NWTD Implementation Guide

Introduction

This document attempts to outline NWTD's ecommerce application so that future developers can better understand what's been built, and where various functions and business logic implementations take places.

This is not a functional speciation or an API document, rather a reference document for anyone coming up to speed on the project to use for finding valuable information, especially when it comes to customization of ECF for NWTD's purposes.

NWTD ECF Overview

Multi-Site

The site is set up so that the database and file system are shared between both the NWTD site and the MSSD site. Both sites use the same catalog. The places where the sites "split" is in the Themes and in the CMS data itself. The way this is configured is through eh Console Manager.

Console Manager

The console manager is the backend for both the NWTD and MSSD sites. All CMS and commerce functions can be performed in the Console Manager, such as page creation, user modification, site import/export, and so on. Please note that you should use the front end websites for User Creation.

Getting to the Console Manager

The staging version of the console manager is probably all you need to use. It can be accessed using the following information:

- **Location**: http://admin.staging.nwtd.oaktree.com
- **Username**: admin
- **Password**: prAqac3e
- **Application**: ecommerceframework

If you want to run the console manager in a development environment, it's quite easy. Open up the solution, expand **AdminLayer**, Right-click **ConsoleManager**, and choose **View in Browser**. The application will build and use the built-in Visual Studio web browser.

Site Import/Export

Sites can be imported by doing the following:

1. Log into the console manager.
2. In the left menu, click **Content Management**.
3. In the tree that appears under the Content Management section, click **Sites** (but don't click the little plus button to expand it).
4. Click the **More Actions** menu item and choose **Import Site**.
5. Click **Add New File.**
6. Click **Browse**, and browse to the site export file you want to import
7. Click **Upload**.
8. When your file appears in the list below, select it, and click **Start Import**.

Note that you can only import a site if it doesn't already exist.

Sites can be exported by doing the following:

1. Log into the console manager.
2. In the left menu, click **Content Management**.

3. In the tree that appears under the Content Management section, click **Sites** (but don't click the little plus button to expand it).
4. Check the box next to the site you want to export.
5. Click the **More Actions** menu item and choose **Export Site**.
6. Click **Start Export**.

Note, when the export completes, it's a good idea to download the site export, put it in /Documents/SiteExports, and check it into Subversion.

Site Settings

The site settings page configures information about the site such as the site name, url (for multi-site setups) the site theme, and any custom information stored in the site that might need to be accessed from code.

The site settings screen can be accessed in the following way:

1. Log into the console manager.
2. In the left menu, click **Content Management**.
3. In the tree that appears under the Content Management section, click **Sites** (but don't click the little plus button to expand it).
4. In the list of sites that appears on the right, click the name of the site you want to modify site. Don't click either of the icons, click the link.
5. Make your changes to the site and click **OK** to save.

For each NWTD site (NWTD and MSSD), there is some additional info saved in the "Additional Settings" tab that identifies the depository. This information is accessed programmatically during certain business processes.

There are portions of the code that actually look at the title and other attributes of the site to determine actions. For example, the information send to the password recovery email template uses the site title. Also, as part of the login process, we check to see if a user is an NWTD user and that the site's "Depository" attribute is "NWTD" before allowing him or her in. Same thing for MSSD... if a user's assigned depository is MSSD and the site's "Depository" attribute is "MSSD" we let them in. The title and attributes for a site are set in the ECF backend in the content management section.

Setting up the Default Site

If you're working in a development environment and run locally using the built-in visual studio web server, you'll only be able to see one of the two front end sites. The site you'll see when viewing in a browser will depend on which site is set up as the "default site." The change the default site, do the following:

1. Go to the Site Settings page for the site you want to make the default site.
2. Click the **Yes** radio button next to "Is Default."
3. Click **OK**.

4. Go to the Site Settings page for the site you want to make *not* the default site.
5. In the site settings page that appears, click the **No** radio button next to "Is Default."
6. Click **OK**.

User Creation

Because of the Organization Bug, you can't set a user's Business Partner from the backend. It's therefore better to create a user in the front end sign-up form located at ~/profile/register.aspx.

Level A and B Users

A Level A user is technically a user that has been assigned the role of "Level A." This can be done in the Multi-Site

The site is set up so that the database and file system are shared between both the NWTD site and the MSSD site. Both sites use the same catalog. The places where the sites "split" is in the Themes and in the CMS data itself. The way this is configured is through eh Console Manager.

Console Manager by editing an account, checking the box next to the role labeled "Level A," and saving the user.

A Level B user is simply an authenticated user who does not have the Level A role assigned.

Level A users have a somewhat different experience using the site:

- Level A users can access the CSS Order Status sections of the site and will see a different panel on the homepage with links to this section.
- Level A users can submit a cart (whereas non-level A users can only print).
- Level A users cannot enter a shipping address, rather must choose from a list of addresses associated with that user's Business Partner
- Level A users are automatically assigned a billing address.

The bulk of Level A branching is handled in the home page template (which shows different content areas depending on the user) and in the controls that handle the checkout process. There is a static method in the NWTD.Profile class that allows a programmer to check to see if a user is a Level A user, a Level B user, or an Anonymous user.

Page (and Other Types of) Templates

The templates for site pages are located in the front end site's Templates directory. Put page templates in the NWTD/PageTemplates directory.

There are other templates in the Templates/NWTD directory that are used for various groupings of controls (such as the main seach control on the SearchResults.aspx page.

When you edit templates, it is very important to be mindful of CSS selectors, especially class names. Some CSS selectors are used for finding elements using JavaScript. Removing classes from elements could potentially break clientside functionality.

If you want to create a new Page template:

1. Create a new control in the appropriate template directory.
2. Make sure it has a unique class name (i.e. don't simply copy an existing one).
3. Make sure it inherits BaseStoreUserControl, and implements IPublicTemplate.
4. Add a public string property called ControlPlaces that returns a comma-separated list of control ID's that should be user-editable.

Email Templates

Email templates use XSLT to build an email an XML serialized object. Email templates are configured in the web.config file arount line 142 in the XslTemplateProvider property:

```
<templateService defaultProvider="XslTemplateProvider">
    <providers>
        <add
        name="XslTemplateProvider"

type="Mediachase.Commerce.Engine.Template.Providers.XslTemplateProvider,
Mediachase.Commerce"
        applicationName="eCommerceFramework"
        templateSource="d:\Clients\nwtd\PublicLayer\Templates\{0}\{1}.xsl"/>
    </providers>
</templateService>
```

Note that the location of the email templates is a path on the machine. You'll have to configure this for each environment. The email templates are stored in SVN in \PublicLayer\Templates.

