Sitecore Ecommerce Enterprise Edition

Software Developer Kit

System Overview Documentation

Chapter 1: Introduction

In the first chapter you will get the background and development philosophy of Sitecore Enterprise Commerce including:

- ✓ Background of product
- ✓ Sitecore Ecommerce Enterpise Edition Components
- ✓ Architecture & Philosophy

Goal of this manual

The goal of this manual is to provide the student or reader a basic understanding of the Sitecore Ecommerce Enterprise Edition toolkit. Together with the SDK training curriculum and other documentation, the student should be able to leverage their .NET expertise to use Sitecore Ecommerce Enterprise Edition to rapidly build and maintain their own, robust, fully e-commerce enabled web sites.

Components of the SDK

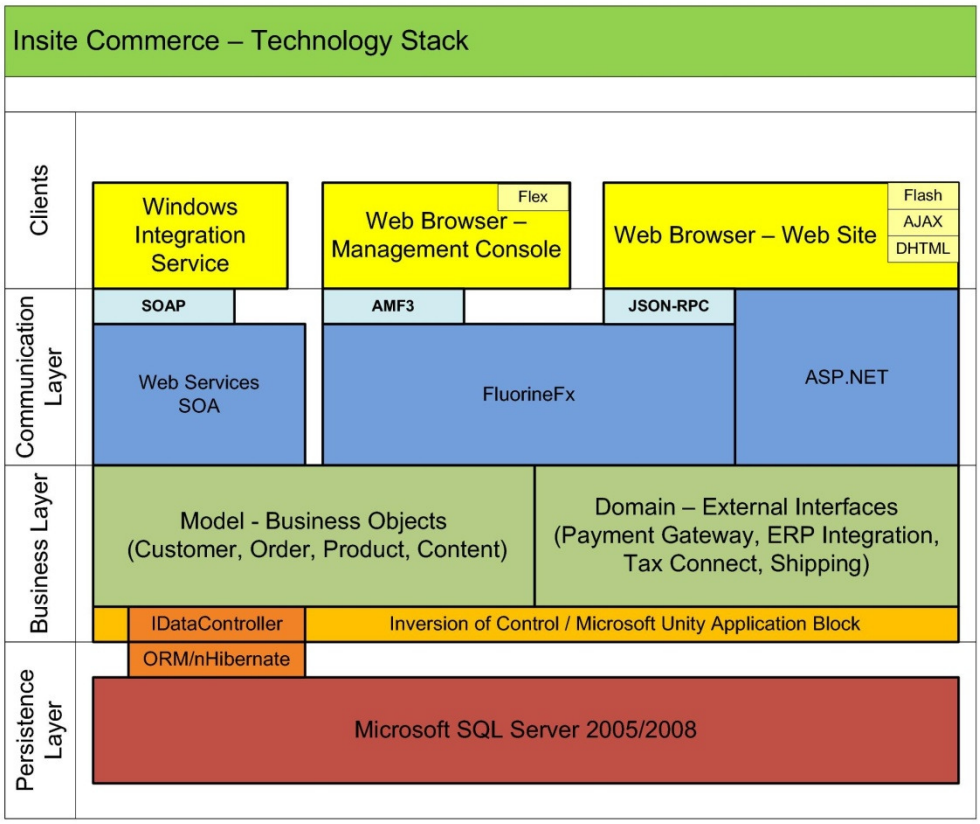
Sitecore Ecommerce Enterprise Edition (SEEE for the rest of this document) is a set of software components that can either add eCommerce functionality to an existing Sitecore site or help rapidly create and deploy custom web sites.

SEEE contains several core components including:

- Management Console – this is an Adobe Flex-based system that allows for maintenance of data associated with the site. This includes application configuration settings, shipping data, promotions, content, order history and much more. This is the set of pre-defined functionality that allows a non-programmer to maintain the site.
- Insite.Model – this is the set of classes that is used to access the data and methods within SEEE.
- Insite.Domain – this is a set of classes and functions that interfaces externally to various functions such as credit card processors and tax services.
- Nicam Demo Site – this is a sample web site complete with sample C#, ASP.NET pages, HTML and CSS. This is the starting point for the student to do their exercises and to create new projects.
- Template database – this is an MSSQL 2005 database complete with sample data to be used with the template site.
- Sample ‘’Generic” code – several of the core components of the application use Inversion of Control to allow customized extensions for customer-specific use. The generic versions of these programs are provided so the student may create their own versions of the code as needed.

- Integration Service –along with the sample code, the integration service is provided so that the user can create their own integrations with SEEE. In general, the supported integration is either through a flat-file transfer or a direct database connection to the ERP. Web services and APIs may also be used.

Architecture



Architectural Philosophy

Sitecore Ecommerce Enterprise Edition's underlying architectural philosophy is to leverage existing technology that is appropriate for business environments that is scalable, robust and proven. The designers of Sitecore Ecommerce Enterprise Edition believe that .NET along with other, open source componentry brings a state of the art, flexible, and scalable technology to Sitecore customers and partners.

KISS Principle. Keep It Simple, Stupid. This philosophy states that elegance is in simplicity, not intentional complexity. We try to teach our internal developers to use existing functionality rather than creating new facilities, adherence to standards and naming conventions. If some piece of code can be written in one line of code using 5 embedded functions or in three separate lines that makes it more readable, then we prefer more lines of code. This can be anathematic to some developers, but our experience teaches us to write easily understood, maintainable code. Remember, you may not be the one that has to maintain the code.

Standard Nomenclature. Our goal in our designs is to have properties and methods be self-describing. We do this by standardizing contractions or, preferentially, spelling out words. For example, if we want to refer to the 'quantity' of an item, Qty is the standard contraction. The current standards are readily apparent when browsing the object model. Additionally, for simplicity and ease of use, we do not repeat the data type within the property in general, so Activate is used consistently as a date the entity becomes active and Deactivate is the date the entity becomes inactive. In cases where the active status of an entity is Boolean, the IsActive property would be used. Promotions are a good example of when something should become activated/deactivated.

Framework Orientation. The model, which covers the bulk of the underlying functionality you will work with, is explicitly written to be a framework. Thus, there is a ModelBase class that everything inherits from so if we want to add core functionality, we are able to do so at the appropriate layer of logic.

Programming to Interfaces/Design Patterns. The construction of SEEE is heavily influenced by the notion of design patterns. We strongly suggest that if you are not familiar with design patterns that you obtain a book and read through it to understand them. Design Patterns bring an orientation to programming whereby best practices are used to determine the way to approach a problem. Very few problems in the domain of web programming are unique and there are many proven approaches to solving these issues. We recommend the *Heads First: Design Patterns* book published by O'Reilly as an excellent resource. This approach provides a common language when working with our developers and will make the overall model make even more sense when studied with the knowledge of design patterns as a foundation. Programming to an interface allows us to seamlessly swap out components of the application as needed. For example, the OrderPackager is called through a consistent interface so that the underlying, specific version of the order package can work very differently from site to site if needed. This approach also allows us to swap out major components of the architecture if needed or desired.

Use the Page to your advantage. When building the web site, minimize the amount of code in the code behind wherever possible. Use code behind whenever there is business logic. Each page is generally associated with a single collection.

Extensibility. While the user may add new tables and classes to the application, they will not have the ability to extend the properties of existing objects. The SDK developer *may* certainly use extension methods to add business functionality to the underlying classes. The Management Console is not extensible by the SDK developer.

Use of GUIDs/Key structure. We have chosen explicitly to use GUIDs for all primary keys. This allow for globally unique entries and also allows entries to be made and relationships honored within the NHibernate session *prior to* the data being committed to the database while ensuring that multiple users would never generate the same primary key.

Table Standards. When a table is named it is normally given a description that denotes a single entry in the table – for example Customer or OrderLine rather than Customers or OrderLines. The primary key of the table will be the name of the table followed by "Id" for example CustomerId and OrderLineId. Note the "init cap" format –

each word in a compound column or table name has the first letter capitalized and not others, therefore we use “Id” rather than “ID.”

Database/ORM/IData Controller

SEEE currently uses Microsoft SQL 2005 or Microsoft SQL 2008 as the underlying database engine. There are a small number of stored procedures and full-text indexing is used for searching, but other than that, there is not specific requirement for SQL Server to be the underlying database.

