ERP Provider Sync Specification (ERPS)

v3.0

For AspDotNetStorefront ML v8.1 or higher

(Portions released in BETA on v8.0)

Document Revision Number:	12
Last Revised By:	Dan Van Kuren
Last Revised On:	1/6/2009 6:57 PM

Table of Contents

1. ERPS Introduction 3
 1.1. Overview and Scope 3
 1.2. Objectives 3
 1.3. Operational Modes and Definitions....................3
 1.4. Performance Considerations 5
 1.5. Supported Data 5
 1.6. Reconciling Dirty/Out of Sync Data 6
 1.7. Size and Scaling....................6
 1.8. Stateless....................6
2. ASPDNSF Data Structure Review....................7
 2.1. Products and Product Variants 7
 2.2. Entities 9
 2.3. Customers and Related Data 11
 2.4. Orders 13
 2.5. Special Data Type – ML Data 14
 2.6. Order Data Field Definitions....................15
 2.7. Order Data Structure 27
 2.8. Products and Product Variants with Mappings and Related Tables (inc. Inventory) 28
 2.9. Core Customer Data 29
3. External Providers (Typically Batch Push With Event Listeners) 30
 3.1. Introduction to WSI....................30
 3.2. Event Handlers 31
 3.3. WSI Data Types....................33
 3.4. Updating and Retrieving Data Using WSI....................35
 3.5. Code Sample: Communicating with WSI 35
 3.6. Code Sample: Event Listener Service....................38
 3.7. WSI Sample: Categories 40
 3.8. WSI Sample: Manufacturers 42
 3.9. WSI Sample: Products and Product Variants....................44
 3.10. WSI Sample: Product Inventory....................48
 3.11. WSI Sample: Customers 49
 3.12. WSI Sample: New Orders....................53
4. In-Process Providers (AspDotNetStorefront Add-In Model) 61
 4.1. Introduction to the AspDotNetStorefront Add-In Model 61
 4.2. Using Add-Ins in AspDotNetStorefront 66
 4.3. Extending AspDotNetStorefront with Add-Ins....................62
 4.4. Code Sample #1: Shopping Cart Inventory Update 62
 4.5. Code Sample #2: ERP-Provided Shipping Methods and Rates**Error! Bookmark not defined.**
 4.6. Sample Add-in Contract 64
5. Security Considerations 74
 5.1. General Security Considerations....................74
 5.2. WSI....................74
 5.3. Event Handlers 74
 5.4. Event Listeners 74
 5.5. Add-Ins....................75
6. Additional Resources 76
7. Appendix A: Considerations for Integration 77
 7.1. Customer Records 77
 7.2. Orders 77

7.3. Inventory 77

1. ERPS Introduction

1.1.Overview and Scope

This specification covers the ERP Provider Sync Specification (ERPS) v1.0 for integrating AspDotNetStorefront ML v8.1 or higher with external ERP systems.

NOTE: This spec also refers to, and relies on our Web Service Automation Interface (WSI) specification. Portions of this specification (e.g. WSI) are implemented in ML v8.0.

The ERPS is defined with the objective of allowing real-time, near real-time, and batch sync with external master ERP systems, to allow control of AspDotNetStorefront (ASPDNSF) enabled e-commerce sites.

In this specification, the ERP system is considered to be the "master" data source and ASPDNSF to be the "slave".

1.2.Objectives

In an ideal sense, the objective of this specification is to allow ERP systems and ASPDNSF to be fully integrated in real-time, near real-time, or batch mode with a master ERP management system.

At a high level conceptually, the ERP master provides (and updates) all manufacturer, category (entity), product, price, inventory and other similar data to the ASPDNSF slave. Newly created Customers and Orders in ASPDNSF are sent to the ERP master for fulfillment, handling and processing also.

In an ideal implementation, the existing ASPDNSF "admin site" (merchant control panel) would not even have to be used to fully control the ASPDNSF website. That objective is beyond the scope of ERPS v1.0 specification however.

NOTE: in particular, if they wanted to start with an "empty" storefront using in-process providers only.

A likely implementation scenario will use WSI push for bulk data population, supported by real-time (or very near real-time) in-process providers for highly dynamic, and typically incremental, data updates (e.g. push ERP catalog from master using WSI, but provide in-process provider for real-time price and inventory status).

1.3.Operational Modes and Definitions

ERPS should support external synchronization with:

a) **External Subsystems**, typically used for push and batch type update via WSI, and
b) **In Process Pluggable Components**, for maximum real-time pull on demand data during web page load and execution. Note the in-process provider as nearly full access to the ASPDNSF application, server, web page, and db state, allowing a number of techniques to be utilized

Data directional modes include both:

a) **Push Provider:** injecting data from ERP system into ASPDNSF, typically implemented along with event listeners, and

b) **Pull Provider:** ASPDNSF pulls data on demand, typically using in-process pluggable component

Typically, push mode will be used when batch updating ASPDNSF data from the master ERP system, either in real-time as it's changed in ERP or through a timed or batch period bulk update. Typically push mode will run through our Web Service Automation Interface (WSI, see separate spec). Typically pull mode will be made as ASPDNSF makes real-time calls to in-process plug-in providers. Note that pull mode doesn't have to be real-time, it could also include in-process (and even in database) caches that the pluggable provider does to maximize performance, and minimize latency on calls to the master ERP system.

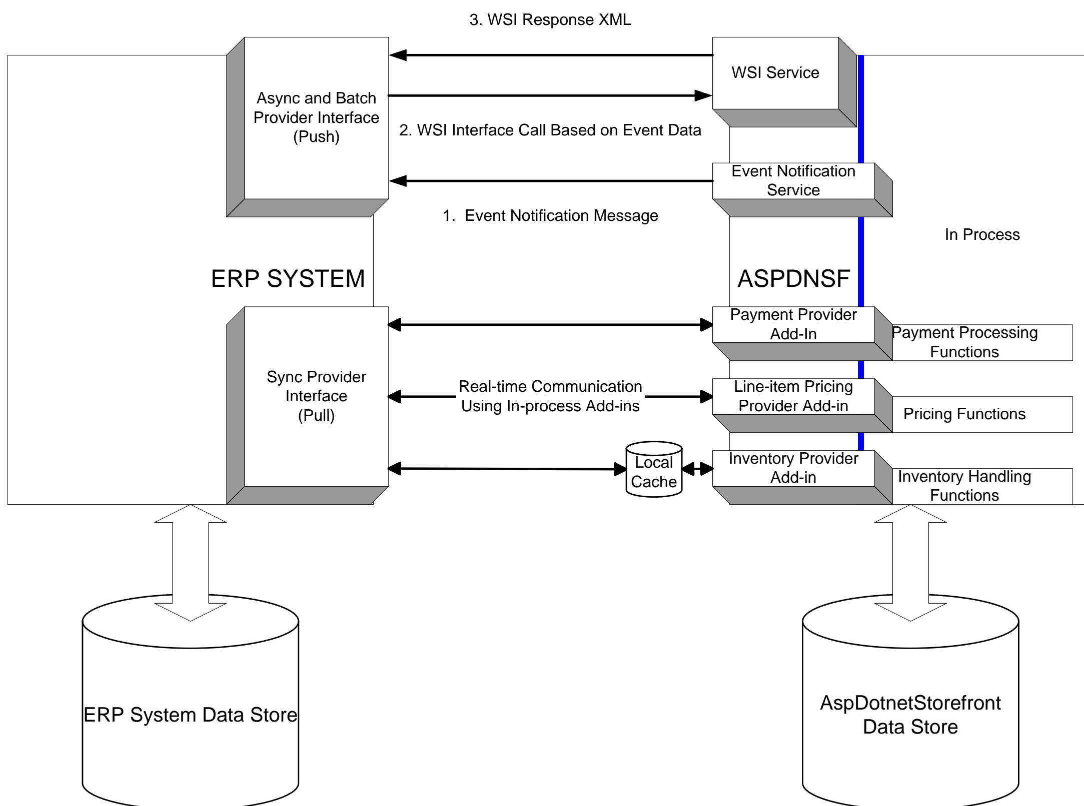


Figure 1: Integration Model Diagram

Synch Types supported include:

a) **Batch**: this typically is to be used when master ERP system makes bulk WSI calls to update ASPDSF data

b) **Near Real-Time**: typically used when the ERP system schedules either batch push updates to ASPDNSF or the pluggable in-process component provider implements caching or other times internal updates, simulating near real-time caching of master ERP data, available for maximum performance by ASPDNSF, and

c) **Real-Time**: this typically we be either a push on save in ERP direct to ASPDNSF (possibly also triggering ASPDNSF internal cache flush or update, either in part or

total), and/or pull calls from ASPDNSF to a highly tuned in process provider, making efficient calls to the master ERP system for instant data.

We also assume at this time, that no db schema changes are planned, or anticipated in ASPDNSF. This may mean that the ERPS providers have to do some data shaping or conversion to match the internal ASPDNSF data structures.

1.4.Performance Considerations

Our internal goal is to complete website construction (base page) in 50ms or less. This scales well given proper hardware on high traffic sites, and provides crisp page response. The initial page construction involves whatever server side is needed to completely send HTML back to browser on initial request. This then allows the browser to start queuing the external files (images, js files, external css, flash, etc) so that the full page load can be given in 1-2s maximum.

Given this stringent demand on website page loads for maximum performance, considerations are made in all cases to allow the provider to tune the sync mode in each integration point to match goals of website performance. This usually does mean however that sacrifices are often made in architectures (such as ERPS) to fully prioritize performance over the most "ideal" API (or subsystem) architecture, as all communications with external subsystems involves latency, which can be quite substantial, as multiple ERPS calls will typically be done multiple times on EACH page load of the website.

We would also expect that some websites would run with near real-time data, and others may run with full real-time data, allowing flexibility to merchant business requirements.

Additionally, different data sets could be synced differently. E.g. inventory checks may be real-time, and product definition (description, SKU, image, name, price, etc) updates are done in near real-time modes.

The concurrent use of different sync modes by data is supported.

We also assume, for simplicity in this specification, that the ERP system is "nearby" to the ASPDNSF website (actually the ASPDNSF SQL database), for minimum latency on communications. Provisions are made using batch, near real-time, and via caching, to allow for slower connections. Those details typically are involved in implementation of the ERPS providers, and are covered only briefly in this specification.

1.5.Supported Data

ERPS should support control of the following data:

- Manufacturers
- Categories (entities)
- Products
- Variants
- Pricing
- Inventory
- Images
- Shipping Rules (e.g. shipping provider)
- Tax Rules (e.g. tax provider)
- Payment Provider (e.g. pluggable gateway, providing auth, capture, void, refund, etc)
- Customers

- Orders

1.6.Reconciling Dirty/Out of Sync Data

AspDotNetStorefront will make the assumption that, whenever data becomes out of sync, the ERP system will control the master data set. Because of the critical nature of ERP System data, the ERP system will make the final determination as to what happens to irreconcilable changes made via the AspDotNetStorefront website. For example, an account representative may make updates to an order record in the ERP system (such as voiding an order placed by a customer via the website). Should for some reason the call to the website fail to update the order state, one would not desire that AspDotNetStorefront "re-open" the voided order. Therefore, during any synchronization process, the order state (using our previous example) would be set to a voided state based on data received from the ERP system.

1.7.Size and Scaling

Primary consideration for size and scaling would be the efficiency and design of the data sent to be consumed by AspDotNetStorefront, specifically in cases of bulk updating large numbers of objects from the ERP system. For scalability, systems integrators should consider failure scenarios when writing code to determine how those scenarios would be handled and later reconciled. For example, it would be ill-advised when using WSI to build a document that contains 100,000 nodes and try to process those all at once where a failure might occur at 99% completion. Instead, it is advised break the transactions into small batches where the chance of failure of would be exponentially smaller, and the subsequent rollback and resubmission would put far less load on both systems.

1.8.Stateless

We also assume, in nearly all cases, that each and every call, to a web page is completely stateless, as properly defined with web protocols. We do cache ASPDNSF application state in some cases, but that is ONLY done for performance, and with proper access protocols to update cache internally.

The consequences for this is that an API specification like ERPS must be assumed to be 100% stateless also, which typically means more overhead, as nothing can be assumed from a prior ERPS API call, page invocation, or customer/session state.

This can lead to what appears to be additional data passed around, or additional overhead in fulfilling ERPS API calls, but again, this is done for stateless preservation, minimizing any assumptions being built in that assume a stateful operation.

Note that this DOES NOT prevent the ERPS plug-ins, or external subsystems, from being stateful, if they support that, and can manage state properly, in the stateless calls back to ASPDNSF. Typically, ERPS add-on components or subsystems will do internal caching, to minimize communications latency and maximize performance. Obviously, caching must be done carefully, and with full knowledge and awareness of the ASPDNSF stateless requirement stated here.

2. ASPDNSF Data Structure Review

To help provide the base to understand what data and information is being synchronized a review of the primary ASPDNSF database elements is given here. This data provides the live ASPDNSF data to the website. A review of this data is helpful when writing ERPS providers.

Also, ASPDNSF uses a SQL Server database (SQL 2005+); the database (db) either co-located on the web server, or on LAN-accessible SQL Server. That architecture is assumed for ERPS also (e.g. ASPDNSF will continue to make direct database calls using the existing schema. The in-process pull providers can override, or provide near real-time, or real-time, updated data however). At this time, it is not possible to entirely sever ASPDNSF from its database to make provider calls for all data, nor would doing so be feasible from a performance standpoint.

For integration purposes, the primary objects of interest are customers, addresses, products (and their variants), categories and departments (entities used to organize products), and order information. The following overview will assist a systems integrator in determining what fields and relationships would need to be accounted for when integrating with AspDotNetStorefront. Database diagrams are also provided later in this section to provide a visual reference, however it is important to note that the primary key definitions shown in the diagrams are logical. AspDotNetStorefront enforces referential integrity within code, not at the database level. Because some of the diagrams contain many tables and are impractical for print purposes, an ERD file is also provided that can be restored on any computer with SQL 2005 or later installed (including express versions).

2.1.Products and Product Variants

Each purchasable item on an AspDotNetStorefront e-commerce website comprises of a single product variant, comprised of its own unique data along with data from its parent product. To describe the relationship between a product and its variants, we'll use the following example:

A store owner may wish to sell the movie "The Lion King" in their online store, however "The Lion King" can be purchased in either DVD or Blu-Ray format. Traditionally, a store owner would have to create two separate and distinct products (one for the standard DVD and one for the Blu-Ray DVD). This can be time consuming and unwieldy however, as the vast majority of the details would be duplicated for both products. For instance, both the DVD and Blu-Ray products would likely have the exact same product description, images, and product settings (e.g. is the product downloadable, is it a kit item, or whether or not you can purchase it through Google checkout).

Looking at the same scenario using the AspDotNetStorefront product/variant model, the situation is simplified. A store owner would create one product, generically called "The Lion King." The description, images, and general options would be configured at the product level. Once this was completed, the store owner would then create two variants for "The Lion King," (DVD and Blu-Ray). All pricing is assigned at the variant level, so the fact that the Blu-Ray disc is more expensive doesn't cause any problems. Product variants also allow the store owner to append-to (or fully set) data such as the product SKU and manufacturer part number. For example, both items might share the same SKU prefix (TLK845), but have a different suffix (such as SD for standard definition DVD and BR for Blu-Ray). Using the SKU suffix field at the variant level, when a customer ordered "The Lion King" DVD variant, the SKU would end up being TLK845SD.

Every product must have one variant, but can have many if necessary. It is up to the store owner to decide how they wish to organize and consolidate their products.

Product variants also support modifiers, which by default, are sizes and colors. To look at a bit more complex product, we'll use a shirt. A store owner may have the same basic shirt that comes in two versions (long and short-sleeved), and each version of the shirt can be purchased in multiple sizes and colors. Since variants support variant-level attributes for sizes and colors, this scenario is fully supported. Sizes and colors can be further extended to also modify the product/variant SKU as well as affect the price of the final purchased variant. Using the above shirt, the store owner can specify that big and tall sizes are $2.00 more than the standard price, and that the shirt is $1.00 cheaper when purchased in white. Size/Color SKU modifiers can also be defined so that the final purchased item SKU is SHIRT-LS-XL-RED (where SHIRT is the product SKU, -LS is the variant SKU suffix for long sleeves, -XL is the size SKU modifier, and –RED is the color SKU modifier).

If your product has no more than two customer-selectable attributes other than size and color, the size and color fields can also be used to store these other attributes. By changing the Size and Color Option Prompts at the product level, the store owner can determine what these attributes are called. For example, if a store owner sold a picture frame, they might want to allow the customer to select the finish of the wood used in the frame. The store owner would change the Color Option Prompt for their picture frame product to "Finish," and set the color options to Cherry,Oak,Natural Maple. When viewing the product page, the customer would be presented with a dropdown titled "Finish" that allowed them to select one of those options.

Inventory tracking is done either at the variant level, or at the variant size/color combination level. This option is specified by setting the "Track Inventory by Size and Color" option at the product level. When inventory is tracked by size and color, the inventory records for each size/color combination are stored in the Inventory table. Otherwise, inventory is stored in the variant table in the Inventory field.

Tracking inventory by size and color also enables a feature called VendorFullSKU. The VendorFullSKU can be used to assign a completely unique SKU to a size/color combination as opposed to relying on modifiers to create the SKU. This is helpful in cases where SKU numbers for size/color options do not follow a set naming convention.

In addition to multi-variant products, AspDotNetStorefront provides the capability to create "kit" products. A kit product is an item that contains many complex options that may or may not change the price and weight of the final purchased product. A good example of a kit product would be a computer. A computer manufacturer may offer a base system (eg. an Intel Core2-based computer), however the system has many user-configurable options associated with it. For example, the user might be asked to choose from a list of one of four different processor types, select the amount of memory they need, choose whether or not they want to receive a monitor with the system, select what size hard drive they want in the computer, and pick none, one, or many OEM software titles they wish to be shipped with the system. Kit items offer the flexibility of doing just that.

When creating kit items, the administrator first defines Kit Item Option Groups, which will contain one or many kit item options. Additionally, the administrator chooses which type of data option group will contain. The following field types are supported:

1. Single Select Dropdown List
2. Single Select Radio Button
3. Multi Select Checkboxes
4. Single-Line Text Box
5. Multi-Line Text Area

The numbers above correspond with the KitGroupTypeID in the KitGroupType table.

After defining the group, including the group type, the administrator then defines the Kit Items, each of which is assigned to a single item group, and will be displayed according to the kit group type. For example, if the store owner defines a kit group type of Single Select Dropdown List, and then adds four kit items to that group (eg. Intel Core2 Duo E4600, Intel Core2 Duo E4700, Intel Core2 Duo E6540, Intel Core2 Quad Q6600), when viewing the item the shopper will be presented with a dropdown list where they can select one of those four choices. Likewise, if the kit option group type were set to multi-select checkbox, and the administrator creates kit items assigned to that group for Symantec Antivirus and Adobe Acrobat, the user could choose to add both, one, or none of those items to their cart. Kit groups can also be marked as required, forcing the user to choose a selection should they leave a certain group blank.

Each kit item can also have an associated price delta and weight delta. The price of the finished kit will be the sum of the kit base price (as defined on the default variant in the ProductVariant table's price field, and the sum of all selected kit item price deltas). The weight delta functionality works in the same manner.

Finally, AspDotNetStorefront provides access to one or more "ExtensionData" fields in nearly every database tables. These fields are not used by AspDotNetStorefront, and not shown on the website without customization, and in some cases are not accessible via the admin site either. The purpose of the extension data fields is to store custom data defined by the store owner. Extension data fields are the prescribed way to store specific integration data, such as an ERP system primary key field to tie an AspDotNetStorefront product or variant to an inventory item in the ERP system of other non-custom data (such as the product or variant GUID) cannot be used.

