ektron

What do you want your website to do?

Ektron eCommerce QuickStart Guide

For Administrators and Developers

Using CMS400.NET to Generate Revenue and Grow your Business

ektron CMS400.net

EKTRON, INC. SOFTWARE LICENSE AGREEMENT

YOUR RIGHT TO USE THE PRODUCT DELIVERED IS SUBJECT TO THE TERMS AND CONDITIONS SET OUT IN THIS LICENSE AGREEMENT. USING THIS PRODUCT SIGNIFIES YOUR AGREEMENT TO THESE TERMS. IF YOU DO NOT AGREE TO THIS SOFTWARE LICENSE AGREEMENT, DO NOT DOWNLOAD.

CUSTOMER should carefully read the following terms and conditions before using the software program(s) contained herein (the "Software"). Downloading and/or using the Software or copying the Software onto CUSTOMER'S computer hard drive indicates CUSTOMER'S acceptance of these terms and conditions. If CUSTOMER does not agree with the terms of this agreement, CUSTOMER should not download.

Ektron, Inc. ("Ektron") grants, and the CUSTOMER accepts, a nontransferable and nonexclusive License to use the Software on the following terms and conditions:

1. Right to use: The Software is licensed for use only in delivered code form. Each copy of the Software is licensed for use only on a single URL. Each license is valid for the number of seats listed below (the "Basic Package"). Any use of the Software beyond the number of authorized seats contained in the Basic Package without paying additional license fees as provided herein shall cause this license to terminate. Should CUSTOMER wish to add seats beyond the seats licensed in the Basic Package, the CUSTOMER may add seats on a block basis at the then current price for additional seats (see product pages for current price). The Basic Packages are as follows:

Ektron CMS400.NET — Licensed for ten seats (10 named users) per URL.

Ektron eWebEditPro — Licensed for ten seats (10 named users) per URL.

Ektron eWebEditPro+XML — Licensed for ten seats (10 named users) per URL.

For purposes of this section, the term "seat" shall mean an individual user provided access to the capabilities of the Software.

The CUSTOMER may not modify, alter, reverse engineer, disassemble, or decompile the Software. This software product is licensed, not sold.

2. Duration: This License shall continue so long as CUSTOMER uses the Software in compliance with this License. Should CUSTOMER breach any of its obligations hereunder, CUSTOMER agrees to return all copies of the Software and this License upon notification and demand by Ektron.

3. Copyright: The Software (including any images, "applets," photographs, animations, video, audio, music and text incorporated into the Software) as well as any accompanying written materials (the "Documentation") is owned by Ektron or its suppliers, is protected by United States copyright laws and international treaties, and contains confidential information and trade secrets. CUSTOMER agrees to protect the confidentiality of the Software and Documentation. CUSTOMER agrees that it will not provide a copy of this Software or Documentation nor divulge any proprietary information of Ektron to any person, other than its employees, without the prior consent of Ektron; CUSTOMER shall use its best efforts to see that any user of the Software licensed hereunder complies with this license.

4. Limited Warranty: Ektron warrants solely that the medium upon which the Software is delivered will be free from defects in material and workmanship under normal, proper and intended usage for a period of three (3) months from the date of receipt. Ektron does not warrant the use of the Software will be uninterrupted or error free, nor that program errors will be corrected. This limited warranty shall not apply to any error or failure resulting from (i) machine error, (ii) Customer's failure to follow operating instructions, (iii) negligence or accident, or (iv) modifications to the Software by any person or entity other than Company. In the event of a breach of warranty, Customer's sole and exclusive remedy, is repair of all or any portion of the Software. If such remedy fails of its essential purpose, Customer's sole remedy and Ektron's maximum liability shall be a refund of the paid purchase price for the defective Products only. This limited warranty is only valid if Ektron receives written notice of breach of warranty within thirty days after the warranty period expires.

5. Limitation of Warranties and Liability: THE SOFTWARE AND DOCUMENTATION ARE SOLD "AS IS" AND WITHOUT ANY WARRANTIES AS TO THE PERFORMANCE, MERCHANTIBILITY, DESIGN, OR OPERATION OF THE SOFTWARE. NO WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE IS OFFERED. EXCEPT AS DESCRIBED IN SECTION 4, ALL WARRANTIES EXPRESS AND IMPLIED ARE HEREBY DISCLAIMED.