NHibernate for .NET is our object-relational mapping tool which handles database interactions. There are a few targeted areas (such as pricing calculations or integrations) that utilize direct SQL for speed, but, in general, NHibernate manages the database interactions directly, including complex objects such as many:many and one:many (parent/child) relationships. Lazy loading is one of the excellent functions of the tool so that related objects are only loaded when they are needed during processing.

It is important to understand that NHibernate uses a session context to store data. It is important to flush the session in order to commit data to the database. Additionally, if you are working with large sets of data, it can improve performance significantly by both flushing and evicting data within an NHibernate session.

IData Controller is the data wrapper and includes the functions to tell NHibernate to flush, evict, commit, etc. The reason the IData Controller exists is to allow us to abstract out the data interactions from the chosen ORM. If a better ORM than NHibernate was chosen, it would be a reasonable level of effort to swap it out without affecting any of the standardized data access logic.

Insite.Model

The model is the container of all of the core **business objects** and related business logic in the overall model. It contains all of the classes that are used for building and maintaining the web site and, other than some of the functions within the Insite.Domain, it is the core of the SDK. Significant detail about key components of the model will be discussed later in this manual.

Data Store Singleton. This component of the model is what handles session information and inversion of control. This is also where the Windsor Container is implemented (see Castle Windsor project at http://www.castleproject.org), optionally swapping out customer-specific componentry as follows:

- IWindsorContainer
- IDataController
- IPaymentGateway
- IShippingEngine
- IOrderPackager
- IPromotionEngine

- IPriceCalculator
- IEmailProcessor
- IBuildEmailValues
- ITaxCalculator
- ProductUpdate
- CustomerUpdate
- CustomerOrderUpdate
- ProductRefresh
- TaskBuilder

Additionally, this component does a license check whereby the system will 'phone home' to the SEEE licensing server to validate the current license.

These optional plug-ins will be automatically loaded if a custom DLL implementing the plugin interface is located in the bin directory. If none is found, it will use the standard one in the Model or Domain.

This component is also a service provider used to retrieve specific sets of data as needed and is the primary conduit for retrieving data directly through the IData Controller. For example, see the following code as an example of how this is constructed:

```
public static String GetNextCustomerNumber(String prefix, String format)
{
    int nextCustomerNumber = 1;
    NameValueCollection queryOptions = new NameValueCollection();
    queryOptions.Set("UseSqlQuery", "True");
                nextCustomerNumber =
                DataStoreSingleton.DataController.GetCount<Customer>("SELECT
                ISNULL(MAX(CAST(REPLACE(CustomerNumber,'" + prefix + "','') AS INT)),0) + 1 AS
                Count FROM Customer WITH(NoLock) WHERE CustomerNumber LIKE '" + prefix +
                "%' AND ISNUMERIC(REPLACE(CustomerNumber,'" + prefix + "','')) = 1", new
                Hashtable(), queryOptions);
    return prefix + nextCustomerNumber.ToString(format);
}
```

ModelBase. This is the base class from which all other classes inherit. Common methods in this class are available including:

- CompareTo
- Get
- GetCount
- GetList
- GetPropertiesCollection
- ImportXLS

Some of the other functions this class services includes:

- List of states
- List of countries
- Load local tax rates
- Load application settings
- Current web site

Insite.Domain

The Domain component represents everything that is *outside* of the model and dishes up **business services**. This includes integration components to the ERP and payment gateways. Calls that will eventually filter through the inversion of control (IOC) container are called in the domain as the interface which are then filtered through the IOC container into the model which has invoked/loaded the customer-specific version of the functionality, such as the promotions engine.

The domain also handles the integration with Adobe Flex which is used to run the Management Console.

Insite.Library

The library is a set of helper functions for common tasks in the web site such as regular expression helper, hashtable generator, script helper, etc. General descriptions of each are shown in a table in the Web Site Components section.

Adobe Flex

The Management Console is written using the Adobe Flex framework along with FluorineFx for remoting to .NET (see www.fluorinefx.com) using AMF3 (Adobe Messaging Format) for compact transfer of serialized objects and data between the model and the management console. This component of the SDK is delivered as is and is not extensible.

Management Console

The Management Console (MC) is the maintenance component of the SDK. This Flex-based interface allows the user to set up web sites, configuration parameters, product information, web content, product categories, etc. It is the heart of the maintenance capabilities of the SDK.

The entire MC is controlled by an Application Dictionary which is designed to dynamically allow changes to the labels and behavior of data elements. The Application Dictionary controls will be extended to the menu objects and various other UI components such as buttons and tabs in the near future.

The application dictionary controls, for example, what properties are displayed or are read-only by user role. It provides a very flexible way of creating user-role based security and interface, including support for multi-lingual MC installations.

The MC also allows for user-specific saving of settings such as the data grid width, column order, and column sort. The inherent import/export capability of the MC allows data to be easily extracted, manipulated and re-imported when batch updates are desired to be done.

The MC uses the Model-View-Controller design pattern in its implementation.

Global vs. Website Managed Items

When you access the Management Console you will notice that there are sections for Global Settings, Global Management, Website Settings and Website Management.

Settings are intended for setup and occasional modification. Menu functions like application settings, states, and carrier information are controlled in Settings predominantly because it is gone into relatively rarely.

Management is intended for areas that are more frequently changed or modified.

Global is used for areas that are shared across the various sites in the MC. Products and Customers are managed globally as are application settings.

Website is used for site-specific functions such as assigning promotions to a specific site.

Chapter 2: Order Flow

This chapter is designed to explain the basic order flow through Sitecore Ecommerce Enteprise Edition.

How the user creates an order…

When a user is in the web site, they are typically browsing for products and adding them to the cart. As soon as an item is added to the cart for the first time, an order object will be created and its status will be 'Cart'.

This order will persist throughout the session. Even if the user navigates away from the current web site to another web site and then returns, the cart will be completely intact.

If you wish to garner statistics on abandoned carts, simply look for all the orders with a status of 'Cart'.

Steps in Order Processing

There is a generic way of processing orders

1) When an item is initially added to the cart, the CustomerOrder object is created

2) When the user adds a product to the cart, the system will use the quantity ordered and calculate the price and create an OrderLine and update the value on the CustomerOrder

3) When the user Views Cart, the system will display a list of the cart contents with the currently calculated price

4) When the user chooses to Go To Checkout, the system will then go to a login page which may allow the user to continue as a guest.

    a. Note that every user is associated with a user profile in the aspnet_Membership table. SEEE uses a combination of standard membership services and our own user and customer tables. A user must be established in the database to even begin the checkout process and subsequently submit an order. A customer must exist in the system in order to submit an order and customers are created automatically as needed and associated with the user.

        i. Aspnet_Membership is related to aspnet_Users on column userid

        ii. Aspnet_USers is related to UserProfile on column username

    b. The user is typically allowed to register at this point and establish themselves with an email/login and password. This allows them to return to the site and see historical orders, depending on the site design.

c. New Users are automatically associated with a new customer record. In typical commercial/B2B sites, the relationship between an authorized user and the customer they belong to is normally pre-established.

d. The next page is normally the bill to/ship to page, also known as the address page. The ship-to address can be marked as the 'same as' the billing address. The billing address is typically only really used for validating credit card information.

e. The next step is the Review/Pay page. This is the page that really does the work including the following:

    i. Based on the postal code and/or state of the shipping address (not the billing address) taxes are calculated.

    ii. Automatic promotions are also calculated on this page. For example, orders > $100 get free shipping may impact the shipping dropdown.

    iii. The shipping options are calculated on this page as well based on the shipping address and contents of the order. Rules may be established to control if the customer-specific shipper is used or if certain products cannot be sent to a particular location such as a ban on liquor shipments into certain states.

    iv. This page then captures the credit card information and processes the authorization (or sale transaction, depending on application setting) as well, prior to confirming the order.

f. Once the user accepts the order by clicking on the submit order button, the order will be completed and the status will be changed to 'Submitted'. An entry will be made in the Scheduled Task table to send the order over the to ERP.

g. The final step that the user sees is the Order Confirmation page telling them the order number generated by SEEE.

h. When the integration service picks up the order and successfully submits it to the ERP, the status of the order is changed to either 'Processing' or 'Review', depending on an application setting.

i. If the ERP system generates its own order number, that number is normally posted back into the SEEE order during the Order Refresh process but, in some cases, comes back at the same time that the order is successfully submitted and the ERPOrderNumber is captured along with the change to the status being 'Processing' or 'Review'.

j. During the order refresh process, if shipments are processed, the status may change to 'Shipped' or 'Complete' but could be a variety of status codes such as Paid, Canceled, etc. depending on the specific integration.