Summary

- A purchasable item consists of a single product variant.
- Generic product-level options are set at the product level. This might include the name, the SKU prefix, whether the product is downloadable or not, product images, and SEO information.
- Pricing and special attributes (such as size and color) are set at the variant level
- Product variants contain fields for description and images, however these are seldom used and may require customization to display on the site.
- The final purchased SKU of an item consists of the product SKU prefix, the variant sku suffix, and any defined size/color SKU modifiers.
- Inventory can be tracked at the variant level (stored in the ProductVariant Inventory field) or tracked by individual size and color (stored in the Inventory table)

2.2. Entities

The term entity within the context of AspDotNetStorefront is used primarily to define an organization unit used to classify products for display and processing purposes. AspDotNetStorefront supports the following entities out of the box:

- Categories
- Departments (also known as sections)
- Manufacturers

AspDotNetStorefront also supports the following special entities that provide additional functionality within the product:

- Affiliates
- Customer Levels
- Distributors

The primary and frequently the sole organization unit used by store owners for display and organization purposes is the category. Categories can be nested within one another to any reasonable level, and products can be assigned to one or many categories. As an example (following our Lion King convention), a store owner may have a root-level category called "Movies." Within that root level category, they could create a sub category called "Animated Features." The Lion King product could then be assigned to the Animated Features category.

Departments operate exactly the same as categories, and simply provide an additional tool for organizing large numbers of products, making them easier to find. Our imaginary store owner may decide he also wants customers to be able to browse his/her store based on the genre of movie they are looking for. In that case, he/she could also define Drama, Action, Comedies, and Cartoons as departments within his/her store. The store owner would then assign The Lion King to the Cartoons department, in addition to being assigned to the Movies → Animated Features category.

A manufacturer is required for each product, and can be used to allow customers to view all products assigned to a given manufacturer. Store owners that do not wish to allow this level of complexity on their site can simply create a single default manufacturer and assign all products to it.

Distributors are an optional entity that can be assigned to a product on a one to basis. They are primarily used for the purpose of allowing store owners to sell "Drop-ship" items on their store. When a distributor is created, the administrator defines an email address for that distributor, and then proceeds to assign products which that distributor. When an order is placed on the site and payment is captured, the distributor will receive an email notification of any products contained in the order assigned to it. Orders can consist of items from multiple distributors, in which case each distributor will receive a notification that contains only those items to which it is assigned. Because this functionality tends to duplicate existing processes already in place within an environment using a fully functional ERP system, this functionality will likely be ignored.

Affiliates are another optional entity that tends to be used by store owners to track inbound links from other websites. An affiliate can be created via the store's admin site (or the affiliate can create their own account via the customer-facing side of the site if the administrator allows this) and is assigned an affiliate id. The administrator/store owner then provides the affiliate with a link to their site that contains the affiliate ID in the following format:

http://www.samplesuperstore.com/default.aspx?affiliateid=12345

When a shopper enters the site through that link, the affiliate ID will be cookied to that shopper. If the shopper proceeds to make a purchase or register an account on the website, the affiliate ID will be permanently stored in their customer record and any subsequent order

records. If this customer then enters the site from a different affiliate in the future, the original affiliate ID will be overwritten in their customer record with the new affiliate ID (however previously placed orders will still contain the old affiliate ID).

Customer Levels are one of the most functional special entities offered within AspDotNetStorefront. Customer levels allow a store owner to create containers to which customers can be assigned that provide special terms and pricing options for those customers. Customer levels can impact pricing (either by defining flat percentage or dollar-based discount) as well as allow special payment methods such as purchase orders. Additionally, customer levels can modify tax (eg. Wholesale level customers do not pay sales tax) and shipping, and determines whether that level allows the usage of coupons and access to quantity discounts. The aforementioned customer level functionality would be duplicated by the add-in pricing provider written by the system integrator, and it is strongly recommended that these feature not be used to prevent conflicting pricing, tax, and shipping logic.

One of the most powerful features of customer levels that would be appropriate when integrated with a third party system is the ability for an AspDotNetStorefront website to filter products by customer level. Products may be mapped to one or many customer levels. When mapped to any customer level, and the store owner chooses to "FilterProductsByCustomerLevel," that product will only be visible to a customer that is assigned to a customer level to which that product is mapped. This allows a store owner to effectively hide products on their site from everyone except those customers allowed to view them.

Summary

- The majority of stores use categories alone to organize their products
- Products can be assigned to one or many categories
- Departments mimic the functionality of categories, and can be used if an extra level of organization is required
- A product must be assigned to a manufacturer, but stores that do not wish to use this feature can simply assign all products to a single generic manufacturer
- Distributors are optional and used for "drop-ship" items. Distributors likely mimic processes or capabilities already in place within the ERP system
- Affiliates are used primarily to track inbound links to the website
- Customer levels offer a variety of capabilities, however it is strongly recommended that they be used only for filtering products when integrated with another system that provides pricing, shipping, and tax rules

2.3.Customers and Related Data

Customers and their related database tables are the primary method for tracking shoppers on the website, as well as creating and fulfilling orders. Before delving into customers in depth, it is important to understand that there are two types of customers within AspDotNetStorefront (registered and anonymous customers).

Registered customers are shoppers that have gone through the process of registering for and creating an account on the website, complete with a password, allowing them to save personal settings such as address information, track purchase history, and buy multiple times without re-submitting their personal information each time.

Anonymous customers are used first by AspDotNetStorefront to allow tracking things such as shopping cart items prior to the customer registering on the site. The first time a shopper visits the website and adds something to their shopping cart, an anonymous customer record is created in the Customer table. As the customer continues to shop, additional items are added to the cart under this record.

At some point, an anonymous customer will want to proceed through the checkout process. A few things can happen at this time. If the store administrator allows anonymous checkout on the website, the shopper will be presented with an option to log into an existing account, register an account, or continue anonymously. Should they choose to continue anonymously, the anonymous customer record is used temporarily to link things like addresses to the anonymous shopper. Upon completion of the order, the pertinent information (such as the customer's name and address) are moved into the order header record. In most cases, a third party application would have no use for anonymous customer records, and would instead want to assign those orders to an internal default anonymous customer.

If the store owner does not allow anonymous checkout, or the shopper chooses to register for an account, the shopper will be required to enter their information (name, email address, phone number, billing and shipping addresses, etc.) as well as create a password. Upon completion of the process, the anonymous customer record is promoted to a registered customer record, and the shopper proceeds with checkout.

In the event that the shopper already has an account registered on the site, the shopper can simply log in at the time of checkout, or at any other point. Upon logging in, their cart contents will be migrated to the existing customer account, and the shopping will continue shopping under their registered customer record.

The following diagram provides an overview of the use of customer records within AspDotNetStorefront.

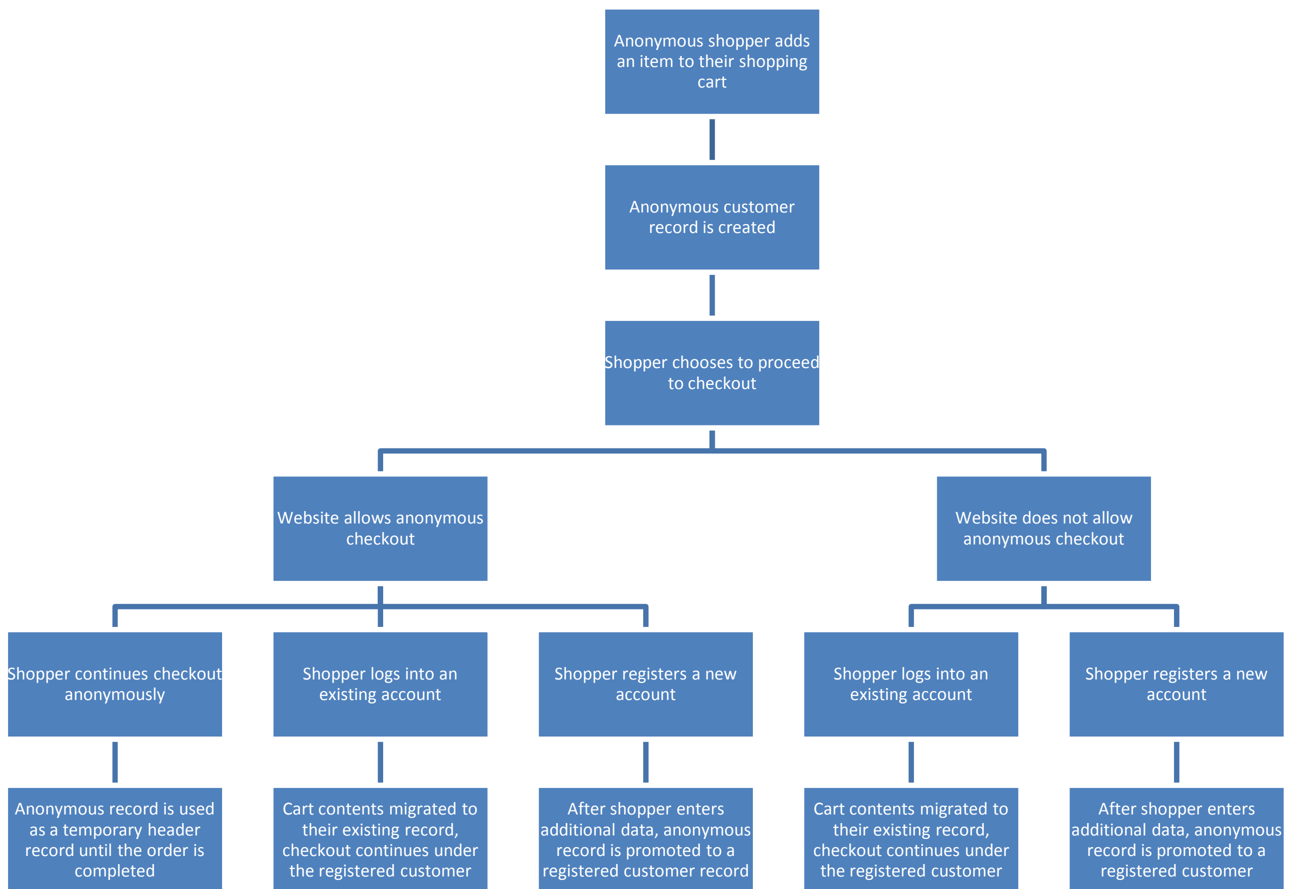


Figure 2: Customer Record Creation

The important thing to keep in mind with customer records is that in most cases, anything in the customer table with the IsRegistered flag set to zero or null should be ignored. These are anonymous records that may or may not be linked to any valid transaction. More so, since any customer adding an item to the cart creates a customer record, this can lead to thousands (or more over time) of anonymous records with little or no useful data being added to the ERP system database. AspDotNetStorefront contains a monthly maintenance function used for the purpose of pruning these records.

2.4.Orders

Along with customers, orders comprise the bulk of the information returned to the ERP system for integration purposes. We will briefly discuss the structure of AspDotNetStorefront orders here, and later provide examples of orders in XML format.

Orders consist of a header record (stored in the Orders table) along with individual line item records (stored in the Orders_Shoppingcart table). Additionally, line items may have additional data in the form of kit item details stored in the Orders_KitCart table. Upon creation, all details for an order (including billing and shipping addresses, product details and pricing, and customer information) are stored within the orders table. This prevents having to join "live" tables when displaying order data (which may no longer be available, or may have been changed since the order record was created).

2.5.Special Data Type – ML Data

For the purpose of storing Multi-lingual data, AspDotNetStorefront supports a special ML-data type. This data type uses an XML fragment to store data for multiple languages within a single database table field, which can be queried from XML (eg. a WSI data document) using standard xpath queries.

An example of data contained within an ML field would appear as:

```
<ml>
   <locale name="en-US">en us value</locale>
   <locale name="ja-JP">ja jp value</locale>
</ml>
```

2.6.Order Data Field Definitions

Table Name: Orders

ColumnName	Datatype	Length	Nullable	DefaultValue	Is ML Field	FieldDescription
OrderNumber	int	4	0			
OrderGUID	uniqueidentifier	16	0	newid()		
ParentOrderNumber	int	4	1			Used when generating adhoc orders to link an adhoc order to the order it was created from
StoreVersion	nvarchar	50	1			The version of the store that the order was created under
QuoteCheckout	tinyint	1	0	0		Whether the order was generated as a quote
IsNew	tinyint	1	0	-1		Determines whether the order shows in the list of new orders on the orders page
ShippedOn	datetime	8	1			Date the order was marked as shipped by an administrator
CustomerID	int	4	0			CustomerID (from the customer table) of the customer who placed the order
CustomerGUID	uniqueidentifier	16	0			GUID of the customer who placed the order
Referrer	ntext	16	1			Used to track the referring URL of the site that the customer orginally linked to the store from
SkinID	int	4	0	-1		SkinID of the customer who placed the order
LastName	nvarchar	100	1			Customer last name
FirstName	nvarchar	100	1			Customer first name
Email	nvarchar	100	1			Customer email
Notes	ntext	16	1			Any notes migrated from the customer record on checkout
BillingEqualsShipping	tinyint	1	0	0		Determines whether the customer chose to use the same address for billing and shipping purposes
BillingLastName	nvarchar	100	1			Last name from the customer's billing address record in the address table
BillingFirstName	nvarchar	100	1			First name from the customer's billing address record in the address table
BillingCompany	nvarchar	100	1			Company from the customer's billing address record in the address table

BillingAddress1	nvarchar	100	1		Address1 from the customer's billing address record in the address table
BillingAddress2	nvarchar	100	1		Address2 from the customer's billing address record in the address table
BillingSuite	nvarchar	50	1		Suite from the customer's billing address record in the address table
BillingCity	nvarchar	100	1		City from the customer's billing address record in the address table
BillingState	nvarchar	100	1		State abbreviation from the customer's billing address record in the address table
BillingZip	nvarchar	10	1		Postal code from the customer's billing address record in the address table
BillingCountry	nvarchar	100	1		Country name from the customer's billing address record in the address table
BillingPhone	nvarchar	25	1		Phone number from the customer's billing address record in the address table
ShippingLastName	nvarchar	100	1		Last name from the customer's shipping address record in the address table
ShippingFirstName	nvarchar	100	1		First name from the customer's shipping address record in the address table
ShippingCompany	nvarchar	100	1		Company from the customer's shipping address record in the address table
ShippingResidenceType	int	4	0	0	ResidenceType from the customer's shipping address record in the address table (eg. Residential or Company) - used in shipping calculations
ShippingAddress1	nvarchar	100	1		Address1 from the customer's shipping address record in the address table
ShippingAddress2	nvarchar	100	1		Address2 from the customer's shipping address record in the address table
ShippingSuite	nvarchar	50	1		Suite from the customer's shipping address record in the address table
ShippingCity	nvarchar	100	1		City from the customer's shipping address record in the address table
ShippingState	nvarchar	100	1		State abbreviation from the customer's shipping address record in the address table
ShippingZip	nvarchar	10	1		Postal code from the customer's shipping address record in the address table

ShippingCountry	nvarchar	100	1			Country name from the customer's shipping address record in the address table
ShippingMethodID	int	4	0	0		ID of the shipping method chosen by the customer from the ShippingMethod table
ShippingMethod	ntext	16	1		Yes	Full Xml Field Copy - Shipping method chosen by the customer
ShippingPhone	nvarchar	25	1			Phone number from the customer's shipping address record in the address table
ShippingCalculationID	int	4	1			Corresponds with the calculation type (eg. realtime rates) from the ShippingCalculation type used when the order was placed
Phone	nvarchar	20	1			Phone number from customer table
RegisterDate	datetime	8	1			Date customer registered their account on the site (or anonymous customer record creation date)
AffiliateID	int	4	1			ID of the affiliate (if any) the customer entered the site through
CouponCode	nvarchar	50	1			Coupon code entered when the order was placed
CouponType	int	4	0	0		0, 1, or 2. 0 = Order level coupon, 1 = product level coupon, 2 = gift card
CouponDescription	ntext	16	1			Description specified when the coupon was created
CouponDiscountAmount	money	8	1			The (dollar) discount amount (if any) of the coupon used on the order
CouponDiscountPercent	money	8	1			The percentage discount amount of the coupon used on the order
CouponIncludesFreeShipping	tinyint	1	1			Determines whether the coupon used on the order allowed free shipping
OkToEmail	tinyint	1	1			Whether the customer is "Ok to email" based on their preference set in the customer record
Deleted	tinyint	1	0	0		Soft delete flag
CardType	nvarchar	20	1			Credit card type from CreditCardTypes table
CardName	nvarchar	100	1			Name customer entered during payment processing
CardNumber	ntext	16	1			Credit card number (if stored)
CardExpirationMonth	nvarchar	10	1			Credit card expiration month
CardExpirationYear	nvarchar	10	1			Credit card expiration year
CardStartDate	ntext	16	1			Card start date (used for some european credit cards)
CardIssueNumber	ntext	16	1			Card issue number (used for some european credit cards)
OrderSubtotal	money	8	0	0		The order total minus any tax or shipping

OrderTax	money	8	0	0	Amount of tax charged on the order
OrderShippingCosts	money	8	0	0	Amount of shipping charged on the order
OrderTotal	money	8	0	0	Order total including all taxes, shipping, and discounts
PaymentGateway	nvarchar	50	1		Payment gateway used to process the order
AuthorizationCode	nvarchar	100	1		Auth code returned by the payment gateway
AuthorizationResult	ntext	16	1		The raw result of the authorization request as returned by the payment gateway
AuthorizationPNREF	nvarchar	100	1		Used by certain gateways as an additional transaction reference number
TransactionCommand	ntext	16	1		The data sent to the gateway to obtain the initial authorization
OrderDate	datetime	8	0	getdate()	Date order record was created
LevelID	int	4	1	0	Customer level ID of customer placing the order
LevelName	nvarchar	100	1		Full XML Field Copy - Name of the customer level ID of the customer placing the order
LevelDiscountPercent	money	8	1		The amount of discount as a percent of the order total (if any) provided by the customer level under which the order was placed
LevelDiscountAmount	money	8	1		The amount of discount in dollars (if any) provided by the customer level under which the order was placed
LevelHasFreeShipping	tinyint	1	1		Determines whether the customer level under which the order was placed has free shipping
LevelAllowsQuantityDiscounts	tinyint	1	1		Determines whether the customer level under which the order was placed allows quantity discounts
LevelHasNoTax	tinyint	1	1		Determines whether the customer level under which the order was placed is charged tax
LevelAllowsCoupons	tinyint	1	1		Determines whether the customer level under which the order was placed allows coupons to be used
LevelDiscountsApplyToExtendedPrices	tinyint	1	1		Determines whether the customer level under which the order was placed allows customer level discounts to apply to extended prices
LastIPAddress	nvarchar	20	1		Last IP address of the customer placing the order as recorded by the site
PaymentMethod	nvarchar	100	1		Method of payment used as specified in the PaymentMethods AppConfig parameter
OrderNotes	ntext	16	1		Any order notes entered by the customer during checkout

PONumber	nvarchar	50	1		Purchase order entered during checkout (if PO's are enabled and selected by the customer)
DownloadEmailSentOn	datetime	8	1		Date the download email was sent on (only used for orders containing downloadable items)
ReceiptEmailSentOn	datetime	8	1		Date customer was sent the order receipt
DistributorEmailSentOn	datetime	8	1		Date that the distributor dropship emails were sent if the order contains items assigned to a distributor
ShippingTrackingNumber	nvarchar	100	1		Tracking number as entered or imported by the store admin
ShippedVIA	nvarchar	100	1		Carrier that was used to ship the order (entered by admin)
CustomerServiceNotes	ntext	16	1		Additional private notes entered by customer service staff directly on the order via the admin site
RTShipRequest	ntext	16	1		XML request sent to the shipper(s) to retrieve shipping rates when using realtime shipping
RTShipResponse	ntext	16	1		XML response returned by the shipper(s) containing rates when using realtime shipping
TransactionState	nvarchar	20	1		The result of the initial payment gateway call as returned by the gateway
AVSResult	nvarchar	50	1		Result of the address verification check returned by the gateway
CaptureTXCommand	ntext	16	1		Command sent to the gateway when processing a capture
CaptureTXResult	ntext	16	1		Raw response returned by the gateway when processing a capture
VoidTXCommand	ntext	16	1		Command sent to the gateway when processing a void
VoidTXResult	ntext	16	1		Raw response returned by the gateway when processing a void
RefundTXCommand	ntext	16	1		Command sent to the gateway when processing a refund
RefundTXResult	ntext	16	1		Raw response returned by the gateway when processing a refund
RefundReason	ntext	16	1		Admin-entered reason for processing a refund on the order
CardinalLookupResult	ntext	16	1		Used by cardinal commerce (3rd party VBV/3DSecure provider)
CardinalAuthenticateResult	ntext	16	1		Used by cardinal commerce (3rd party VBV/3DSecure provider)