NEITHER COMPANY NOR ITS SUPPLIERS SHALL BE LIABLE FOR ANY LOSS OF PROFITS, LOSS OF BUSINESS OR GOODWILL, LOSS OF DATA OR USE OF DATA, INTERRUPTION OF BUSINESS NOR FOR ANY OTHER INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES OF ANY KIND UNDER OR ARISING OUT OF, OR IN ANY RELATED TO THIS AGREEMENT, HOWEVER, CAUSED, WHETHER FOR BREACH OF WARRANTY, BREACH OR REPUDIATION OF CONTRACT, TORT, NEGLIGENCE, OR OTHERWISE, EVEN IF COMPANY OR ITS REPRESENTATIVES HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH LOSS.

6. Additional Terms and Conditions apply

When using the CMS400 map control, Subject to the terms and conditions of the Map provider (Microsoft Bing Maps for Enterprise or Google maps)

Microsoft Bing Maps for Enterprise - http://www.microsoft.com/virtualearth/product/terms.html

If you have any questions would like to find out more about a MWS/VE Agreement, please contact maplic@microsoft.com for information.

Google Maps - http://code.google.com/apis/maps/terms.html

7. Miscellaneous: This License Agreement, the License granted hereunder, and the Software may not be assigned or in any way transferred without the prior written consent of Ektron. This Agreement and its performance and all claims arising from the relationship between the parties contemplated herein shall be governed by, construed and enforced in accordance with the laws of the State of New Hampshire without regard to conflict of laws principles thereof. The parties agree that any action brought in connection with this Agreement shall be maintained only in a court of competent subject matter jurisdiction located in the State of New Hampshire or in any court to which appeal therefrom may be taken. The parties hereby consent to the exclusive personal jurisdiction of such courts in the State of New Hampshire for all such purposes. The United Nations Convention on Contracts for the International Sale of Goods is specifically excluded from governing this License. If any provision of this License is to be held unenforceable, such holding will not affect the validity of the other provisions hereof. Failure of a party to enforce any provision of this Agreement shall not constitute or be construed as a waiver of such provision or of the right to enforce such provision. If you fail to comply with any term of this License, YOUR LICENSE IS AUTOMATICALLY TERMINATED. This License represents the entire understanding between the parties with respect to its subject matter.

YOU ACKNOWLEDGE THAT YOU HAVE READ THIS AGREEMENT, THAT YOU UNDERSTAND THIS AGREEMENT, AND UNDERSTAND THAT BY CONTINUING THE INSTALLATION OF THE SOFTWARE, BY LOADING OR RUNNING THE SOFTWARE, OR BY PLACING OR COPYING THE SOFTWARE ONTO YOUR COMPUTER HARD DRIVE, YOU AGREE TO BE BOUND BY THIS AGREEMENT'S TERMS AND CONDITIONS. YOU FURTHER AGREE THAT, EXCEPT FOR WRITTEN SEPARATE AGREEMENTS BETWEEN EKTRON AND YOU, THIS AGREEMENT IS A COMPLETE AND EXCLUSIVE STATEMENT OF THE RIGHTS AND LIABILITIES OF THE PARTIES.

Version 7.6.6, September 28, 2009.

Rev 2.2

1 Ektron eCommerce

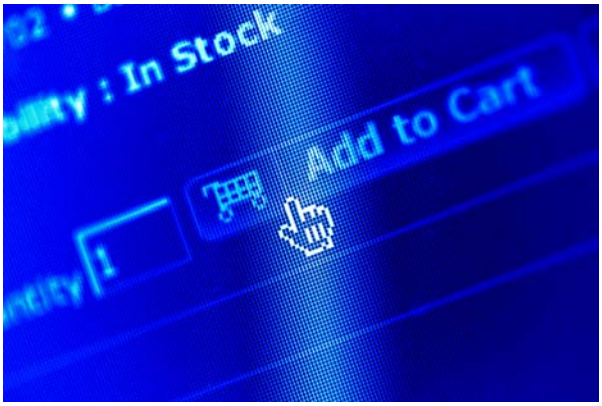
An overview

Ektron's eCommerce solution is a flexible and feature-rich platform that offers all of the functionality that shoppers have come to expect in an online marketplace. For Administrators, eCommerce allows them to easily manage products, review inventory and use Ektron's powerful taxonomy to enhance product SEO. For developers, eCommerce allows them to use the flexible and scalable API to import an existing store, configure a customized integration with any payment gateway and integrate with financial software.