Chapter 3: Web Site Construction & Components

In this chapter we will explore the components of the actual web site project:

- ✓ The template web site
- ✓ Categories
- ✓ Left-Navigation
- ✓ The Cart

Web Site Components – Base Constructs

When you install the template project, you will have references to a variety of components that will be included in your installation including:

- Castle.Core – part of Castle Windsor project for Inversion of Control
- Castle.DynamicProxy
- Castle.MicroKernel
- Castle.Windsor
- FluorineFx – part of access to Management Console
- Insite.Model
- Insite.Domain
- Insite.Library
- Microsoft.SqlServer.Connection

There are a variety of other references that will be part of the template site, some of which you may be using in your site and many of which you will not.

The following is a list of included, base or sample components for building a web site.

This is source code that is included in the SDK and will allow you use the existing source as a model for making your own pages. Almost all of these are 'base' code from which you can extend and add additional functionality.

Web Site App_Code

With the SDK you will receive a fully functioning template site that has many pages with code. There are some common components that are used in the structure of the site that are discussed briefly below. Many of these are wrappers to access their corresponding component in the model but then allow for extension by having them be site code.

Component	Description
CategoryPageBase	A category page specific instance of CommonCategoryPageStrategy
IHeader	This was implemented as an interface because some browsers had difficulty changing the header/title information directly and this approach seemed to solve the propblem
PageBase	Wrapper to CommonPageBase and base class for all pages.
ProductPageBase	Wrapper to CommonProductPageStrategy
ProductSearchController	Controller to help create site-specific searching
SearchResultsControlBase	Common base for search results
SiteLinks	Single place to contain common site links
UserControlBase	Wrapper to CommonUserControlBase

Web Site Controls

Component	Description
CommonApplicationMessageDisplay	Common control allows the display of an application message on the page
CommonCategoryPageStrategy	Constructs the base page for a category based on current values
CommonPageBase	Base page constructor and adds in hooks to add/support css, javascript, set page title, etc.
CommonProductPageStrategy	Loads products into a product detail page and constructs the metadata and page title appropriately so that the page is web searchable and may be indexed by web crawlers.
CommonUserControlBase	Hold context objects such as current web site, current customer, current user profile, current customer order, current category, current product, etc.
CommonWebPageContent	Basic control to load content
PageStrategy	Holding control to load a page

Insite.Library

Component	Description
ControlHelper	Helps with date controls to fill in years and months
DynamicComparer	Used in sorting lists, use Linq instead
FileHelper	Functions to read data from an Excel or csv files and write Excel data

Component	Description
ListHelper	Creates a sorted list from any IList
LowerCaseHashtable	Function to add items to a hash table with generics and always in lower case for sorting
ObjectHelper	Copies objects and resolves references so NHibernate can properly track and flush
RegularExpressionLibrary	Uses regular expressions to help validate data such as mail addresses or postal codes
ScriptHelper	Positions a div as a popup in the center of the screen – requires jQuery.
SortProperty	Used in sorting lists – use Linq instead
SQLHelper	Forms up full-text searches while preventing SQL Injection
StringHelper	A series of functions for manipulating strings
WebControlHelper	Adds javascript so that an enter keypress in the given TextBox will fire off the given LinkButton. For some reason asp.net default buttons do not play well with LinkButtons
WebPathResolver	Helps load a given image and, if there is no image, it loads a default image

Chapter 4: Key Model Components

In this chapter we will explore several of the key components of the model. This overview will familiarize you with some of the items and concepts you will be dealing with regularly. Not all classes will be defined in this documentation – refer to the Sitecore Ecommerce Enterprise Edition component help file for all of the available classes, properties and methods.

At the end of this chapter you should be familiar with:

- ✓ Common Methods
- ✓ Products & Categories
- ✓ Customers
- ✓ Promotions Engine
- ✓ Shipping Engine and Order Packager
- ✓ Document Manager
- ✓ Payment Gateway
- ✓ Users

ModelBase

Since the architecture is organized around a framework, ModelBase is something that each base class in the mode inherits from. This has allowed us, for example, to create generic SOA wrapper methods on all of the classes which are used for the Management Console which is written in Adobe Flex.

The ModelBase class has many useful methods that you will find useful throughout the application. Feel free to browse this class and learn them. Some of the most useful methods include:

- Evict – tells NHibernate to free memory – used in conjunction with flush
- Get – retrieve a specific object
- GetList – returns an IList of objects

Common Methods in Classes

Most of the classes in the model have the following common methods:

- Add[associated class] – this is a useful method to create relationships between entities. For example, a category is composed of products and products are in one or more categories. For example, Product has an AddCategory method and Category has an AddProduct method which allows the relationship to be defined from either direction.

- Remove[associated class] – this is the opposite of the add and removes the association.
- GetBy[specified property] – this is a way to retrieve a single object or an IList of objects, depending on the method call. For example, Customer.GetByNumber allows the developer to retrieve a specific customer using the CustomerNumber field (human-readable key to the table). Gets in general return an IList of an object in a hash table but typically a GetBy returns a single object, but not always.
- Get[specified property] – this is a way to retrieve a specific related object. For example, Customer.GetCustomerProperty will allow the developer to retrieve a specific customer property for the customer in scope.

Overview of Key Classes

The following classes are the primary areas of Insite.Model that the SDK developer will be working with. They are listed alphabetically as opposed to topically to help find them in this list. There are many classes that are data holders (i.e. State or City) where we believe the classes are enough self-evident and may be viewed in the help file so therefore are not included in this list.

Affiliate

The affiliate class is a simple class that can be set up to help track affiliate programs. The URL is used to capture what affiliate sent the user to the site so that special offers could be made or for the site to actually display itself differently by using the affiliate-specific image.

This class must be implemented in the site by design.

ApplicationLog

The application log is used to capture messages in the system for a variety of occurrences.

One of the types of log entries is a 'debug' entry. These are controlled by the Application Setting 'ApplicationLog_LogDebugMessages' - if set to true, then the debug message will be logged and if not, it will be skipped. These log entries will be marked as type Debug.

When logging an error message in the application, simply call ApplicationLog.LogError(source, message, batchnumber). The *batch number* is an integer that can be used to trace back to the set of data that caused the problem if appropriate. The *source* would be the page or class that invoked the error and the message is what the developer deemed applicable to report to the log. These log entries will be logged as type Error.

You may also use the ApplicationLog.LogMessage method with the source, type (error, information, security, debug) and message.

ApplicationMessage

The application message is used to store common messages to use in various places in the application, especially useful for pop-ups. For example, OrderConfirmationMessage can have the text 'Your order was successfully placed'.

Carrier

Carrier and its associated classes CarrierZone, CarrierShipCharge, CarrierZoneRate, and CarrierZoneZipCodeRange classes support the shipping engine. In essence, the carrier represents the actual shipping carrier (i.e. FedEx, UPS, Yellow Freight) and then the ShipVia classes relate to the Carrier and represent a specific service (i.e. Overnight Air or Ground).

The carriers may use a pre-defined rating method (based on property RatingService) or may use custom data tables which is what most of the carrier-related classes are for. The following objects are used when the carrier is being rated within the application and not relying on an external interface.

CarrierZone is a delivery area and the zone is typically defined by the ZoneZipCodeRange. For example, when shipping from Minneapolis to Florida, the zone may be zone 5.

CarrierZoneZipCodeRange defines the range of zip codes used to narrow down the zone itself and from that calculate the rate. The presumption is that a particular carrier as set up in the system has a singular set of zones and rates. If you have multiple ship-from sites, you may need to use separate carriers to encode the zone/rate structure properly.

CarrierZoneRate is what holds the actual rate for a given carrier/zone combination. These are coded by weight so that each particular weight of the package would have a specific rate. This is typical of parcel carriers.