CardinalGatewayParms	ntext	16	1		Used by cardinal commerce (3rd party VBV/3DSecure provider)
AffiliateCommissionRecorded	tinyint	1	0	0	Not Used
OrderOptions	ntext	16	1		Comma seperated list of order option IDs selected by the user during checkout
OrderWeight	money	8	0	0	Total weight of the order
eCheckBankABACode	ntext	16	1		ABA Code stored when using eCheck services
eCheckBankAccountNumber	ntext	16	1		Account number stored when using eCheck services
eCheckBankAccountType	ntext	16	1		Account type stored when using eCheck services
eCheckBankName	ntext	16	1		Bank name stored when using eCheck services
eCheckBankAccountName	ntext	16	1		Account name stored when using eCheck services
CarrierReportedRate	ntext	16	1		Not Used
CarrierReportedWeight	ntext	16	1		Not Used
LocaleSetting	nvarchar	10	1		Locale setting customer used when placing the order
FinalizationData	ntext	16	1		Not Used
ExtensionData	ntext	16	1		Field for containing custom user data - Not Used
AlreadyConfirmed	tinyint	1	0	0	Used on checkoutconfirmation to determine if the user has alread clicked the "Place Order" button - prevents multiple charges for the same order during browser refresh
CartType	int	4	0		Used in this case primarily to determine if the order was a normal one-time checkout or recurring autobill order. 0 = normal cart, 2 = recurring
THUB_POSTED_TO_ACCOUNTING	char	1	1	'N'	Used by THUB (third party QuickBooks integration provider)
THUB_POSTED_DATE	datetime	8	1		Used by THUB (third party QuickBooks integration provider)
THUB_ACCOUNTING_REF	char	25	1		Used by THUB (third party QuickBooks integration provider)
Last4	nvarchar	4	1		Last four digits of the credit card used to place the order
ReadyToShip	tinyint	1	0	0	Determines whether the admin has marked the order as "Ready to ship"
IsPrinted	tinyint	1	0	0	Determines whether the order has been printed

AuthorizedOn	datetime	8	1		Date order was authorized on
CapturedOn	datetime	8	1		Date payment was captured for the order
RefundedOn	datetime	8	1		Date (if any) that the order was refunded
VoidedOn	datetime	8	1		Date (if any) that order was voided on
FraudedOn	datetime	8	1		Date (if any) on which the order was marked as fraud
TrackingURL	ntext	16	1		URL to track the order as entered by the admin
ShippedEMailSentOn	datetime	8	1		Date the shipment notification was sent to the customer
InventoryWasReduced	int	4	0	0	Whether inventory has been reduced for the order already (prevents multiple inventory reductions for the same order)
MaxMindFraudScore	decimal	5	1	0	Used for integration with MaxMind fraud screening service
MaxMindDetails	ntext	16	1		Used for integration with MaxMind fraud screening service
VATRegistrationID	ntext	16	1		VAT registration ID entered by the customer
Crypt	int	4	0	-1	Not Used
TransactionType	int	4	0	0	0 = Unknown, 1 = Charge, 2 = Credit, 3 = Recurring_Auto

Figure 3: Order Date Field Definitions - Orders Table

Table Name: Orders_ShoppingCart

ColumnName	Datatype	Length	Nullable	DefaultValue	Is ML Field	FieldDescription
OrderNumber	int	4	0			Corresponding order number for cart line item record
ShoppingCartRecID	int	4	0			Unique shoppingcart record id for this line item
CustomerID	int	4	0			Customer ID of the customer that placed the order
ProductID	int	4	0			ProductID of the product purchased
VariantID	int	4	0			VariantID of the specific variant purchased
Quantity	int	4	0			Quantity of this item purchased
ChosenColor	nvarchar	100	1		Yes	Full Xml Field Copy - Color (if any) selected for the item purchased
ChosenColorSKUModifier	nvarchar	50	1			SKU modifier (if any) defined for the color purchased
ChosenSize	nvarchar	100	1		Yes	Full Xml Field Copy - Size (if any) selected for the item purchased
ChosenSizeSKUModifier	nvarchar	50	1			SKU modifier (if any) defined for the size purchased
OrderedProductName	ntext	16	1		Yes	Full Xml Field Copy - Name of the parent product for the purchased variant
OrderedProductVariantName	ntext	16	1		Yes	Full Xml Field Copy - Name of the variant purchased (if any) from the productvariant table
OrderedProductSKU	nvarchar	100	1			Full SKU (product + variant + modifiers) of the item purchased
OrderedProductManufacturerPartNumber	nvarchar	50	1			Full part number (product plus variant)
OrderedProductWeight	money	8	1			Total weight of the variant purchased
OrderedProductPrice	money	8	1			Actual price paid for the variant purchased * quantity
OrderedProductRegularPrice	money	8	1			Regular price of the variant (may be different than above due to sale prices, etc.)
OrderedProductSalePrice	money	8	1			Sale price of the variant purchased (if any)
OrderedProductExtendedPrice	money	8	1			Customer level extended price of the product purchased (if any)
OrderedProductQuantityDiscountName	nvarchar	100	1			Name of the quantity discount table (if any) applicable to this product

OrderedProductQuantityDiscountID	int	4	1			ID of the quantity discount (if any) this line item qualified for
OrderedProductQuantityDiscountPercent	money	8	1			Percentage (if any) subtracted due to quantity discount
IsTaxable	tinyint	1	0	0		Determines whether this item is taxable or not
IsShipSeparately	tinyint	1	0	0		Determines whether this item ships seperately from all other items
IsDownload	tinyint	1	0	0		Determines whether this item is downloadable
DownloadLocation	ntext	16	1			Download URL (if any) for this item
FreeShipping	tinyint	1	0	0		Determines whether this item ships for free
IsSecureAttachment	tinyint	1	0	0		Not Used
TextOption	ntext	16	1			Customer entered text for product variants that require a text option
CartType	int	4	0	0		Used in this case primarily to determine if the order was a normal one-time checkout or recurring autobill order. 0 = normal cart, 2 = recurring
SubscriptionInterval	int	4	1			For subscription items, the number of X added to the subscription, where X = the SubscriptionIntervalType
ShippingAddressID	int	4	0	0		Used for multi-ship orders to record the address ID this line item ships to
ShippingDetail	ntext	16	1			Not Used
ShippingMethodID	int	4	1			For multi-ship orders, used to determine the ShippingMethodID assigned to this line item
ShippingMethod	ntext	16	1		Yes	Full Xml Field Copy - For multi-ship orders, stores the name of the shipping method used to ship this id
DistributorID	int	4	1			The distributorID (if any) of the distributor this product was assigned to
GiftRegistryForCustomerID	int	4	1			CustomerID of the gift registry owner this item was purchased for
Notes	ntext	16	1			Any line item notes as entered by the customer
DistributorEmailSentOn	datetime	8	1			Date distributor email for this line item was sent
ExtensionData	ntext	16	1			Custom field for user defined data - Not Used
SizeOptionPrompt	ntext	16	1		Yes	Full Xml Field Copy - Product's size option prompt
ColorOptionPrompt	ntext	16	1		Yes	Full Xml Field Copy - Product's color option prompt
TextOptionPrompt	ntext	16	1		Yes	Full Xml Field Copy - Product's text option prompt
CreatedOn	datetime	8	0	getdate()		Date this record was created
SubscriptionIntervalType	int	4	0	-3		1= Day, 2 = Week, 3 = Month, 4 = Year

CustomerEntersPrice	tinyint	1	0	0		Determines whether the customer entered the price for this item (eg. a donation product)
CustomerEntersPricePrompt	ntext	16	1		Yes	Full Xml Field Copy - Customer enters price prompt for this item
IsAKit	tinyint	1	1			Determines whether this item is a kit
IsAPack	tinyint	1	1			Determines whether this item is a pack
IsSystem	tinyint	1	1			Determines whether this is a system product (eg. ad-hoc refund or MicroPay)
TaxClassID	int	4	0	-1		TaxClassID of this item from the taxclass table
TaxRate	money	8	0	0		Rate of tax charged on this item

Figure 4: Order Data Field Definitions - Orders_ShoppingCart Table

Table Name:
Orders_KitCart

ColumnName	Datatype	Length	Nullable	DefaultValue	Is ML Field	FieldDescription
OrderNumber	int	4	0			Corresponding order number for this kit cart
KitCartRecID	int	4	0			Primary key
CustomerID	int	4	0			CustomerID of the customer that placed the order
ShoppingCartRecID	int	4	0			Corresponds with the Orders_ShoppingCart ShoppingCartRecID to tie this kit item to a specific line item on the order
ProductID	int	4	1			ProductID associated with this kit item
VariantID	int	4	1			VariantID associated with this kit item
ProductName	nvarchar	255	1		Yes	Full Xml Field Copy - Name of the product this kit item is associated with
ProductVariantName	nvarchar	255	1		Yes	Full Xml Field Copy - Name of the variant that this kit item is associated with
KitGroupID	int	4	1			ID of the kit item option group this item belonged to
KitGroupName	nvarchar	255	1		Yes	Full Xml Field Copy - Name of the kit group this item belonged to
KitGroupIsRequired	tinyint	1	1			Determines whether this item belong to a group which required a selection
KitItemID	int	4	1			Kit item ID of this kit item (corresponds to the KitItem table)
KitItemName	nvarchar	255	1		Yes	Full Xml Field Copy - Name of this kit item
KitItemPriceDelta	money	8	1			Price delta (if any) of this kit item
Quantity	int	4	1			Quantity of this item chosen (generally is 1)

TextOption	ntext	16	1			Text entered by the customer placing the order for Text Option-type kit items
ExtensionData	ntext	16	1			Custom field for user defined data - Not Used
KitGroupTypeID	int	4	0			Corresponds to the KitGroupTypeID in the KitGroupTypeTable 1= Single Select Dropdown List, 2 = Single Select Radio Button, 3= Multi Select Checkoxes, 4 = Single-Line Text Box, 5= Multi-Line Text Area
InventoryVariantID	int	4	1			VariantID (if any) associated with this kit item for inventory control purposes
InventoryVariantColor	nvarchar	100	1		Yes	Full Xml Field Copy - Variant color associated with this kit item for inventory control purposes
InventoryVariantSize	nvarchar	100	1		Yes	Full Xml Field Copy - Variant size associated with this kit item for inventory control purposes
CreatedOn	datetime	8	0	getdate()		Date this record was created
CartType	int	4	0	0		Used in this case primarily to determine if the order was a normal one-time checkout or recurring autobill order. 0 = normal cart, 2 = recurring

Figure 5: Order Data Field Definitions - Orders_KitCart Table

2.7.Order Data Structure

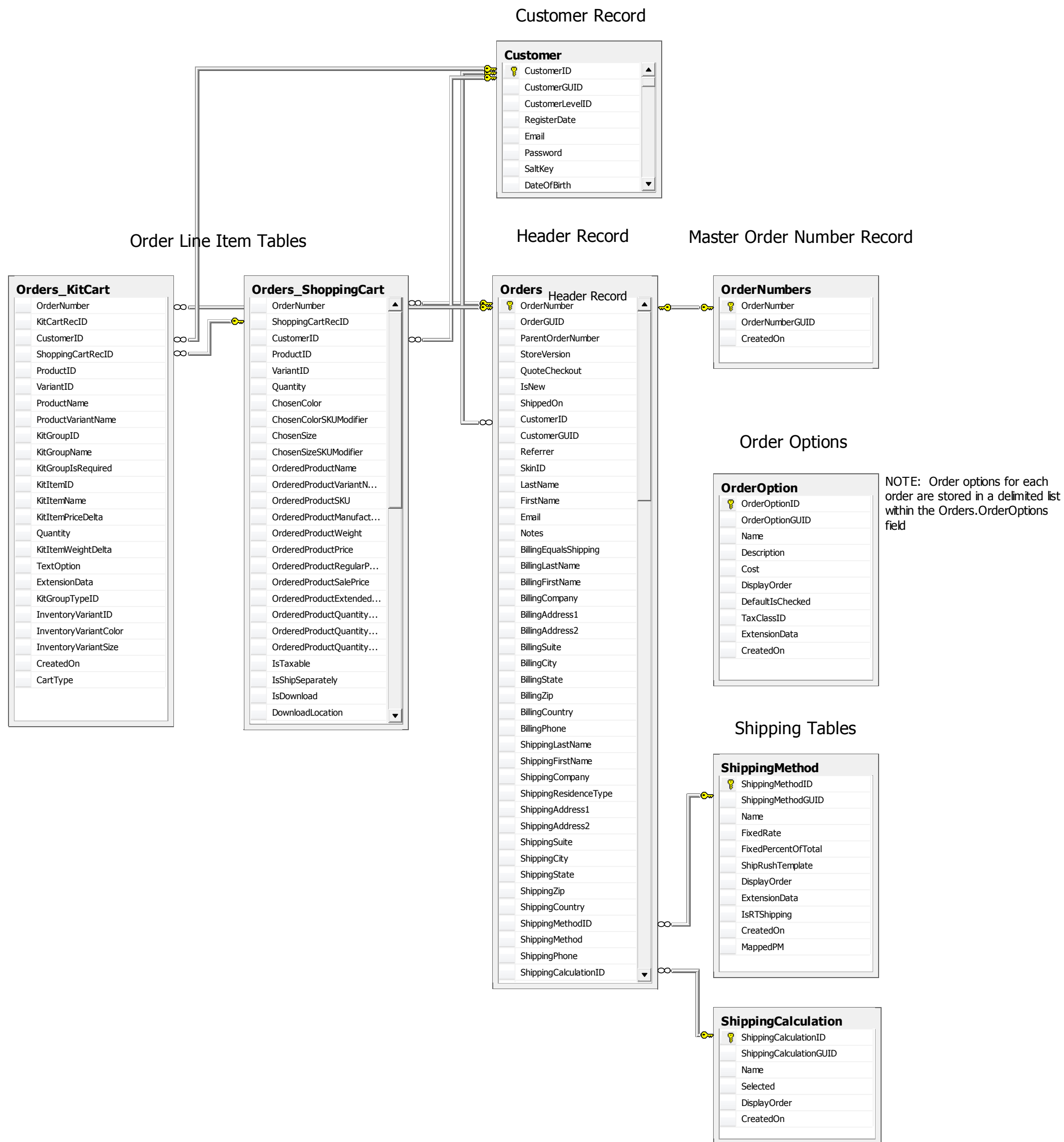


Figure 6: Order Data Structure Diagram

2.8.Products and Product Variants with Mappings and Related Tables (inc. Inventory)

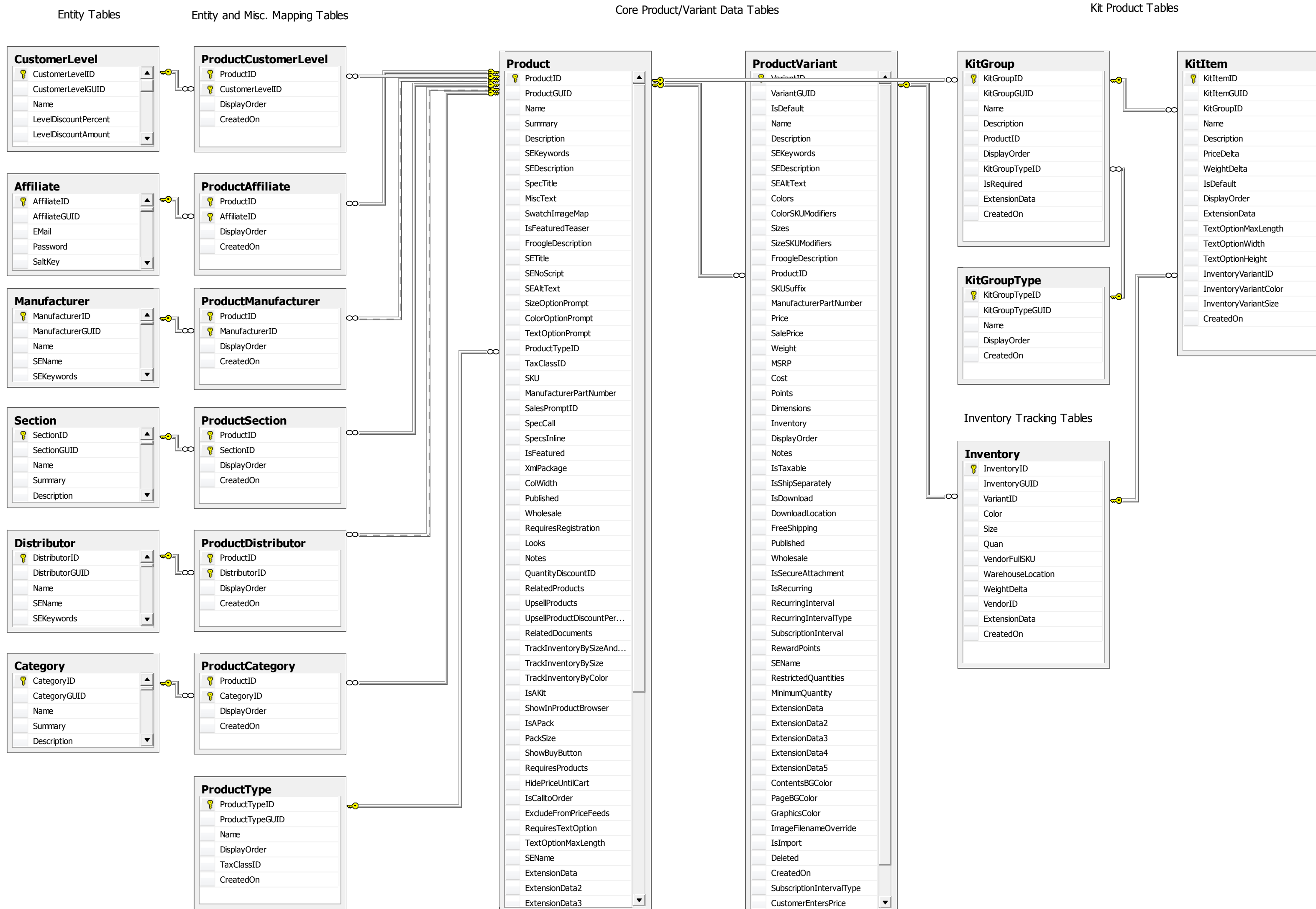


Figure 7: Products and Product Variants with Mappings and Related Tables (inc. Inventory) Diagram

2.9.Core Customer Data

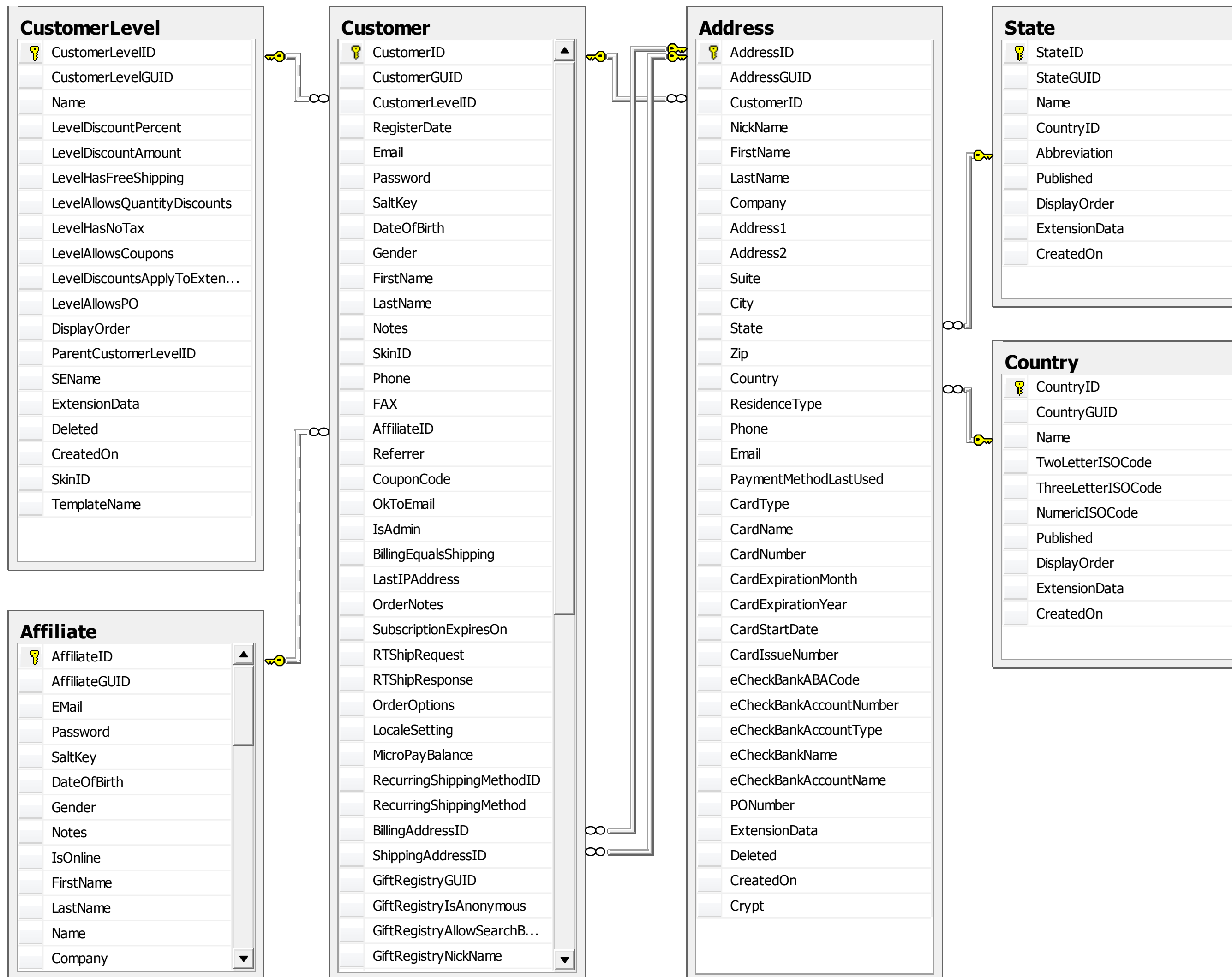


Figure 8: Core Customer Data Diagram

3. External Providers (Typically Batch Push With Event Listeners)

3.1.Introduction to WSI

The AspDotNetStorefront Web Service Automation Interface (WSI) version 1.0 will be the primary technology for pushing batch data into AspDotNetStorefront, in conjunction with event-driven notifications, which can be acted upon by the ERP system to either push or retrieve data from AspDotNetStorefront.

