ASP.NET Development with Castle

John C. Zablocki

john at zblock.net

Fairfield University

webloyalty.com

http://www.codevoyeur.com

http://www.dllhell.net

Fairfield / Westchester .NET User Group

December 4, 2007

Agenda

- Castle Project Overview
- MonoRail
- ActiveRecord
- Questions

What is the Castle Project?

- *Castle is an open source project for .net that aspires to simplify the development of enterprise and web applications. Offering a set of tools (working together or independently) and integration with others open source projects, Castle helps you get more done with less code and in less time.*

  - As defined by the Castle team

Castle Project History

- Started as a subproject of the Apache Avalon project (reusable component framework for server applications)
- Mid 2003 Castle started as an attempt to build a simple IoC container
- As the scope of Castle went beyond IoC (DynamicProxy), Castle separated from Avalon
- Current version is RC3, final pre-1.0 release
- http://www.castleproject.org

The Castle Projects

MonoRail	ASP.NET MVC Framework based on ActionPack from Ruby on Rails
ActiveRecord	Implementation of Active Record data mapping pattern defined by Martin Fowler
MicroKernel	Lightweight IoC container
Windsor Container	Extends MicroKernel to include common enterprise features
Components	Currently contains support for business object validation, DynamicProxy, etc.
Services	Currently contains support for transaction management and logging

MonoRail Overview

- Model View Controller implementation
- Uses view engines and controller classes instead of WebForms and code behind files
- Enforces separation of concerns (very difficult to include business logic in a view)
- Uses Convention vs. Configuration to facilitate rapid development

Model View Controller

Model	Domain objects containing business and data persistence logic
View	Display of information from the model
Controller	Handles requests (user input), manipulates the model, causes updates on the view

Model View Controller continued

- Separation of Presentation from model
  - Develop different presentations for single reusable model (Web, Windows, Mobil, WS, etc.)
  - Easier to test non-visual model
- Separation of View and Controller
  - In practice, this is a byproduct

MonoRail - Model

- The Model is not implemented explicitly by any MonoRail classes
- ActiveRecord may be used for the Model, but is not required
- More to come on ActiveRecord

MonoRail – View

- MonoRail uses View Engines for displaying model data
- Multiple view engines are available, most popular seem to be NVelocity and Brail (we'll look at the former)

MonoRail - Controller

- Controllers are any classes that directly or indirectly extend MonoRail's Controller class
- Controller base class provides subclasses with access to Request/Response properties and methods (much like Page base class in ASP.NET WebForms)

MonoRail Configuration Basics

- MonoRail's HTTP Handler and HTTP Module need to be configured
  - HTTP Handler is responsible for controller and action invocation
  - HTTP Module manages services (extensions, configuration, etc.)

```
<system.web>
    <httpHandlers>
      <add verb="*" path="*.aspx" type="Castle.MonoRail.Framework.MonoRailHttpHandlerFactory, Castle.MonoRail.Framework" />
    </httpHandlers>
    <httpModules>
      <add name="monorail" type="Castle.MonoRail.Framework.EngineContextModule, Castle.MonoRail.Framework" />
    </httpModules>
  </system.web>
```

MonoRail Configuration - cntd.

- Register the config section handler
- List the assemblies containing controllers
- Set the view engine

```
<configSections>
    <section name="monorail"
             type="Castle.MonoRail.Framework.Configuration.MonoRailSectionHandler, Castle.MonoRail.Framework" />
  </configSections>

<monorail>
    <controllers>
      <assembly>AssemblyName</assembly>
    </controllers>
    <viewEngine customEngine="Castle.MonoRail.Framework.Views.NVelocity.NVelocityViewEngine,
     Castle.MonoRail.Framework.Views.NVelocity" />
  </monorail>
```

MonoRail Request/Response

- MonoRail uses the request path to determine which controller and action to invoke
- Consider the URL http://localhost:49425/Speaker/Register.aspx
- In the controller assemblies MR should find:
  - A Controller subclass named SpeakerController, inferred by the path /Speaker/
  - A method Register on the SpeakerController class, inferred by the action Register.aspx

MR Request/Response cntd.