CarrierShipCharge is a separate set of records that may be used to encode additional shipping charges. For example, there may be a residential delivery charge or an oversized package charge. There is no specific interface to expose this data to the user, so these should be used carefully and are a flexible structure to hold charges that are package or service-oriented as opposed to weight-oriented.

Make sure to read the class information on ShipRule, ShipRate, ShipCharge which is related to the carrier class.

Category

Category is a major area of the system and is a way to group like items together. Categories contain products and other categories and are used, in general, to pull multiple products or other categories together for presentation on the site.

Important Point. Categories belong to specific web sites and must be established and maintained by site. Products are global, so the way one associates which products belong to a given site is specifically through the category construct.

Categories may also be leveraged for some specialty uses and provide a simpler mechanism to apply business rules to an entire set of products in a single place. For example, you can associate a particular dealer to a particular category or group of categories. This may then be implemented in the site as a way to only present those categories (and subsequently products) that are available to a specific dealer. This logic must be implemented explicitly in the site, but the constructs are presented through the model to do so.

Many of the same classes that may be associated with a product may also be associated with a category. Examples would include Property, Specification, Restriction, Tax Exemption, and CrossSell. It is up to the site designer to know when to use this functionality.

Finally, a category may also be assigned one or more subcategories which is done by assigning the parent category to a subcategory to prevent an infinite loop. A given category can only have a single parent to keep the integrity of the category tree.

Content (and related classes)

Because much of what makes up a web site is content, we have several classes that are related to content management including:

- Category
- EmailTemplate
- NewsArticle
- Product
- Review
- Specification
- WebPage
- WebPageContent

The Content class is intended to help with the workflow and approval of given contents and stores the actual HTML content. It contains properties such as Approved, PublishToProduction, and SubmittedForApproval.

Every Content entry has a link to the ContentManager which effectively manages all the links to a given piece of content. For example, a Product will have a pointer to the ContentManager and there could be multiple entries of Content pointing to the same ContentManager. The ContentManager is intended to keep a single link between the source entity (i.e. Product, Category, WebPage) and the current instance of its content.

WebPage is intended to store *content managed pages*. These pages may be added to the site as needed and have no active data – they are static pages.

WebPageContent is intended for a section of content on a page such as a current promotion slash on the home page.

Customer

This class is much as you would expect – it holds data related to a purchaser on the system. What makes customer distinct from User is that the User is an *individual person* and a Customer is a *purchasing entity*. Every user that purchases from the site will be established as a customer in the system.

There may be multiple users associated with a given customer. Conversely, a given user may be associated with multiple Customers – this is typically for a sales rep, dealer or sales manager – someone who is responsible for or to a group of customers.

The customer contains a great deal of information including name/address, tax information, credit limit, etc. Customers may be imported/updated (we use the term *refreshed*) from the underlying ERP system as well. This function is normally done through the Sitecore Integration Service and is run nightly but may be configured to run more frequently as needed.

Customers may have the following classes associated with them:

- Carrier: Specifically allows only particular carriers to be used for this customer
- Product: This would be used to assign specific products to a customer
- ProductSet: This is analogous to WishLists and works much the same way – see WishList. This allows saving named lists of products by customer as opposed to user which is what Wish Lists do.
- Property: Customer Properties associated with the customer – see discussion below on Property
- Salesman: which sales person(s) are associated with the customer
- User: which user(s) are associated with the customer
-
- It is work noting that Customers may have multiple ship-to locations. This is denoted by the difference between the CustomerNumber and the CustomerSequence. This can be very ERP-specific. Guest users or public customers tend to have a single customer record where the ship-to information is the same as the *billing* information. Corporate customers, however, may have a common billing record, typically denoted by CustomerSequence 0 and additional ship-to locations with other sequences.

CustomerOrder/OrderLine

This class represents the actual order in the system as represented by the *cart*. As soon as someone adds something to their cart, the order is instantiated. An order is associated with the currently logged in user (even if checking out as a guest). As products are added to the cart, each one either creates or adds to an OrderLine object.

Orders are always associated with a particular customer.

The **status** of the order begins life as 'CART'. Once an order is completed and payment taken, it is changed to **Submitted**. Once the order makes it to the ERP

system via the integration service, it is changed to either **Review** or **Processing** depending on the application setting indicating if the order will be reviewed prior to accepting it into the ERP.

When the order is refreshed back from the ERP and it has been shipped, the status will be changed to **Complete**. Other available status codes are **Saved, Void,** and **Ready For Pickup** (a special case status when the order is being picked up in the store).

Depending on the application setting, the credit card is either hit for an authorization or it is actually hit for the entire sale transaction.

There is also an **order type** field which is either **Order** or **Quote**. The developer may choose to start orders as a quote and then convert them to an order when ready to check out – it is a way to save the order for later recall as well. All orders stay in the system, including *cart* orders (sometimes referred to as *abandoned cart* orders).

The key constructs in an order include:

- Customer Bill To (used for credit card verification)
- Customer Ship To (used for shipping calculations)
- Order Lines
- Product ordered
- Quantity Ordered
- Price
- Promotions applied
- Ship Via
- Credit Card Payment (current system allows a single credit card payment per order)
- Gift Card – allows one or more gift cards to be redeemed as part of the payment transaction
- Order/OrderLine properties are available
-

There are many functions that are called as the order starts to move through the payment page including the following:

- Apply automatic promotions (i.e. free shipping on orders > $500)
- Determine valid shipping methods based on product restrictions and/or customer restrictions
- For rated carriers, run the OrderPackager to determine how the order will be shipped and calculate all the shipping rates by valid ShipVia
- Calculate sales tax
- Allow entry and application of an entered promotion

- Authorize or capture funds against the credit card if entered

Dealer

A dealer is generally used to define brick & mortar locations where the customers may go to purchase products advertised on the site. The model contains name/address information along with geocode data for use in a dealer locator. Dealers may also be assigned to specific categories or products to allow context-sensitive lists of dealers.

Document Management

Documents are generally constructs associated with a category or product and are structured similarly to web content. Examples will include PDF's, Word documents, Excel spreadsheets and image files. These are used for specification detail, product schematics or whatever other document-oriented content you wish to expose to the users. It could be considered a library of digital assets.

The classes involved in document management include **Document, DocumentManager,** and **DocumentType.**

DocumentType is used to define the type of document by its extension. An icon may be associated with the document type for presentation on the site. This is not currently exposed in the Management Console.

DocumentManager is the central link to the list of documents. Each category and product will have its own document manager link and each document is then associated with a specific document manager.

Document is the actual digital asset, or, more precisely, the link to the actual asset on the system.

Email Processing

There are several classes specifically associated with emails. The creation of an email in the Management Console combines the EmailTemplate and EmailList object. There are some standardized times when emails are sent out (see document on Application Settings) such as when an error occurs, when an order is placed, and when shipment is confirmed.

EmailTemplate is the holder of the actual email contents and is associated with an EmailList. Data fields are tokenized and replaced within the template.

EmailList defines the attributes of the email and points to the template that contains the content. It includes the subject line, who the email is from, and the optional from address. If the template is marked as being a subscription email, the user must have *opted in* or they will not receive the requested email. In general, once a specific EmailList object is created, a set of users is added to it and then the data, if present, will replace the tokens for the construction of the email body.

IBuildEmailValues is an interface to create email processors for specific purposes. In order to know what data should be read and incorporated into the email, this inversion

of control container is used per email to send. The BuildOrderEmailToList passing in a customerOrder shown below is an example of this. This function grabs the actual data, currently from either the Order or the Shipment, and maps the data into a common token set to be replaced by the EmailPricessor.

IEmailProcessor is an interface to manage what emails to send and whom to send them to. Its basic method is SendEmail.

