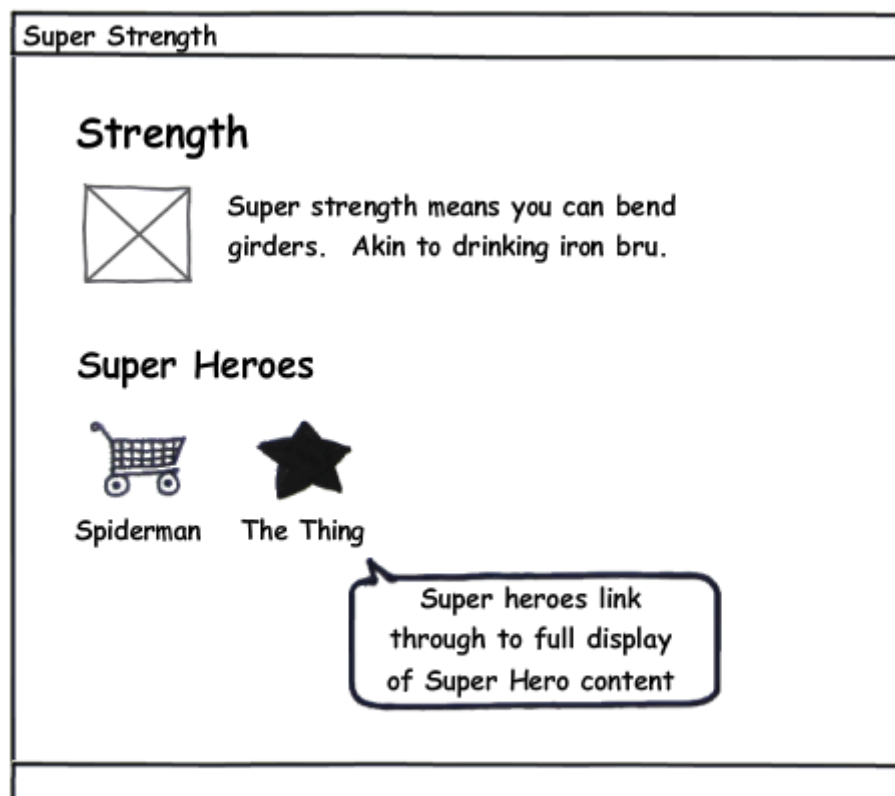
Super Groups



Super Heroes



Super Powers



Type Webskins (aka List Views)

A "type" or "list" webskin is a special type of view that is bound to the content type and not a specific content item. They're often used to list multiple objects of a specific type. For example, the listing of Super Groups in the next walkthrough.

Type webskins should be prefixed with `displayType`. Any webskin with this prefix will be available to attach to a navigation folder in the site overview tree.



Type webskins do not have a specific database record associated with them; it's effectively a view on the type itself rather than a content item. However, they still have an `stobj` structure - it's not normally referenced and contains system properties that be useful in more advanced templating.

Building Queries with `getContentObjects()`



There are a couple of dopplegangers for this functionality in core, buried in FourQ and the database gateways. Both are awkward, and offer little more than arguments for the where and order by clauses. All are now deprecated and replaced with `getContentObjects()` from FarCry 6.0.1.

Of course you can always write your own SQL statement if you have to, but rather than building a query in your view or writing a specific function for your content type, consider using the mighty `getContentObjects()` – it's awesome.

`application.fapi.getContentObjects()` massively improves on older methods by:

- using dynamic arguments for filtering
- handling filters on date columns that can have null values
- automatically filtering results by their status
- using `cfqueryparam` for all filter values

The arguments are:

- `typename` (req)
- `IProperties` (default=`objectid`)
- `status` (default=`request.mode.IValidStatus`)
- `orderBy` (default=`unordered`)
- property filters*

Property filters are arguments in the form `propertyname_comparison=value`. Supported comparisons are:

- `eq`, `neq` (equality filters)
- `in`, `notin` (list filters)
- `like` (standard DB like comparison)
- `isnull` (boolean value to specify whether to return null properties or not)
- `gt`, `gte`, `lt`, `lte` (standard comparisons, null dates always pass these filters)

Examples `getContentObjects()` in Action

```
qLatestNews =
application.fapi.getContentObjects( typename="dmNews", lProperties="objectid,title",publishDate_lt=n
desc" )
qLockedHTML = application.fapi.getContentObjects( typename="dmHTML", locked_eq=1 )
qNotActiveUsers =
application.fapi.getContentObjects( typename="dmUser", userstatus_in="inactive,pending" )
```



application.fapi

`application.fapi` has a bunch of great short cuts to more complex areas of the FarCry framework and APIs. The

FAPI should be the first place you look for doing anything vaguely tricky 😊

Walkthrough: List of Super Groups (typewebskin)

1. Create a new webskin in /webskins/superGroup and call it "displayTypeBody"
2. Update the `displayname` with "Super Group List"
3. Generate a listing of Super Groups, using something similar to the code below:

```

./myproject/webskins/superGroup/displayTypeBody.cfm

<cfsetting enablecfoutputonly="true" />
<!--- @@displayname: Super Group List --->

<!--- tag libraries --->
<cfimport taglib="/farcry/core/tags/webskin/" prefix="skin" />

<!--- get query of super groups --->
<cfset stlocal.qGroups =
application.fapi.getContentObjects(typename="superGroup",orderBy="title")

<cfoutput><h1>Super Groups</h1></cfoutput>

<cfloop query="stlocal.qGroups">
  <skin:view typename="superGroup" objectid="#stlocal.qGroups.objectid#"
webskin="displayTeaserStandard" />
</cfloop>

<cfsetting enablecfoutputonly="false" />

```

4. Build out a `displayTeaserStandard` (called from the listing above) for Super Groups to show in this list:

```

<cfsetting enablecfoutputonly="true" />
<!--- @@displayname: Super Group Teaser --->

<!--- tag libraries --->
<cfimport taglib="/farcry/core/tags/webskin/" prefix="skin" />

<cfoutput>
<div class="featurebox">
  <div class="thumbnailLeft">
    <skin:buildLink objectid="#stobj.objectid#">
      
    </skin:buildLink>
  </div>

  <br class="clear" />
</div>
</cfoutput>

<cfsetting enablecfoutputonly="false" />

```

5. Reload the COAPI Metadata



You can reference the type skin or list view on the URL with the following convention:
<http://localhost:8888/index.cfm?type=superGroup&view=displayTypeBody>

6. Login to the webtop and open the Site overview tree; we're going to hook our type view into the site menu structure
7. Locate the "Hero Hotline" navigation folder, and delete the "Hero Hotline" HTML Page ****NOT THE NAVIGATION FOLDER.... THE HTML PAGE ****
8. Create a new Include item: right mouse click on the Navigation folder or select Create Include from the right hand menu accordion
9. Fill in the fields for the Include, calling it "Hero Hotline"
10. Under OPTION 1, select the type "Super Group" and the type webskin "Display Type Listing"
11. Preview this page in your site

Related Content

Related content are other content items that are related to the object in question by way of array or UUID properties.