With Ektron CMS400.NET, you have a single application running both your Web site and online marketplace, letting you manage both from the same interface. Total integration means total control over your site through the author-friendly Workarea which enables you to set up your eCommerce content (catalog information). Because eCommerce reuses many standard Ektron CMS400.NET components, you can leverage existing "know-how" to quickly build the store and use server controls to create the actual Web pages that site visitors use to purchase goods and services on the site.

eCommerce is the perfect complement to your Web site, going beyond consumer goods:. CMS400.NET enables you to sell memberships and/or access to premium content. Also, state and local governments can process online vehicle registration and tax payments, and event sites can sell tickets.

eCommerce gives you all of the tools and capabilities you expect from an online sales application, with the added bonus of running side-by-side with CMS400.NET. With your online commerce being powered by the same content management system running your overall Web site, you will have complete control of your online marketing and sales strategy at your fingertips.



eCommerce Features

Ektron CMS400.NET includes all of the features and tools you need to build out your online marketplace. This includes:

- Simple to complex product support. You can package several products together into a single purchase, or let site visitors choose extra cost options.
- Subscription-based products, which means a customer pays for access to an area of your Web site, such as confidential content or a download page that provides the latest version of software.
- Cross-sell/up-sell recommendations.
- Flexibility in setting up credit card and payment gateways.
- Inventory control, with the ability to interface with a third-party inventory system.
- Flexible pricing functionality, including support for any currency and volume discounts.
- Coupon generation that allows for flat rate, percentage, and free shipping options.
- Link to shipping providers out-of-the-box (UPS or FedEx) and access real-time rates from their servers.

- Extensive server controls that manage the shopping cart and checkout process.
- A powerful, scalable API that enables developers to customize the store in any way.
- Existing product data can be imported into eCommerce and the CMS using the API.

Catalog Management and Attributes

eCommerce supports all product types, from single items, to complex products (for example, movies that are available in DVD, Blu-ray or other formats), bundles and kits.

eCommerce takes advantage of key CMS400.NET technologies, including Smart Forms and market-leading taxonomy and search. Product entry Smart Forms standardize the way that products are entered — all of the information, attributes, and descriptive text is entered in one place and goes into the database as structured XML data. This makes it easy to repurpose this content wherever and whenever you need it.

Enhance product SEO in the Smart Form by setting attributes, adding metadata and including photos and other media. Adding images of products auto-generates thumbnails and gives you complete control over sizes and number of images, as well as a clickthrough image gallery. Images can also be defined for the product type through Ektron's taxonomy.

Taxonomy

Control over products extends to cross-sell and up-sell features and Ektron's taxonomy enables products to be classified in multiple ways, providing even more options for you to direct your customers to the product they are looking for.

Item Review

CMS400.NET's Web 2.0 content review tools find new uses here, letting your customers post their opinions and five-star ratings on products they have purchased. Searching for product uses Ektron's unparalleled taxonomy, and those searches can be filtered by price, ratings, reviews and more.

Coupons, Taxes and Pricing

eCommerce supports a complete array of pricing schemes: individual pricing, separate pricing within complex products, tiered pricing to support volume discounts and more. Online coupon capability is also included, letting you set the type (dollar amount or percentage) and at what level those coupons are applied at (product level, basket level and type of item).

Localization

For global business opportunities, all international currencies are available. You are able to use conversion rates to determine prices or set flat costs for each product. This goes hand-in-hand with the tax options available. Set tax classes, including country, state, and local (by postal code) as well as taxes for specific classes of goods, and be able to sell to even more customers.

Personalization

There is no login necessary for purchases (but you can make it a requirement), and when a customer is a member of your eCommerce site, they can name and save shopping baskets for future purchases or as wishlists and registries. Your site administrators can always see what baskets are still open.