An example of sending an order confirmation email:

```
// Send Order Confirmation Email
if (EmailList.GetByName("OrderConfirmation") != null)
{
    String emailToList =
DataStoreSingleton.BuildEmailValues.BuildOrderEmailToList(customerOrder);
    if (emailToList.Length > 0)
    {
        EmailList emailList = EmailList.GetByName("OrderConfirmation");
        EmailListProcessor emailListProcessor = new EmailListProcessor(emailList);
        emailListProcessor.Send(emailToList,
DataStoreSingleton.BuildEmailValues.BuildOrderEmailValues(customerOrder));
```

Filtering/Advanced Filtering

Advanced Filtering is a construct by which the site builder creates some generic ways to group products together for presentation and the web master is able to set up their own filtering groups. It is essentially a way of grouping products together explicitly without the search function trying to ascertain the user's intent and searching for, say, 'Red' products where it could return a search set that includes 'Redolent Ginger' as a result. Another example would be 'Shop by Brand' separated from 'Shop by Category'.

Advanced Filtering is typically reserved for sites with a large number of products where a user would want to be able to find groups of items in various ranges of attributes. This is considered a filtering operation rather than a search operation although this is a nuanced differentiation.

Advanced Filtering is made up of the following classes:

- CategoryFilterSection
- FilterSection
- FilterValue

The Advanced Filter starts with the concept of a filter section. Examples would be Price, Color, or Style. Once the section is created, categories may be assigned directly to the section if appropriate – the entire category should fulfill the intended criterion of the filter section.

Next, the FilterValues are created with the description to display (i.e. $50-100, $100-150 for a price section and 'Red/Crimson, Blues/Greens' for a color section).

Next, assign the specific products or categories that belong to that specific filter value.

When the advanced filter set is presented, it will use the data constructed here to create the filtered and presented list and know which parts belong to the specific selection.

Note that Advanced Filtering is a single-level construction to pre-group items together. Searching could then be done beyond that within the filtered list.

GiftCard/GiftCardTransaction

A gift card is normally sold in a transaction on the web site (product property IsGiftCard controls this behavior). So, let's say a user purchases a $50 gift card on the site. When the order is submitted, the system will then generate a gift card entry and associated transaction to show the value of the gift card and establish an expiration date if applicable. This code must be generated within the site since there could be several variations on how this process ought to work.

When the gift card transaction is created, it is associated with the order on which it was purchased.

When a gift card is redeemed, the system will check that it is a valid gift card (card exists, is not expired) and that there is a balance on the card. Only the current balance will be allowed to be redeemed against the order.

Interface Classes – General

There are several *Interface* classes that are intentionally created to allow for being swapped out. As mentioned earlier, the Castle Windsor Inversion of Control Container approach is being used and the DataStoreSingleton object will load the appropriate version of the class at runtime by scanning the bin directory looking for any DLL implementing one of the plug-in interfaces. Some of the interface classes are only expressed currently with a single service class.

In general, the expressed instances of the interfaced class are contained in the Insite.Domain. Examples include OrderPackager_Generic, BuildEmailValues_Generic, or PaymentGateway_Dummy.

Interface: IDataController

This is the core of how the model communicates with the database. Currently, the IDataController is implemented to speak to NHibernate as the ORM. IDataController is used extensively to *speak* to the database and manage transactions.

Interface: IEmailProcessor

This was discussed above with Emails and is used to send the actual emails.

Interface: IOrderPackager

This class is used to return a set of packages based on a specific order.

The standard OrderPackager will first add up the total weight of the order using the Products in the order. The system will then determine the smallest package that is

established by carrier that can accommodate the contents. If no single package can handle the entirety of the shipment (exclusively by weight), then the system will close one package and open another.

This process will proceed until a list of packages with their dimensions and weights are completed which is then typically used to pass to the various rating services to arrive at an estimated freight amount by service.

Interface: IPaymentGateway

This is the interface used for handling transactions against the selected payment gateway. Its only method is to SubmitTransaction which passes in the type of transaction, the credit card data (in-memory only) and customer order object. The gateway must then be able to process the various transactions such as authorize, capture sale or void.

Interface: IPriceCalculator

This interface is used to control pricing logic within the system. While there is already a pretty comprehensive pricing capability within PriceCalculator_Generic, it is another example of how pricing could be extended for a given site by changing out the pricing engine.

Interface: IPromotionEngine

This is another example where it is unlikely to replace the existing engine, but it is available to be done. Promotions, in particular, are a complex structure where different actual classes are invoked depending on the setup. See the section on Promotions for additional information.

Interface: IShippingEngine

This is very analogous to the promotions engine. The shipping engine is used both to determine what carriers/services are *available* given the contents of the order and the destination along with the estimated shipping cost.

Integration: ITaxCalculator

The TaxCalculator_Generic is a version that will use the tax constructs already in Sitecore Ecommerce Enterprise Edition to calculate tax. This means that the state and postal code will drive the aggregated rate of tax and the tax code on the product or exceptions defined in the tax entities will determine if a given item is excluded from taxation. The customer may also be established as tax excempt.

PackageLine

This is a class that is relatively rarely used. It is currently designed to integrate with Insite Software's InSiteShip application to pull in, during the Shipment Refresh, the packages and the package contents. This class is used to show what items were actually packed in each package.

Product

This is the most central class in the entire model and there are many classes that may be associated with it. Products, at their core, are simply those items that may be sold on the web site. Only Active classes will be generally available to the site and there are many properties that may control specific behavior on a given site. A good number of these properties are truly placeholders and are available for the site designer to leverage, but do not assume that they are implemented as standard.

Note that products can be an item sold directly or be considered a cross-sell or accessory, they can be associated with one or more categories, or can be specialty products (i.e. Gift Card)

Products may have any or all of the following associated with them:

- Category – indicates what products belong to a specific category
- Category Cross Sells – products associated with a category's cross-sells
- Customer – products associated with a customer
- Dealer – products associated with a dealer
- Inventory Transaction – inventory transactions for a product
- OrderLine – product being purchased on an order line
- ProductFilterValue – part of advanced filtering
- ProductAccessory – accessories associated with a product
- ProductCrossSell – cross sells associated with a product
- ProductProperty – custom property for a product
- ProductSpecification – specification data for a product
- ProductTaxExemption – if the product is part of a tax exemption
- PromotionResult – product-specific promotion results
- PromotionRule – product-specific promotion rule
- Restriction – product is restricted from being sold in certain areas
- Review – review data for a product
- ShipRule – product-specific shipping rule
- Subscription – a specific subscription on a specific order where the product is the product being subscribed. SubscriptionLines then define the products that will be shipped
- SubscriptionLine – product within the subscription that is being fulfilled
- Subscription – standard products within a subscription
- WishList – product on a wish list

Some of the key properties of the Product that may not be obvious include:

- **Active** (date) – this determines if the product is active and is returned in various searches or is able to be added to the cart. The Active date must be less than or equal to the current order date and the DeactivateDate must be either null or greater than the order date.
- **AllowAnyGiftCardAmount** – if set to true the amount may be entered by the user and the gift card will typically be generated dynamically. If set to false, this would indicate that the gift cards are already established in the system with pre-defined amounts.
- **CategoryTree** – this is hierarchical list of categories that the item is assigned to. It is used in the PromotionRules engine and TaxExemption to enable the system to traverse up the tree from a given category to its parent categories to see if an item 'belongs' to a category. For example, if a part is assigned to category "Bed" which belongs to "Bedroom" which belongs to "Furniture" and Furniture is marked as tax exempt, the system needs to also make all products belonging to subcategories of Furniture to also be tax exempt.
- **ERPManaged** – this property indicates if the product should be updated by the ERP integration update. If this field is not marked as true, then even if the part exists in the ERP, it will not be updated.
- **ERPNumber** – while this may be the same as the product number (name) in the system, this field is used for connecting the ERP product/item number to the SEEE version of the product.
- **IsConfigured** – there is a base production configuration capability in the system and this flag indicates that the product is a configured product. The actual configuration must be expressed through the site design.
- **IsGiftCard** – this flag indicates that the product, rather than being a physical product necessarily, is actually a gift card. See the Gift Card section for additional information.
- **IsSubscription** – this is another capability of the system which allows for an item to be created as a subscription item which then contains other products that are included in the subscription. See the Subscriptions section for additional information.
- **Name** – this is a unique key on the product table and is used to identify the human-readable part number for the product. This is also used when presenting the title on a product detail page, so this is often either the ERP number or the short description.
- **PriceCode** – depending on the ERP or the way pricing was implemented, this field is used to group products into pricing categories. Customers may also be grouped into pricing categories and pricing may be impacted by how a given product is set up, so this field could be meaningful. Other than this, there is no standard functionality to it.
- **SubscriptionMonths** – this is a way to indicate the standard months for which a given subscription product is to be fulfilled. It is used in the repeating task that will generate new orders from a subscription order.
- **TaxCodes** – these are used in the tax processor. Note that there is a site setting called the Non-Taxable TaxCode. This is the current way that the

system knows that an item is tax exempt. For example, if the non-tax code is 'NT' and the item's tax code 1 is marked as 'NT', it effectively makes the item non-taxable.