- Again - http://localhost:49425/Speaker/Register.aspx
- A SpeakerController instance is created and the Register method is invoked

```
public class SpeakerController : Controller
   {
      public void Register()
      {
           ...
      }
   }
```

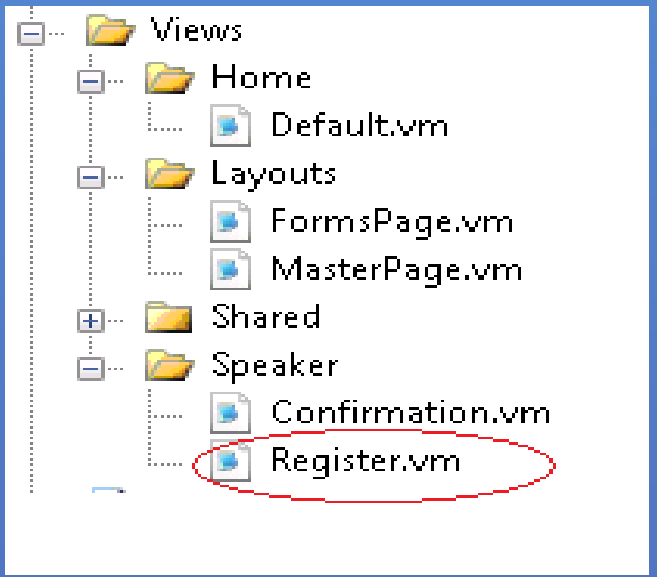
MR Request/Response cntd.

- http://localhost:49425/Meetings/Speaker/Register.aspx
- Areas are used to group related controllers
  - The Meetings path assumes that the attribute below has been added to the class

```
[ControllerDetails(Area="Meetings")]
public class SpeakerController : Controller
   {
      public void Register()
      {
           …
      }
   }
```

MR Request/Response - Views

- http://localhost:49425/Speaker/Register.aspx
- The path is also used to map a view template
- By convention, Views are to be found in a Views directory (under the site root)
- The path (and area if applicable) should map to a directory under Views in which a template matching the action is found



NVelocity

- The Castle team forked NVelocity from an abandoned Apache project
- Port of Apache's Jakarta Velocity project
- NVelocity View Engine uses NVelocity as its template engine
- Uses the Velocity Template Language (VTL) for rendering model data, conditional logic, looping, etc.

NVelocity Layouts

- NVelocity supports a MasterPage like construct called a Layout
- Layouts maybe set declaratively or programatically at the class (controller) or method (action) level
- The requested view is merged into the layout
- The layout may also declare sections for rendering shared widgets defined by the view
- Layouts are stored under the view root in a Layouts folder

NVelocity Layouts

- http://localhost:49425/Speaker/Register.aspx
- The Register.vm view file is merged into the $ChildContent ($ precedes variables in VTL)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Fairfield / Westchester .NET User Group Speaker Registry</title>
</head>
<body>
        <div id="main">
            $ChildContent
        </div>
    </body>
</html>
```

Controllers and Views

- Controllers make model data available to the Views through the PropertyBag
  - The PropertyBag is a Dictionary similar to HttpContext.Current.Items

```
//Controller code
public void List()
{
    PropertyBag["Speakers"] = Speaker.FindAll();
}

<!-- View Code -->
#foreach($speaker in $Speakers)
   #if ($speaker.IsActive)
      <div>$speaker.LastName, $speaker.FirstName</div>
   #end
#end
```

Controllers and Views continued

- Controllers actions may render views other than the default action associated view by using RenderView(“viewname”)
- Controllers may cancel a view altogether using CancelView
- Controllers have a number of methods for redirecting to actions or URLs (Redirect, RedirectToAction, RedirectoReferer)
- Controllers may use the Flash dictionary to make data available to a view after a redirect

Filters

- Filters – classes that implement IFilter – are used to execute code before, after or before and after a controller action executes

```
public class AuthorizationFilter : IFilter
   {
         public bool Perform(ExecuteEnum exec, IRailsEngineContext context, Controller controller)
         {
            if (!context.CurrentUser.IsInRole(RoleConstants.ACTIVE_USERS))
            {
               NameValueCollection parameters = new NameValueCollection();
               parameters.Add("ReturnUrl", context.Url);
               controller.Redirect("Membership", "LoginRequired", parameters);
               return false;
            }
            return true;
         }
   }

[Filter(ExecuteEnum.BeforeAction, typeof(AuthorizationFilter))]
   [Layout("FormPage")]
   public class ProfileController : ControllerBase
   {
      ...
   }
```

View Components

- Extend ViewComponent for resuable UI code
- Built in components (security, pagination, ...)