Data Migration Between SBO and ECF

Data Import

On a nightly basis NWTD plans to run a SQL script that migrates catalog and organization data from their SBO system into ECF. Along with the data will migrate an updated searchFilters.config and catalog index.

Data Export

NWTD will periodically run a SQL script that harvests Level-A generated orders. This will actually be shopping carts with a submitted status.

NOTE: It might be a good idea to have NWTD mark these “harvested” orders in some way.

Custom Code

This section provides an outline of some of the custom code that has been implemented for NWTD's application.

Development Environment

To get going on development, you need Visual Studio 2010 Professional or higher. The solution to open is eCommerce Framework 5.0 SDK.sln.

This solution contains many class libraries provided with ECF, a few web applications, and some libraries written by OakTree Digital.

The two web applications for this solution are:

- PublicLayer/FrontEnd - this is the front end commerce site
- AdminLayer/ConsoleManager - this is the admin side

There are two additional class libraries that contain custom code:

- Oaktree.Web.UI
- NWTD

ECF Tips

MetaData

ECF's schema can be expanded using its MetaData system. NWTD has added quite a bit of MetaData for both business logic purposes (which is often times required for the codebase to run) and for presentation purposes. For details of some of the more important meta data additions, see Schema.

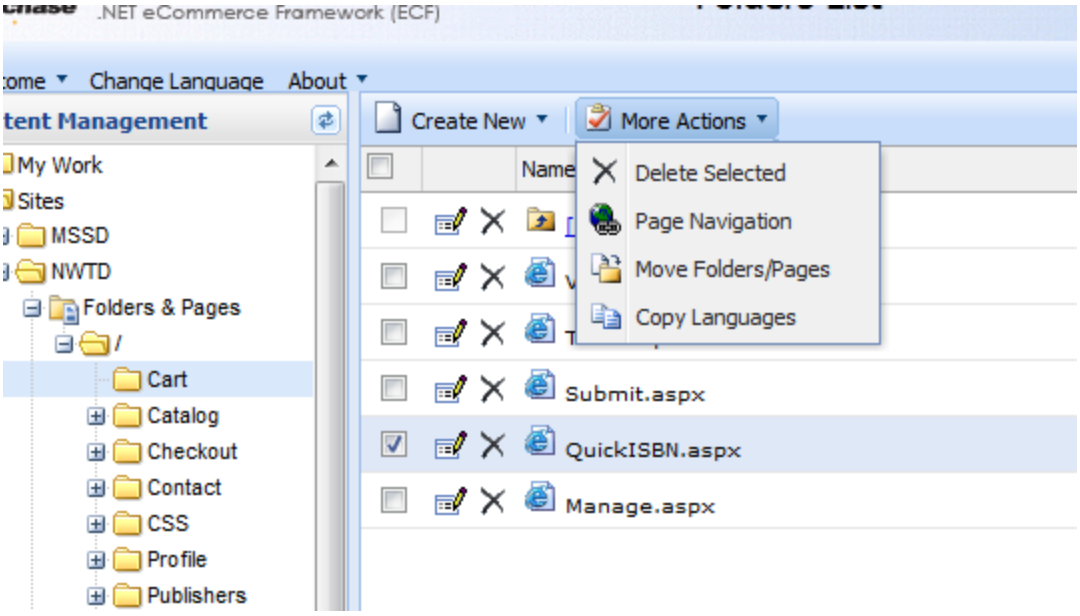
Navigation

Use the NavigationManager.GetUrl method to get fully qualified URLs for navigation from navigation ite names. You can store URLs in the ECF navigation system that expects parameters, and you can programmatically navigate to them using this method as well.

For example, in the Cart.ascx control you'll find the following code:

```
string quickISBNUrl = NavigationManager.GetUrl("QuickISBN", new
object[] { "cart", this.SelectedCartName });
```

If you go to the QuickIsbn page in the Console Manager, check the box next to it, click the **More Actions** menu item, you'll see an option called **Page Navigation**.



Click it and you'll see an entry called **QuickISBN**. This is the first argument for the code above. The second argument is the query string parameters that will be added to the url. This makes it possible to move the page around and still generate links to it.

NWTD Assembly

The NWTD assembly contains a class library that supplies custom code that helps control developers conducts NWTD-specific business logic. This library includes mostly static helper methods, static property getters, enums, and a few overrides of ECF core functionality (such as a search indexer).

For detailed information about the classes in the NWTD assembly, see the generated Code Documentation.

NWTD.Profile

The NWTD.Profile class mostly consists of static methods and properties that facilitate custom NWTD business logic that deals with user accounts.

Examples of functionality this class provides include:

- Retrieving the current user's level (e.g. Level A)
- Retrieving the current user's Business Partner
- Retrieving a the current user's Business Partner state
- Assigning an active cart to a customer
- Setting user sale information

NWTD.Orders.Cart Class

- When creating new carts use the CreateCart method found here
- Static methods for creating order numbers

NWTD.Web.UI.UserControls.InfoManagerUserControl

This is a control that is designed to be inherited from. It hooks up to the InfoManager Service automatically, and provides the OrderInfoManagerClient class to any inheriting controls.

NWTD.Search.Extensions.Indexers.CatalogIndexBuilder

This class overrides the ECF search indexer, providing addional search index functionality. In particularl, this indexer breaks up ranges of grades (e.g. 6-12) and indexes each grade individually. For the search index to work as NWTD requires, this indexer should be configured for use as opposed the built-in indexer provided by MediaChase in the mediachase.Search.config (in the Commerce Manager AND in the frontend site) file.

OakTree.Web.UI Assembly

The OakTree.Web.UI assembly contains custom helpers and controls developed by OakTree to solve various common ASP.NET development challenges. Several controls (such as a control for phone numbers), and several helper static methods (such as a helper method for including embedded resources) are used in the NWTD application. Additionally, some embedded JavaScript libraries such as OakTree JavaScript Library are pulsed from this library.

Custom User Controls

The majority of the end user experience and business logic is handled in a series of custom user controls. These user controls are registered with the CMS and then can be dragged onto a page within the page editor for the CMS.

Control File Locations

All NWTD-specific controls are stored uniformly in the FrontEnd project in the directory /Structure/User/NWTDControls in sub-directories that represent their functionality within the system.