- **TrackInventory** – this flag indicates if the part should actually decrement inventory balances in the InventoryTransaction table. If the site is not tracking inventory, then this flag should be turned off to help improve performance.

Promotions

Promotions are a major function of Sitecore Ecommerce Enterprise Edition and it is not often modified for specific sites. See the documentation on the Management Console for additional information on how Promotions are defined and used. The following is a list of the classes that are involved in the promotions engine:

- **Promotion** – this is the top level of a promotion. Promotions are created as global and then assigned to specific web sites.
- **PromotionCode** – this is a table that identifies which promotions have been applied to which orders
- **PromotionProcessor** – this is the parent class that is actually called do to the work of applying the promotion. It uses an IList of promotions and the order and will then apply the valid promotions against the order.
- **PromotionRule –** the promotion rules (and there are several specific ones) is used to determine if an order qualifies for a particular promotion
- **PromotionResult –** this is how the promotion is expressed if it passes the rules. Examples are taking an amount off the order, off of shipping, shipping being free, getting a product for free or adding a product to the cart from a qualified list of products.

Property Class (generic)

The property class is something defined in the ModelBase and is associated with:

- Customer
- CustomerOrder
- Dealer
- Product
- OrderLine
- UserProfile

The intent of the Property class is to provide a generic way to associate custom data with an object. These are simply name/value pairs in the database and are used to store any sort of attributes that are not already exposed by the model.

Salesman

The salesman (sales rep) class is used to denote which internal sales rep in the company is responsible for their customers. There is a parent salesman property to create a sales manager hierarchy that may be used.

Customers may be assigned to multiple sales reps.

Each user may be assigned to a specific sales rep. The sales rep itself may be assigned to a specific user to help automatically identify them if desired.

Scheduled Task

This is one of the core constructs used for integration. See the separate section on Integration Services for additional information.

Shipments

There are several classes involved in tracking shipment history and these are dependent on how the integration service was written for the specific ERP.

Shipment – this is the class that saves actual shipments. It contains the order that was shipped, the shipment date, and if a shipment confirmation email was sent.

ShipmentPackage – this is the construct that will save the tracking numbers. If integrated with Insite Software's InSiteShip application, the shipment packages and package lines will be created automatically. If not, then an integration needs to be created in order to create the packages associated with a shipment. The packages will basically encode the size, weight, and tracking number and optionally the ship via code and freight charge.

PackageLine – this class represents the contents of the shipment package

Shipping Rules & Constructs

Shipping is a fairly deep set of objects and constructs and was based on the approach used by Promotions. It is overrideable using inversion of control.

ShipCharge – the intent is to allow adding specific, additional shipping charges to the calculated charges. The standard behavior is to add all shipping charges associated with a carrier to the overall cost of the shipping charges. This allows the site to add other charges to the calculated charge and simply add it to the calculated charge.

ShippingClassification – this is intended to be used to define products into specific groupings for calculating shipping charges. For example, if the FreightQuote rating service is used, this classification would represent the standard freight class (i.e. 50, 55, 62.5). When the shipping engine is invoked, it would aggregate weight by each product's shipping classification and use that data to calculate the overall shipping charge on a truck shipment.

ShipRate – for carriers that are have custom rate tables, the ship rate is what encodes the rate and, at this level, is applied against a particular ship via code and order range. The ship rates are used to apply fixed shipping rates per order value. This can be either a fixed amount or a percentage of the order, depending on the calculation method on the carrier.

ShipRule – this is part of the calculation engine and is used to determine if a particular carrier or service is available for a given order. If a rule is at the carrier then the ShipViaId is null and, conversely, if the rule is at a particular shipvia, then the CarrierId is null.

ShipRule…. – there are a series of specific ship rules types that make specific checks to see if the particular rule applies or not. While ShipRule is in the database, ShipRulexxxx is purely a code construct.

ShipVia – this is the actual service within the carrier that is being used. For example, FedEx has an overnight service and a second day service – each of these would be a different ShipVia. The ShipVia also encodes the ERP's version or code for the same service. The Ship Code field is intended to represent the carrier's identification for the service and for the services that are using on-line rating services such as FedEx and UPS, these codes must match the carrier's code list.

ShipViaShipCharge – this is the same as the ShipCharge but applies to the specific service. The same distinction used for ShipRule is applied since the data for both levels of ship charge are in the same physical table.

Specification

Specifications are associated with both categories and products and are a combination of optional content and name/value data. Specifications can contain formatted informational content or they can really be used as product-specific specifications such as color, size, weight, Ohm impedence, etc. Specifications are typically presented on a product detail page and could be used for comparing like products. There is also a SortOrder that may be used to define the order in which the specification data is displayed.

Subscriptions

The concept of the subscription is that there is a part, marked as a subscription, that contains one or more other parts that will be automatically fulfilled on a periodic basis. Think of a tune-up kit for a lawnmower, as an example. The customer orders the subscription item which indicates the price of the subscription and the frequency of fulfillment. The contents of the subscription then may be a spark plug, quart of oil, and oil filter.

There is a process that runs checking for when subscriptions are to be fulfilled. When a subscription comes due, based on the original date of the subscription and the date comes due for another fulfillment, an order will be created automatically and submitted to the ERP.

Please note that the Authorize.Net CIM module must be in place in order to support subscriptions if they are to be paid by credit card. This module/payment gateway is the only way for Sitecore Ecommerce Enterprise Edition to be able to charge a credit card or create a new authorization without having to store credit card data. When the subscription is initially created, if there is no CIM profile, the system will automatically set one up and retain the customer/card profile id.

SubscriptionProduct – this is the set of products associated with the product marked as 'IsSubscription'. This class is used to denote which products belong within a subscription.

Subscription – this is the placeholder in a specific order that initiates the subscription itself.

SubscriptionLine – a line for each SubscriptionProduct will be created and associated with the subscription.

Tax Calculations

If the standard tax calculator is being used, there are several areas in the system that are involved in tax calculations. These are outlined as follows:

WebSite – In the WebSite itself is a property called TaxFreeTaxCode. This is often set to 'NT' (non-taxable). This is the trigger to determine if a customer or product is tax exempt. The website also defines the tax calculation method (dollar, percent, none, calculate). Only the *calculate* method goes through the state/local tax rate and tax rate exemption logic.

Customer – in the Customer class is property TaxCode1. If this is set to the TaxFreeTaxCode, the customer is considered tax exempt.

Product – in the Product class there is also a property TaxCode1 that works the same as with the customer – if set to the TaxFreeTaxCode, the product is tax exempt in all jurisdictions.

State – this construct is used to capture the tax rate and exemptions at a state level. The state has a taxrate and taxable property. If the state is taxable then the tax rate is applied against the order total. If freight is also marked as taxable, freight and additional handling are added to the taxable order total.

LocalTaxRate – this class is used to define tax rates at a zip code level. This rate is calculated separately from the state tax rate and the two are added together. The taxfreight property is honored at the local level irrespective of the flag at the state level.

TaxCalculator_Generic – this is the primary tax calculation engine that honors the various parameter selections as defined above. This program may be overridden if needed.

TaxExemption – this class is used to define specific products or categories that are tax exempt in a particular state or locality. Exemptions may be applied over specific dates, so, for example, Florida has a tax holiday over the Memorial Day weekend, the tax

exemption could be applied for just that period of time. A tax exemption is first established by tax jurisdiction (state/local) and then by product or category. When the tax calculator runs, it checks each line to see if it has a current exemption and, if so, the amount of that line is excluded from the taxable total.

UserProfile

The UserProfile is used in conjunction with the aspnet_Users construct to track users in the system. A User is defined by their login and the system stores their email address, password, password reset question/answer and roles along with some other parameters. The username property must be unique and it is traditional, but not required, to use the email address as the username.