Checkout, Payment and Shipping

Checking out is a multi-step process. You can display all of the steps on a single page or distribute them through multiple pages. While they are checking out, users can save as many shipping addresses as they want, or simply check “same as billing”.

Shipping uses real-time rates by plugging into external providers or you can use your own rates by setting up an internal shipping provider. Rates allow for a great deal of customization. They can be set up by package size, product weight and even product price ranges.

Payment options are flexible with eCommerce. You can select what credit cards to accept and what gateways to use to authorize and capture transactions (PayFlow and Authorize.Net are supported out-of-the-box). Or, override this to create your own payment gateway (one that allows terms or any other payment option).

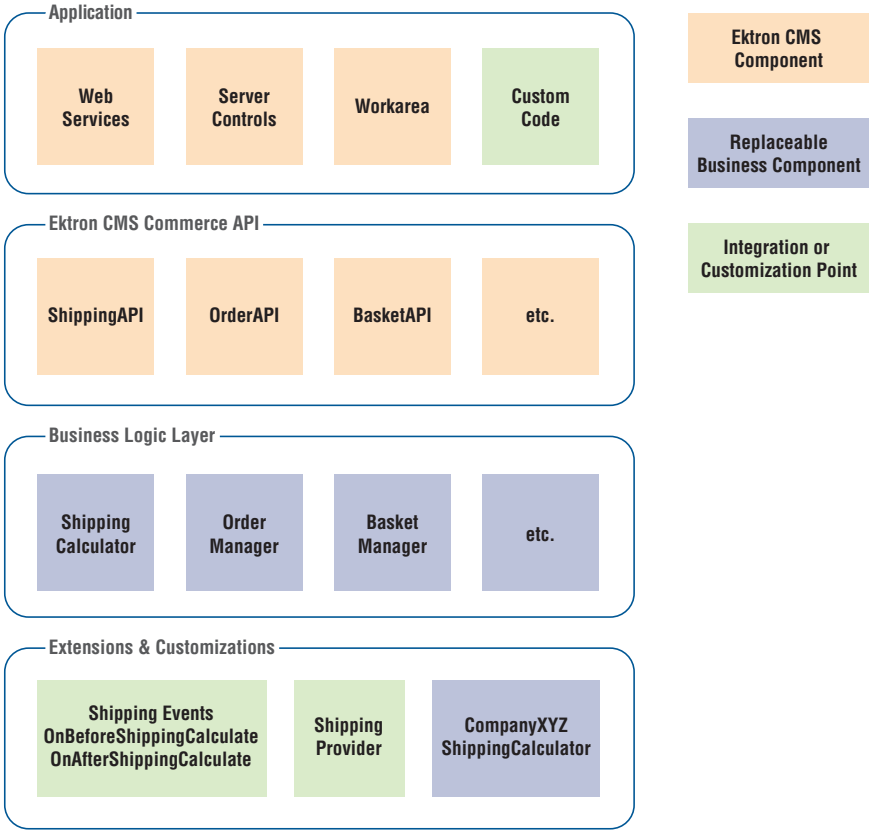
When the order is reviewed and submitted (and payment is authorized), it can go through a simple or complex routing process. An order can process through your own processing setup or plug-in to your existing CRM or ERP. When the order is processed, the CRM/ERP notifies CMS400.NET that it is complete. Online inventory is kept up to date, and you can set properties so that items that drop below a certain level trigger notifications that are sent to your inventory control system.

Architecture

The CMS400.NET eCommerce architecture is composed of several key systems:

- Catalog/Inventory systems
- Order system
- Shipping system

The eCommerce API sits within the core CMS system with customization and integration points. The figure below shows how the eCommerce API fits into the overall CMS architecture.