Each control has a configuration file that registers it with the CMS in the directory /Structure/User/NWTDControls/Configs. ECF automatically detects these configuration files and makes their associated controls available to the CMS. The configuration files contain information about the control's location, name, unique ID, and so forth.

Adding a new control to the system therefore requires the generation of the control itself, and creating a new corresponding configuration file (you could use an existing one as a template—just don't forget to change the GUID).

Controls that have custom properties that need to be set during the CMS are possible, provided you are willing to create a control for editing such properties. The control doing the editing must implement the IPropertyPage interface and be registered in the control's configuration file.

Order Submission Controls

Like many built-in ECF controls, the order submission-related controls have been completely replaced.

Order submission flows through three pages, each of which has a custom user control associated with it. Each control is located in the Mediachase.Cms.Website.Structure.User.NWTDControls.Controls.Cart

namespace. Their files are located in `\FrontEnd\Structure\User\NWTDControls\Controls\Cart\Cart.ascx`.

Cart.ascx

This control displays the contents of the cart and is visible when you go to the ~/Cart/Views.aspx page. One interesting thing to note (this is a requirement from NWTD) is that *the act of viewing a cart makes it your currently active cart*.

The cart uses code built specifically for NWTD to generate totals, handle tax, shipping, and gratis choices.

When a user clicks the **Add Shipping Info** button, the cart is saved, and the user moves to the Shipping and Billing Information page.

Checkout.ascx

This is the control where shipping and billing information is added to the cart. It is one of the more complicated controls in the system because of the amount of branching that takes place between Level A users and Level B users and different depositories. This control is located on the page ~/Cart/Submit.aspx and is typically accessed by clicking the **Add Shipping Info** button on the view carts page.

Here are some things Checkout.ascx does:

When the page loads, if the user is level A, we look up the user's possible shipping addresses and bind it to a dropdown list, and if it's already been selected and saved by the user, we re-select the correct item. Level A billing addresses are pre-determined and stored against the user's organization. There should only ever be one billing address. If the user is level B, we present a form for the user to fill out for both shipping and billing, and re-populate its fields if that information has been entered in the past.

Any Cart MetaData is also bound to other form fields on the page for both Level A and B users.

When either the **Return to Cart** button or the **Review Order** button is clicked, the data entered into this page is saved. Clicking the Review Order button takes the user to the Order Summary page, which contains the CheckoutSummary.ascx control.

CheckoutSummary.ascx

This control displays the summary of an order, and for Level A users. allows submission. Usually this page is reached by clicking **Review Order** from the Shipping and Billing Entry page.

When this control loads, the first thing that happens is to check whether the user is attempting to view an already-submitted order. If so, a message saying so is displayed. Otherwise the normal information is displayed.

In the case of an unsubmitted (or unsubmittable) order, a print button is displayed, and for Level A users, a submit button.

When a Level A user submits a cart, the following happens:

- The status of the cart changes
- The cart's name is changed to an amalgamation of the cart id and name (this allows users to create future carts with the same name)
- The Cart's WebConfirmation metafield is generated using the GenerateOrderNumber method of the NWTD.Orders.Cart Class.
- A confirmation email is sent (using one of ECF's Email Templates), which includes the WebConfirmation
- The order complete message is displayed, including the WebConfirmation.

At this point, the cart is available to be "harvested" by NWTD during Data Export.

It's important to note that the default behavior of ECF was to convert a cart to an order and then delete the cart. This does not occur for NWTD. Instead, the cart's status is changed.

Custom Templates

ECF includes the ability to register templates. For information on templates, see Page (and Other Types of) Templates.

While most templates simply provide different layouts, on some occasions there is some custom code going on in templates that integrates NWTD business logic. For example, in the home page templates (\FrontEnd\Templates\NWTD\PageTemplates\NWTDHomeTemplate.ascx and \FrontEnd\Templates\NWTD\PageTemplates\MSSDHomeTemplate.ascx), there is quite a bit of branching that goes on, exposing different areas of content to different users depending on their level. In fact, there are three conditions: LevelA users, LevelB users, and Anonymous users. But because a content editor needs to be able to edit the content of all of these areas, in design mode they are all shown.

Another complex template is the book search template control (\PublicLayer\FrontEnd\Templates\NWTD\Modules\BookSearchControl.ascx). This control is actually used for conducting searches for publications.

Cart Service

There is a web service located in the FrontEnd project under /Services/Cart.svc. This service provides a means for changes to a shopping cart to take place using Ajax, and is heavily consumed using the Cart JavaScript Actions written for the application.

JavaScript

jQuery Modal plugin

A custom jQuery plugin was created for the numerous modal dialogs found within the site. The plugin is located in `/FrontEnd/Structure/User/NWTDControls/Scripts/jquery.modal.js`. It provides modal dialog functionality, and can add buttons and callbacks.

It's fairly easy to implement. You create the markup for the modal dialog in the page, and then use a jQuery selector to find the element. Once the element is found, you can call the plugin, passing it the options like you would with most jQuery plugins.

Here's an example:

```
var editDialog = $('.nwtd-editCartDialog').modalDialog({
    title: 'Rename Cart',
    onSubmit: function (e, data) {
        e.preventDefault();
        data = data[0];
        if (data.newName == data.cartName) {
            editDialog.modal.close();
            return;
        }
        NWTD.Cart.updateCart(
            data.cartName,
            data.newName,
            function (data) {
                if (data.Status != 0) {

editDialog.modal.showMessage(data.Message);
                    return;
                }
                editDialog.modal.close();
                refreshCarts();
            }
        );
    }
});
```

In the above example, any element on the page with the CSS class .nwtd-editCartDialog will be wrapped in a modal dialog, given a title of "Rename Cart" and will execute the inline function as a callback when the submit button is clicked.

Cart JavaScript Actions

There is a JavaScript library that allows very easy use of the Cart Service located in `/FrontEnd/Structure/User/NWTDControls/Scripts/NWTD.js`.

This library basically makes it so you can call the service's methods without having to repeatedly write the ajax-related code. For example, to add a new item to the current user's current cart, you can simply call the following JavaScript function:

```
//create some callbacks
var successCallback = function(responseData){ alert('success!')};
var failureCallback = function(responseData){ alert('failure!')};

//add an item to the cart
NWTD.Cart.addItem('myitemcode',2, successCallback, failureCallback );
```

JavaScript Registration

There's a static method named NWTD.Web.UI. AddRequiredScripts the adds the required JavaScript files for the custom aspects of NWTD's application to function. Most of these scripts are Embedded in the

OakTree JavaScript Library

There's a set of JavaScript functions and classes in the OakTree.Web.UI Assembly. These are embedded scripts that provide helper functionality (such as getting and setting cookies) along with a very convenient way to register a script to a page, ensuring it only gets added once, and to register information about a control from server-side script that makes client-side script aware of its existence.