WSI is an XML web service that can consume a well-formed XML document and return like data based upon the actions provided in the initial call. Both WSI and event listeners support XML-package technology, allowing for some data shaping on the web server before the data is return. Because XML packages are compiled at runtime and invoked via the WSI call or event handler definition, special integration requirements can be met without making changes to AspDotNetStorefront code. Instead, custom XML packages could be provided along with other ERP-system specific add-in modules and simply dropped into the site.

Authentication is supported using both username/password authentication over SSL, or via Microsoft's WSE (Web Services Enhancements) 3.0 in combination with a username and secure hashed password. Microsoft WSE 3.0 is interoperable with Microsoft WCF-based clients (Windows Communication Foundation).

An example process-flow for an event-driven WSI-based transaction would look like the following:

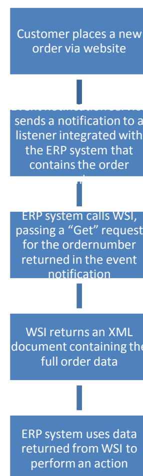


Figure 9: Event-Driven WSI Transaction Process Flow

WSI is also fully capable of simply consuming data provided to it, making it ideal for "Push" type and batch updates to AspDotNetStorefront. For example, if an integrator wished to add a new customer to AspDotNetStorefront whenever a new customer was added to the ERP system, the process would work as follows:

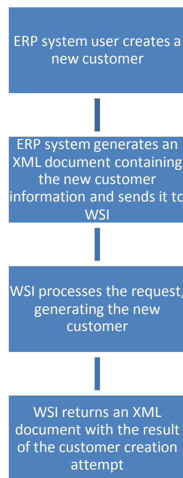


Figure 10: Data Push Process Flow via WSI

3.2.Event Handlers

Event handlers in AspDotNetStorefront allows the website to asynchronously send XML-formatted notifications to a target URL via an HTTP web request. Event notification data is fully extensible using XML-package technology, allowing systems integrators to tune data being returned by the notification to meet specific requirements without making changes to AspDotNetStorefront source code.

By default, AspDotNetStorefront supplies a single XML package that handles event notifications, returning primary key data to the listener service to act upon (for example, for a new order notification, an XML document containing an order number would be returned). The follow example shows a simple XML package that would handle returning an order number for a new order event. Additional XML packages can be defined on a per-event basis. Specifications for building XML Packages can be found at http://manual.aspdotnetstorefront.com/pdf/XmlPackages2.1.pdf.

```
<?xml version="1.0" standalone="yes" ?>
<package version="2.1" displayname="Event Handler" debug="false" includeentityhelper="true">
	<!-- ########################################################################### -->
	<!-- Copyright AspDotNetStorefront.com, 1995-2008.  All Rights Reserved.            -->
	<!-- http://www.aspdotnetstorefront.com                                             -->
	<!-- For details on this license please visit the URL above.                        -->
	<!-- THE ABOVE NOTICE MUST REMAIN INTACT.                                           -->
	<!-- ########################################################################### -->

	<PackageTransform>
		<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:aspdnsf="urn:aspdnsf" exclude-result-prefixes="aspdnsf">

		<xsl:output method="xml" omit-xml-declaration="no" />

		<xsl:template match="/">
			<EventNotification>
				<EventDate>
					<xsl:value-of select="/root/System/Date" />
				</EventDate>
				<EventTime>
					<xsl:value-of select="/root/System/Time" />
				</EventTime>
				<EventSite>
					<xsl:value-of select="aspdnsf:AppConfig('LiveServer')" />
				</EventSite>
				<xsl:if test="/root/Runtime/NewOrder">
				<Event>NewOrder</Event>
				<EventData>
					<OrderNumber>
						<xsl:value-of select="/root/Runtime/OrderNumber" />
					</OrderNumber>
				</EventData>
				</xsl:if>
			</EventNotification>
		</xsl:template>
		</xsl:stylesheet>
	</PackageTransform>
</package>
```

Using the sample XML package provided, the event notification data for a new order would look like:

```
<?xml version="1.0" encoding="utf-8"?>
	<EventNotification>
		<EventDate>12/14/2008</EventDate>
		<EventTime>10:23AM</EventTime>
		<EventSite>yourdomain.com</EventSite>
		<Event>NewOrder</Event>
		<EventData>
			<OrderNumber>100001</OrderNumber>
		</EventData>
	</EventNotification>
```

Event notifications are currently supported for the following actions, and additional notifications can be added as needs arise for integration purposes:

Event Type	Page	Called On
AddToCart	Addtocart.aspx	Page_Load
DeleteCustomer	Admin/customers.aspx	Page_Load
OrderShipped	Admin/orderframe.aspx	Page_Load
ViewEntityPage	ShowEntityPage.aspx	Page_Load
CheckoutPayment	CheckoutPayment.aspx	Page_Load
CheckoutReview	CheckoutReview.aspx	Page_Load
CheckoutShipping	CheckoutShipping.aspx	Page_Load
CheckoutShipping	CheckoutShippingMult.aspx	Page_Load
CreateAccount	CreateAccount.aspx	CreateAccount
BeginCheckout	ShoppingCart.aspx	ProcessCart
UpdateCustomer	AspDotNetStorefrontCommon.Customer	UpdateCustomerStatic
UpdateCustomer	AspDotNetStorefrontCommon.Customer	UpdateCustomer
CreateCustomer	AspDotNetStorefrontCommon.Customer	MakeAnonCustomer
RemoveFromCart	AspDotNetStorefrontCommon.ShoppingCart	RemoveItem
NewOrder	AspDotNetStorefrontGateways.Gateway	MakeOrder
OrderVoided	AspDotNetStorefrontGateways.Gateway	DispatchVoid
OrderVoided	AspDotNetStorefrontGateways.GoogleCheckout	ProcessOrderStateChangeNotification

Figure 11: Event Notification Callers

3.3.WSI Data Types

STRING fields

String fields can contain any text (properly XmlEncoded) that you need. CDATA field values can also be used to specify string data contents if markup/encoding issues become problematic.

ML fields (Multi-Lingual)

The WSI interface supports specification of multi-lingual fields. Any "ml" field can have its value set as a sub Xml fragment in this format:

```
<ml>
   <locale name="en-US">en us value</locale>
   <locale name="ja-JP">ja jp value</locale>
</ml>
```

e.g.

```
<Name>
   <ml>
      <locale name="en-US">en us value</locale>
      <locale name="ja-JP">ja jp value</locale>
   </ml>
</Name>
```

CDATA Fields

Almost any "string" field can also be specified in CDATA syntax if that is more convenient to avoid encoding problems, e.g.:

```
<Description >
        <![CDATA[
        Complex product description here with non-xml compliant characters such as
        &#<>, etc...
        ]]>
</Description >
```

BOOLEAN Fields

All Boolean fields can be any of:

1, "true", "TRUE", "True", "yes", "Yes", "YES" meaning logical "true" or

0, "false", "FALSE", "False", "no", "No", "NO" meaning logical "false"
DECIMAL Fields

All Decimal fields (e.g. prices, weights, etc) must be in en-US decimal format, without commas or leading currency signs (e.g. xx.xx). The format is always xxxxx.xx, regardless of your store master locale setting (e.g. do NOT use xx.xxx,00 international formats)

DATETIME Fields

All DateTime field values MUST be in your store master locale format. That is controlled by the locale settings in your web.config file. The text "NULL" can be used to assign a date field to "no value".

WSI Import Doc Header Attributes

The import doc has a number of header attributes, which can affect how the document is processed.

```
<AspDotNetStorefrontImport Version="7.1" AutoLazyAdd="true|false"
AutoCleanup="true|false" Verbose="true|false" TransactionsEnabled="true|false">
...
</AspDotNetStorefrontImport>
```

Each attribute is listed and described below.

Version: This should be set to 7.1

AutoLazyAdd (Default=false): This should be set to either true or false. This flag affects how an update action behaves if an item doesn't exist (e.g. you issue an update action on a product that doesn't exist). If AutoLazyAdd="false", then an error will be raised in the case you update a non-existent item. If AutoLazyAdd="true", the product will be automatically added, using the values in the update action node.

AutoCleanup (Default=false): This flag is not yet supported at the document level. See the AutoCleanup attribute on various processing nodes, which are supported.

Verbose (Default=false): This flag indicates whether or not to output information processing nodes in the resulting XmlDocument. Setting Verbose="true" will enable easier debugging of WSI actions and allow you to trace through all execution steps, and SQL statements. Your client should ignore any informational nodes produced, when processing the output for automation.

UseImplicitTransactions (Default=false): This flag control whether or not nodes in the input document are handled in transaction mode or not implicitly. This flag does NOT affect explicit <Transaction>...</Transaction> blocks which are always honored. If UseImplicitTransactions="true", then if there are processing nodes in the input Xml document which are not nested inside a <Transaction> block, the WSI will create an implicit default transaction for EACH node it processes. This can allow you to ensure the node and all resulting data either commits or rolls back on any error (e.g. when adding a Product). If UseImplicitTransactions="false", then any non Transaction block nodes are just executed as is. They can partially succeed or fail, and you will have to examine the output result Xml document to determine what was performed.

In this root node, you can set the following attributes. Defaults are marked with a *.

Version = 7.1

SetImportFlag = true | *false {this is not supported yet in BETA}

AutoLazyAdd = true | *false. This flag is supported and controls how an update node performs, if the targeted data doesn't exist in the database. If an update is

Execution Sequence

The WSI processes elements sequentially, IN THE ORDER THEY APPEAR in your input Xml document!

Transactions

WSI supports transactional processing (and rollback) for many operations (e.g. bulk adding products, categories, etc.). The input Xml doc header attribute must have TransactionsEnabled="true". Note that some nodes CANNOT be put inside a transaction as there is no way to roll them back (e.g. updating an order transaction state from AUTH to CAPTURED)...See the notes on each action element description on whether it can be inside a transaction or not.

You can have more than one transaction in a single input document also, and each transaction can optionally be given a name.

3.4.Updating and Retrieving Data Using WSI

The following sample code and examples show ways in which data can be updated and retrieved using WSI.

3.5.Code Sample: Communicating with WSI

Communication with WSI can be accomplished in several ways; using SOAP 1.1 or 1.2 requests or an HTTP POST with one of two available operations:

DoItUsernamePwd
XmlDocument Input (as String). XmlDocument Output. This method is less secure, but does not require Microsoft Web Services 3 UsernameToken authentication!. This method

can be used over HTTPS to do username and password check on the call itself. Password should be clear text master password here.

DoItWSE3
XmlDocument Input (as String). XmlDocument Output. This method requires Microsoft Web Services 3 UsernameToken authentication! When using WSE3 authentication, you must send in the full admin hashed password (you can find this in the master database record for the admin user customer record).

A sample SOAP 1.2 request (placeholders shown need to be replaced with actual values) using DoItUsernamePwd would look like:

```
POST /8000CSLA/ipx.asmx HTTP/1.1
Host: localhost
Content-Type: application/soap+xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap12:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap12="http://www.w3.org/2003/05/soap-
envelope">
  <soap12:Body>
    <DoItUsernamePwd xmlns="http://www.aspdotnetstorefront.com/">
      <AuthenticationEMail>string</AuthenticationEMail>
      <AuthenticationPassword>string</AuthenticationPassword>
      <XmlInputRequestString>string</XmlInputRequestString>
    </DoItUsernamePwd>
  </soap12:Body>
</soap12:Envelope>
```

and a SOAP 1.2 response would look like:

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap12:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap12="http://www.w3.org/2003/05/soap-
envelope">
  <soap12:Body>
    <DoItUsernamePwdResponse xmlns="http://www.aspdotnetstorefront.com/">
      <DoItUsernamePwdResult>string</DoItUsernamePwdResult>
    </DoItUsernamePwdResponse>
  </soap12:Body>
</soap12:Envelope>
```

Using UsernameToken Authentication (DoItWSE3), a sample SOAP 1.2 request would be:

```
POST /8000CSLA/ipx.asmx HTTP/1.1
Host: localhost
Content-Type: application/soap+xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap12:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap12="http://www.w3.org/2003/05/soap-
envelope">
  <soap12:Body>
    <DoItWSE3 xmlns="http://www.aspdotnetstorefront.com/">
      <XmlInputRequestString>string</XmlInputRequestString>
    </DoItWSE3>
  </soap12:Body>
</soap12:Envelope>
```

And the response would be:

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap12:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap12="http://www.w3.org/2003/05/soap-
envelope">
  <soap12:Body>
    <DoItWSE3Response xmlns="http://www.aspdotnetstorefront.com/">
      <DoItWSE3Result>string</DoItWSE3Result>
    </DoItWSE3Response>
  </soap12:Body>
</soap12:Envelope>
```

If references to Microsoft.Web.Services3 are added to an application, and Web References are made to the web service (in this example we're just using a simple windows application with a Web Reference to the WSI namespace), a sample application to send a request and retrieve the response might look something like:

```
private void DoIt(String XmlInputRequestString)
{
    String Result = String.Empty;
    Boolean UsingWSE3 = false; // or true if you are using UsernameToken authentication
    if (UsingWSE3)
    {
        UsernameToken token = new UsernameToken("admin@emailaddress.com", "hashed admin
password", PasswordOption.SendHashed);
        WSI.AspDotNetStorefrontImportWebServiceWse svc = new
WSI.AspDotNetStorefrontImportWebServiceWse();
        svc.Url = "URL where the web service can be consumed";
        svc.RequestSoapContext.Security.Tokens.Add(token);
        Result = svc.DoItWSE3(XmlInputRequestString);
    }
    else
    {
        WSI.AspDotNetStorefrontImportWebService svc = new
WSI.AspDotNetStorefrontImportWebService();
        svc.Url = "URL where the web service can be consumed";
        Result = svc.DoItUsernamePwd("admin@emailaddress.com", "plain-text admin password",
XmlInputRequestString);
    }
}
```

In this case, the "Result" would be a string of XML that could then written to an xml file or parsed programmatically to then be utilized from the ERP system.

3.6.Code Sample: Event Listener Service

You've already seen how a new order event would be formatted (section 3.2):

```
<?xml version="1.0" encoding="utf-8"?>
<EventNotification>
  <EventDate>1/6/2009</EventDate>
  <EventTime>1:16 PM</EventTime>
  <EventSite>yourdomain.com</EventSite>
  <Event>NewOrder</Event>
  <EventData>
    <OrderNumber>100025</OrderNumber>
  </EventData>
</EventNotification>
```

but something still needs to be done with the data. To retrieve the event xml, you could use something similar to the following code in the page that you have set as your CalloutURL in the event handler configuration section:

```
try
{
    StringBuilder sb = new StringBuilder();
    int streamLength;
    int streamRead;
    Stream s = Request.InputStream;
    streamLength = Convert.ToInt32(s.Length);

    Byte[] streamArray = new Byte[streamLength];
    streamRead = s.Read(streamArray, 0, streamLength);
    for (int i = 0; i < streamLength; i++)
    {
        sb.Append(Convert.ToChar(streamArray[i]));
    }
    s.Dispose();

    if (sb.Length > 0)
    {

        XDocument xdoc = XDocument.Parse(sb.ToString());

        XElement xeEvent = xdoc.Element("EventNotification");

        if (xeEvent != null)
        {
            String EventType = xeEvent.Element("Event").Value;

            if (EventType.Equals("NewOrder", StringComparison.InvariantCultureIgnoreCase))
            {
                // we have a new order event, get the order number
                String OrderNumber =
xeEvent.Element("EventData").Element("OrderNumber").Value;

                // create a request to WSI to get all order information
                XElement xeImport = new XElement("AspDotNetStorefrontImport",
                new XAttribute("Version", "7.1"),
                new XElement("Get",
                    new XAttribute("Table", "Orders"),
                    new XAttribute("Name", "NewOrders"),
                    new XElement("XmlPackage", "DumpOrder.xml.config"),
                    new XElement("DefaultWhereClause", "OrderNumber=" + OrderNumber)));

                XDocument xdResponse = XDocument.Parse(DoItReturnString(xeImport));

                /*
                 * DoItReturnString(XElement xe) is your WSI requst sending method
                 * that returns a response in string format.
                 * Now you have an xmldocument (xdResponse) with all order information.
                 * Logic to do something in ERP system.
                 *
                 */

            }
        }
    }
}
catch (Exception ex)
{
    // TODO: Error logging here
}
```

Now you have a way of tieing new orders in AspDotNetStorefront to the ERP system in near real-time fashion.