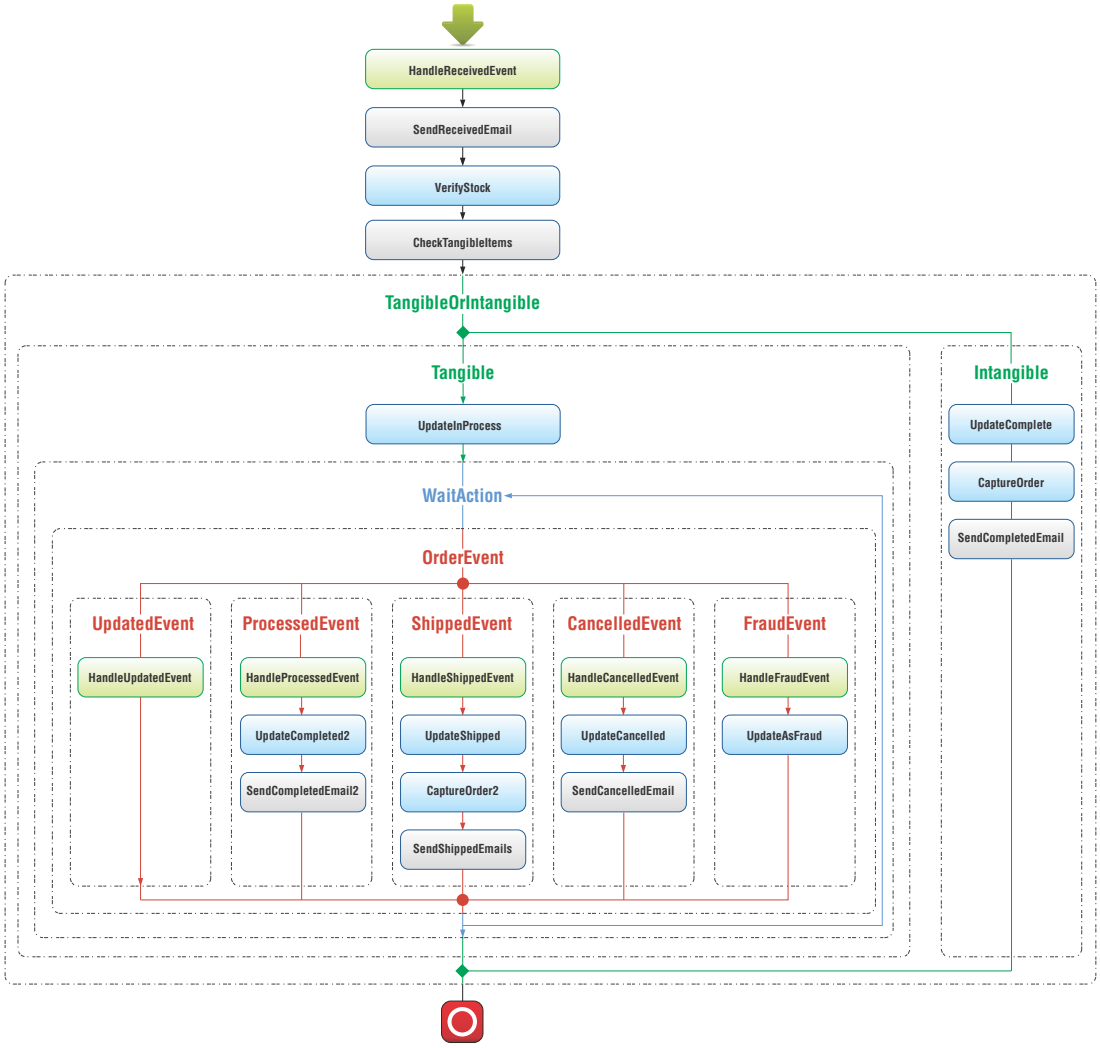


Order Workflow

CMS400.NET leverages Windows Workflow Foundation for order processing.