An example of this latter function can be seen in the Page_Load method in the Mediachase.Cms.Website.Structure.User.NWTDControls.Controls.Cart.ManageCarts control:

```
OakTree.Web.UI.ControlHelper.RegisterControlInClientScript(
    Page.ClientScript,
    this, "ManageCarts",
    "{updatePanelID:'" + this.udpUserCarts.ClientID + "',
cartGridID:'"+this.gvUserCarts.ClientID+"'}"
);
```

In this case, we're making an UpdatePanel so that information about it to the client side script. This information will include the update panel id, and the id of a GridView within that control (represented as a JSON object we're manually building). This control will be registered in the JavaScript namespace OakTree.Web.UI.WebControls.ManageCarts.

The client script that wants to access information about this control can simply inspect the object at OakTree.Web.UI.WebControls.ManageCarts, which will be an Array of Objects, each with the information we passed to it on the serverside present:

```
jQuery.each(OakTree.Web.UI.WebControls.ManageCarts, function (i,
ManageCart) {
    console.log(ManageCart); //logs the object to the console
});

/*logs
[Object { controlID="ctl15_MainContentArea_ctl00_ctl00",
updatePanelID="ctl15_MainContentArea_ctl00_ctl00_udpUserCarts",
cartGridID="ctl15_MainContentArea_ctl00_ctl00_gvUserCarts"}]
*/
```

Theming

ECF uses the ASP.NET theme system. NWTD's theme is called "NWTD," and is located in the NWTD theme directory in the front-end web site MSSD's theme is called MSSD. All CSS edits should take place there.

Because of the great similarities between applications, many edits performed on the CSS files for NWTD will need to be performed on the CSS files for MSSD and vice-versa.

Schema

Accounts

Account Metadata

In the Console Manager, account metadata is configured und3er Adminstration->Profile System->Meta Classes. The Element to choose is Principal and the Type is Account.

The following metadata has been added to the system and is not out-of-the-box metadata for ECF.

Active Cart

This field is used to determine a user's active cart. Any time a user's active cart is changed through the UI, this value gets changed.

Organizations (School Districts/Business Partners)

NWTD's business logic requires that each Account (customer) be associated with an Organization (school district or private school/business partner). Organizations are determined by a user when they sign up. The connection between a user and an organization is a built-in feature of ECF, and during user sign-up, standard ECF APIs are called to make this association.

Some static methods and properties exist in the NWTD.Profile class to help programmers determine the current user level.

Organization Bug

There is a bug in ECF 5.0 that prevents more than five organizations from being listed in the backend admin page for editing an account. Therefore, an organization can only be set for a user when the user first registers, or by changing the

This bug also makes it impossible to use the ECF API to get a list of organizations. Therefore, in controls that pertain to signing up for an account, a direct SQL query must be used to generate a list of organizations from which to choose.

Organization MetaData

Organizations have been extended in for NWTD using the standard ECF metadata system. This metadata is then used to enforce business logic throughout the system.

In the Console Manager, account metadata is configured und3er Adminstration->Profile System->Meta Classes. The Element to choose is Principal and the Type is Organization.

The following custom metafields are important from a business logic perspective:

IsTaxExempt

Sometimes NWTD Organizations are except from tax. This metafield indicates this. The main place this information is used is in the CartTax and CartTotal static methods in the NWTD.Orders.Cart Class, and is used to determine cart totals.

ShipRate, ShipMinCharge, ShipFlatCharge

These metafields determine shipping charges. The logic is in code and can be found in the CartShippingCharge and CartTotal static methods in the NWTD.Orders.Cart Class, and is used to determine cart totals.

BusinessPartnerPriceGroup

This information is used to determine special pricing for a business partner. See Pricing.

BusinessPartnerID

This metafield associates an organization in ECF with a District/Public school in NWTD's SBO system.

BusinessPartnerState

This metafield associates an organization with a US state, and can therefore be used to associate a user with a state. Business logic in the NWTD application requires this metadata for some specific business logic:

- Per-State price calculation
- Per-State catalog availability
- Per-State catalog flags
- Extra column in search results for Nevada Users

There is a utility function for retrieving a Business Partner state in the NWTD class library.

Addresses

ECF stores two types of addresses: customer addresses and order addresses. A customer address is associated with an account or an organization, while an order address is associated with a cart or an order. NWTD imports customer addresses SBO into the system and associates them with organizations.

During checkout, Level A users may select a shipping address, but will have a billing automatically assigned to them. Level B users will generate a new address by entering text fields. During the process new Order Addresses are created, which are stored against the cart instead of against the organization.

Address MetaData

In the Console Manager, account metadata is configured under Adminstration->Profile System->Meta Classes. The Elements to choose are Customer Profile Address and the Type is Customer Profile Address Extended.

The following custom metafields are important from a business logic perspective:

SBOAddressId

This is used for linking a business partner to an address in NWTD's SBO system.

TaxRate

Orgnaizations have different tax rates, and this metafield is used to determine custom tax rates imported by NWTD. The main place this information is used is in the CartTax and CartTotal static methods in the NWTD.Orders.Cart Class, and is used to determine cart totals.

IsFreightTaxable

Sometimes NWTD Organizations should not have their shipping charges taxes. This metafield indicates this. The main place this information is used is in the CartTax and CartTotal static methods in the NWTD.Orders.Cart Class, and is used to determine cart totals.

Type

There are several address types in the system, some of which should not be made available in locations in the site where address is selected. The type field is used to filter such addresses out. For example, only shipping addresses, which have a type of "S" are displayed to Level A users during the order submission process.

Carts

Carts have been extended in ECF to support more than one cart per customer. The only task involved in doing so has been to write the code to interact with ECF's API to create additional carts for a customer.

Cart MetaData

You can view the configured cart metadata by doing the following:

In the Console Manager, account metadata is configured under Adminstration->Order System->Meta Classes. The Element to choose is **Order Group** and the Type is **Cart Order Class**.

The Cart itself has some metadata associated with it, mostly for storing customer supplied information against a cart. These metafields are set during the order submission process and include things like **PurchaseOrder**, **OrderContactName**, **OrderContactPhone**, and so on.