UserProfiles are assigned roles and that role will define security and other functionality.

UserProfiles may also be related to Customers, EmailList, WishList, Custom Properties, and a single Salesman.

Vendor

A vendor is typically the manufacturer of a given product. Each product may be associated to a specific Vendor. The vendor construct may be used for pricing when the idea is to markup from cost a specific percentage by vendor.

WebSite

While much of the data in the system is global, the WebSite class is used to define various rules and options for a given web site. Licensing is also enforced at the web site level using the URL of the site.

It is expected that each web site has its own URL landing page. Typical uses of different web sites in the Sitecore Ecommerce Enterprise Edition perspective is a public-facing site and a private site. The public site (often called a B2C site) is used for casual users to come into the system and look at products but may or may not be able to purchase them. It is common to leave various bits of data, such as current inventory levels, off the product information pages in these sites.

The flip side to the public site is a private site. These are typically called B2B and require a login to gain access to any of the pages in the site and certainly to place orders. Since the audience is quite different for these two types of sites, it is common to have different functionality in addition to difference content.

There are several parameter options associated with the web site that you should reference the Management Console Reference Guide for additional information.

While some classes are built directly in context of the site, others are simply assigned at the site level. The following are the constructs associated with a site:

- Carrier
- Category
- Country

- CrossSell
- NewsArticle
- Promotion
- State
- WebPage
- WebPageContent

WishList/WishListProduct

The concept of the Wish List is a list of products that a given user is interested in purchasing some time later. Since the system supports multiple wish lists, it is possible to use this construct to hold repeating order information, top-10 items to purchase, etc. It is basically nothing more than a list of products with an optional quantity and its associated User.

The WishListProduct is what stores the actual product that's on the wish list, along with a quantity.

There is no standard logic to decrement the quantity of a wish list once a user adds a wishlist item to their cart. Likewise, there is nothing that will automatically notify a user that a wishlist item is in stock or out of stock – these are all functions that the developer may add to their site as needed.

Chapter 5: Integration

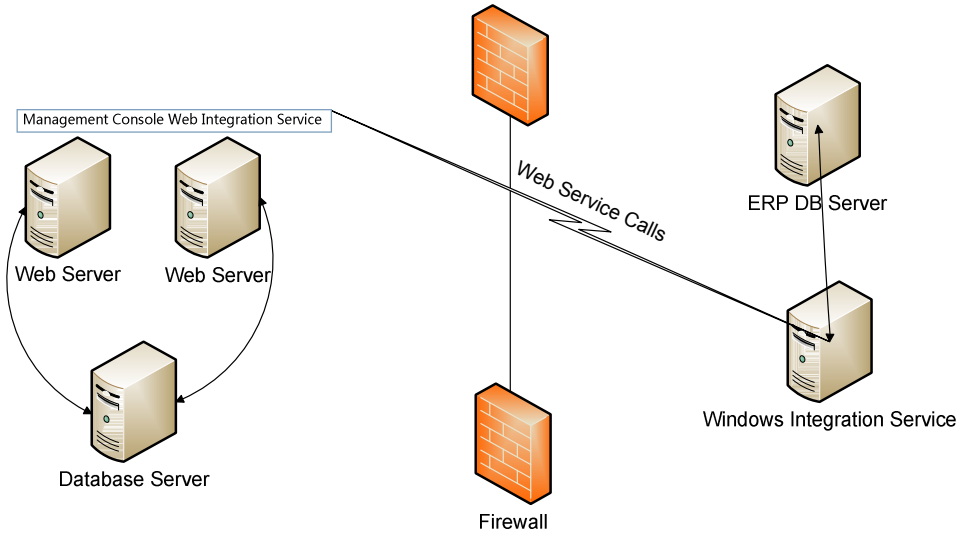
In this chapter we will provide an overview of the structure of integration including:

- ✓ General flow of integration
- ✓ Windows Integration Service
- ✓ Integration Web Service

General Integration Flow

The following diagram shows the general flow of integration when there is a direct database connection on the Windows Integration side. In the case where a web service is called from within the Management Console, the flow is different.

Integration Architecture



Basic Workflow

There are two primary components to the integration process:

- Windows Integration Service – sits behind the firewall and controls communications between the web site and the ERP. It basically *wakes up* every 30 seconds and checks for work to do from the web server.
- When the service initially connects, it checks for a newer version of itself from the web server. If there is a newer version of the integration DLL, it will be unloaded from memory in the Windows Service, pull down the new one, and restart the DLL. This allows for in-line updates of integration from the web

server directly without having to get on the client machine and perform updates.

- Web Integration Service – this is a web service that is connected to the Management Console and responds to requests made by the Windows Service.

Refreshes

A **Refresh** is our generic term indicating that data is retrieved and synched up between the ERP and the web site. The basic approach is to pull a full set of data from the ERP using some parameterized queries and then send this data up to the web site. On the web site, there will be a specific, targeted refresh class (i.e. CustomerRefresh, ProductRefresh, DealerRefresh, etc.) which will take this data that should come up in a standardized format and update the SEEE database.

As the synch/refresh process is running, the integration service quietly waits until it gets a signal that it is complete and then moves on to process the next request. Specific refreshes may also be controlled by an application setting indicating to perform the refresh.

Looking at something like the ProductRefresh, there are some typical business rules such as:

- If the application setting ERP_RefreshProducts is set to FALSE, skip the refresh
- If the product is **not** ERP Managed, then the data from the ERP is ignored
- If the product does not come up in the dataset and it is ERP managed, it will be marked as deactivated on the date of the refresh in the web database
- There is an application setting (ERP_ RefreshProductColumnsNotToOverwrite) that defines specific columns to retain, regardless of the data coming back from the ERP

There are many application settings that help control the flow and behavior of integration, refer to the Sitecore Ecommerce Enterprise Edition AppSettings documentation for more information about specific application settings.

As refreshes are run, they will typically log their progress in the Application Log.

Almost all integrations are data pulls with the specific exceptions of OrderSubmit, which will often create customers or ship-to data and the order and SubmitPayment when the credit card is charged immediately on the order.

Primary Classes

There are several tables and classes related to the integration process. Besides numerous application settings, the following classes help control what happens – most of which are in Insite.Domain:

Insite.Integration is the project is used to contain base/standard integration components and then each of the ERP-specific integrations. These ERP-specific code

files will control some of the order submission control and defaults used specific to an ERP.

Insite.IntegrationService is the Windows service that calls the Web Service.

XXXRefresh: these classes (InventoryRefresh, DealerRefresh, etc.) are designed to take the provided dataset and transcribe it into the data structures through the business model.

ScheduledTask stores the work to be completed as requested by the web site. Repetitive tasks, such as the ProductRefresh will have a RunDateTime and LastRunDateTime and a MinuteInterval property which is the number of minutes before running the next iteration (typically 1440, which is 24 hours). The system will not run the integration until the RunDateTime and, once that is hit, it will add the minute interval to the RunDateTime to ensure that it is time to run the integration again by looping through the date, adding the minute interval until the date/time is greater than the current date/time.

There is also a parameter property to the scheduled task which is typically used to process a single entry. For example, the entry type SubmitOrders will normally have the order # being submitted. When an order is submitted on the web site, for example, an entry is created in the scheduled tasks table which is then picked up the next time the Windows Service kicks off.

TaskBuilder: Typically this is created per ERP system and contains a method for each of the refreshes that are supported. The list of tasks from ScheduledTask are passed into the TaskBuilder which then processes additional information to break down specific sets of SQL to run on the Windows Service side to retrieve various sets of data. For example, RefreshProduct may retrieve a product masterfile, product pricing, product cross-sells, product categories, etc. This is entirely dependent on the specific integration code.