There are many ways of gathering this information. This is simplified by way of the `<skin:relatedContent />` tag. Throw it the `objectId`, the filtering `typename` and the `webskin` you want to render on each related object and you're done.

relatedContent tag

```
<!-- @@displayname: Related Content Tag -->
<cfimport taglib="/farcry/core/tags/webskin/" prefix="skin" />
<skin:relatedContent objectId="#stobj.objectid#" filter="superHero"
webskin="displayTeaserStandard" renderType="unordered" />
```

Walkthrough: displayBody View (displayBody.cfm) of Super Groups



Overriding A View

You can override a view in your project by creating a view of the exact same name. Reload the application, and the view should be replaced with the template you have created inside your project. For example, `displayLabel` is a view defined in the core framework for all content types. To override this for a content type called `superGroup` in your project you would create a template called `./webskin/superGroup/displayLabel.cfm`.

1. Build out a full page display for a Super Group by creating a `displayBody.cfm` (this will become embedded in the common `displayPageStandard.cfm` template located in `./webskin/types`)
 - Output the title and the description fields
 - Put the headquarters image somewhere on the page too
2. Your code might look something like this:

./myproject/webskins/superGroup/displayBody.cfm

```
<cfsetting enablecfoutputonly="true" />
<cfoutput>
<h1>#stobj.title#</h1>

#stobj.description#


</cfoutput>

<cfsetting enablecfoutputonly="false" />
```

3. Using `<skin:relatedContent />` tag, list the Super Heroes related to this Group at the end of the page:

```
<!-- import tag library -->
<cfimport taglib="/farcry/core/tags/webskin/" prefix="skin" />

<cfoutput><h2>The Heroes</h2></cfoutput>

<skin:relatedContent objectId="#stobj.objectid#" filter="superHero"
webskin="displayLabel" rendertype="unordered" />
```

4. Preview your Page; link to any of the Super Groups from your Super Group listing page
5. Build out a `displayTeaserMugShot` view (`./webskin/superHero/displayTeaserMugShot.cfm`) for Super Hero to use in the Super Group page that just shows the Hero's mugshot with a link to the full Hero display page:

```
<cfsetting enablecfoutputonly="true" />
<!-- @@displayname: Mugshot Teaser for superHero -->

<!-- tag libraries -->
<cfimport taglib="/farcry/core/tags/webskin/" prefix="skin">

<skin:buildLink objectID="#stobj.objectid#">
  <cfoutput></cfoutput>
</skin:buildLink>

<cfsetting enablecfoutputonly="false" />
```

6. Update the `<skin:relatedContent />` tag to use your new `displayTeaserMugShot` webskin

Lab: displayBody View (displayBody.cfm) of Super Hero

Based on what was done for Super Groups we want to do something very similar here.

- Build out a full page display for Super Hero using the displayBody view: create a webskin ./webskin/superHero/displayBody.cfm (this will become embedded inside the common displayPageStandard.cfm layout template located in ./webskin/types)
 - Output details of the Super Hero including their mugshot, secret hideout, biography, etc.
 - Using <skin:relatedContent />, list their:
 - Super Groups
 - Super Powers
 - Use a simple <skin:view .. /> call to display their Side Kick (calling their displayTeaserMugShot webskin), if they have one



Check the last step of the lab for some hints on building your webskin.

- Build out a displayTeaserIcon webskin (./webskin/superPower/displayTeaserStandard.cfm) for the Super Power that displays the imgPower property in an image tag () and use that webskin on the displayBody view of the superHero:

./myproject/webskin/superPower/displayTeaserStandard.cfm

```
<cfsetting enablecfoutputonly="true" />
<!--- @@displayname: Power Teaser --->

<cfimport taglib="/farcry/core/tags/webskin" prefix="skin" />

<skin:buildLink objectid="#stobj.objectid#">
  <cfoutput><br /></cfoutput>
</skin:buildLink>

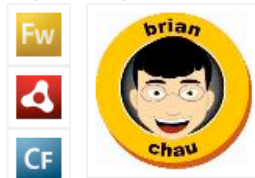
<cfsetting enablecfoutputonly="false" />
```

- Your page should look something like:

Brian Chau

Secret Hideout: Melbourne, Australia

Super Groups: [WEBDU](#) [Adobe](#)



Brian Chau is the Web Market Development Manager for Adobe Asia Pacific. He has been working for Adobe/Macromedia for over 10 years and is an Adobe Certified Expert in Dreamweaver and Flash.

Side Kick



- Your final code might look something like this:

./myproject/webskin/superHero/displayBody.cfm

```

<cfsetting enablecfoutputonly="true">
<!--- @@displayname: Hero Body --->
<cfoutput>
<h1>#stObj.title#</h1>

<div><b>Secret Hideout</b>: #stobj.secretHideout#</div>
<div>
  <b>Super Groups</b>:
  <skin:relatedContent objectid="#stobj.objectid#" filter="superGroup"
webskin="displayLabel" rendertype="none" />
</div>
<div class="thumbnailLeft">
  <skin:relatedContent objectid="#stobj.objectid#" filter="superPower"
webskin="displayTeaserIcon" />
</div>
<div class="thumbnailLeft">
  
</div>
<br class="clear" />

#stobj.biography#

<cfif len(stobj.sidekickID)>
  <h2>Side Kick</h2>
  <skin:view typename="superHero" objectid="#stobj.sidekickID#"
webskin="displayTeaserMugshot" />
</cfif>

</cfoutput>
<cfsetting enablecfoutputonly="false">