3.7.WSI Sample: Categories

Update Direction: From ERP

Supported Actions: Add, Update, Delete (soft delete), Nuke, Lookup

A Delete, Nuke, and Lookup Action would look fairly similar except for the specification of the action itself. These actions require the knowledge of the Category ID or GUID from the Category table in the database, and the request can be done programmatically using the following code:

```
XElement xeImport = new XElement("AspDotNetStorefrontImport", new XAttribute("Version",
"7.1"));

XElement xeCategory = new XElement("Entity",
    new XAttribute("Action", "Delete"), // or ("Action", "Nuke") or ("Action", "Lookup")
    new XAttribute("EntityType", "Category"),
    new XAttribute("ID", 1)); // or ("GUID", CategoryGUID)

xeImport.Add(xeCategory);

DoIt(xeImport.ToString(SaveOptions.DisableFormatting));
```

The resulting request (the xeImport variable) in this case would resemble:

```
<AspDotNetStorefrontImport Version="7.1">
  <Entity Action="Delete" EntityType="Category" ID="1" /> <!-- or Action="Nuke" or Action="Lookup"-->
</AspDotNetStorefrontImport>
```

Updating a category also requires knowledge of the CategoryID or the CategoryGUID from the Category table in the database. To update the Name, XmlPackage, Description, and Published values for the Category, you could create the request as following:

```
XElement xeImport = new XElement("AspDotNetStorefrontImport", new XAttribute("Version",
"7.1"));

XElement xeCategory = new XElement("Entity",
    new XAttribute("Action", "Update"),
    new XAttribute("EntityType", "Category"),
    new XAttribute("ID", 1),
        new XElement("Name", "updated category name"),
        new XElement("Description", new XCData("updated category description")),
        new XElement("Display",
            new XElement("XmlPackage", "entity.gridwithprices.xml.config")),
        new XElement("Published", true));

xeImport.Add(xeCategory);

DoIt(xeImport.ToString(SaveOptions.DisableFormatting));
```

The resulting xeImport in this case would be:

```
<AspDotNetStorefrontImport Version="7.1">
  <Entity Action="Update" EntityType="Category" ID="1">
    <Name>updated category name</Name>
    <Description><![CDATA[updated category description]]></Description>
    <Display>
      <XmlPackage>entity.gridwithprices.xml.config</XmlPackage>
    </Display>
    <Published>true</Published>
  </Entity>
</AspDotNetStorefrontImport>
```

To add a new category with a Name, XPath (used to specify location in the category is a child of another category), Description, XmlPackage, and Published value, you could use as follows:

```
XElement xeImport = new XElement("AspDotNetStorefrontImport", new XAttribute("Version",
"7.1"));

XElement xeCategory = new XElement("Entity",
    new XAttribute("Action", "Add"),
    new XAttribute("EntityType", "Category"),
    new XElement("Name", "CategoryName"),
    new XElement("XPath", "/Main Category/Sub Category/CategoryName"),
    new XElement("Description", new XCData("category description")),
    new XElement("Display",
        new XElement("XmlPackage", "entity.gridwithprices.xml.config")),
    new XElement("Published", true));

xeImport.Add(xeCategory);

DoIt(xeImport.ToString(SaveOptions.DisableFormatting));
```

The resulting xeImport in this case would be:

```
<AspDotNetStorefrontImport Version="7.1">
  <Entity Action="Add" EntityType="Category">
    <Name>CategoryName</Name>
    <XPath>/Main Category/Sub Category/CategoryName</XPath>
    <Description><![CDATA[category description]]></Description>
    <Display>
      <XmlPackage>entity.grid.xml.config</XmlPackage>
    </Display>
    <Published>true</Published>
  </Entity>
</AspDotNetStorefrontImport>
```

3.8.WSI Sample: Manufacturers

Manufacturers work much in the same way as Categories, with the one major difference being that Manufacturers can have additional values such as address values, a website URL, and an email address. The Lookup, Delete, and Nuke requests would be built the same as categories with the exception of the EntityType attribute:

```
XElement xeImport = new XElement("AspDotNetStorefrontImport", new XAttribute("Version",
"7.1"));

XElement xeManufacturer = new XElement("Entity",
    new XAttribute("Action", "Delete"), // or ("Action", "Nuke") or ("Action", "Lookup")
    new XAttribute("EntityType", "Manufacturer"),
    new XAttribute("ID", 1)); // or ("GUID", ManufacturerGUID)

xeImport.Add(xeManufacturer);

DoIt(xeImport.ToString(SaveOptions.DisableFormatting));
```

The resulting xeImport in this case is:

```
<AspDotNetStorefrontImport Version="7.1">
  <Entity Action="Delete" EntityType="Manufacturer" ID="1" />
</AspDotNetStorefrontImport>
```

You can add the address values, URL, and email address the same as you would the other details of the manufacturer:

```
XElement xeImport = new XElement("AspDotNetStorefrontImport", new XAttribute("Version",
"7.1"));

XElement ManufacturerAddressData = new XElement("AddressData",
    new XElement("Address1", "manufacturer address 1"),
    new XElement("Address2", "manufacturer address 2"),
    new XElement("Suite", "manufacturer suite"),
    new XElement("City", "manufacturer city"),
    new XElement("State", "OH"),
    new XElement("ZipCode", "11111"),
    new XElement("Country", "United States"),
    new XElement("Phone", "1112223333"),
    new XElement("FAX", "1112223333"));

XElement xeManufacturer = new XElement("Entity",
    new XAttribute("Action", "Add"),
    new XAttribute("EntityType", "Manufacturer"),
    new XElement("Name", "ManufacturerName"),
    new XElement("Description", new XCData("manufacturer description")),
    ManufacturerAddressData.Nodes(),
    new XElement("Display",
        new XElement("XmlPackage", "entity.gridwithprices.xml.config")),
    new XElement("Published", false));

xeImport.Add(xeManufacturer);

DoIt(xeImport.ToString(SaveOptions.DisableFormatting));
```

The xeImport here would be:

```
<AspDotNetStorefrontImport Version="7.1">
  <Entity Action="Add" EntityType="Manufacturer">
    <Name>ManufacturerName</Name>
    <Description><![CDATA[manufacturer description]]></Description>
    <Address1>manufacturer address 1</Address1>
    <Address2>manufacturer address 2</Address2>
    <Suite>manufacturer suite</Suite>
    <City>manufacturer city</City>
    <State>OH</State>
    <ZipCode>11111</ZipCode>
    <Country>United States</Country>
    <Phone>1112223333</Phone>
    <FAX>1112223333</FAX>
    <Display>
      <XmlPackage>entity.gridwithprices.xml.config</XmlPackage>
    </Display>
    <Published>false</Published>
  </Entity>
</AspDotNetStorefrontImport>
```

Manufacturers can be updated in similar fashion:

```
XElement xeImport = new XElement("AspDotNetStorefrontImport", new XAttribute("Version",
"7.1"));

XElement ManufacturerAddressData = new XElement("AddressData",
    new XElement("Address1", "updated manufacturer address 1"),
    new XElement("Address2", "updated manufacturer address 2"),
    new XElement("Suite", "updated manufacturer suite"),
    new XElement("City", "updated manufacturer city"),
    new XElement("State", "NV"),
    new XElement("ZipCode", "22222"),
    new XElement("Country", "United States"),
    new XElement("Phone", "1112223333"),
    new XElement("FAX", "1112223333"));

XElement xeManufacturer = new XElement("Entity",
    new XAttribute("Action", "Update"),
    new XAttribute("EntityType", "Manufacturer"),
    new XAttribute("ID", 1),
    new XElement("Name", "Updated ManufacturerName"),
    new XElement("Description", new XCData("updated manufacturer description")),
    ManufacturerAddressData.Nodes(),
    new XElement("Display",
        new XElement("XmlPackage", "entity.grid.xml.config")),
    new XElement("Published", true));

xeImport.Add(xeManufacturer);

DoIt(xeImport.ToString(SaveOptions.DisableFormatting));
```

resulting in an xeImport of:

```
<AspDotNetStorefrontImport Version="7.1">
  <Entity Action="Update" EntityType="Manufacturer" ID="1">
    <Name>Updated ManufacturerName</Name>
    <Description><![CDATA[updated manufacturer description]]></Description>
    <Address1>updated manufacturer address 1</Address1>
    <Address2>updated manufacturer address 2</Address2>
    <Suite>updated manufacturer suite</Suite>
    <City>updated manufacturer city</City>
    <State>NV</State>
    <ZipCode>22222</ZipCode>
    <Country>United States</Country>
    <Phone>1112223333</Phone>
    <FAX>1112223333</FAX>
    <Display>
      <XmlPackage>entity.grid.xml.config</XmlPackage>
    </Display>
    <Published>true</Published>
  </Entity>
</AspDotNetStorefrontImport>
```

3.9.WSI Sample: Products and Product Variants

Products and variants are more involved and there are many more values for these than for categories or manufacturers. Delete, Nuke, and Lookup remain similar however, requiring the ProductID or ProductGUID from the Product table in the database, and the VariantID or VariantGUID from the ProductVariant table in the database. Deleting just the variant of a product can be done as follows:

```
XElement xeImport = new XElement("AspDotNetStorefrontImport",
    new XAttribute("Version", "7.1"),
        new XElement("Product",
            new XAttribute("Action", "Update"),
            new XAttribute("ID", 1),
            new XElement("Variants",
                new XElement("Variant",
                    new XAttribute("Action", "Delete"),
                    new XAttribute("ID", 1))))));

DoIt(xeImport.ToString(SaveOptions.DisableFormatting));
```

The xeImport here would be:

```
<AspDotNetStorefrontImport Version="7.1">
  <Product Action="Update" ID="1">
    <Variants>
      <Variant Action="Delete" ID="1" />
    </Variants>
  </Product>
</AspDotNetStorefrontImport>
```

Deleting the product doesn’t require the Variants or Variant nodes to be included in in the request:

```
XElement xeImport = new XElement("AspDotNetStorefrontImport",
    new XAttribute("Version", "7.1"),
        new XElement("Product",
        new XAttribute("Action", "Delete"),
        new XAttribute("ID", 1)));

DoIt(xeImport.ToString(SaveOptions.DisableFormatting));
```

which yields:

```
<AspDotNetStorefrontImport Version="7.1">
  <Product Action="Delete" ID="1" />
</AspDotNetStorefrontImport>
```

The Lookup and Nuke actions are performed in the same format.

There is also a GetProduct node which can be used in the request to retrieve all information from a given product. It requires an ID or GUID attribute (the ID or GUID of the Product) but can also accept a GetAll attribute (when true, returns all information for all products and ignores the ProductID or ProductGUID) and an IncludeVariants attribute (when true, the response will include a Variants node with each Variant of the product specified in a Variant node…when false the Variants node is omitted from the response)

```
<AspDotNetStorefrontImport Version="7.1">
  <GetProduct GetAll="true" IncludeVariants="false" />
</AspDotNetStorefrontImport>
```

Adding a product is done via the Product node with the Add Action attribute and be done without variants by omitting the Variants and any Variant nodes (though it's recommended to include at least one variant. Updating is done in the same format but with the "Update" Action attribute instead of "Add". To add a product with 2 variants you could use something like:

```
XElement xeImport = new XElement("AspDotNetStorefrontImport",
                new XAttribute("Version", "7.1"));

String ProductDescription = "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam
et orci at dui dignissim lacinia. Morbi luctus turpis et libero. Sed ultricies mi id odio.
Pellentesque tincidunt viverra enim. Ut eu elit vitae lorem dignissim fermentum.";
String VariantDescription = "Sed fermentum risus eu est. Nam purus quam, luctus in, pulvinar
nec, pulvinar ut, tellus. Ut ipsum. Nulla in urna. Praesent lectus. In vehicula est eu felis
vulputate laoreet. Nullam tincidunt cursus dolor. Phasellus laoreet turpis sit amet dolor.";

// Start a Product Node and add the data we want to add
XElement xeProduct = new XElement("Product",
        new XAttribute("Action", "Add"),
        new XElement("Name", "the product name"),
        new XElement("Description", new XCData(ProductDescription)),
        new XElement("SKU", "001PNRMSKU"),
        new XElement("Display",
                new XElement("XmlPackage", "product.variantsinrightbar.xml.config")),
        new XElement("Published", true));

// Create the Variants Node
XElement xeVariants = new XElement("Variants");

// Create a Variant Node and add the data we want to add
XElement xeVariant = new XElement("Variant",
        new XAttribute("Action", "Add"),
        new XElement("IsDefault", true),
        new XElement("Name", "variant name 1"),
        new XElement("Description", new XCData(VariantDescription)),
        new XElement("SKUSuffix", "-SKUSFX"),
        new XElement("Price", 20.99),
        new XElement("SalePrice", 17.99),
        new XElement("Weight", 2.2));

// Add the Variant Node to the Variants Node
xeVariants.Add(xeVariant);

// Create another Variant Node and add the data we want to add
xeVariant = new XElement("Variant",
        new XAttribute("Action", "Add"),
        new XElement("IsDefault", false),
        new XElement("Name", "variant name 2"),
        new XElement("Description", new XCData(VariantDescription)),
        new XElement("SKUSuffix", "-SKUSFX2"),
        new XElement("Price", 30.99),
        new XElement("SalePrice", 27.99),
        new XElement("Weight", 4.2));

// Add the second Variant Node to the Variants Node
xeVariants.Add(xeVariant);

// Add the Variants Node to the Product Node
xeProduct.Add(xeVariants);

// Start a Mappings Node and add the entities we want to map the product to,
// to the child nodes
XElement xeMappings = new XElement("Mappings",
        new XElement("Entity",
                new XAttribute("EntityType", "Manufacturer"),
                new XAttribute("Name", "ManufacturerName")),
        new XElement("Entity",
                new XAttribute("EntityType", "Category"),
                new XAttribute("ID", 1)));

// Add the Mappings Node to the Product Node
```

```
xeProduct.Add(xeMappings);

// Add the Product Node to the Import Node
xeImport.Add(xeProduct);

DoIt(xeImport.ToString(SaveOptions.DisableFormatting));
```

And the resulting xml request would look like:

```
<AspDotNetStorefrontImport Version="7.1">
  <Product Action="Add">
    <Name>the product name</Name>
    <Description><![CDATA[Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam et orci at dui dignissim lacinia. Morbi luctus turpis et libero. Sed ultricies mi id odio. Pellentesque tincidunt viverra enim. Ut eu elit vitae lorem dignissim fermentum.]]></Description>
    <SKU>001PNRMSKU</SKU>
    <Display>
      <XmlPackage>product.variantsinrightbar.xml.config</XmlPackage>
    </Display>
    <Published>true</Published>
    <Variants>
      <Variant Action="Add">
        <IsDefault>true</IsDefault>
        <Name>variant name 1</Name>
        <Description><![CDATA[Sed fermentum risus eu est. Nam purus quam, luctus in, pulvinar nec, pulvinar ut, tellus. Ut ipsum. Nulla in urna. Praesent lectus. In vehicula est eu felis vulputate laoreet. Nullam tincidunt cursus dolor. Phasellus laoreet turpis sit amet dolor.]]></Description>
        <SKUSuffix>-SKUSFX</SKUSuffix>
        <Price>20.99</Price>
        <SalePrice>17.99</SalePrice>
        <Weight>2.2</Weight>
      </Variant>
      <Variant Action="Add">
        <IsDefault>false</IsDefault>
        <Name>variant name 2</Name>
        <Description><![CDATA[Sed fermentum risus eu est. Nam purus quam, luctus in, pulvinar nec, pulvinar ut, tellus. Ut ipsum. Nulla in urna. Praesent lectus. In vehicula est eu felis vulputate laoreet. Nullam tincidunt cursus dolor. Phasellus laoreet turpis sit amet dolor.]]></Description>
        <SKUSuffix>-SKUSFX2</SKUSuffix>
        <Price>30.99</Price>
        <SalePrice>27.99</SalePrice>
        <Weight>4.2</Weight>
      </Variant>
    </Variants>
    <Mappings>
      <Entity EntityType="Manufacturer" Name="ManufacturerName" />
      <Entity EntityType="Category" ID="1" />
    </Mappings>
  </Product>
</AspDotNetStorefrontImport>
```

3.10. WSI Sample: Product Inventory

There are several ways to update product inventory depending on the type of product configured and when you are making the inventory updates. If you wish to update the inventory for a product that does not Track Inventory by Size and Color and are already updating the Product and/or Variant (section 3.9), you can update the inventory directly in the Variants -> Variant node:

```
XElement xeImport = new XElement("AspDotNetStorefrontImport",
    new XAttribute("Version", "7.1"),
    new XElement("Product",
        new XAttribute("Action", "Update"),
        new XAttribute("ID", 1),
        new XElement("Variants",
            new XElement("Variant",
                new XAttribute("Action", "Update"),
                new XAttribute("ID", 1),
                new XElement("Inventory", 9999)))));

DoIt(xeImport.ToString(SaveOptions.DisableFormatting));
```

which yields a request of:

```
<AspDotNetStorefrontImport Version="7.1">
  <Product Action="Update" ID="1">
    <Variants>
      <Variant Action="Update" ID="1">
        <Inventory>9999</Inventory>
      </Variant>
    </Variants>
  </Product>
</AspDotNetStorefrontImport>
```

You can also update inventory for a product that does not have Track Inventory by Size and Color turned on, through the InventoryUpdate node:

```
XElement xeImport = new XElement("AspDotNetStorefrontImport",
        new XAttribute("Version", "7.1"),
            new XElement("InventoryUpdate",
                new XComment("Simple inventory updates only require the VariantID"),
                new XElement("Inv",
                    new XAttribute("VariantID", 1),
                    new XAttribute("Quantity", 1000)),
                new XElement("Inv",
                new XAttribute("VariantID", 2),
                new XAttribute("Quantity", 8888))));

DoIt(xeImport.ToString(SaveOptions.DisableFormatting));
```

and the request becomes:

```
<AspDotNetStorefrontImport Version="7.1">
  <InventoryUpdate>
    <!--Simple inventory updates only require the VariantID-->
    <Inv VariantID="1" Quantity="1000" />
    <Inv VariantID="2" Quantity="8888" />
  </InventoryUpdate>
</AspDotNetStorefrontImport>
```

For products that do have Track Inventory by Size and Color turned on, you can use the InventoryUpdate node and an attribute on the Inv node called “MatchKey”, which let’s you match on specific attributes of the inventory record. Here, you can use MatchKey=”VendorFullSKU” to update the quantity of the items with the VendorFullSKU defined:

```
XElement xeImport = new XElement("AspDotNetStorefrontImport",
        new XAttribute("Version", "7.1"),
        new XElement("InventoryUpdate",
        new XComment("When using 'MatchKey' the product must have 'Track Inventory " +
                     "by Size and Color' set to 'true'"),
                new XElement("Inv",
                        new XAttribute("MatchKey", "VendorFullSKU"),
                        new XAttribute("VendorFullSKU", "PSKUNUM-red-large"),
                        new XAttribute("Quantity", 9999)),
                new XElement("Inv",
                        new XAttribute("MatchKey", "VendorFullSKU"),
                        new XAttribute("VendorFullSKU", "PSKUNUM-red-medium"),
                        new XAttribute("Quantity", 6666))));

DoIt(xeImport.ToString(SaveOptions.DisableFormatting));
```

3.11. WSI Sample: Customers

Customers can use Email, CustomerID, or CustomerGUID for all of the actions except for Add (Update, Delete, Nuke, and Lookup). A typical Delete request could be built with:

```
XElement xeImport = new XElement("AspDotNetStorefrontImport",
        new XAttribute("Version", "7.1"),
        new XElement("Customer",
            new XAttribute("Action", "Delete"),
            new XAttribute("EMail", "Customer@emailaddress.com")));

DoIt(xeImport.ToString(SaveOptions.DisableFormatting));
```

and would yield the request:

```
<AspDotNetStorefrontImport Version="7.1">
  <Customer Action="Delete" EMail="Customer@emailaddress.com" />
</AspDotNetStorefrontImport>
```

Updating or adding a customer is similar in format to updating or adding other data. A customer will also have addresses, and multiple addresses can be updated or added at once by using Address nodes within an Addresses node. An Add may be built like:

```
XElement xeImport = new XElement("AspDotNetStorefrontImport",
        new XAttribute("Version", "7.1"));

    // Start a Customer Node with the Add Action, and add the data we want to populate
    // to the child nodes
    XElement xeCustomer = new XElement("Customer",
        new XAttribute("Action", "Add"),
        new XElement("EMail", "Customer@emailaddress.com"),
        new XElement("FirstName", "Customer First Name"),
        new XElement("LastName", "Customer Last Name"),
        new XElement("Password", "customerpassword!$*"),
        new XElement("Phone", "1112223333"),
        new XElement("IsRegistered", 1));

    // Start an Addresses Node
    XElement xeAddresses = new XElement("Addresses");

    // Start an Address Node with all of the address information for the customer
    XElement xeAddress = new XElement("Address",
        new XAttribute("Action", "Add"),
        new XElement("NickName", "Address Nick Name"),
        new XElement("FirstName", "first name"),
        new XElement("LastName", "last name"),
        new XElement("Company", "company"),
        new XElement("Address1", "address 1"),
        new XElement("Address2", "address 2"),
        new XElement("Suite", "address suite"),
        new XElement("City", "address city"),
        new XElement("State", "OH"),
        new XElement("Country", "United States"),
        new XElement("ResidenceType", (int)AddressTypes.Billing),
        new XElement("Phone", "1112223333"),
        new XElement("Email", "Customer@emailaddress.com"));

    // Add the Address Node to the Addresses Node
    xeAddresses.Add(xeAddress);

    // Let's add another address
    xeAddress = new XElement("Address",
        new XAttribute("Action", "Add"),
        new XElement("NickName", "2nd Address Nick Name"),
        new XElement("FirstName", "different first name"),
        new XElement("LastName", "different last name"),
        new XElement("Company", "different company"),
        new XElement("Address1", "2nd address 1"),
        new XElement("Address2", "2nd address 2"),
        new XElement("Suite", "2nd address suite"),
        new XElement("City", "2nd address city"),
        new XElement("State", "OH"),
        new XElement("Country", "United States"),
        new XElement("ResidenceType", (int)AddressTypes.Shipping),
        new XElement("Phone", "1112223333"),
        new XElement("Email", "Customer@emailaddress.com"));

    // Add the 2nd Address Node to the Addresses Node
    xeAddresses.Add(xeAddress);

    // Add the Addresses Node to the Customer Node
    xeCustomer.Add(xeAddresses);

    xeImport.Add(xeCustomer);

    DoIt(xeImport.ToString(SaveOptions.DisableFormatting));
```