```
public class MapComponent : ViewComponent
   {
      private string _mapProvider = string.Empty;
      public override void Initialize()
      {
         _mapProvider = ComponentParams["MapProvider"];
      }

      public override void Render()
      {
         RenderText("<script type=\"text/javascript\">");
         RenderText(string.Format("    var mapstraction = new Mapstraction('mapstraction','{0}');", _mapProvider));
         ...
      }
   }

<!-- VTL Usage -->
<div class="map">
    #blockcomponent(MapComponent with "MapProvider=Yahoo")
     #end
</div>
```

Data Binding

- By extending SmartDispatcherController, which in turn extends Controller it is possible to bind request parameters to action arguments automatically
- In the sample below, firstname and lastname are automatically bound to request params

```
<form action="$SiteRoot/Profile/Save.aspx">
    <div>First Name: $FormHelper.TextField("firstname")</div>
    <div>First Name: $FormHelper.TextField("lastname")</div>
    <div><input type="submit" value="Submit" /></div>
</form>

//request is something like http://localhost:49425/User/Save.aspx?firstname=John&lastname=Zablocki
//post would work  same way
public void Save(string firstname, string lastname)
{
    ...
}
```

Data Binding with Objects

- It is possible to data bind reference types with the DataBind attribute
- A prefix is used in form field naming to map an object to its properties
- In the example below, if the PropertyBag contains a “profile” entry with a User object, the form fields are bound on display as well

```
<form action="$SiteRoot/Profile/Save.aspx">
   <div>First Name: $FormHelper.TextField("profile.firstname")</div>
   <div>First Name: $FormHelper.TextField("profile.lastname")</div>
   <div><input type="submit" value="Submit"></div>
</form>

public void Save([DataBind("profile")]User user)
{
    if (user.IsValid)user.Save();
}
```

Some Other MR Features

- Helpers – classes made available to views by controllers (for advanced formatting, etc.)
- Rescues – controller or action level exception handling mechanism (catch all)
- FormHelper – provides support for form field rendering and bidirectional data binding
- CaptureFor – view component for replacing layout variable with content defined in view
- AjaxHelper – along with other helpers, facilitates Scriptaculous integration

ActiveRecord Pattern

- *An object that wraps a row in a database table or view, encapsulates the database access, and adds domain logic on that data*
  - Martin Fowler in PoEAA
- An Active Record class maps fields to the columns in the mapped table
- Contains static finder methods
- Contains instance methods for create, update and delete
- Contains some business logic

ActiveRecord Overview

- Implementation of the AR pattern
- Built on top of NHibernate
  - Port of Java Hibernate ORM
  - Uses XML to map objects to tables
- Encapsulates XML mapping through easy to use attributes

ActiveRecord Configuration

- Register config section handler
- Include required NHibernate configuration
- isWeb=”true” is used to handle behavior of threads in a web application

```
<section name="activerecord" type="Castle.ActiveRecord.Framework.Config.ActiveRecordSectionHandler, Castle.ActiveRecord" />

<activerecord isWeb="true">

  <config>
    <add key="hibernate.connection.driver_class" value="NHibernate.Driver.SqlClientDriver" />
    <add key="hibernate.dialect" value="NHibernate.Dialect.MsSql2005Dialect" />
    <add key="hibernate.connection.provider" value="NHibernate.Connection.DriverConnectionProvider" />
    <add key="hibernate.connection.connection_string" value="Data Source=.\SQLEXPRESS;AttachDbFilename=|
   DataDirectory|\registry.mdf;Integrated Security=True;User Instance=True;" />
  </config>

 </activerecord>
```

ActiveRecord Initialization

- ActiveRecord must be started once and only once for an application
  - Necessary to create the XML mappings

```
protected void Application_Start(object sender, EventArgs e)
        {
            ActiveRecordStarter.Initialize(
                Assembly.Load("SpeakerRegistry"),
                ActiveRecordSectionHandler.Instance);
        }
```

Simple Mapping

- Consider a table products with three columns, product_number, description and manufacturer
  - The class attribute ActiveRecord maps the class to the table
  - The PrimaryKey attribute maps the PK property to the PK column
  - The Property attribute maps simple properties to columns