```

Walkthrough: displayBody View (displayBody.cfm) of Super Power

1. Build out a displayBody view (./webskin/superPower/displayBody.cfm) of the Super Power
2. Use <skin:relatedContent /> to work out what Super Heroes possess this Power, and provide links back to the Super Heroes using the displayTeaserMugshot view for superHero:

./myproject/webskin/superPower/displayBody.cfm

```

<cfsetting enablecfoutputonly="true">
<cfimport taglib="/farcry/core/tags/webskin" prefix="skin" />

<cfoutput>
<h1>#stObj.title#</h1>

  <p>#stobj.description#</p>

  <h2>Heroes with this Power</h2>
  <skin:relatedContent objectid="#stobj.objectid#" filter="superHero"
webskin="displayTeaserMugshot" />
</cfoutput>
<cfsetting enablecfoutputonly="false">

```

Lab: Super Hero

Based on what was done to link back to Super Heroes from Powers, try building a similar feature linking Super Heroes to the Groups they might belong to.

1. Update the displayBody view (./webskin/superHero/displayBody.cfm) of the Super Hero content type
2. Use <skin:relatedContent /> to work out what Super Groups the Hero belongs to, and provide links back to the Super Groups by overriding the displayLabel for superGroup in the correct webskin folder

Bonus Lab: Other Things To Try

1. Try your hand at the `skin:pagination` tag; a marvel to behold
2. Add a `skin:breadcrumb` and discuss options for "homing" your content types with nav aliases

UNIT 08 - ORM, Object Broker, View Caching

Objectives

By the end of this unit you will be able to apply caching to various aspects of your applications to dramatically increase performance.

Object Broker

The Object Broker is a super piece of caching technology for the integrated ORM of FarCry Core. In simple terms it speeds up the access to data in the database by only retrieving the data once, then using in-memory storage to provide fast access for subsequent requests. You really want to turn this on. The goodness provided is only limited by the the memory available to your ColdFusion instance.

The Object Broker manages the population and refreshing of its cache. You don't really need to do anything more than activate the service for the specific content types you desire.

In addition to specific database calls, the Object Broker can also manage caching of all views. Remember a view or webskin is like a fragment of output, such as HTML. Generally a page request is made up of one or more views. The Object Broker's webskin cache will keep track of embedded views, flushing all the related views whenever a relevant content item is changed. This ensures that your changes are directly reflected in the view without having to be involved in any complex cache management.



The **objectbroker** cache is stored per application. In clustered solutions its possible for application instances to get out of synch. Daemon has a commercial plugin for the FarCry Platform to help provide high availability in clustered solutions: [FarCry HA Plugin](#).

Activating Object Broker

The Object Broker is activated by adding component level metadata to your content type.

```
<cfcomponent extends="farcry.core.packages.types.versions" displayname="Article"
    hint="Standard article used in the site."
    bObjectBroker="true"
    objectBrokerMaxObjects="10000">
```

bObjectBroker

Setting this to `true` activates object-level caching for the specific content type. The default if you do not specify this attribute is `false`.

objectBrokerMaxObjects

The maximum number of objects to be held in the Broker for this content type. The default if you do not specify this attribute is 100. Typically you want to set this to a number that is high enough to hold all the records for this content type. The only reason not to is if you lack enough physical memory on the ColdFusion instance to accommodate them.

Object Broker Report

You can check which content types have been activated for the Object Broker by running the Object Broker Report under the webtop Admin tab, Cache Management menu. This should indicate those content types using the Object Broker, their maximum threshold and the current number of objects in the Broker.

Walkthrough: Activating the Object Broker for Content Types

Let's activate the Broker for the three content types we created earlier; `superHero`, `superGroup` and `superPower`.

1. Open your Super Group type (`./packages/types/superGroup.cfc`) and update the component metadata

```
<cfcomponent
    name="superGroup" extends="farcry.core.packages.types.types" output="false"
    displayname="Super Group"
    bObjectBroker="true" objectBrokerMaxObjects="1000">
```

2. Open your Super Hero type (`./packages/types/superHero.cfc`) and update the component metadata

```
<cfcomponent
  name="superHero" extends="farcry.core.packages.types.types" output="false"
  displayname="Super Hero"
  bObjectBroker="true" objectBrokerMaxObjects="1000">
```

3. Open your Super Power type (./packages/types/superPower.cfc) and update the component metadata

```
<cfcomponent
  name="superPower" extends="farcry.core.packages.types.types" output="false"
  displayname="Super Power"
  bObjectBroker="true" objectBrokerMaxObjects="1000">
```

4. Re-initialize the application to get your component changes registered in the application (ie. Update App!)
5. Wander about the website clicking on a few pages; this will activate the Object Broker and start caching objects behind the scenes
6. Open up the webtop Admin tab and review the Object Broker Report

View Caching

You can also cache the view layer very easily, by specifying a caching directive in the webskin's decorator.



lObjectBrokerWebskins is Deprecated

Do not use `lObjectBrokerWebskins` in your content type component. This feature has been deprecated and should not be used. If you see it in your `cfcomponent` metadata you should remove it. Use the **webskin decorator** instead.

Caching From the View

5.1 introduced the ability to nominate metadata within a specific webskin or view to control webskin caching.

- `@@cacheStatus`: 1 caching, 0 default, -1 nothing can cache
- `@@cacheTimeout`: (in minutes)

A `@@cacheStatus` is always required if you want to nominate a `@@cacheTimeout` value.



Setting a `@@cacheStatus` of "-1" will ensure that the template is never cached. Be aware that this may have a significant impact on your caching strategy as all views that enclose a webskin with a negative `@@cacheStatus` will in turn not be cached.