WebConfirmation

This metafield gets set during checkout, and is displayed to the user upon submission and in the confirmation email. NWTD can then retrieve this information from the database to make a connection when a customer enquires about an order.

Cart Line Items

MetaData

Line items can have some extra fields added that make certain business logic possible.

To see the fields, In the Console Manager, account metadata is configured under Adminstration->Order System->Meta Classes. The Element to choose is **LineItem** and the Type is **LineItem Extended**.

Line Items only have one additional field, which is **Gratis**. This field allows customers to enter choose a "gratis quantity" as opposed to a quantity for which they'd be normally charged.

Books

Books MetaData

There are lots of metafields associated with books that are used in search results, search filtering, and sorting. There are a few that are used programmatically for custom business logic. These fields are listed below.

TypeSort

This is a special string that allows sorting by type according to a very specific search string generated by NWTD and added during data import. This field is used for sorting in search results and sub-search results.

Status_PDX

This is a status code for items that should only be visible for NWTD Customers. It is hidden in the search results if the user is not an NWTD user. See \PublicLayer\FrontEnd\Structure\User\NWTDControls\Controls\Catalog\SearchResults.ascx

Status_SLC

This is a status code for items that should only be visible for MSSD Customers. It is hidden in the search results if the user is not an NWTD user. See \PublicLayer\FrontEnd\Structure\User\NWTDControls\Controls\Catalog\SearchResults.ascx

StateAvail_WA, StateAvail_AK, StateAvail_OR, StateAvail_NV, StateAvail_UT

These fields are used to determine whether a book is available in a certain state in book searches and subsearches. Before the search is conducted, the a filter is added to the search based on the user's state. For example, a Washington user will have a filter added that add a *where* clause to the search query only returning results where the StateAvail_WA field is set to "y." For examples, see \PublicLayer\FrontEnd\Templates\NWTD\Modules\BookSubSearch.ascx.cs

StatusCode_PDX

This field is used to check status codes for MSSD. In search results, if this field's value is "CX" the item is listed as unavailable.

StatusCode_SLC

This field is used to check status codes for MSSD. In search results, if this field's value is "CX" the item is listed as unavailable.

AdoptDist_NV

This is a special field that displays special information to members of the Nevada Adopted District.

NoPriceAvailable

This is a special code indicating no price is available. This is metavalue is used in several controls:

- \FrontEnd\Structure\User\NWTDControls\Controls\Catalog\EntryPrice.ascx.cs
- \FrontEnd\Structure\User\NWTDControls\Controls\Cart\LineItemPrice.ascx.cs

Pricing

ECF has a built-in pricing system that NWTD Takes advantage of. See http://archives.mediachase.com/doku.php?id=ecf:50:developerreference:reference:catalog:pricing.

The way it works is as follows:

ECF looks in the session for certain variables and then compares them to special prices set in the database. We just need to set both the business partner key and the state key against the currently logged in user during runtime (probably during login). Mediachase will choose the lower price between those two (if either are present).

For example, if user Jeremy has two variables set when he logs in:

```
Session[StateGroup] = OR;
Session[BusinessPartnerGroup] = 1234;
```

ECF looks to see if there's a price defined for the product for each of these two sales type (stored as a numeric value, and defined in the config file). Among the prices returned for that product, ECF checks to see if there's a matching SaleCode. ECF then chooses the lower of those prices, and presents it to the user. We do not need to add any custom logic, as the tiered system will be working (assuming there are no cases where a certain district is charged more than the state to which it belongs).

When NWTD imports data into ECF from their SBO system, the only thing you need to do is make sure that when you add a price for a state, you make add a sale type to the Sales table:

If the price is for a state, make the SaleType be 3 and the SaleCode be the state name, or whatever way we're going to reference a state.

If the price is for a Business Partner, make the SaleType be 4, and the SaleCode be whatever way we identify a Business Partner.

With this data in place, and the appropriate information in the ecf.catalog.config file, the only code that needs to be executed involves determining the business partner and sate states a user is associated with, and letting ECF know by storing it as a session variable.

This is accomplished in various places in the application by calling static SetSaleInformation method in the NWTD.Profile class.

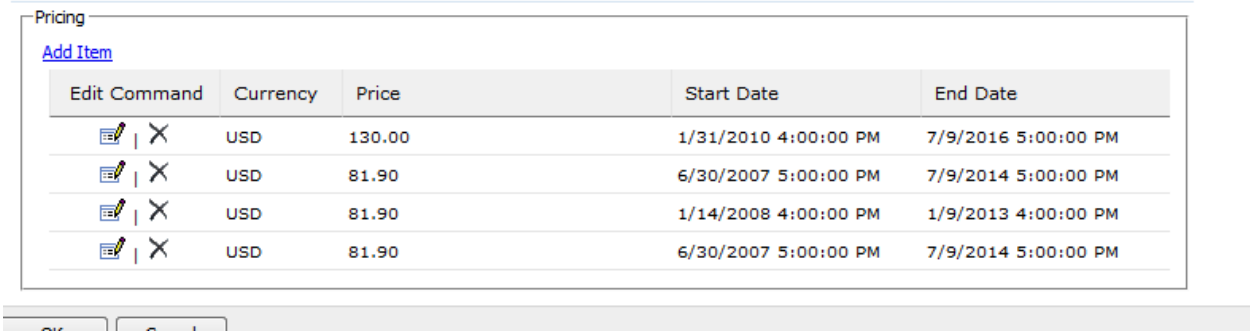
The Current Implementation

Here's an example of how the current implantation works:

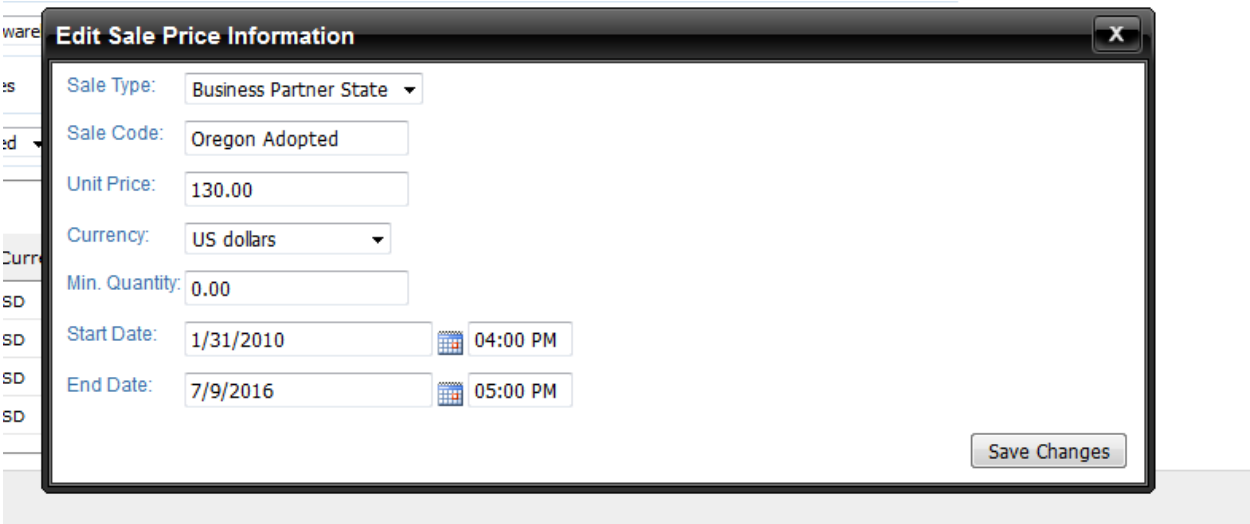
In the ecf.catalog.configfile there's a SalePriceTypes item configured with a key of "BusinessPartnerPriceGroup" that has a value of "3."

If we look in the Organization table there is a column named "BusinessPartnerPriceGroup" that has a value of "Oregon Adopted." Finally in the SalePrice table we see an item with a SaleType of "3" and a SaleCode of "Oregon Adopted." Therefore any user with a BusinessPartnerPriceGroup of "Oregon Adopted" will get at minimum the sale price in the SalePrice table that has a SaleType of 3 and a SaleCode of Oregon Adopted.

You should be able to take a look at the item in the catalog with the code 9780030931086 and see four prices.



In the database each of these prices should have a sale type and a sale code associated with it.



This is what makes the pricing automatically work.

InfoManager Service

NWTD has written the *InfoManager* web service that the application is hooked up to. This service provides two different sets up information outlined below. The parts of the application that present this information are often referred as “Customer Self-Service” or CSS.

Publisher Information

The InfoManager service can be used to retrieve a list of publishers for a given depository and respective representatives using the Service’s GetPublishersByDepository and GetPublisherRepresentatives methods.

Order Status Information

Information on order status can be obtained using the service. There will be a delay between order status showing up here and actual submitted orders due to the amount of time it takes for NWTD to “harvest” orders from ECF and place them in their SBO system.

In the controls developed for this application, only Level A users can view this information, and when viewing the information, they can see orders for ALL other level A users.

Search

Searches are conducted in three places in the application:

- In the Search Results page
- In the Individual Book detail page as a search for related items
- In the search box itself to generate a list of Publishers

In all cases, as much caching is done as possible. The search filter facets on the search results page are generated by some very complex ECF code that uses the results of the search in conjunction with the searchFilters.config.

In order to accommodate some of NWTD's search requirements, several changes were made, including the creation of a custom indexer and some Site Exports

ECF provides the capability to export CMS site data such as pages, menu items, site settings, etc. to an XML file that can be used to import the site into a different environment. Because OakTree's environment serves as the master copy of the site, it is important, especially after CMS changes, to export the site and deliver it to NWTD along with any builds or source code deliveries.

Sites should be saved in \Docs\Site Export and then checked into SVN.

Source Code Changes to enable partial ISBN search.

Custom Indexer

The custom indexer (NWTD.Search.Extensions.Indexers.CatalogIndexBuilder) provides the ability to index grades in such a way that grade ranges (e.g. k-8) can be indexed so that they would be returned in searches for grades within the range (e.g. grade 6).

This indexer gets configured to be used in the mediachase.Search.config (in the Commerce Manager AND in the frontend site) file.

Configuration

Configuration Files

The following configuration files should be set on a per-environment basis. For more information on configuration files, you can view the ECF 5.0 documentation as well: http://archives.mediachase.com/doku.php?id=ecf:50:start:config and http://archives.mediachase.com/doku.php?id=ecf:50:start:searchindexserviceconfig

web.config

The web.config file is a standard web.config file. The only per-environment setting that deviates from the standard configuration settings you'd see in a web.config file is the template path location for Email Templates.

mediachase.Search.config (in the Commerce Manager AND in the frontend site)

There is such a file in both the FrontEnd and ConsoleManager, and configures where search indexes will be stored and retrieved. It contains important two settings:

- The search index directory. This is a system path and should point to the location where the site index binary data should be stored. For OakTree, NWTD typically generates this data and hands it to OakTree and placed in the /SearchIndex folder in the trunk of the SVN project. If NWTD supplies a new one, you should SVN Delete the old files first.
- The search indexer class. OakTree has generated a In order to accommodate some of NWTD's search requirements, several changes were made, including the creation of a custom indexer and some Site Exports

ECF provides the capability to export CMS site data such as pages, menu items, site settings, etc. to an XML file that can be used to import the site into a different environment. Because OakTree's environment serves as the master copy of the site, it is important, especially after CMS changes, to export the site and deliver it to NWTD along with any builds or source code deliveries.

Sites should be saved in \Docs\Site Export and then checked into SVN.

Source Code Changes to enable partial ISBN search.

- Custom Indexer that should be set here.

connectionStrings.config

This file holds the three connection strings to each of the pertinent NWTD ECF databases.

searchFilters.config

This file is located in the PublicLayer project in the App_Data directory, and determines which items show up in the search filter tree. The file is generated by NWTD and provided to OakTree periodically.

ecf.catalog.config

One important bit of data this file contains is sale rules.

```
<!-- Sale Price Types -->
<SalePriceTypes>
    <add key="AllCustomers" value="0" description="All Customers" />
    <add key="Customer" value="1" description="Customer"/>
    <add key="CustomerPriceGroup" value="2"
        description="Customer Price Group"/>
    <add key="BusinessPartnerPriceGroup" value="3"
        description="Business Partner Price Group" />
    <add key="BusinessPartnerID" value="4"
        description="Business Partner ID" />
</SalePriceTypes>
```

Documentation

Code Documentation

The C-Sharp projects that contain OakTree-generated source code have been configured to automatically build XML documentation, which ends up in the bin directory of the project.

A post-build script has been created in the FrontEnd web application that executes an html-based code documentation application called ImmDocNet. This application is located in \Docs\ImmDocNet\ImmDocNet.exe, and generates html-based source code documentation that will be output to the \Docs\CodeDoc directory. The output is not stored in SVN, since it is regenerated every time the application upon build.