```
[ActiveRecord("products")]
  public class Product : ActiveRecordBase<Product>
  {
     private int _number;
     private string _description;
      private string _manufacturer;

     [PrimaryKey(PrimaryKeyType.Identity, "product_number")]
     public int Number
     {
        get { return _number; }
        set { _number = value; }
     }

     [Property("description")]
     public string Description
     {
        get { return _description; }
        set { _description = value; }
     }

     [Property("manufacturer")]
     public string  Manufacturer
     {
        get { return _manufacturer; }
        set { _manufacturer = value; }
     }
  }
```

The ActiveRecordBase Class

- Provides support for CRUD
  - Create, Update and Delete are instance methods
  - Numerous Find methods for simple searches

```
Product p0 = new Product();
p0.Description = "Roland Juno-G";
p0.Create();

Product p1 = Product.Find(12345);
p1.Description = "Fender American Standard Telecaster";
p1.Update();

Product[] products0 = Product.FindAllByProperty("Description", "Zoom H-2 Handy Recorder");
Product[] products1 = Product.FindAll();
```

Complex Finds

- SQL expressions containing or, and, like, between, etc. are made available to ActiveRecord through NHibernate's Expression library.

```
Product zoomH2 = Product.FindOne(Expression.Eq("Description", "Zoom H-2 Handy Recorder"));

Product[] rolandAndYamahaProducts = Product.FindAll(Expression.Or(Expression.Eq("Manufacturer", "Roland"),
                                                                  Expression.Eq("Manufacturer", "Yamaha")));

Product[] fenderProducts = Product.FindAll(Expression.InsensitiveLike("Description", "Fender",MatchMode.Anywhere));

Product[] incompleteEntries = Expression.IsNull(Expression.IsNull("Description"));
```

Relations – Many-to-One

- If the products table were updated so that the manufaturer column became a FK to a manufacturers table, the AR model would be changed to use a BelongsTo attribute
  - BelongsTo maps many-to-one relationships
  - Sample assumes a new AR class Manufacturer was created

```
[ActiveRecord("products")]
   public class Product : ActiveRecordBase<Product>
   {
      ...

       private Manufacturer _manufacturer;

      [BelongsTo("manufacturer_id")]
      public Manufacturer Manufacturer
      {
         get { return _manufacturer; }
         set { _manufacturer = value; }
      }
   }
```

Relations One-to-Many

- Consider the relationship from manufacturer to products
  - HasMany attribute maps one manufacturer to its set of products

```
[ActiveRecord("manufacturers")]
  public class Manufacturer: ActiveRecordBase<Manufacturer>
  {
     ...

     private IList _products;

     [HasMany(typeof(Product), Table="products", ColumnKey="manufacturer_id")]
     public IList Products
     {
        get { return _products; }
        set { _products = value; }
     }
  }
```

Relations - Many-to-Many

- Consider a tagging scheme for products that adds two new tables, tags (tag_id, tag_name) and products_tags (product_tag_id, product_id, tag_id)
- Composite key is possible by rolling PK columns into separate key class

```
[ActiveRecord("products_tags")]
   public class ProductTag : ActiveRecordBase<ProductTag>
   {
      private int _id;

      [PrimaryKey(PrimaryKeyType.Identity, "product_tag_id")]
      public int Id
      {
         get { return _id; }
         set { _id = value; }
      }

      private Product _product;

      [BelongsTo("product_id")]
      public Product Product
      {
         get { return _product; }
         set { _product = value; }
      }

      private Tag _tag;

      [BelongsTo("tag_id")]
      public Tag Tag
      {
         get { return _tag; }
         set { _tag = value; }
      }

   }
```

Relations – Many-To-Many cntd.

- The relations from products and tags to products_tags may be mapped using HasAndBelongsToMany attribute
- The Product class would map a Tags collection in a similar way, simply reversing ColumnKey and ColumnRef

```
[ActiveRecord("tags")]
  public class Tag : ActiveRecordBase<Tag>
  {
      ...

      private IList _products;