Example of Caching for 15 minutes

```
<cfsetting enablecfoutputonly="true" />
<!--- @@displayname: Mugshot Teaser for Super Hero --->
<!--- @@cacheStatus: 1 --->
<!--- @@cacheTimeout: 15 --->

<!--- tag libraries --->
<cfimport taglib="/farcry/core/tags/webskin/" prefix="skin">

<skin:buildLink objectID="#stobj.objectid#">
  <cfoutput></cfoutput>
</skin:buildLink>

<cfsetting enablecfoutputonly="false" />
```


Example of Enforcing No Caching

```

<cfsetting enablecfoutputonly="true" />
<!--- @@displayname: Mugshot Teaser for Super Hero --->
<!--- @@cacheStatus: -1 --->

<!--- tag libraries --->
<cfimport taglib="/farcry/core/tags/webskin/" prefix="skin">

<skin:buildLink objectID="#stobj.objectid#">
  <cfoutput></cfoutput>
</skin:buildLink>

<cfsetting enablecfoutputonly="false" />

```

Environment Changes

In some cases you need the view to respond to changes on the URL, form post, client session, security role or other environmental factor. For example, in the case of a paginated result set you may be changing the displayed list of teasers based on a URL parameter such as "&pg=2". This is often awkward for caching regimes as you really need to use a hash of the query string in order to be sure you are looking at the right cache.

There are two specific options for managing environmental cues for caching:

- **@@cacheByVars**: used for any arbitrary variable you may wish to key the cache by. To correctly set the cache, you need to specify the full variable name. For example, if you want to cache according to a url parameter (such as 'page') you need to set **@@cacheByVars: url.page**
- **@@cacheByRoles**: used to key webskin caches by security roles (to allow for different views dependent on different privileged access levels to be cached in the presentation tier) **@@cacheByRoles** is simply a boolean, so set it to 0 or 1.

**cacheByURL, cacheByForm, hashURL Deprecated**

The following inline template metadata options have been deprecated as of 5.1: **@@cacheByURL**, **@@cacheByForm**, **@@hashURL**. These are all replaced by the **@@cacheByVars** metadata option.

Walkthrough: Add Caching to Super Hero



Don't forget to give both **webskin tracer** and **farcry profiler** a workout. These developer tools are especially helpful when working with caching. See *Unit 4* for more information.

1. Run some pages in the sample Fandango site and review the debugging output with your instructor using the Profiler; we're looking for views that might be beneficial to cache
2. Open the `./packages/types/superHero.cfc` for editing
3. Modify the opening `<cfcomponent>` tag to include references to the Object Broker caching engine

```
<cfcomponent bObjectBroker="true">
```

4. Add the **@@cacheStatus: 1** and **@@cacheTimeout: 15** webskin decorators to the views you want to cache
5. Restart the application to activate the changes to the component metadata you have made
6. Re-run the pages in your web site and review your findings with your instructor

Lab: Add Object Broker Caching To Super Group and Super Power

1. Open up `superGroup` and `superPower` content types for editing
2. Activate caching for all relevant views

Bonus Lab: @@cachetypewatch

1. Add a cache "type watch" to your listing pages to immediately capture changes to lists of a certain content type



For a list of all the available webskin decorators review:

<https://farcry.jira.com/wiki/display/FCDEV60/Summary+of+View+Decorators>

UNIT 09 - The Webtop

Objectives

By the end of this unit you will be able to modify the webtop tabs, sub-sections and menus.

Custom Admin

The webtop administration area in FarCry is fully configurable. This includes the addition of your own functionality and the modification or removal of existing areas of the webtop. Within your project you will find a sub-directory called **projectName/customadmin** and all code pertaining to modifying the webtop for your application should be stored there.

Object Admin

The "object admin" is a sophisticated grid for listing and providing hooks for maintenance tasks on any content type. It is fully configurable, and is a workhorse for providing administration in the FarCry framework.

If you've used the scaffold utility to build an administration screen for your custom content type you will find a subdirectory called **projectName/customadmin/customlists** complete with a simple ColdFusion template referencing the object admin custom tag:

Sample Object Admin Scaffold
<pre> <cfsetting enablecfoutputonly="true"> <cfimport taglib="/farcry/core/tags/formtools" prefix="ft" /> <cfimport taglib="/farcry/core/tags/admin/" prefix="admin" /> <!-- set up page header --> <admin:header title="Case Studies" /> <ft:objectAdmin title="Case Studies" typeName="dmCaseStudy" plugin="daeBase" columnList="title,urlWebSite,datetimelastupdated" sortableColumns=" " lFilterFields=" " sqlOrderBy=" " /> <admin:footer /> <cfsetting enablecfoutputonly="no"> </pre>

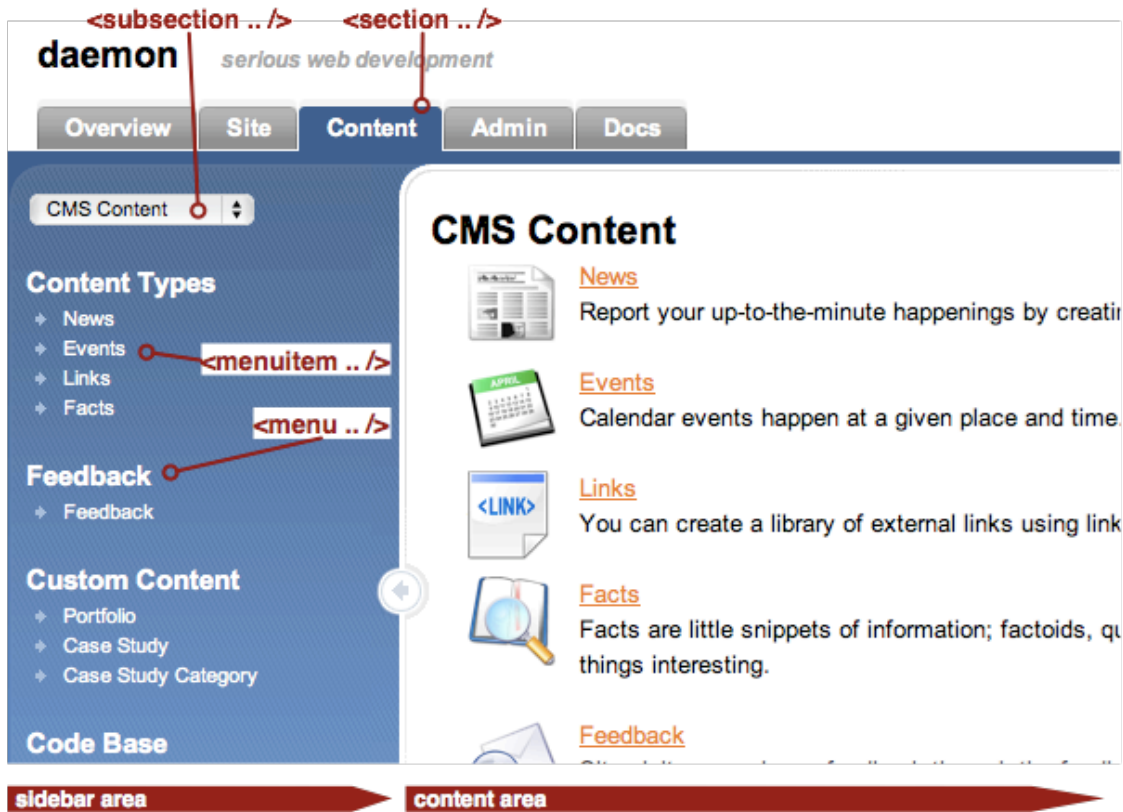
You can create as many Object Admin screens as needed with different attributes to easily provide different types of admin options for users.