You can also manually generate the code documentation without building by running \Docs\ImmDocNet\NWTDDocs.bat

Written Documentaion

Written documentation (including the document your reading can be found in the \Docs directory of the site.

ECF Documentation

ECF 5.0 online documentation is located at http://archives.mediachase.com/doku.php?id=ecf:intro.

Deployment

Deploy to Staging

To deploy a build to stating

1. Open the Project in visual studio
2. Make sure the project is set to Release Mode
3. In the Solution Explorer, select the FrontEnd Project
4. Click the **Build** menu, and select **Publish FrontEnd**. You should see a dialog that has the Target Location already set to be \\CHERRY\Clients\NWTD\web\temp.staging.nwtd.oaktree.com
5. Click **Publish**
6. Copy all the files from \\CHERRY\Clients\NWTD\web\temp.staging.nwtd.oaktree.com to \\CHERRY\Clients\NWTD\web\staging.nwtd.oaktree.com EXCEPT the following:
    - Web.Config
    - Web.CHERRY.Config
    - mediachase.Search.config
    - mediachase.Search.CHERRY.config
    - connectionStrings.config
    - connectionStrings.CHERRY.config

Staging URLs

NWTD: http://staging.nwtd.oaktree.com

MSSD: http://staging.mssd.oaktree.com

Admin for both sites: http://admin.staging.nwtd.oaktree.com

Deploy to Production

Here are MediaChase's instructions on deploying to production:

http://archives.mediachase.com/doku.php?id=ecf:50:operations:deployment

Basically the suggested made by MediaChase are standard deployment steps such as turnining on canching and turning off debugging.

IIS Permissions

One thing that's very important is to properly configure IIS permissions. Starting with IIS 7.5 a virtual account (AppPoolIdentity) can be used with each Appplication pool, making it possible to easily sandbox write permissions to a singlee user for a single site. Here's a great article on this topic:

http://learn.iis.net/page.aspx/624/application-pool-identities/

As for permissions, MediaChase instructs you to grant the account running the worker process full permissions to the entire website. A more secure approach would be to figure out specific directories that the website needs to write to (such as the App_Data folder) and grant write access there.

Importing Data Provided by NWTD

NWTD will provide the following:

- Search Index
- Database Backups
- Facet Configuration File

You must do the following to start using NWTD's data:

1. Use the Multi-Site
2. The site is set up so that the database and file system are shared between both the NWTD site and the MSSD site. Both sites use the same catalog. The places where the sites "split" is in the Themes and in the CMS data itself. The way this is configured is through eh Console Manager.
3. Console Manager to make backups of both NWTD and MSSD sites
4. Import the Database backups into SQL server
5. Point your dev/staging site's connectionStrings.config files to the new databases. Don't forget to do this for the staged Multi-Site
6. The site is set up so that the database and file system are shared between both the NWTD site and the MSSD site. Both sites use the same catalog. The places where the sites "split" is in the Themes and in the CMS data itself. The way this is configured is through eh Console Manager.
7. Console Manager too.
8. Use the Multi-Site
9. The site is set up so that the database and file system are shared between both the NWTD site and the MSSD site. Both sites use the same catalog. The places where the sites "split" is in the Themes and in the CMS data itself. The way this is configured is through eh Console Manager.
10. Console Manager to delete the MSSD and NWTD sites that came with the database.
11. Import the sites you backed up in step 1.
12. Replace the Facet configuration file with the one NWTD supplied
13. Replace the Search index with the one NWTD supplied

Site Exports

ECF provides the capability to export CMS site data such as pages, menu items, site settings, etc. to an XML file that can be used to import the site into a different environment. Because OakTree's environment serves as the master copy of the site, it is important, especially after CMS changes, to export the site and deliver it to NWTD along with any builds or source code deliveries.

Sites should be saved in \Docs\Site Export and then checked into SVN.

Source Code Changes

Because of bugs or other unmet requirements, a few changes have been made to the source code of ECF.

Incorrect Sorting

As part of troubleshooting issues with incorrect sorting, Mediachase had us do two things (the latter of which solved the problem).

Build and add the latest version of Lucene.Net

OakTree downloaded the source code for Lucene.Net 2.9.2.2, compiled it, and integrated it into ECF. Because of some API changes, there were a few necessary code changes in ECF

```
SearchResults results = new SearchResults(searcher.Reader, hits, criteria);
```

Becomes:

```
SearchResults results = new SearchResults(searcher.GetIndexReader(), hits,
criteria);
```

Execute Mediachase Patch

Run the following SQL script on the MAIN database:

```
DECLARE @Major int, @Minor int, @Patch int, @Installed DateTime

SET @Major = 5;
SET @Minor = 0;
SET @Patch = 73;

SELECT @Installed = InstallDate FROM SchemaVersion_CatalogSystem WHERE Major=@Major AND
Minor=@Minor AND Patch=@Patch

IF(@Installed IS NULL)
BEGIN
--## Schema Patch ##
EXEC dbo.sp_executesql @statement = N'CREATE FUNCTION [dbo].[ecf_splitlist_with_rowid]
(
      @List nvarchar(max)
)
RETURNS
@ParsedList table
(
      Item nvarchar(100),
      RowId int
)
AS
BEGIN
      DECLARE @Item nvarchar(100), @Pos int, @RowId int
      SET @RowId = 0

      SET @List = LTRIM(RTRIM(@List))+ '',''
      SET @Pos = CHARINDEX('','', @List, 1)

      IF REPLACE(@List, '','', '''') <> ''''
      BEGIN
            WHILE @Pos > 0
            BEGIN
                  SET @Item = LTRIM(RTRIM(LEFT(@List, @Pos - 1)))
                  IF @Item <> ''''
                  BEGIN
                        SET @RowId = @RowId + 1
                        INSERT INTO @ParsedList (Item, RowId)
                        VALUES (CAST(@Item AS nvarchar(100)), @RowId) --Use Appropriate
conversion
                  END
                  SET @List = RIGHT(@List, LEN(@List) - @Pos)
                  SET @Pos = CHARINDEX('','', @List, 1)
            END
      END
      RETURN
END'

EXEC dbo.sp_executesql @statement = N'ALTER PROCEDURE
[dbo].[ecf_CatalogEntry_SearchInsertList]
      @SearchSetId uniqueidentifier,
      @List nvarchar(max)
AS
BEGIN
      SET NOCOUNT ON;
      INSERT INTO [CatalogEntrySearchResults]
           ([SearchSetId]
           ,[CatalogEntryId]
           ,[Created]
           ,[SortOrder])
     select @SearchSetId, L.Item, getdate(), L.RowId from ecf_splitlist_with_rowid(@List) L
inner join CatalogEntry E ON E.CatalogEntryId = L.Item ORDER BY L.RowId

      SET NOCOUNT OFF;
END'
--## END Schema Patch ##
INSERT INTO SchemaVersion_CatalogSystem(Major, Minor, Patch, InstallDate) VALUES (@Major,
@Minor, @Patch, GetDate())

Print 'Schema Patch v' + Convert(Varchar(2),@Major) + '.' + Convert(Varchar(2),@Minor) + '.' +
Convert(Varchar(3),@Patch) + ' was applied successfully '

END
GO
```