      [HasAndBelongsToMany(typeof(Product), Table="products_tags",
          ColumnKey="tag_id", ColumnRef="product_id")]
      public IList Products
      {
          get { return _products; }
          set { _products = value; }
      }
  }
```

Other Relations

- One-To-One relations are mapped using the OneToOne attribute
- Consider a table payment_detail with columns payment_method_id and payment_type_id where the payment type determines whether the payment_method_id is a FK to a PAYPAL_ACCOUNT table vs. a CREDIT_CARD table
  - The Any and HasManyToAny attributes are used for mapping these scenarios

Lazy Loading

- Without lazy loading (Lazy=true on ActiveRecord or relation attribute), all relations are loaded at the time a parent object is loaded
- Lazy loading is somewhat complicated
  - Class level lazy loading requires properties to be virtual as NHibernate generates a proxy
  - AR requires lazy loading to occur within a SessionScope

NHibernate Sessions (briefly)

- ISessionFactory – application level factory for managing instances of ISession
- ISession – responsible for opening/closing database connections, monitoring changes to objects, querying and committing changes to the database
- ISessionScope – Castle construct for extending the life of an ISession instance until the ISessionScope instance is disposed

Lazy Load Alternatives

```
private IList _products = null;

public IList Products
{
    get
    {
        if (_products == null)
            _products = Product.FindByManufacturer(_id);
        return _products;
    }
}
```

- Remove collection
  - Instead use static find methods on the collection class
- Implement collection property to call Find

```
[ActiveRecord("products")]
  public class Product : ActiveRecordBase<Product>
  {
      ...
      public static Product[] FindByManufacturer1(int mid) {
          return FindAllByProperty("Manufacturer.Id", mid);
      }

      public static Product[] FindByManufacturer2(int id) {
          string hql = @"select p
                         from Product p
                         join p.Manufacturer m
                         where p.Manufacturer.Id = ?";

          SimpleQuery<Product> query = new
                              SimpleQuery<Product>(hql, id);
          return query.Execute();
      }

      public static Product[] FindByManufacturer3(int mid) {
          DetachedCriteria criteria =
                           DetachedCriteria.For<Product>();
          criteria.Add(Expression.Eq("Manufacturer.Id", mid));

          return FindAll(criteria);
      }
  }
```

Hibernate Query Language (HQL)

- Database agnostic language for querying object model
- Supports joins, aggregate functions and various expressions

```
[ActiveRecord("products_tags")]
   public class ProductTag : ActiveRecordBase<ProductTag>
   {
      ...
      public static Tag[] FindProductTags(int pid)
      {
         string hql = @"select t
                     from ProductTag pt
                     join pt.Product p
                     join pt.Tag t
                     where p.Id = ?";

         SimpleQuery<Tag> query = new SimpleQuery<Tag>(hql, pid);
         return query.Execute();
      }

      public static long GetCountByTag(string tagName)
      {
         string hql = @"select count(p.Id)
                     from ProductTag pt
                     join pt.Product p
                     join pt.Tag t
                     where t.Name = ?";

         ScalarQuery<long> query = new ScalarQuery<long>(
                                          typeof(ProductTag),
                                    hql, tagName);
         return query.Execute();
      }

   }
```

NHibernate Criteria API

- Expressions (see slide “Complex Finds”)
- Projections for aggregation, partial column set queries
- DetachedCriteria for reusable or out of session criteria, working with Projections

```
public static int GetCountByTag(string tagName)
        {
            DetachedCriteria criteria = DetachedCriteria.For<ProductTag>();
            criteria.CreateCriteria("Tag", "t", JoinType.InnerJoin);
            criteria.CreateCriteria("Product", "p", JoinType.InnerJoin);
            criteria.Add(Expression.Eq("t.Name", tagName));

            ProjectionList pList = Projections.ProjectionList().Add(Projections.Count("p.Id"));

            ScalarProjectionQuery<ProductTag,int> query =
                new ScalarProjectionQuery<ProductTag, int>(pList, criteria);

            return query.Execute();

        }
```

ActiveRecord Validation

- Extend ActiveRecordValidationBase
- Use Castle Component (not AR specific) validation attributes for common routines

```
[ActiveRecord("manufacturers")]
   public class Manufacturer: ActiveRecordValidationBase<Manufacturer>
   {
      ...
      private string _name;