Walkthrough: Object Admin Customisation

1. Open all the **projectName/customlists** templates created by the scaffold utility for editing
2. Update the attributes to provide more appropriate column lists, and filterable fields

Webtop XML

FarCry webtop tabs (sections), drop downs (sub-sections) and menu items are managed through the webtop.xml document. You can supplement this XML file, or even override sections of the document simply by adding an appropriately formatted customadmin.xml document to your project's **projectName/customadmin** folder. FarCry merges webtop.xml and customadmin.xml and uses the resulting xml document object to build the UI for the webtop. The XML document is also used to nominate all the relevant permissions for access to areas of the webtop.



Multiple Webtop Configs

In fact the framework will automatically merge all suitable webtop configuration xml files it finds. So you can have multiple files in your **projectName/customadmin** folder if needed. The scaffold utility will produce multiple files by default. Configs will also be merged from active plugins.

Webtop XML Nodes

- The root node is 'webtop'
- Webtop has 'section' nodes as children
- Section has 'subsection' nodes as children
- Subsection has 'menu' nodes as children
- Menu has 'menuitem' nodes as children

Sample Webtop XML Config File

```
<?xml version="1.0" encoding="utf-8"?>

<webtop>
  <section mergeType="merge" id="content">
    <subsection mergeType="merge" id="farcrycmsSubSection">
      <menu id="farcrycorecontent" label="Code Base Management">
        <menuitem sequence="1" id="codebase" label="Code Base"
link="/admin/customadmin.cfm?module=customlists/coreCodeBase.cfm&plugin=daeBase" />
        <menuitem sequence="2" id="codebranch" label="Code Branch"
link="/admin/customadmin.cfm?module=customlists/coreCodeBranch.cfm&plugin=daeBase" />
        <menuitem sequence="3" id="release" label="Code Release"
link="/admin/customadmin.cfm?module=customlists/coreRelease.cfm&plugin=daeBase" />
      </menu>
    </subsection>
  </section>
</webtop>
```

Webtop

<webtop> has no attributes, except an optional mergeType (see Modifying Core Admin section below).

Common Node Attributes

(Not including <webtop>; see **Webtop** above)

Attribute	Purpose
id	Uniquely identifies the node
permission	FarCry permission required for the user to be able to access the screen element that results from the node
label	Text to display on the resulting screen element; this can be either straight text, or it could be a ColdFusion expression/variable to evaluate (see <code>labelType</code>)
labelType	Possible values are 'text' (default), 'expression', and 'evaluate'; if set to 'expression' or 'evaluate' the value of the <code>label</code> attribute is passed to the ColdFusion Evaluate function, and the result is displayed
mergeType	See Modifying Core Admin below

Attributes of the <subsection> Node

Attribute	Purpose
sidebar	Path to a page to use as the left sidebar; the path is relative to the FarCry admin section root (e.g. <code>sidebar="custom/sidebar.cfm"</code> ; <code>sidebar="admin/customadmin.cfm?module=...."</code>)
content	Path to a page to use as the content pane (right side); the path is relative to the FarCry admin section root (e.g. <code>content="inc/content.html"</code> ; <code>content="admin/customadmin.cfm?module=...."</code>)

Attributes of the <menuitem> Node

Attribute	Purpose
link	Path the link carries the user to (in the content pane); the path is relative to the FarCry admin section root (e.g. <code>link="content/dmimage.cfm"</code> ; <code>link="admin/customadmin.cfm?module=...."</code>)
sequence	The order in which the menu items should be displayed.

Custom Admin

Creating custom admin sections is very easy. The `customadmin.xml` file in ***projectName/customadmin*** has the same format as the core's `webtop.xml` file (nodes and attributes described above).



Config File Names

You can give your webtop config file any name (must not include spaces and must have the `.xml` extension) and you can have multiple files. Essentially any file ending in `.xml` found in the ***projectName/customadmin*** folder will be processed; `"customadmin.xml"` is just a common name.

To Add a Custom Admin Section:

Add a <section> node to your `customadmin.xml` with all the appropriate <subsection>, <menu>, <menuitem> nodes within.

Modifying Core Admin

The existing core admin sections, subsections, menus, and/or menuitems can be modified, extended, or replaced.

Replacing a Core Admin Section

To replace an entire core admin section, implement the same node in your `customadmin.xml`. Be sure to include the `id` attribute and make it the same as the existing section in the core. Add a `mergeType="replace"` attribute to your section node.

For example, to replace the core's security section:

Replacing A Section

```
<webtop>
  <section id="security" mergeType="replace" permission="blah" ...>
    <!-- implement whatever subsections/menus/menuitems -->
    <!-- you would like to in here -->
  </section>
</webtop>
```

Replacing a Core Admin Subsection/Menu/Menuitem

Replacing a subsection, menu, or menuitem is very similar. You will have to include all parent nodes of the node you are replacing, and their `id` attributes must match the corresponding nodes in the core's `webtop.xml` file.

For example, to replace the core's 'User' menu of the 'User Management' subsection of the 'Security' section:

Replacing A Menu

```
<webtop>
  <section id="security">
    <subsection id="user">
      <menu id="user" mergeType="replace" label="-">
        <!-- implement whatever menuitems -->
        <!-- you would like to in here -->
      </menu>
    </subsection>
  </section>
</webtop>
```



Replacing Config

When you do a replace, you must supply all of the normally required attributes (such as `sidebar`, `content`, `label`) of the node you are replacing.

Merging with a Core Admin Section

To merge a node means to supplement the existing core node(s) with your own.