…and would yield the following request:


```xml
<AspDotNetStorefrontImport Version="7.1">
  <Customer Action="Add">
    <EMail>Customer@emailaddress.com</EMail>
    <FirstName>Customer First Name</FirstName>
    <LastName>Customer Last Name</LastName>
    <Password>customerpassword!$*</Password>
    <Phone>1112223333</Phone>
    <IsRegistered>1</IsRegistered>
    <Addresses>
      <Address Action="Add">
        <NickName>Address Nick Name</NickName>
        <FirstName>first name</FirstName>
        <LastName>last name</LastName>
        <Company>company</Company>
        <Address1>address 1</Address1>
        <Address2>address 2</Address2>
        <Suite>address suite</Suite>
        <City>address city</City>
        <State>OH</State>
        <Country>United States</Country>
        <ResidenceType>1</ResidenceType>
        <Phone>1112223333</Phone>
        <Email>Customer@emailaddress.com</Email>
      </Address>
      <Address Action="Add">
        <NickName>2nd Address Nick Name</NickName>
        <FirstName>different first name</FirstName>
        <LastName>different last name</LastName>
        <Company>different company</Company>
        <Address1>2nd address 1</Address1>
        <Address2>2nd address 2</Address2>
        <Suite>2nd address suite</Suite>
        <City>2nd address city</City>
        <State>OH</State>
        <Country>United States</Country>
        <ResidenceType>2</ResidenceType>
        <Phone>1112223333</Phone>
        <Email>Customer@emailaddress.com</Email>
      </Address>
    </Addresses>
  </Customer>
</AspDotNetStorefrontImport>
```

An update would be done in a similar format:

```
XElement xeImport = new XElement("AspDotNetStorefrontImport",
        new XAttribute("Version", "7.1"));

    // Start a Customer Node with the Update Action, and add the data we want to update
    // to the child nodes
    XElement xeCustomer = new XElement("Customer",
        new XAttribute("Action", "Update"),
        new XAttribute("EMail", "Customer@emailaddress.com"),
        new XElement("FirstName", "Updated Customer First Name"),
        new XElement("LastName", "Updated Customer Last Name"),
        new XElement("Phone", "4445556666"));

    // Start an Addresses Node
    XElement xeAddresses = new XElement("Addresses");

    // Start an Address Node with all of the address information for the customer
    XElement xeAddress = new XElement("Address",
        new XAttribute("Action", "Update"),
        new XAttribute("ID", 27),
        new XElement("FirstName", "updated first name"),
        new XElement("LastName", "updated last name"),
        new XElement("Company", "updated company"),
        new XElement("Country", "United States"));

    // Add the Address Node to the Addresses Node
    xeAddresses.Add(xeAddress);

    // Add the Addresses Node to the Customer Node
    xeCustomer.Add(xeAddresses);

    xeImport.Add(xeCustomer);

    DoIt(xeImport.ToString(SaveOptions.DisableFormatting));
```

and would result in a request of:

```
<AspDotNetStorefrontImport Version="7.1">
  <Customer Action="Update" EMail="Customer@emailaddress.com">
    <FirstName>Updated Customer First Name</FirstName>
    <LastName>Updated Customer Last Name</LastName>
    <Phone>4445556666</Phone>
    <Addresses>
      <Address Action="Update" ID="27">
        <FirstName>updated first name</FirstName>
        <LastName>updated last name</LastName>
        <Company>updated company</Company>
        <Country>United States</Country>
      </Address>
    </Addresses>
  </Customer>
</AspDotNetStorefrontImport>
```

3.12. WSI Sample: New Orders

New orders can be retrieved using the Get node, which allows you to “get” any data from any table in the database. You can also use an XmlPackage to format the response for ease of use in the ERP system. To get new orders, you could use something like:

```
XElement xeImport = new XElement("AspDotNetStorefrontImport",
        new XAttribute("Version", "7.1"),
        new XElement("Get",
            new XAttribute("Table", "Orders"),
            new XAttribute("Name", "NewOrders"),
            new XElement("XmlPackage", "DumpOrder.xml.config"),
            new XElement("OrderBy", "OrderDate asc"),
            new XElement("DefaultWhereClause", "IsNew=1")));

DoIt(xeImport.ToString(SaveOptions.DisableFormatting));
```

which would provide a request:

```
<AspDotNetStorefrontImport Version="7.1">
  <Get Table="Orders" Name="NewOrders">
    <XmlPackage>DumpOrder.xml.config</XmlPackage>
    <OrderBy>OrderDate asc</OrderBy>
    <DefaultWhereClause>IsNew=1</DefaultWhereClause>
  </Get>
</AspDotNetStorefrontImport>
```

When you send that request, a response will be similar to:


```
<?xml version="1.0" encoding="utf-8"?>
<AspDotNetStorefrontImportResult Version="7.1" DateTime="1/6/2009 12:57:18 PM">
  <Get Table="Orders" Name="NewOrders" XmlPackage="DumpOrder.xml.config"
IDColumn="OrderNumber" DefaultWhereClause="IsNew=1" OrderBy="OrderDate asc">
    <Order OrderNumber="100000" ShowCardNumber="">
      <OrderNumber>100000</OrderNumber>
      <OrderGUID>8e7b414c-7b3a-49a4-a543-c53e84114410</OrderGUID>
      <ParentOrderNumber>
      </ParentOrderNumber>
      <StoreVersion>AspDotNetStorefront ML 8.0.0.0/8.0.0.0</StoreVersion>
      <QuoteCheckout>0</QuoteCheckout>
      <IsNew>1</IsNew>
      <ShippedOn>
      </ShippedOn>
      <CustomerID>58639</CustomerID>
      <CustomerGUID>a2a983be-8753-40bb-9a3e-e1afd609c591</CustomerGUID>
      <Referrer>
      </Referrer>
      <SkinID>1</SkinID>
      <LastName>User</LastName>
      <FirstName>User</FirstName>
      <Email>user@email.com</Email>
      <Notes>
      </Notes>
      <BillingEqualsShipping>0</BillingEqualsShipping>
      <BillingLastName>User</BillingLastName>
      <BillingFirstName>User</BillingFirstName>
      <BillingCompany>
      </BillingCompany>
      <BillingAddress1>123 Main St</BillingAddress1>
      <BillingAddress2>
      </BillingAddress2>
      <BillingSuite>
      </BillingSuite>
      <BillingCity>New York</BillingCity>
      <BillingState>NY</BillingState>
      <BillingZip>10451</BillingZip>
      <BillingCountry>United States</BillingCountry>
```

```
<BillingPhone>123-456-7890</BillingPhone>
<ShippingLastName>User</ShippingLastName>
<ShippingFirstName>User</ShippingFirstName>
<ShippingCompany>
</ShippingCompany>
<ShippingResidenceType>0</ShippingResidenceType>
<ShippingAddress1>123 Main St</ShippingAddress1>
<ShippingAddress2>
</ShippingAddress2>
<ShippingSuite>
</ShippingSuite>
<ShippingCity>New York</ShippingCity>
<ShippingState>NY</ShippingState>
<ShippingZip>10451</ShippingZip>
<ShippingCountry>United States</ShippingCountry>
<ShippingMethodID>0</ShippingMethodID>
<ShippingMethod>
</ShippingMethod>
<ShippingPhone>123-456-7890</ShippingPhone>
<ShippingCalculationID>7</ShippingCalculationID>
<Phone>
</Phone>
<RegisterDate>11/18/2008 1:42:24 PM</RegisterDate>
<AffiliateID>0</AffiliateID>
<CouponCode>
</CouponCode>
<CouponType>0</CouponType>
<CouponDescription>
</CouponDescription>
<CouponDiscountAmount>0.0000</CouponDiscountAmount>
<CouponDiscountPercent>0.0000</CouponDiscountPercent>
<CouponIncludesFreeShipping>0</CouponIncludesFreeShipping>
<OkToEmail>1</OkToEmail>
<Deleted>0</Deleted>
<CardType>AMEX</CardType>
<CardName>User User</CardName>
<Last4>1234</Last4>
<CardExpirationMonth>03</CardExpirationMonth>
<CardExpirationYear>2009</CardExpirationYear>
<OrderSubtotal>24.7500</OrderSubtotal>
<OrderTax>0.0000</OrderTax>
<OrderShippingCosts>0.0000</OrderShippingCosts>
<OrderTotal>24.7500</OrderTotal>
<PaymentGateway>MANUAL</PaymentGateway>
<AuthorizationCode>0</AuthorizationCode>
<AuthorizationPNREF>6D38543A-A440-4D60-8584-DA0ABCFE12A0</AuthorizationPNREF>
<OrderDate>11/19/2008 12:54:26 AM</OrderDate>
<LevelID>0</LevelID>
<LevelName>
</LevelName>
<LevelDiscountPercent>
</LevelDiscountPercent>
<LevelDiscountAmount>
</LevelDiscountAmount>
<LevelHasFreeShipping>
</LevelHasFreeShipping>
<LevelAllowsQuantityDiscounts>
</LevelAllowsQuantityDiscounts>
<LevelHasNoTax>
</LevelHasNoTax>
<LevelAllowsCoupons>
</LevelAllowsCoupons>
<LevelDiscountsApplyToExtendedPrices>
</LevelDiscountsApplyToExtendedPrices>
<LastIPAddress>::1</LastIPAddress>
<PaymentMethod>CREDITCARD</PaymentMethod>
<OrderNotes>
</OrderNotes>
<RecurringSubscriptionID>
</RecurringSubscriptionID>
<PONumber>
</PONumber>
<DownloadEmailSentOn>
```

```
</DownloadEmailSentOn>
<ReceiptEmailSentOn>11/19/2008 12:54:34 AM</ReceiptEmailSentOn>
<DistributorEmailSentOn>
</DistributorEmailSentOn>
<ShippingTrackingNumber>
</ShippingTrackingNumber>
<ShippedVIA>
</ShippedVIA>
<CustomerServiceNotes>
</CustomerServiceNotes>
<RTShipRequest></RTShipRequest>
<RTShipResponse></RTShipResponse>
<TransactionState>CAPTURED</TransactionState>
<AVSResult>OK</AVSResult>
<CardinalLookupResult>
</CardinalLookupResult>
<CardinalAuthenticateResult>
</CardinalAuthenticateResult>
<CardinalGatewayParms>
</CardinalGatewayParms>
<AffiliateCommissionRecorded>0</AffiliateCommissionRecorded>
<OrderOptions>
</OrderOptions>
<OrderWeight>2.5000</OrderWeight>
<eCheckBankABACode>
</eCheckBankABACode>
<eCheckBankAccountNumber>
</eCheckBankAccountNumber>
<eCheckBankAccountType>
</eCheckBankAccountType>
<eCheckBankName>
</eCheckBankName>
<eCheckBankAccountName></eCheckBankAccountName>
<CarrierReportedRate>
</CarrierReportedRate>
<CarrierReportedWeight>
</CarrierReportedWeight>
<LocaleSetting>en-US</LocaleSetting>
<FinalizationData>&lt;root&gt;&lt;/root&gt;</FinalizationData>
<ExtensionData>
</ExtensionData>
<AlreadyConfirmed>1</AlreadyConfirmed>
<CartType>0</CartType>
<THUB_POSTED_TO_ACCOUNTING>N</THUB_POSTED_TO_ACCOUNTING>
<THUB_POSTED_DATE>
</THUB_POSTED_DATE>
<THUB_ACCOUNTING_REF>
</THUB_ACCOUNTING_REF>
<ReadyToShip>1</ReadyToShip>
<IsPrinted>0</IsPrinted>
<AuthorizedOn>11/19/2008 12:54:26 AM</AuthorizedOn>
<CapturedOn>11/19/2008 12:54:27 AM</CapturedOn>
<RefundedOn>
</RefundedOn>
<VoidedOn>
</VoidedOn>
<InventoryWasReduced>1</InventoryWasReduced>
<MaxMindFraudScore>-1.00</MaxMindFraudScore>
<MaxMindDetails>
</MaxMindDetails>
<CardStartDate>
</CardStartDate>
<CardIssueNumber>
</CardIssueNumber>
<TransactionType>1</TransactionType>
<Crypt>1</Crypt>
<VATRegistrationID>
</VATRegistrationID>
<OrderItems>
  <Item>
    <OrderNumber>100000</OrderNumber>
    <ShoppingCartRecID>1</ShoppingCartRecID>
    <CustomerID>58639</CustomerID>
```

```xml
      <ProductID>1</ProductID>
      <VariantID>1</VariantID>
      <Quantity>5</Quantity>
      <ChosenColor />
      <ChosenColorSKUModifier />
      <ChosenSize />
      <ChosenSizeSKUModifier />
      <OrderedProductName>Simple Product 1</OrderedProductName>
      <OrderedProductVariantName />
      <OrderedProductSKU>01-0001</OrderedProductSKU>
      <OrderedProductManufacturerPartNumber />
      <OrderedProductWeight />
      <OrderedProductPrice>24.7500</OrderedProductPrice>
      <OrderedProductRegularPrice>4.9500</OrderedProductRegularPrice>
      <OrderedProductSalePrice>0.0000</OrderedProductSalePrice>
      <OrderedProductExtendedPrice>0.0000</OrderedProductExtendedPrice>
      <OrderedProductQuantityDiscountName />
      <OrderedProductQuantityDiscountID>0</OrderedProductQuantityDiscountID>
<OrderedProductQuantityDiscountPercent>0.0000</OrderedProductQuantityDiscountPercent>
      <IsTaxable>1</IsTaxable>
      <IsShipSeparately>0</IsShipSeparately>
      <IsDownload>0</IsDownload>
      <DownloadLocation />
      <FreeShipping>1</FreeShipping>
      <IsSecureAttachment>0</IsSecureAttachment>
      <TextOption />
      <CartType>0</CartType>
      <SubscriptionInterval>0</SubscriptionInterval>
      <ShippingAddressID>1</ShippingAddressID>
      <ShippingDetail></ShippingDetail>
      <ShippingMethodID />
      <ShippingMethod />
      <DistributorID />
      <GiftRegistryForCustomerID>0</GiftRegistryForCustomerID>
      <Notes />
      <DistributorEmailSentOn />
      <ExtensionData />
      <SizeOptionPrompt />
      <ColorOptionPrompt />
      <TextOptionPrompt />
      <CreatedOn>11/19/2008 12:54:26 AM</CreatedOn>
      <SubscriptionIntervalType>3</SubscriptionIntervalType>
      <CustomerEntersPrice>0</CustomerEntersPrice>
      <CustomerEntersPricePrompt />
      <IsAKit>0</IsAKit>
      <IsAPack>0</IsAPack>
      <IsSystem>0</IsSystem>
      <TaxClassID>1</TaxClassID>
      <TaxRate>0.0000</TaxRate>
    </Item>
  </OrderItems>
  <OrderPackDetail />
  <OrderKitDetail />
</Order>
<Order OrderNumber="100001" ShowCardNumber="">
  <OrderNumber>100001</OrderNumber>
  <OrderGUID>59bf7a76-34d6-40e5-a5b4-3329f521b9b9</OrderGUID>
  <ParentOrderNumber>
  </ParentOrderNumber>
  <StoreVersion>AspDotNetStorefront ML 8.0.0.0/8.0.0.0</StoreVersion>
  <QuoteCheckout>0</QuoteCheckout>
  <IsNew>1</IsNew>
  <ShippedOn>
  </ShippedOn>
  <CustomerID>58642</CustomerID>
  <CustomerGUID>5bd69835-bd92-4242-8927-bae45abf9d6f</CustomerGUID>
  <Referrer>
  </Referrer>
  <SkinID>1</SkinID>
  <LastName>Customer</LastName>
  <FirstName>Tom</FirstName>
  <Email>tom@email.com</Email>
  <Notes>
```