Multiple Sort on Search Results

Out of the box, ECF 5.0 did not allow sorting search results on multiple files. As a response to our need for this for NWTD, Mediachase built a way to do this and send us updated source code. The updated files included:

- /BusinessLayer/Search/MediachaseSearch/SearchManager.cs (modified)
- /BusinessLayer/Search/MediachaseSearch/SearchSortField.cs (added)
- /BusinessLayer/Search/MediachaseSearch/SearchSort.cs (modified)
- /BusinessLayer/Search/SearchExtensions/AbstractSearchCriteria.cs (modified)

The files and corresponding emails can be viewed in `\ECF Support\HotFixes\Multiple Sort` in the source code.

Bug in Cart Prepare Workflow

There was a bug detected in the Cart Prepare work flow. Mediachase provided us with a fix for it that is located in `\ECF Support\NewVersionOfValidateLineItemsActivity.zip` in the source code.

Partial ISBN Search

Here's what Mediachase had us do to make this possible:

In ~\BusinessLayer\Search\SearchExtensions\CatalogEntrySearchCriteria.cs, please make the following changes:

1. Add in Line 240: parser.SetAllowLeadingWildcard(true);
2. Comment the line with "Query searchQuery..."
3. Uncomment the line right after (with "WildcardQuery searchQuery...") and change to:
4. WildcardQuery searchQuery = new WildcardQuery(new Term("_content", "*" + SearchPhrase + "*"));

Furthermore, in order for this kind of search to not kill performance, date ranges have been removed from the CatalogEntrySearchCriteria object. (`\BusinessLayer\Search\SearchExtensions\CatalogEntrySearchCriteria.cs` lines 250-254).

In order to really get things cooking, the ECF source code had to be further modified to use the following logic:

- To the CatallogEntrySeachCriteria Class, a Boolean property called IsISBN was added that checks the search string against a regular expresion to see if the search is probably for an ISBN:

```
public bool IsISBN {
    get {
        //string keyWords = this.SearchPhrase;
        //check to see if the KeyWords matches an ISBN Pattern
        if
(System.Text.RegularExpressions.Regex.IsMatch(this.SearchPhrase,
@"^[0-9-]+[a-z-0-9]?$")) { //any string of all hyphens or numbers and
possibly an alpha character at the end
            return true;
        }
    return false;
    }
}
```

- At this point the getter for CatallogEntrySeachCriteria.Query was editied to check this property and if the search phrase was for an ISBN, widlcards would be put at the beginning of the search phrase as well as at the end. Otherwise the wildcard would be only added to the end.
- This property is also used in the SearchFilterHelper class. If the search phrase is not an ISBN, fuzzy search will be turned on when the initial search yeilds no results. Fuzzy search will not happen if the search is probably for an ISBN. Also at this point the MaxClauseCount property for Lucene is altered depending on whether IsISBN returns true or false.

```
/// <summary>
/// Searches the entries.
/// </summary>
/// <param name="criteria">The criteria.</param>
/// <returns></returns>
public virtual SearchResults SearchEntries(CatalogEntrySearchCriteria
criteria)
{
 try
   {
    if (criteria.IsISBN)
Lucene.Net.Search.BooleanQuery.SetMaxClauseCount(Int32.MaxValue);
     _Results = Manager.Search(criteria);
    }
  catch (SystemException)
  {
   if (HttpContext.Current.IsDebuggingEnabled)
    throw;
  }
 // Perform fuzzy search if nothing has been found AND the search
phrase is NOT a big number (e.g. ISBN)
 if (_Results.TotalCount == 0 && !criteria.IsISBN) {
  Lucene.Net.Search.BooleanQuery.SetMaxClauseCount(1024);
  criteria.IsFuzzySearch = true;
  criteria.FuzzyMinSimilarity = 0.7f;
  _Results = Manager.Search(criteria);
 }
 Lucene.Net.Search.BooleanQuery.SetMaxClauseCount(1024);
 return _Results;
}
```

Upgrading

These are the notes that were taken during an investigation on upgrading from ECF 5.0 to 5.1.

- This is not going to be an easy upgrade, and is going to consume quite a bit of time
- ECF's solution setup and hierarchy has changed so much that the solution will need to be re-built from scratch, with the NWTD customizations added (I've got some instructions from ECF on how to do this, but they're pretty vague)
- Debugging/running by hitting F5/shift-F5 is no longer supported in the way they've set things up. To debug, you'll now have to set up a new IIS application with multiple mapped directories, then attach the IIS Worker process on your machine while viewing the site.
- There are many, many namespace changes that I've run into so far, and it appears that there may be some method changes as well. It's going to take awhile to correct all of these in our templates alone and get the solution to actually build. Hopefully must of it is just namespace issues.
- There may be changes to some of their base controls that we'll want to implement in our site.
- They appear to have completely removed the "Profile System" from their metadata scheme. I've written them to ask what we should do since we were depending on that for one feature
- It appears that the catalog/metadata/site information from our 5.0 site imports ok, though I haven't got the solution running to be able to see If they search/present properly, etc.

Here are some notes in an email from Richard in regards to his end up the upgrade:

> I have completed updating the synchronization scripts for the 5.1 upgrade. I had to make quite a few changes to the Business Partner script, however, it greatly simplified the script … nice . I have exported the Meta data (xml files) and uploaded them to our FTP server for your consumption. In addition, there is a back up of the 5.1 database fully loaded with catalog and biz partner data. The new tax and shipping fields are also included. I exported the Order subsystem's Meta data as well, but don't recall adding any fields to this subsystem (yet). (see folder 1_18_2010 on the FTP site)