A duplicate node is one that exists in the core and also exists in `customadmin.xml`, with the same value for the `id` attribute (same goes for the parents of this node). This is no different than the case above where we had to include the parents with correct `id`'s to replace a certain child node.

When a duplicate node is encountered while merging the webtops, `mergeType="merge"` is assumed, if no `mergeType` attribute is present.

`mergeType="merge"`

Any attributes in the core's node that are also implemented in the customadmin duplicate will be overwritten by the values in the customadmin duplicate. Attributes of the customadmin duplicate that are NOT in the corresponding core's node are added.

The children of a customadmin duplicate are appended to the corresponding core node's array of children. If a child of a customadmin duplicate does NOT have a corresponding duplicate in the core, it is added as it is. Otherwise, it is merged with the corresponding duplicate in the core following the normal merge rules (based on their `mergeType` attributes).

`mergeType="mergeNoReplace"`

This will behave mostly the same as "merge", except existing core attributes will NOT be overwritten. New attributes will still be appended.

Walkthrough: Webtop Configuration

So far we've generated webtop links by relying on the automatically generated scaffold code. In this walkthrough we will merge these files together into a single configuration file, placing all our administration menus under a single section or tab.

1. Open the all the *.xml files under **projectName/myproject/customadmin** for editing
2. Merge all the XML files into a single file called `customadmin.xml`
3. Tidy up the XML by removing all the extraneous webtop, section and subsection elements

4. Place all menus under a single section called "Super"

```
<webtop>
  <section id="super" label="Super">
    <subsection id="SuperSubSection" label="Super Section">
      <!-- your menus go here! -->
    </subsection>
  </section>
</webtop>
```

Bonus Lab: Webtop Customisations

In addition to the tabs and menus, its also possible to build custom views for the webtop overview tabs and custom cell renderers and actions for the content grids.

1. Discuss creating a simple cell view for the grid
2. Discuss creating a custom webtop overview for our custom content types; just enough to bring it up to par with something like dmNews

UNIT 10 - Building Forms

Objectives

After completing this unit you will be able to build your own FarCry forms to edit and save content objects.

Formtool Tag Library

Up until this point, all of our forms have been generated for us by the system. But what if we need more flexibility? We must be able to create our own edit forms.

FarCry has its own set of tags enabling you to build forms to add, edit and save content yourselves. This tag library is located at **/farcry/core/tags/formtools** and is imported into your webskin using:

```
<cfimport taglib="/farcry/core/tags/formtools" prefix="ft" />
```

We are going to look at this unit in 2 stages:

1. Building the Form
2. Processing the Form



Wizard Tools

A similar custom tag library exists for building Wizards, but these are not covered in this unit.

Building the Form

The first stage is to simply build the form. The following code is all we need to do:

```
<ft:form>
    <ft:object objectid="#stobj.objectid#" lFields="" />

    <ft:button value="Save" />
    <ft:button value="Cancel" validate="false" />
</ft:form>
```

You will see 3 simple tags:

<ft:form />

This tag virtually replaces the html <form> tag. Behind the scenes, however, it is setting up all the necessary plumbing so that FarCry knows what object you are saving, hooks for client-side and server-side validation and so on.

<ft:object />

This tag actually renders each of the requested properties in edit mode. All the properties passed through in the `lFields` attribute will be available for editing. If you want to edit ALL fields, you can leave this attribute blank.

This tag also has a number of attributes that you can use to affect the output. Most interesting is the `legend` attribute. This will allow you to create your own fieldsets.

<ft:button />

This tag will provide the buttons to perform actions on the form, essentially replacing the standard `<input type="submit" />` html markup.

Source of a Sample Form

Below you will see an example of the generated html markup that the simple form tags we used above generates. Although it may look a bit intimidating, there is a method to the madness 😊. You do not need to understand the output at all to use formtools, but curious minds should feel free to quiz the instructor.


```

<form
  action="/webtop/conjuror/invoke.cfm?objectid=BC139D3D-C12A-A68F-5AD1A9660401F83A"
  method="post"
  id="farcryForm590063094"
  name="farcryForm590063094"
  enctype="multipart/form-data"
  onsubmit=""
  class="formtool"
  style="">

  <input type="hidden" name="FarcryFormPrefixes" value="BC139D3DC12AA68F5AD1A9660401F83A">

  <fieldset class="formSection ">

    <legend class="">General Details</legend>

    <div class="fieldSection string ">
      <label for="BC139D3DC12AA68F5AD1A9660401F83Atitle" class="fieldsectionlabel ">Title</label>
      <div class="fieldAlign">
        <input type="Text"
          name="BC139D3DC12AA68F5AD1A9660401F83Atitle"
          id="BC139D3DC12AA68F5AD1A9660401F83Atitle"
          value="Andrew Spaulding" class="" style="">
      </div>
      <br class="clearer">
    </div>

    <div class="fieldSection string ">
      <label for="BC139D3DC12AA68F5AD1A9660401F83Asecrethideout" class="fieldsectionlabel ">Secret
      Hideout</label>
      <div class="fieldAlign">
        <input type="Text"
          name="BC139D3DC12AA68F5AD1A9660401F83Asecrethideout"
          id="BC139D3DC12AA68F5AD1A9660401F83Asecrethideout"
          value="Sydney, Australia" class="" style="">
      </div>
      <br class="clearer">
    </div>

  </fieldset>

  <input type="hidden" name="BC139D3DC12AA68F5AD1A9660401F83AObjectID"
  value="BC139D3D-C12A-A68F-5AD1A9660401F83A">
  <input type="hidden" name="BC139D3DC12AA68F5AD1A9660401F83ATypename"
  value="superhero">
  <input type="hidden" name="FarcryFormPrefixes"
  value="BC139D3DC12AA68F5AD1A9660401F83A">

  <span id="f-btn-BE7AC5F1-ED26-9DBF-49B46B4A5036804A-wrap">
<button id="f-btn-BE7AC5F1-ED26-9DBF-49B46B4A5036804A"
  name="FarcryFormsubmitButton=Save"
  type="submit" value="Save" class="f-btn-text">Save</button>
  </span>
  <span id="f-btn-BE7AC5FA-9506-5268-D038424D119EAD9A-wrap">
<button id="f-btn-BE7AC5FA-9506-5268-D038424D119EAD9A"
  name="FarcryFormsubmitButton=Cancel"
  type="submit" value="Cancel" class="f-btn-text">Cancel</button>
  </span>

  <input type="hidden" name="FarcryFormPrefixes" value="">
  <input type="hidden" name="FarcryFormSubmitButton" id="FarcryFormSubmitButton" value="">
  <input type="hidden" name="FarcryFormSubmitButtonClickedfarcryForm590063094"
  id="FarcryFormSubmitButtonClickedfarcryForm590063094" class="fc-button-clicked" value="">
  <input type="hidden" name="FarcryFormSubmitted" value="farcryForm590063094">
  <input type="hidden" name="SelectedObjectID" class="fc-selected-object-id" value="">
  <input type="hidden" name="farcryFormValidation"
  id="farcryFormValidationfarcryForm590063094" value="1">

</form>