      [Property("manufacturer_name")]
      [ValidateNonEmpty("Manufacturer name is required.")]
      [ValidateLength(4, 30, "Manufacturer name must be between 4 and 30 characters.")]
      [ValidateRegExp(@"^[\w\s]*$", "Name may contain numbers, letters and spaces.")]
      public string Name
      {
         get { return _name; }
         set { _name = value; }
      }
      ...
   }
```

ActiveRecord Validation cntd.

- Creates and Updates will fail (exception thrown) if any validation rule fails
- IsValid property used to check for errors
- Each validation failure has its message added to the ValidationErrorMessages collection

```
public void Save()
    {
        Manufacturer m = new Manufacturer();
        m.Name = "Roland";

        if (m.IsValid())
            m.Create();
        else
            foreach (string errorMessage in m.ValidationErrorMessages)
                Console.WriteLine(errorMessage);

    }
```

Transactions

- ActiveRecord supports transactions using TransactionScope

```
public void Save()
{
    using (TransactionScope ts = new TransactionScope())
    {
        try
        {
            Manufacturer m = new Manufacturer();
            m.Name = "Fender";
            m.Create();

            Product p = new Product();
            p.Description = "Tex Mex Stratocaster";
            p.Manufacturer = m;
            p.Create();

            Product p2 = new Product();
            p.Description = "American Standard Telecaster";
            p.Manufacturer = m;
            p.Create();

            ts.VoteCommit();
        }
        catch
        {
            ts.VoteRollBack();
            throw;
        }
    }
}
```

Multiple Databases

- Create a base class that is mapped to a new <config> block under the AR config section
  - Classes extending this base type use the configured DB
- Use DifferentDatabaseScope

```
public void Save()
{
    SqlConnection conn = new SqlConnection();
    using (new DifferentDatabaseScope(conn))
    {
        Manufacturer m = new Manufacturer();
        m.Name = "Fender";
        m.Create();

        Product p = new Product();
        p.Description = "Tex Mex Stratocaster";
        p.Manufacturer = m;
        p.Create();
    }
}
```

Stored Procedures and SQL

- ActiveRecordMediator can be used to get ISession instance from ISessionFactoryHolder instance
- ISession instance exposes IDbConnection instance which can be used to execute arbitrary ADO.NET code

```
ISessionFactoryHolder sessionFactory = ActiveRecordMediator.GetSessionFactoryHolder();
ISession session = sessionFactory.CreateSession(typeof(StoryTag));

IDbConnection conn = session.Connection;
IDbCommand cmd = conn.CreateCommand();

//ADO.NET code to execute SP or arbirary SQL goes here...
```

MonoRail and ActiveRecord

- Enhanced data binding through ARDataBind and ARFetch attributes
  - Used for fetching a row before saving form sumbitted values, enforce validation, etc.
  - Requires ARSmartDispatcherController subclass

```
public class ProductController : ARSmartDispatcherController
   {
      public void Save([ARDataBind("product",
         AutoLoad=AutoLoadBehavior.Always, Validate=true)]Product product)
      {
         ...
      }
   }
```

Scaffolding

- Quick and Dirty CRUD forms
- Extend ARSmartDispatcherController
- Mark controller with Scaffolding attribute
- List, Create, Edit forms are auto-generated

```
[Scaffolding(typeof(Manufacturer))]
    [ControllerDetails(Area="admin")]
    public class ManufacturersController : ARSmartDispatcherController
    {
    }
```

Links

- Castle Project - http://www.castleproject.org
- NHibernate - http://www.nhibernate.org
- Velocity Project - http://velocity.apache.org
- Ayende's Blog - http://www.ayende.com/
- Hammet's Blog - http://hammett.castleproject.org
- Code Voyeur - http://www.codevoyeur.com
  - OK, there's nothing there now, but I'll be putting the slides and samples up along with some Castle tutorials
- dll Hell - http://www.dllhell.net
  - OK, again nothing yet... Future home of my blog where I'll be focusing on OSS, teaching and other tech topics

Other Resources

- Martin Fowler's PoEAA - http://www.bookpool.com/sm/0321127420
- Podcasts on MonoRail and NHibernate-
  - http://www.hanselminutes.com/default.aspx?showID=71
  - http://www.dotnetrocks.com/default.aspx?showNum=224
- The Killers “Sawdust” - http://www.amazon.com/Sawdust-Killers/dp/B000WCDI5K/ref=pd_bbs_sr_1?ie=UTF8&s=music&qid=1196790635&sr=8-1

Questions?