</Notes>
<BillingEqualsShipping>0</BillingEqualsShipping>
<BillingLastName>Customer</BillingLastName>
<BillingFirstName>Tom</BillingFirstName>
<BillingCompany>
</BillingCompany>
<BillingAddress1>1234 Main St.</BillingAddress1>
<BillingAddress2>
</BillingAddress2>
<BillingSuite>
</BillingSuite>
<BillingCity>Some City</BillingCity>
<BillingState>OH</BillingState>
<BillingZip>44145</BillingZip>
<BillingCountry>United States</BillingCountry>
<BillingPhone>1231231234</BillingPhone>
<ShippingLastName>
</ShippingLastName>
<ShippingFirstName>
</ShippingFirstName>
<ShippingCompany>
</ShippingCompany>
<ShippingResidenceType>0</ShippingResidenceType>
<ShippingAddress1>
</ShippingAddress1>
<ShippingAddress2>
</ShippingAddress2>
<ShippingSuite>
</ShippingSuite>
<ShippingCity>
</ShippingCity>
<ShippingState>
</ShippingState>
<ShippingZip>
</ShippingZip>
<ShippingCountry>
</ShippingCountry>
<ShippingMethodID>0</ShippingMethodID>
<ShippingMethod>
</ShippingMethod>
<ShippingPhone>
</ShippingPhone>
<ShippingCalculationID>7</ShippingCalculationID>
<Phone>1231231234</Phone>
<RegisterDate>11/19/2008 7:38:27 PM</RegisterDate>
<AffiliateID>0</AffiliateID>
<CouponCode>
</CouponCode>
<CouponType>0</CouponType>
<CouponDescription>
</CouponDescription>
<CouponDiscountAmount>0.0000</CouponDiscountAmount>
<CouponDiscountPercent>0.0000</CouponDiscountPercent>
<CouponIncludesFreeShipping>0</CouponIncludesFreeShipping>
<OkToEmail>0</OkToEmail>
<Deleted>0</Deleted>
<CardType>DISCOVER</CardType>
<CardName>Tom Customer</CardName>
<Last4>1234</Last4>
<CardExpirationMonth>02</CardExpirationMonth>
<CardExpirationYear>2010</CardExpirationYear>
<OrderSubtotal>1.9500</OrderSubtotal>
<OrderTax>0.0000</OrderTax>
<OrderShippingCosts>7.6100</OrderShippingCosts>
<OrderTotal>9.5600</OrderTotal>
<PaymentGateway>MANUAL</PaymentGateway>
<AuthorizationCode>0</AuthorizationCode>
<AuthorizationPNREF>56F55C17-F647-4B24-8484-A2F54088005F</AuthorizationPNREF>
<OrderDate>11/19/2008 7:39:40 PM</OrderDate>
<LevelID>0</LevelID>
<LevelName>
</LevelName>
<LevelDiscountPercent>

```
</LevelDiscountPercent>
<LevelDiscountAmount>
</LevelDiscountAmount>
<LevelHasFreeShipping>
</LevelHasFreeShipping>
<LevelAllowsQuantityDiscounts>
</LevelAllowsQuantityDiscounts>
<LevelHasNoTax>
</LevelHasNoTax>
<LevelAllowsCoupons>
</LevelAllowsCoupons>
<LevelDiscountsApplyToExtendedPrices>
</LevelDiscountsApplyToExtendedPrices>
<LastIPAddress>::1</LastIPAddress>
<PaymentMethod>CREDITCARD</PaymentMethod>
<OrderNotes>
</OrderNotes>
<RecurringSubscriptionID>
</RecurringSubscriptionID>
<PONumber>
</PONumber>
<DownloadEmailSentOn>
</DownloadEmailSentOn>
<ReceiptEmailSentOn>11/19/2008 7:39:47 PM</ReceiptEmailSentOn>
<DistributorEmailSentOn>
</DistributorEmailSentOn>
<ShippingTrackingNumber>
</ShippingTrackingNumber>
<ShippedVIA>
</ShippedVIA>
<CustomerServiceNotes>
</CustomerServiceNotes>
<RTShipRequest></RTShipRequest>
<RTShipResponse></RTShipResponse>
<TransactionState>CAPTURED</TransactionState>
<AVSResult>OK</AVSResult>
<CardinalLookupResult>
</CardinalLookupResult>
<CardinalAuthenticateResult>
</CardinalAuthenticateResult>
<CardinalGatewayParms>
</CardinalGatewayParms>
<AffiliateCommissionRecorded>0</AffiliateCommissionRecorded>
<OrderOptions>
</OrderOptions>
<OrderWeight>0.5000</OrderWeight>
<eCheckBankABACode>
</eCheckBankABACode>
<eCheckBankAccountNumber>
</eCheckBankAccountNumber>
<eCheckBankAccountType>
</eCheckBankAccountType>
<eCheckBankName>
</eCheckBankName>
<eCheckBankAccountName></eCheckBankAccountName>
<CarrierReportedRate>
</CarrierReportedRate>
<CarrierReportedWeight>
</CarrierReportedWeight>
<LocaleSetting>en-US</LocaleSetting>
<FinalizationData>&lt;root&gt;&lt;/root&gt;</FinalizationData>
<ExtensionData>
</ExtensionData>
<AlreadyConfirmed>1</AlreadyConfirmed>
<CartType>0</CartType>
<THUB_POSTED_TO_ACCOUNTING>N</THUB_POSTED_TO_ACCOUNTING>
<THUB_POSTED_DATE>
</THUB_POSTED_DATE>
<THUB_ACCOUNTING_REF>
</THUB_ACCOUNTING_REF>
<ReadyToShip>1</ReadyToShip>
<IsPrinted>0</IsPrinted>
<AuthorizedOn>11/19/2008 7:39:40 PM</AuthorizedOn>
```

```
        &lt;CapturedOn&gt;11/19/2008 7:39:41 PM&lt;/CapturedOn&gt;
        &lt;RefundedOn&gt;
        &lt;/RefundedOn&gt;
        &lt;VoidedOn&gt;
        &lt;/VoidedOn&gt;
        &lt;InventoryWasReduced&gt;1&lt;/InventoryWasReduced&gt;
        &lt;MaxMindFraudScore&gt;-1.00&lt;/MaxMindFraudScore&gt;
        &lt;MaxMindDetails&gt;
        &lt;/MaxMindDetails&gt;
        &lt;CardStartDate&gt;
        &lt;/CardStartDate&gt;
        &lt;CardIssueNumber&gt;
        &lt;/CardIssueNumber&gt;
        &lt;TransactionType&gt;1&lt;/TransactionType&gt;
        &lt;Crypt&gt;1&lt;/Crypt&gt;
        &lt;VATRegistrationID&gt;
        &lt;/VATRegistrationID&gt;
        &lt;OrderItems&gt;
          &lt;Item&gt;
            &lt;OrderNumber&gt;100001&lt;/OrderNumber&gt;
            &lt;ShoppingCartRecID&gt;10&lt;/ShoppingCartRecID&gt;
            &lt;CustomerID&gt;58642&lt;/CustomerID&gt;
            &lt;ProductID&gt;2&lt;/ProductID&gt;
            &lt;VariantID&gt;2&lt;/VariantID&gt;
            &lt;Quantity&gt;1&lt;/Quantity&gt;
            &lt;ChosenColor /&gt;
            &lt;ChosenColorSKUModifier /&gt;
            &lt;ChosenSize /&gt;
            &lt;ChosenSizeSKUModifier /&gt;
            &lt;OrderedProductName&gt;Simple Product 2&lt;/OrderedProductName&gt;
            &lt;OrderedProductVariantName /&gt;
            &lt;OrderedProductSKU&gt;01-0002&lt;/OrderedProductSKU&gt;
            &lt;OrderedProductManufacturerPartNumber /&gt;
            &lt;OrderedProductWeight /&gt;
            &lt;OrderedProductPrice&gt;1.9500&lt;/OrderedProductPrice&gt;
            &lt;OrderedProductRegularPrice&gt;4.9500&lt;/OrderedProductRegularPrice&gt;
            &lt;OrderedProductSalePrice&gt;1.9500&lt;/OrderedProductSalePrice&gt;
            &lt;OrderedProductExtendedPrice&gt;0.0000&lt;/OrderedProductExtendedPrice&gt;
            &lt;OrderedProductQuantityDiscountName /&gt;
            &lt;OrderedProductQuantityDiscountID&gt;0&lt;/OrderedProductQuantityDiscountID&gt;

  &lt;OrderedProductQuantityDiscountPercent&gt;0.0000&lt;/OrderedProductQuantityDiscountPercent&gt;
            &lt;IsTaxable&gt;1&lt;/IsTaxable&gt;
            &lt;IsShipSeparately&gt;0&lt;/IsShipSeparately&gt;
            &lt;IsDownload&gt;0&lt;/IsDownload&gt;
            &lt;DownloadLocation /&gt;
            &lt;FreeShipping&gt;0&lt;/FreeShipping&gt;
            &lt;IsSecureAttachment&gt;0&lt;/IsSecureAttachment&gt;
            &lt;TextOption /&gt;
            &lt;CartType&gt;0&lt;/CartType&gt;
            &lt;SubscriptionInterval /&gt;
            &lt;ShippingAddressID&gt;1&lt;/ShippingAddressID&gt;
            &lt;ShippingDetail&gt;&lt;/ShippingDetail&gt;
            &lt;ShippingMethodID&gt;5&lt;/ShippingMethodID&gt;
            &lt;ShippingMethod&gt;UPS Ground|7.61&lt;/ShippingMethod&gt;
            &lt;DistributorID /&gt;
            &lt;GiftRegistryForCustomerID&gt;58639&lt;/GiftRegistryForCustomerID&gt;
            &lt;Notes /&gt;
            &lt;DistributorEmailSentOn /&gt;
            &lt;ExtensionData /&gt;
            &lt;SizeOptionPrompt /&gt;
            &lt;ColorOptionPrompt /&gt;
            &lt;TextOptionPrompt /&gt;
            &lt;CreatedOn&gt;11/19/2008 7:39:40 PM&lt;/CreatedOn&gt;
            &lt;SubscriptionIntervalType&gt;3&lt;/SubscriptionIntervalType&gt;
            &lt;CustomerEntersPrice&gt;0&lt;/CustomerEntersPrice&gt;
            &lt;CustomerEntersPricePrompt /&gt;
            &lt;IsAKit&gt;0&lt;/IsAKit&gt;
            &lt;IsAPack&gt;0&lt;/IsAPack&gt;
            &lt;IsSystem /&gt;
            &lt;TaxClassID&gt;1&lt;/TaxClassID&gt;
            &lt;TaxRate&gt;0.0000&lt;/TaxRate&gt;
          &lt;/Item&gt;
```

```
    </OrderItems>
    <OrderPackDetail />
    <OrderKitDetail />
  </Order>
</Get>
     </AspDotNetStorefrontImportResult>
```

4. In-Process Providers (AspDotNetStorefront Add-In Model)

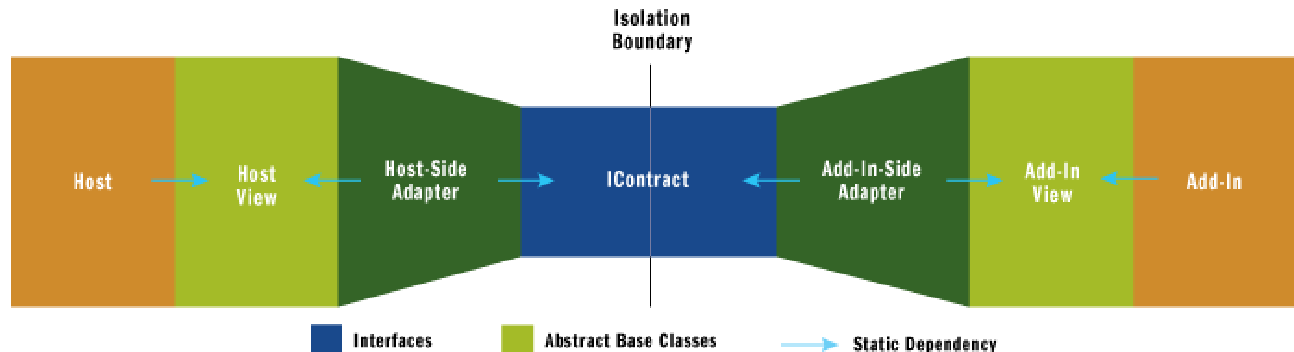
4.1. Introduction to the AspDotNetStorefront Add-In Model

The AspDotNetStorefront Add-In Model provides third party developers with the ability to extend the functionality of AspDotNetStorefront by authoring Add-Ins. An add-in is traditionally thought of as custom code (a customization), written by a third party that is loaded and activated by a host application (AspDotNetStorefront) upon the host starting up. The host makes their programmable API (Object Model) available to the customization assembly. It is expected that developers will use the Add-In model to implement specific business rules, or provide functionality beyond what is included in the base product.

The AspDotNetStorefront Add-In Model is based on the Microsoft Managed Add-In Framework (MAF), included with .NET Framework 3.5 SP1 and higher. MAF defines a programming model, built on top of .NET that allows applications to dynamically load and communicate with third party components at runtime.

Add-Ins run in the same process as the host, and they communicate with the host via an agreed-upon communications protocol, called the Contract. Each of the APIs implements a specific Contract to define communications between the host and the add-in.

Add-Ins (also known as Add-Ons, Extensions, Plug-Ins, and/or Snap-Ins) provide custom extensibility to an existing application. They are dynamically located and activated by a host application based upon some context within the host application (such as application start or a page load). Add-Ins reside in their own folder, have their own application base, load their own assemblies without conflicting with the host or other Add-Ins, and can use their own configuration files. This allows the host application to catch errors in an Add-In and shut the Add-In down without impacting the host or other Add-Ins.



With the Add-In communication pipeline an isolation boundary exists, segregating the host from the Add-In and providing isolation from other components that may be communicating with it as there is no direct reference to the host from the Add-In, or vice-versa.

In the above diagram, the Host is the application that supports the extensibility (AspDotNetStorefront). The Host View defines the way the Host sees any Add-Ins; what abstract classes, methods, or variables are available for the Host to use to call on the Add-In. The Host-Side Adapter is responsible for converting from the Contract to the Host View, and for converting from the Host View to the Contract. The adapter derives from the Host View for a specific Contract and as an implementation calls methods on an instance referenced by the Contract interface.

The Contract interface, IContract, is the interface between the Host and the Add-In and contains the available operations that can be performed from Host to Add-In and back.

The Add-In-Side Adapter is responsible for converting from the Add-In View to the Contract, and from the Contract to the Add-In View. The adapter derives from the BaseContract (System.AddIn.Pipeline.ContractBase) and from the specific Contract, and as an implementation calls methods on an instance of the Add-In View. The Add-In View is the base for Add-Ins, and is the interface defining the operations that the Host can call on an Add-In.

4.2. Extending AspDotNetStorefront with Add-Ins

The following code samples show ways in which Add-Ins can be used to extend the capabilities of AspDotNetStorefront.

4.2.1. Code Sample #1: Implementing a 3rd – Party Shipping Carrier

This example demonstrates how an Add-In can be used to return additional shipping rates while also using a carrier natively integrated into AspDotNetStorefront. This example uses UPS real time shipping rates in AspDotNetStorefront, and integrates FedEx shipping rates.

The Add-In must be decorated with the Add-In attribute from the System.AddIn namespace. Here, the Add-In starts with a class that implements the IShippingRates Add-In View:

```
[System.AddIn.AddIn("Shipping Rates Add In",
    Version = "1.0.0.0",
    Description = "Add-In to retrieve and return FedEx shipping rates externally.")]
public class FedExRates : IShippingRates
```

The IShippingRates view exposes a single method with a return type of IShippingMethodCollection, GetRates(), which accepts a collection of packages being shipped, and a collection of shopping cart items. This method is required by the Add-In View, ergo we must implement it within the Add-In:

```
public IShippingMethodCollection GetRates(IPackageCollection pCollection, IShoppingCart sCart)
{
    var newShipMethods = new ShippingMethodCollection();
```

We then enumerate through each item in the package collection to build a request to be sent to FedEx:

```
foreach (IPackage p in pCollection.Packages())
{
    System.DateTime TomorrowsDate = System.DateTime.Now.AddDays(1);

    XDocument FedExRequest = new XDocument(
        new XDeclaration("1.0", "UTF-8", "no"),
        new XElement("FDXRateAvailableServicesRequest",
        new XAttribute(XNamespace.Xmlns + "api", "http://www.fedex.com/fsmapi"),
        new XAttribute(XNamespace.Xmlns + "xsi", "http://www.w3.org/2001/XmlSchema-instance"),
        new XAttribute(XNamespace.Xmlns + "noNamespaceSchemaLocation",
"FDXRateAvailableServicesRequest.xsd"),
            new XElement("RequestHeader",
                new XElement("CustomerTransactionIdentifer", "RatesRequest"),
                new XElement("AccountNumber", 123456789),
                new XElement("MeterNumber", 0123456),
                new XElement("CarrierCode", "FDXE")),
            new XElement("ShipDate",
                TomorrowsDate.Year.ToString() +
                "-" + TomorrowsDate.Month.ToString().PadLeft(2, '0') +
                "-" + TomorrowsDate.Day.ToString().PadLeft(2, '0')),
```

```
            new XElement("DropoffType", "REGULARPICKUP"),
            new XElement("Packaging", "YOURPACKAGING"),
            new XElement("WeightUnits", "LBS"),
            new XElement("ListRate", false),
            new XElement("Weight", p.Weight()),
            new XElement("OriginAddress",
                new XElement("StateOrProvinceCode", pCollection.OriginStateProvince()),
                new XElement("PostalCode", pCollection.OriginZipPostalCode()),
                new XElement("CountryCode", pCollection.OriginCountryCode())),
            new XElement("DestinationAddress",
                new XElement("StateOrProvinceCode", pCollection.DestinationStateProvince()),
                new XElement("PostalCode", pCollection.DestinationZipPostalCode()),
                new XElement("CountryCode", pCollection.DestinationCountryCode())),
            new XElement("Payment",
                new XElement("PayorType", "SENDER")),
            new XElement("DeclaredValue",
                new XElement("Value", p.InsuredValue().Amount()),
                new XElement("CurrencyCode", p.InsuredValue().CurrencyCode())),
            new XElement("PackageCount", 1),
            new XElement("SpecialServices",
                new XElement("ResidentialDelivery", 1))));
```

We then format the request in a way that the carriers' API expects and send it off:

```
using (StringWriter sw = new StringWriter())
{
    using (XmlTextWriter xtw = new XmlTextWriter(sw))
    {
        FedExRequest.WriteTo(xtw);

        RTShipRequest = sw.ToString();

        xtw.Close();
    }
    sw.Close();
}

String result = POSTandReceiveData(RTShipRequest, "https://gateway.fedex.com:443/GatewayDC");
```

We then take the response and parse it into an xml document so we can get the rates. We can also catch any errors here to provide ourselves with troubleshooting information should something go wrong with the Add-In

```
// Load Xml into a XmlDocument object
XDocument FedExResponse = new XDocument();
try
{
    FedExResponse = XDocument.Parse(result);
}
catch
{
    // you would do your error logging here and return an empty set of shipping
    // methods back to the host.  This prevents the add-in from crashing the application
    // incase anything goes wrong.  However, error handling is also performed on the
    // host side to prevent application failure in the event of add-in failure.
    return new ShippingMethodCollection();
}
```

We would then enumerate through the response document and populate the shipping method collection with the rates and methods returned from the carrier:

```
// Get rates
IEnumerable<XElement> nodesEntries = FedExResponse.Descendants().Where(e => e.Name == "Entry");

foreach (XElement xe in nodesEntries)
{
    string origRateName = FedExGetCodeDescription(xe.Descendants().Single(e => e.Name ==
"Service").Value);
```

```
    string rateName = "(From Shipping Rates Add-In) FedEx " + origRateName;
    decimal totalCharges = decimal.Parse(xe.Descendants().Single(e => e.Name == "NetCharge").Value);

    var newShipMethod = new ShippingMethod();

    newShipMethod.m_carrier = "FEDEX";
    newShipMethod.m_serviceName = rateName;
    newShipMethod.m_serviceRate = new ShippingMoney { m_amount = totalCharges };
    newShipMethod.m_vatRate = new ShippingMoney { m_amount = 0.00M };

    newShipMethods.ShippingMethods().Add(newShipMethod);
}
```

Finally we return the shipping method collection to the Add-In View, which is converted to the Contract via the Add-In-Side Adapter, where the Host-Side Adapter then converts to the Host-Side View for consumption by the Host:

```
return newShipMethods;
```

Because you also have a collection of shopping cart items, you can also do things like penny shipping or free shipping on a particular method/rate combination based on the subtotal of items in the cart:

```
// get the subtotal of items in the shopping cart
decimal cartSubTotal = sCart.Items().Sum(ci => (ci.Price().Amount() * ci.Quantity()));

// if the order is between 50.00 and 150.00 and shipping domestically (U.S.)
// offer penny shipping for the Express Saver shipping method returned by FedEx
if (cartSubTotal > 50.00M && cartSubTotal <= 150.00M &&
pCollection.DestinationCountry().Equals("United States", StringComparison.OrdinalIgnoreCase))
{
    if (origRateName.Equals("Express Saver", StringComparison.OrdinalIgnoreCase))
    {
        newShipMethod.m_serviceRate = new ShippingMoney { m_amount = 0.01M };
    }
}

// if the order is greater than 150.00 and shipping domestically (U.S.)
// offer free shipping for the Express Saver shipping method returned by FedEx
if (cartSubTotal > 150.00M && pCollection.DestinationCountry().Equals("United States",
StringComparison.OrdinalIgnoreCase))
{
    if (origRateName.Equals("Express Saver", StringComparison.OrdinalIgnoreCase))
    {
        newShipMethod.m_serviceName = "FedEx Express Saver (Free Shipping)";
        newShipMethod.m_serviceRate = new ShippingMoney { m_amount = 0.00M };
    }
}
```

In summary, this example shows how any third party shipping provider could be integrated into AspDotNetStorefront to be used in conjunction with existing shipping rates and shipping functionality using an Add-In built against the shipping rates contract.

4.3.Sample Add-in Contract

An Add-In Contract is an agreement between AspDotNetStorefront and a third-party developer. It is guaranteed to never change, ensuring that an Add-In DLL will continue to run in future versions of the software. It establishes the data that the Add-In expects to receive, and the data expected to be returned to AspDotNetStorefront.

Note that these data are the only data available to an Add-In. For security reasons, Add-Ins do not have access to AspDotNetStorefront Core routines or the SQL Server database. They run in the permission model allowed by the host, or defined by the websites' machine.config file.

Although Contracts define the interface between AspDotNetStorefront and the Add-In, it is not what developers will see or use. The Contract generates a "View", which is a set of Interfaces used by the Add-In developer.

Here is the "View" of the Shipping Rates Contract. It consists of one main interface that must be implemented by the Add-In developer:

```
[System.AddIn.Pipeline.AddInBaseAttribute()]
public interface IShippingRates
{
    IShippingMethodCollection GetRates(IPackageCollection packages, IShoppingCart cartContents);
}
```

Note that the interface is decorated with a AddInBaseAttribute from the System.AddIn.Pipeline namespace. This attribute tells the Managed Add-In Framework (MAF) that this interface represents a View that Add-In developers can use.