```

Walkthrough: Building a Super Hero Custom Edit Form

We are going to create a simple custom edit form for our Super Hero content type. You will remember that the system will look for a webskin edit.cfm when editing a content type. If it can't find one, it will build the form dynamically based on the metadata. In this walkthrough, we are going to create the edit.cfm to override the default behaviour.

1. Create a new file **projectName/webskin/superHero/edit.cfm**
2. Paste the following code into the new file

```
<cfsetting enablecfoutputonly="true" />
<!--- @@displayname: Super Hero Edit --->

<!--- importing Tag Libraries --->
<cfimport taglib="/farcry/core/tags/formtools" prefix="ft" />

<!--- Render the Edit Form --->
<cfoutput><h1>Super Hero Edit Form</h1></cfoutput>

<ft:form>

    <ft:object objectid="#stobj.objectid#" lFields="" />

    <ft:button value="Save" />
    <ft:button value="Cancel" validate="false" />

</ft:form>

<cfsetting enablecfoutputonly="false" />
```

3. Update your application to register the new edit.cfm view (i.e. Update App!)
4. Go into the webtop and Edit a Super Hero content item
5. Notice your new form: it is almost complete except it has not broken down the fields into fieldsets; unfortunately, we will need to do this manually, but it's not that tough
6. Replace the current <ft:object /> tag with the following:

```
<ft:object objectid="#stobj.objectid#" lFields="title,secretHideout,teaser,biography"
legend="General Details" />
<ft:object objectid="#stobj.objectid#" lFields="imgHero" legend="Imagery" />
```

7. Reload the form and review
8. But what if you don't like the HTML that is rendered by default? In a desperate bid for control, replace the second <ft:object /> tag with:

```
<ft:object objectid="#stobj.objectid#" lFields="imgHero" legend="Imagery"
r_stFields="stMyFieldInfo" />
<cfdump var="#stMyFieldInfo#" />
```

9. Discuss with your instructor what you see and the potential

Form Processing: Dealing with POST Events

So now we have our lovely looking form, what happens when we press the save button...? At the moment NOTHING.

This brings us to the second part of the unit: we need to process the form POST.

```
<!--- Form processing --->
<ft:processForm action="Save">
    <ft:processFormObjects objectid="#stobj.objectid#" />
</ft:processForm>
```

Here we are introduced to 2 new tags:

<ft:processForm />

This tag is used to capture an event raised by a <ft:button />. You will notice in our form we had a <ft:button value="Save" />. If someone was to click on that button, the "Save" action will be raised and will be captured by the <ft:processForm action="save" /> tag. Any code inside of the tag will only be run IF the [Save] button is pressed.

<ft:processFormObjects />

This is where the action is at. This tag does all the heavy lifting... well saving anyway. This tag says to the framework: if any fields belonging to an object with the objectid of "#stobj.objectid#" has been submitted by a FarCry form, then save the values of those fields to the object in the database.

So lets put it all together.

Walkthrough: Saving the Super Hero

1. Paste the following code below the <cfimport /> tag but above our form:

```
<!-- Form processing -->
<ft:processForm action="Save">
  <ft:processFormObjects objectid="#stobj.objectid#" />
</ft:processForm>
```

2. Now go and edit a Super Hero and this time when you press Save, notice that your data has, in fact, been saved
3. Great, but when we hit Save or Cancel, we don't actually just want to keep refreshing the page; we need it to exit out to where we came from or wherever the calling page tells us to go
4. Paste the following code below our current <processForm /> tag

```
<ft:processForm action="Save,Cancel" exit="true" />
```

5. Now go and save your Super Hero and see what happens

Bonus Lab: Building Forms in the Front End

In addition to changing the default behaviour of forms in the webtop, its also possible to build forms using formtools directly in the front end of your website.

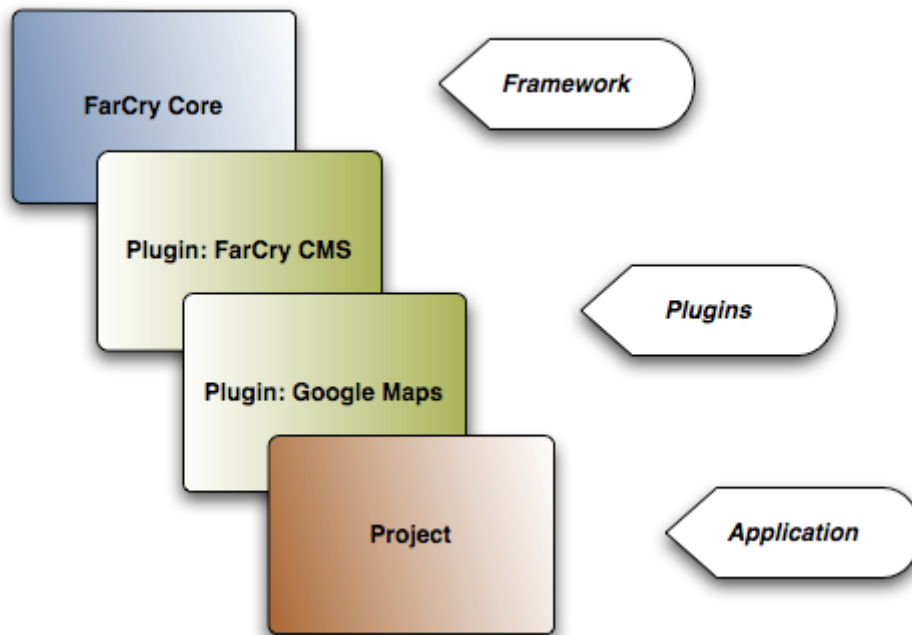
- Build a type webskin listing content items of a particular type
- Put in an "ADD" button and link to a type webskin for adding new content (eg. displayTypeAdd.cfm)
- Create a temporary session object using the "key" attribute so you don't have to create a database record every time someone just views the form
- Call a custom edit handler to update the new record
- Process the form POST and redirect the user to your type listing page

UNIT 11 - Plugins, Skeletons & Extension

Objective

By the end of this unit, we should have an understanding of how FarCry navigates our project and plugins to find both content type metadata and relevant webskin locations.