Creating your own custom integration

Inversion of Control has been implemented within the integration process as well. In order to create your own integration service on the Windows Integration side, do the following:

- Create a Visual Studio project called CustomIntegration (the DLL being looked for will be explicitly CustomIntegration.dll)
- Set a reference to the Insite.Integration dll
- Create a new class inheriting from IntegrationBase
- When you compile the solution and deploy it, the Windows Integration Service will look for the custom integration and use it

Technical Overview – Windows Integration Service

InSite.IntegrationService Project

- Is the Windows Service.
- Has a reference to InSite.IntegrationBroker.
- **Class IntegrationService**
  - Has instance of an IntegrationSite for Production and Pilot getting URLs from the app.config as Production_Url and Pilot_Url.
  - OnStart (called when the Windows Service starts up) creates a timer for Production and a separate timer for Pilot with default interval set to thirty seconds if not specified in app.config as Production_TimerInterval and Pilot_TimerInterval, on elapsed, calls DoSiteWorkOnTimer for Production and Pilot.
  - DoSiteWorkOnTimer calls Production or Pilot IntegrationSite instance DoWork method (depending if it's the Production or Pilot timer that is elapsing).
- **Class IntegrationSite**
  - Creates a new AppDomain named IntegrationDomain pointing to the SiteName (Production or Pilot) sub-directory.
  - Has an instance of IntegrationBroker.RemoteIntegrationControl named Handler created in new AppDomain (IntegrationDomain).
  - Has an instance of the UpdateService.
  - Contains DoWork method which first checks for updates calling the UpdateService FindUpdates method, if there is an update available, unloads the IntegrationDomain AppDomain and downloads the updated dll (the IntegrationDomain property when called will re-load the AppDomain), then calls the UpdateService UpdateComplete method. Then it calls IntegrationBroker.RemoteIntegrationControl (Handler) PerformIntegration method.

InSite.IntegrationBroker Project

- Declares IIntegrationControl Interface that contains SiteUrl Property and PerformIntegration method.
- **Class RemoteIntegrationControl**
  - Has instance of IIntegrationControl that loads InSite.Integration dll and puts InSite.Integration.IntegrationControl in to Windsor Container.
  - Contains PerformIntegration method which calls IntegrationControl PerformIntegration method.
  - Contains InitializeLifetimeService method override which returns null. As handles are cached for a long period of time, return null so they don't expire.

InSite.Integration Project

- Has reference to InSite.IntegrationBroker.
- **Class IntegrationControl**
  - Implements InSite.IntegrationBroker.IIntegrationControl.
  - Has UserName and Password properties
  - Has instance of IntegrationWebService.IntegrationService.
  - Contains PerformIntegration method which calls IntegrationService.GetScheduledTasksForMachine returning an array of IntegrationTasks, then calls PerformTasks passing in that array.
  - Contains PerformTasks method which loops through the IntegrationTask array passed to it. If it is a SystemPause task it sleeps for the time specified, otherwise it calls GetTaskDataSet passing in the task.
    - If the DataSet returned **is not null**, it uses reflection to get the web method to call which is specified in the task.RefreshType and passes the DataSet in to that web method. For example, if the task.RefreshType is "RefreshCustomers", the GetTaskDataSet would return the DataSet of Customers and this would call the web method RefreshCustomers passing in that DataSet.
    - If the DataSet returned **is null**, it calls IntegrationBase.Build passing in the task and reference to IntegrationService. IntegrationBase.Build is a factory method that uses the task.ERPType to create the correct ERP specific instance of the ERP component giving it the task and reference to IntegrationService in the constructor. It then sets the UserName and Password of the returned object and calls PerformTask on it.
  - Contains GetTaskDataSet method which checks the task.ERPType, if it's odbcsql, it creates an OdbcDataAdapter, if sqlsql, it creates a SqlDataAdapter, then passes that and the task.Queries to IntegrationBase.GetData which returns a DataSet. GetTaskDataSet then returns that DataSet.
- **Class IntegrationBase**
  - Is the abstract base class for all the ERP specific components.
  - Contains constants defining the task.RefreshTypes. Any changes to these require corresponding changes to InSite.Model.ScheduledTask according to the comments.
  - Contains abstract PerformTask method that must be implemented in ERP component.
  - Contains Build factory method which returns the ERP component as an IntegrationBase.
  - Contains GetData method which accepts a DbDataAdapter and array of IntegrationQueries. It loops through the array of queries and uses the

data adapter to fill a DataSet, then loops through the query.keys to add primary keys to the DataSet's DataTables.

- **Class <ERP>Integration (where <ERP> is the ERP implemented, for example VisualIntegration)**
  - Inherits from IntegrationBase
  - Must contain PerformTask override method. The standard refresh tasks are normally handled automatically by IntegrationControl, but OrderSubmit and ExecuteQuery is typically implemented in this class. In PerformTask, the IntegrationTask.RefreshType is checked and the corresponding logic is executed. In some cases like Vantage where we call web services for everything, everything is implemented in this class.

Management Console Side

InSite.Management Web Project

- Contains the Web Services used by the Windows Service.
- All integration service web methods require a user name and password. The update service methods do not.
- **UpdateService.asmx / App_Code\UpdateService.cs Class**
  - Contains FindUpdates web method which goes through the /Updates folder and returns the file names and the URL to the files in an array of the InSiteUpdate class.
  - Contains the UpdateComplete web method which accepts a file name and deletes that file from the /Updates folder.
- **IntegrationService.asmx / App_Code\IntegrationService.cs**
  - Contains the refresh web methods (RefreshCustomers, RefreshCustomerOrders, RefreshDealers, RefreshProducts, RefreshSalesmen, RefreshShipments, RefreshInvoices) that accept DataSets and perform the refresh logic.
  - Contains GetOrdersToSubmit web method that returns a DataSet of all orders whose status is Submitted.
  - Contains GetOrderToSubmit web method that accepts an order number and returns a DataSet for that order.
  - Contains GetOrderProducts web method that accepts an order number and returns a DataSet with that orders Products, ProductCosts and Vendors.
  - Contains SetOrderToSubmitted web method that accepts an order number and sets that order's status to Review or Processing depending on the ApplicationSetting SubmitToReviewStatus.
  - Contains GetPaymentsToSubmit web method that returns a DataSet of all CreditCardTransactions whose status is Submitted and not yet SubmittedToERP.

- Contains SetPaymentToSubmitted web method that accepts a CreditCardTransactionId and sets that CreditCardTransactions status to SubmittedToERP.
- Contains ExecuteQuery web method that accepts a DataSet and query (String type) parameters and creates an instance of InSite.Domain.ERPIntegration.ExecuteQuery and sets its DataSet property to the DataSet sent in and calls its Execute method passing in the query parameter. The method is named ExecuteQuery, but the idea is that we are executing a query in the ERP on the Windows Service side and returning the results from that query here.
- Contains GetScheduledTasks web method which gets a list of all scheduled tasks whose RunDateTime is less than the current date time and whose LastRunDateTime is null or it plus it's minute interval (if not zero) is less than or equal to the current date time. Goes through that list of tasks and sets their LastRunDateTime to the current date time then flushes the nHibernate session. Then it calls TaskBuilderBase BuildSystemTasks that accepts the list of tasks and returns a list of the built system tasks that is in that list of tasks. Then it gets the task builder from the Windsor Container that is configured for that client and ERP and calls it's BuildTasks method passing in the list of tasks which returns the list of built tasks. It then checks if any of the tasks in the list of tasks is a ReSubmitOrders task and if so calls CustomerOrderRefresh ResubmitOrders. Then it returns the list of built tasks.
- Contains GetScheduledTasksForMachine web method which accepts the list of tasks and a machine name, validates that that machine name can issue requests, then passes the list of tasks along to GetScheduledTasks.
- Also contains GetApplicationSetting, GetCustomerNumberPrefix, LogError, LogInfo and DebugMessage web methods which are self documenting.

InSite.Domain Project

- ERPIntegration folder and namespace
- RefreshBase (is the base class for all the Refresh classes and ExecuteQuery)
- ExecuteQuery
- CustomerOrderRefresh, CustomerOrderUpdate_Generic and ICustomerOrderUpdate
- CustomerRefresh, CustomerUpdate_Generic and ICustomerUpdate
- DealerRefresh
- InventoryRefresh
- InvoiceRefresh
- ProductRefresh_Generic and IProductRefresh, ProductUpdate_Generic and IProductUpdate
- SalesmanRefresh

- ShipmentRefresh
- ClientSpecificTasks_Generic and IClientSpecificTasks (not used)
- IntegrationTasks folder and namespace
    - IntegrationTask
    - IntegrationQuery
    - ITaskBuilder
    - TaskBuilderBase
    - TaskBuilder<ERP>