The GetRates() method received a collection of packages and a collection of shopping cart items as its parameters. The contents of that collection are defined in the View:

```
public interface IPackageCollection
{
    string OriginCity();
    string OriginStateProvince();
    string OriginZipPostalCode();
    string OriginCountryCode();
    string DestinationCity();
    string DestinationStateProvince();
    string DestinationZipPostalCode();
    string DestinationCountry();
    string DestinationCountryCode();
    ResidenceTypeFlags DestinationResidenceType();
    bool HasDistributorItems();
    bool HasFreeShippingItems();
    decimal Weight();
    IMoney Value();
    IMoney ExtraFee();
    decimal MarkupPercent();
    decimal ShippingTaxRate();
    System.Collections.Generic.IList<IPackage> Packages();
}

public interface IPackage
{
    int Quantity();
    bool IsShipSeparately();
    bool IsFreeShipping();
    IMoney InsuredValue();
    int PackageId();
    bool Insured();
    decimal Width();
    decimal Weight();
    decimal Height();
    decimal Length();
}

public interface IShoppingCart
{
    System.Collections.Generic.IList<ICartItem> Items();
    System.Collections.Generic.IList<IOrderOption> OrderOptions();
}
```

```
public interface ICartItem
{
    string ProductSku();
    decimal Quantity();
    IMoney Price();
    IMoney PriceExtended();
    System.Collections.Generic.IList<ICartItemOption> CartItemOptions();
}
```

The GetRates() method is required to return (provide as output back to AspDotNetStorefront) a collection of shipping methods. The contents of this collection are also defined in the View:

```
public interface IShippingMethodCollection
{
    string ErrorMsg();
    string RTShipRequest();
    string RTShipResponse();
    System.Collections.Generic.IList<IShippingMethod> ShippingMethods();
}

public interface IShippingMethod
{
    int ShippingMethodID();
    string ServiceName();
    IMoney ServiceRate();
    IMoney VatRate();
    string Carrier();
    IMoney FreeItemsRate();
}
```

A developer is required to write his Add-In in such a way as to consume the input parameter (the package collection), and return the output parameter (the shipping method collection). He may choose to ignore the inputs he doesn't care about, and the outputs he wants to leave at their default values.

Beyond those requirements defined by the Contract (and implemented in the View), he can use whatever means necessary to perform his work, including calling external web services, reading configuration parameters from a configuration file (like web.config), or accessing the file system (depending on the trust level afforded to the website by the host). He cannot, however, call AspDotNetStorefront Core routines, or access internal data structures. The only data provided to (and expected from) the Add-In is defined in the Contract, via the View.

4.4. Add-In Reference

AspDotNetStorefront provides the following API's, implemented via Contracts, that can be used by Add-Ins to tie into the runtime, and extend the capabilities of the website. The Contracts (APIs) are:

4.4.1. Inventory Add-In

Description:

This contract allows Add-In developers to retrieve real-time inventory levels for one or more Product SKUs, to restrict shoppers to purchasing only items that are in-stock, or to hide items that are out of stock.

Contract Definition:

```
/// <summary>
/// Inventory AddIn Contract - Represents a collection of ShoppingCart items with
inventories based on query type: Browsing, AddToCart, Checkout
/// </summary>
[AddInContract]
public interface IInventoryContract : IContract
{
    // Provide list of Product SKUs, ProductIds, VariantIds, Color/Size Modifiers
    // return same list, with Inventories
    IShoppingCartContract GetInventories(IShoppingCartContract products,
InventoryRequestTypeFlags requestType);
}
```

Notes:

- The Add-In is expected to return a modified Shopping Cart, based on real-time inventory levels maintained in an existing, external system.
- Items that are restricted from purchase, due to out-of-stock conditions for example, will be removed from the Shopping Cart.
- A Shopping Cart (collection of cart items) is passed to the Add-In.
- A Request Type is also passed, indicating if the Add-In has been invoked due to an Add to Cart or a Checkout. The Add-In can use this information as required, perhaps relying on cached inventory information versus real-time calls to the external system.

4.4.2. Shipping Modifiers Add-In

Description:

This contract will allow developers to implement algorithms for modifying a list of shipping methods, by either injecting new methods into the list (Will-Call Shipping), removing specific methods (limitations based on distance, travel-time, or means of travel, such as air or ground), or adjusting shipping rates for specific shipping methods (free shipping, penny-shipping, handling charges, box / crate fees). Total weight or the items in the cart, as well as the subtotal of the items in the cart are also available here.

Contract Definition:

```
/// <summary>
/// Shipping Methods Modifier Contract - Represents a collection of shipping methods
/// modified from the original collection of shipping methods.
/// </summary>
[AddInContract]
public interface IShippingModifiersContract : IContract
{
    IShippingMethodCollectionContract
ModifyShippingMethods(IShippingMethodCollectionContract originalShipMethods,
IShoppingCartContract cartContents);
}
```

Notes:

- The Add-In is expected to return a collection of Shipping Methods.
- The original collection of shipping methods, and a collection of cart items are passed as parameters to the Add-In.

- The Add-In is expected to either inject new methods into the collection, or adjust shipping rates and properties on existing methods, however the shipping rates add-in should be used for adding entirely new collections of shipping methods.

Example Usage:

```
namespace ShippingModifierAddIn
{
    [AddIn("Shipping Modifier Add In",
    Version = "1.0.0.0",
    Description = "Add in for modifying natively returned shipping methods from UPS.")]
    public class ShippingModifier : IShippingModifiers
    {
        public IShippingMethodCollection ModifyShippingMethods(IShippingMethodCollection
originalShipMethods, IShoppingCart cartContents)
        {
            var newShipMethods = new ShippingMethodCollection();

            foreach (IShippingMethod shipMethod in originalShipMethods.ShippingMethods())
            {
                var newShipMethod = new ShippingMethod();

                if (shipMethod.ServiceName() == "UPS Ground")
                {
                    newShipMethod.m_serviceName = "Free UPS Ground Shipping";
                    newShipMethod.m_serviceRate = new ShippingMoney { m_amount = 0.0M };
                }
                else
                {
                    newShipMethod.m_serviceName = shipMethod.ServiceName();
                    newShipMethod.m_serviceRate = shipMethod.ServiceRate();
                }

                newShipMethod.m_carrier = shipMethod.Carrier();
                newShipMethod.m_freeItemsRate = shipMethod.FreeItemsRate();
                newShipMethod.m_shippingMethodID = shipMethod.ShippingMethodID();
                newShipMethod.m_vatRate = shipMethod.VatRate();


                newShipMethods.ShippingMethods().Add(newShipMethod);
            }

            decimal cartSubTotal = cartContents.Items().Sum(ci => (ci.Price().Amount() *
ci.Quantity()));

            // inject a shipping method into the collection based on a subtotal greater than 500.00
            if (cartSubTotal > 500.0M)
            {
                var newShipMethod2 = new ShippingMethod();
                newShipMethod2.m_carrier = "Call for special rates";
                newShipMethod2.m_serviceName = "Call for special rates";
                newShipMethod2.m_freeItemsRate = new ShippingMoney { m_amount = 0.00M };
                newShipMethod2.m_serviceRate = new ShippingMoney { m_amount = 0.00M };
                newShipMethod2.m_shippingMethodID = 0;
                newShipMethod2.m_vatRate = new ShippingMoney { m_amount = 0.00M };
                newShipMethods.ShippingMethods().Add(newShipMethod2);
            }

            return newShipMethods;
        }
    }
}
```

4.4.3. Shipping Rates Add-In

Description:

This contract will allow developers to invent other shipping rate engines, beyond the capabilities provided by our real-time and table-based shipping rate methods.

Contract Definition:

```
/// <summary>
/// Shipping Rates AddIn Contract - Represents a collection of shipping methods to
/// be calculated and presented from outside the application
/// </summary>
[AddInContract]
public interface IShippingRatesContract : IContract
{
    IShippingMethodCollectionContract GetRates(IPackageCollectionContract packages,
IShoppingCartContract cartContents);
}
```

Notes:

- Add-Ins implementing this API are required to return a collection of Shipping Methods, which the customer will choose from.
- A collection of packages representing a shipment from a Ship-From Address to a Ship-To Address, as well as a collection of shopping cart items will be passed to the Add-In.
- Multiple Add-In calls per order are possible if Distributor shipping or multiple Ship-To addresses are enabled on a website.

Example Usage:

```
using System.Collections.Generic;
using AspDotNetStorefront;
using AspDotNetStorefront.AddInView;

namespace ShippingMethodsFromErp
{
    [System.AddIn.AddIn("Shipping Methods From ERP", Description = "Retrieves Shipping Methods from ERP
System")]
    public class ShippingMethodsFromErpAddIn : IShippingRates
    {
        public IShippingMethodCollection GetRates(IPackageCollection packages)
        {
            var shipMethods = new ShippingMethodCollection();

            // Enumerate over collection of packages in shipment
            foreach (var package in packages.Packages())
            {
                // create a new request object
                var ErpPackage = new ErpRequest();

                // populate required values for request
                ErpPackage.Weight = package.Weight();
                ErpPackage.Height = package.Height();
                ErpPackage.Length = package.Length();
                ErpPackage.Width = package.Width();
                ErpPackage.DestinationCity = packages.DestinationCity();
                ErpPackage.DestinationCountry = packages.DestinationCountry();
                ErpPackage.DestinationPostalCode = packages.DestinationZipPostalCode();

                // call ERP System to retrieve shipping methods and costs for this package
                var ErpShipMethods = GetShippingMethodsFromErp(ErpPackage);
```

```
            // enumerate over returned shipping methods
            foreach (var ErpShipMethod in ErpShipMethods)
            {
                // add shipping method to collection
                shipMethods.ShippingMethods().Add(new ShippingMethod
                {
                    _carrier = "FromErp",
                    _serviceName = ErpShipMethod._serviceName,
                    _serviceRate = ErpShipMethod._serviceRate
                });
            }
        }

        // return shipping methods and costs to shopping cart
        return shipMethods;
    }

    private List<ShippingMethod> GetShippingMethodsFromErp(ErpRequest ErpPackage)
    {
        var shipMethodList = new List<ShippingMethod>();
        shipMethodList.Add(new ShippingMethod()
                           {
                               _carrier = "Erp",
                               _serviceName = "Erp Method #1",
                               _serviceRate = new Money { _amount = 1.23M, _currencyCode = "USD" }
                           });
        shipMethodList.Add(new ShippingMethod()
                           {
                               _carrier = "Erp",
                               _serviceName = "Erp Method #2",
                               _serviceRate = new Money { _amount = 4.56M, _currencyCode = "USD" }
                           });

        return shipMethodList;
    }
}
```

4.4.4. Payment Gateway/Merchant Account Add-In

Description:

This contract will allow developers to implement their own payment processing logic while maintaining the integrity of the existing core logic and native data storage logic.

Contract Definition:

```
/// <summary>
/// Payment Gateway AddIn Contract - Represents the responses provided by a merchant after
/// a transaction has been processed
/// </summary>
[AddInContract]
public interface IGatewayContract : IContract
{
    IGatewayResponseContract ProcessTender(IProcessTenderContract processTender);
    IGatewayResponseContract CaptureTender(ITenderContract tender);
    IGatewayResponseContract RefundTender(ITenderContract tender);
    IGatewayResponseContract VoidTender(ITenderContract tender);
}
```

Notes:

4.4.5. Order Fulfillment Add-In

Description:

This contract defines a method to be called from an Add-In for processing order fulfillment logic. This contract has no returnable data.

Contract Definition:

```
/// <summary>
/// Order Fulfillment AddIn Contract - Represents the action taken when an order
/// requiring fulfillment is fulfilled.
/// </summary>
[AddInContract]
public interface IFulfillmentContract : IContract
{
    void ProcessShipment(IOrderContract order);
}
```

Notes:

4.4.6. Tax Rates Add-In

Description:

This contract will allow a developer to integrate a service to calculate the correct tax based on the address of a purchasing customer.

Contract Definition:

```
/// <summary>
/// Tax Rates AddIn Contract - Represents the tax rate of a given address.
/// </summary>
[AddInContract]
public interface ITaxRatesContract : IContract
{
    decimal GetTaxRate(IAddressContract shipToAddress);
}
```

Notes:

4.4.7. Order Packing Add-In

Description:

This contract will allow developers to create their own order packing logic for sorting and processing the contents of a completed order.

Contract Definition:

```
/// <summary>
```

```
/// Order Packing AddIn Contract - Represents a collection of packages to process for
packing.
/// </summary>
[AddInContract]
public interface IPackOrderContract : IContract
{
    IPackageCollectionContract PackOrder(IShoppingCartContract cart, IAddressContract
shipToAddress, bool includeFreeItems);
}
```

Notes:

4.4.8. Order Options Add-In

Description:

This contract allows developers to present their own list of additional options that can be chosen during the checkout process, such as a list of currently available gift wrap options from an external application.

Contract Definition:

```
/// <summary>
/// Order Options AddIn Contract - Represents a collection of order options that can be
/// presented during the checkout process.
/// </summary>
[AddInContract]
public interface IOrderOptionsContract : IContract
{
    IListContract<IOrderOptionContract> GetOrderOptions();
}
```

Notes:

4.4.9. XSLT Extension Add-In

Description:

This contract allows developers to implement their own extension logic for use in XmlPackages.

Contract Definition:

```
/// <summary>
/// XSLT Extension AddIn Contract - Represents data processed and returned to be used
/// in XmlPackages.
/// </summary>
[AddInContract]
public interface IXsltExtensionContract : IContract
{
    IXsltExtensionResponseContract ProcessXsltExtension(ICustomerContract customer, int
skinId);
}
```

Notes:

4.4.10. Line Item Pricing Add-In

Description:

This contract provides the ability to retrieve/modify pricing on a per-item basis for items currently in a shopping cart during the checkout process.

Contract Definition:

```
/// <summary>
/// Shopping Cart Line-Item Pricing AddIn Contract - Represents a collection of
/// shopping cart items for per-item pricing.
/// </summary>
[AddInContract]
public interface ILineItemPricingContract : IContract
{
    IShoppingCartContract GetLineItemPrice(IShoppingCartContract products);
}
```

Notes:

4.4.11. Order Notification Add-In

Description:

This contract will allow a developer to extend upon the existing new order notification after an order has been successfully placed within the store.

Contract Definition:

```
/// <summary>
/// New Order Notification AddIn Contract - Represents the action taken when a new order has
been
/// successfully placed on the website and internal order processing has completed.
/// </summary>
[AddInContract]
public interface IOrderNotificationContract : IContract
{
    void ProcessOrderNotification(IOrderContract order);
}
```

Notes:

5. Security Considerations

5.1. General Security Considerations

The largest security concern impacting integration will primarily be the proximity of the web server to the ERP system. Nothing in this specification prevents operation over a wide area network, and while for performance reason close proximity to the ERP system is recommended, in some cases this may not be possible or feasible. Whenever communication occurs over a network outside of the administrator's control, extra precaution should be taken to ensure the integrity of communications, especially when transferring sensitive data. The following section covers some basic recommended security practices to help prevent attacks against the different interfaces in use for integration purposes.

5.2. WSI

Install the IPX.asmx files in a subdirectory off of the root of the AspDotNetStorefront. By doing so, you can use IIS to control access to the files. First and foremost, IIS should be configured to deny access to that directory from any IP address access those specifically allowed within the IIS directory security configuration. The directory can also be given a unique name, making it more difficult for a potential attacker to know exactly what to attack.

If possible, use WSE 3 authentication (see WSI examples), which encrypts sensitive authentication data at the message layer. If this is not possible do to environmental constraints (eg. you must connect to WSI using a non-windows-based system), the WSI directory should be specifically restricted to only allow SSL connections. This will cause a failure any time anyone attempts to connect via the HTTP protocol, effectively preventing communication if a developer or store administrator forgets to take proper security precautions.

5.3. Event Handlers

It is important to keep in mind that the event handlers and subsequent notifications were not designed to pass secure or sensitive data, as the notification is an unauthenticated exchange. An event notification would have no concept of whether it just passed the notification to an attacker or a valid event notification end point. Event notifications can occur over SSL, however by default no sensitive information is returned in the event notification, so it is not necessary.

The danger with event handlers is that they allow a developer to take full advantage of the AspDotNetStorefront XML package engine when developing custom event notification packages. This means that a developer could potentially write an event notification package that returns private data over an unauthenticated connection. When developing any custom event handler packages, ensure that the event notification passes only that information which is necessary to retrieve the full details of the transaction via WSI, which implements robust security features.

5.4. Event Listeners

Similar to WSI, event listener services should be restricted (at the application level, network level, or both) to only those IP addresses authorized to communicate with the service. While

following proper precautions as outlined above negates the need for authentication, it is important to treat event notifications as untrusted. For example, inserting data received from an event notification directly into a database, or using it to generic a dynamic SQL query without proper sanitation, would not be a good idea.

5.5.Add-Ins

The add-in model is where the greatest degree of precaution should be exercised by the developer, especially when writing payment provider add-ins. Because payment providers often require sensitive data, it is important that this data be taken into account when developing the add-in. Developers writing add-ins that handle sensitive data are encouraged to build security directly into the add-in component. This would include things such as providing a means of application-level encryption and decryption of sensitive data, allowing for robust authentication (including two-factor authentication such as username/password authentication in conjunction with client certificates), and enforcing transport-layer security such as SSL encryption and IP address restrictions whenever using add-ins to communicate via web services.

6. Additional Resources

AspDotNetStorefront WSI Manual
http://manual.aspdotnetstorefront.com/wsi

AspDotNetStorefront XML Package Specification v2.1
http://manual.aspdotnetstorefront.com/pdf/XmlPackages2.1.pdf

MSDN .NET Add-In Overview
http://msdn.microsoft.com/en-us/library/bb384200.aspx

7. Appendix A: Considerations for Integration

7.1.Customer Records

For stores that support anonymous checkout, the most likely way to link the anonymous customer record to the ERP system would be using a "generic" customer account within the ERP system. This prevents enterprise data from being diluted by potentially thousands of one-time purchasers.

Existing customers could be pushed to the website using WSI. Since most ERP systems don't store customer passwords, a new GUID could be generated and passed to WSI, where it would be hashed and stored in the database. The first time a customer logged into the site they would be able to auto-generate a new password using the built-in lost password functionality within AspDotNetStorefront.

Primary key data for customers (eg. company id, contact id) can be stored within the existing customer table extension data fields as XML fragments. This method would work for nearly any other object in AspDotNetStorefront that needed to be tied to an ERP system primary key as well.

A "tiered" customer system may need to be implemented within AspDotNetStorefront to support standard ERP-style multi-level records (eg. where contacts are part of a parent company, but contacts can also have their own details).

Because customers can update their details on the AspDotNetStorefront website, a system integrator may want to consider placing customer initiated updates in a staging area awaiting administrator approval. The update will "appear" to have completed on the website, however should the administrator not approve the update, the proper details can be pushed back to the website via WSI.

All pricing and tax rules for new customers registering on or browsing the website would likely come from a "generic" customer defined in the ERP system.

7.2.Orders

It is assumed that all order management and history would be provided from the ERP system. Order management actions on the AspDotNetStorefront admin site will be ignored due to its ignorance of processes and rules in place for the enterprise.

7.3.Inventory

It is likely desirable that inventory control on the website support a variety of methods for updates (this can be done using the Inventory Add-In, for which examples provided earlier). For the sake of performance, it is not practical to do real-time checks of inventory levels on entity page loads due to the large number of products that could potentially be queried at once. Entity pages, and possibly product pages, would likely be queried from a local cache. On the other hand, checkout may need to do real-time verification of stock levels prior to actually allowing the order to be placed. A combination of techniques could be used to achieve the desired performance level and stock safety on the website. These techniques would include:

- Pushing inventory updates to the website via WSI from the ERP system any time an inventory update is made
- Querying the local site inventory database for entity page calls, which should be reasonably up to date
- During checkout (and possibly on product pages) use an in-process add-in to pull realtime data from the ERP system (the add-in could also perform caching).

AspDotNetStorefront includes options for allowing backorders and controlling out-of-stock product display, so few, if any, changes should be needed for this functionality. The in-process inventory provider would be used to return the stock level, but standard functionality would control what occurred based on the results of the stock check.