Plugins



Every aspect of a project can be repackaged and used in another project as a plugin. This might be the entire project or just a small part of it. Most applications are combinations of several plugins. Community plugins include solutions for Google Maps, blogging, content management, free-text search engines and many, many more.

In essence, any FarCry application is a seamless merger of the core framework, the library of plugins associated with the application and the project itself. The project inherits everything represented in the collection of core, plugins and project code bases and is available in the running application. Importantly, if the same code is present in more than one location, a cascade model comes into play so that developers can change or override the behaviour of inherited functionality.

The order of the cascade is as follows:

- core; the framework itself is loaded first
- plugins; plugins are loaded in the order they are listed
- project; the project code base is the final arbiter of how the application behaves

Aspects of the framework that participate in this cascade include:

- content types
- webskins or views
- webtop configuration
- publishing rules
- formtools (components used to render form elements)
- components (effectively content types without database persistence)

Installing Plugins

If you nominate a plugin during the initial installation of your project, its code and content types should be automatically registered and deployed. However, if you are deploying a plugin to an established project you will need to get your hands just a little bit dirty.

To add a new plugin to your application you need to register the **pluginname** in the plugin list of the project's **farcryConstructor.cfm** file in the project's web root:

```
./www/farcryConstructor.cfm
```

```
<cfset THIS.plugins = "farcrycms,googleMaps,farcryxud" />
```



The Order Is Important!

Be aware that the order of plugins in the list can be very important. The specific order you nominate dictates the order in which the plugins are loaded on application initialisation. Remember CORE is always first, followed by the plugins in the listed order, and finally your project.

After changing the plugin list you always need to refresh the application to register your changes.



Reloading configuration

You can use `updateapp=1` to reload the entire application, or you can log into the webtop and just reload config data:

1. Admin > Developer Utilities > Reload application > Config Settings (under the Miscellaneous section)
2. Click Update Application

Walkthrough: Installing Google Maps Plugin

1. Download the Google Maps Plugin
2. Copy the code base to the plugins folder of your installation
3. Update your `farcryConstructor.cfm` configuration file to include GoogleMaps
4. Re-initialise the application
5. Go to the Admin > Developer Utilities > COAPI Tools > Types and deploy the Google Maps content types
6. Go to the Admin > Developer Utilities > COAPI Tools > Rules and deploy the Google Maps publishing rules
7. Login to the webtop and update the Config for Google Maps with your API key
8. Create a publish a Google Map

Plugins & Content Types

Now that we understand the cascading relationship between FarCry Core, Plugins and a Project, the order of initialisation of the metadata for a particular content type is straightforward.

A good example is the Super Hero content type we created:

1. FarCry will find our content type definition (cfc) in `/farcry/core/projects/superheroes/packages/types/superhero.cfc`
2. It will then add all the property metadata into the application scope (`application.stcoapi.superhero.stprops`)
3. It will then introspect the content type that it extends (in this case it is `/farcry/core/packages/types/types.cfc`)

So it adds all the property metadata it finds here into our **`application.stcoapi.superhero.stprops`**. As a result we have **`application.stcoapi.superhero.stProps`** filled with the combination of all the properties in both those content types. We can see this using the COAPI Scope Dumper located in the webtop Admin > Developer Utilities > Scope Dump.



Scope Dump

If you know the exact structure you are looking for you can type it up the top and click [dump], or browse using the preset scopes listed on the left of the viewer.

Walkthrough: Extending the News Content Type

Using this concept, we are going to extend the `dmNews.cfc` found in the `farcrycms` plugin and add some new properties.

1. Create a new file in **`projectName/packages/types/dmNews.cfc`**
2. Paste the following:

```
<cfcomponent
    extends="farcry.plugins.farcrycms.packages.types.dmNews"
    displayname="News"
    hint="Dynamic news data">
    <cfproperty
        name="author" type="string"
        ftSeq="10" ftFieldset="Publishing Details" ftWizardStep="General Details"
        ftLabel="Author" />
</cfcomponent>
```

3. Notice the value of `extends`: this says we are extending the properties and functionality found in `farcry.plugins.farcrycms.packages.types.dmNews`
4. Deploy our new property
5. Update our COAPI Metadata
6. Create a new News item in the webtop from "Content / CMS Content / News"
7. Notice our new `author` property now located in the correct position in the wizard

Webskin Inheritance

Once all of the types have been initialized in the fashion above, FarCry then finds all the webskins that are relevant for each type.

In the case of the `superHero.cfc` above, FarCry trawls through the folders in a predefined order to find any relevant webskins.

At this point it is important to remember that we have included 2 plugins into our project (`farcrycms` and `farcrygreybox`), so this process will include those plugins in its search to find relevant webskins.

In this case it will look in the following order for webskins for `superhero.cfc`:

```
/farcry/projects/superhero/webskins/superhero
/farcry/projects/superhero/webskins/types
/farcry/plugins/farcrygreybox/webskins/superhero
/farcry/plugins/farcrygreybox/webskins/types
/farcry/plugins/farcrycms/webskins/superhero
/farcry/plugins/farcrycms/webskins/types
/farcry/core/webskins/superhero
/farcry/core/webskins/types
```

Therefore ALL webskins in any of those locations will be available to the `superHero` content type. It is also important to note that only the FIRST webskin of the same name will be used. So, if we have a `displayTeaserStandard` in two of these folders, the first one the process comes across will be the one the project uses.

This allows us to override behaviour defined in core or a plugin and so on.

Lab: Override the `displayPageFull` News Webskin

Go ahead and update the full page display webskin for `dmNews` to include the new `author` property.