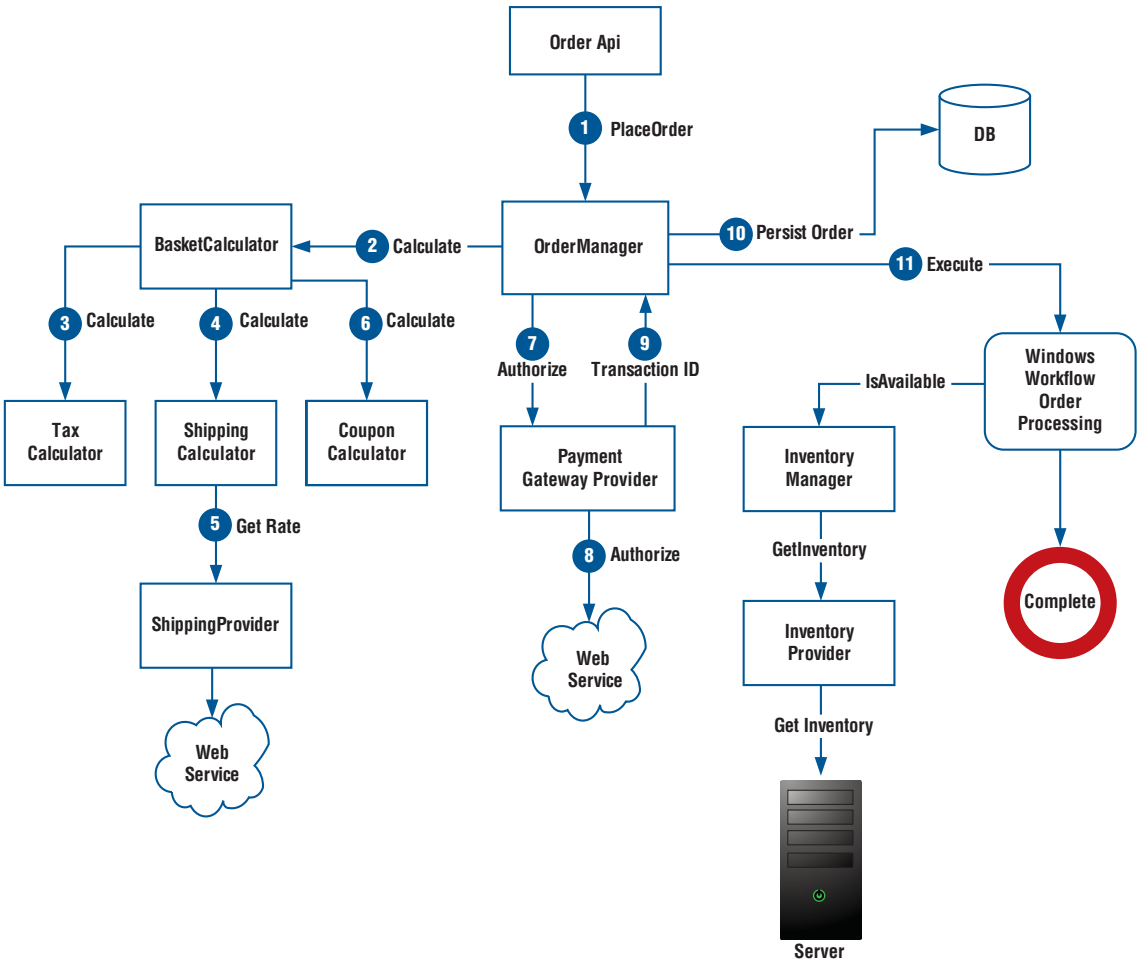
Windows Workflow Foundation is a framework that enables users to create system or human workflows. It consists of a namespace, an in-process workflow engine, and designers for Visual Studio 2005/2008. It comes with a programming model, a re-hostable and customizable workflow engine, and tools for quickly building workflow-enabled applications on Windows.

Windows Workflow Foundation gives site developers complete control over how your store “works”. The workflow is completely extensible and can integrate with third-party ERP/CRM systems. There are default Ektron activities such as email notification and inventory checks, and the workflow can be customized to match the organizations business processes.


HandleReceivedEvent
SendReceivedEmail
VerifyStock
CheckTangibleItems
TangibleOrIntangible
Tangible
UpdateInProcess
WaitAction
OrderEvent
UpdatedEvent
HandleUpdatedEvent
ProcessedEvent
HandleProcessedEvent
UpdateCompleted2
SendCompletedEmail2
ShippedEvent
HandleShippedEvent
UpdateShipped
CaptureOrder2
SendShippedEmails
CancelledEvent
HandleCancelledEvent
UpdateCancelled
SendCancelledEmail
FraudEvent
HandleFraudEvent
UpdateAsFraud
Intangible
UpdateComplete
CaptureOrder
SendCompletedEmail

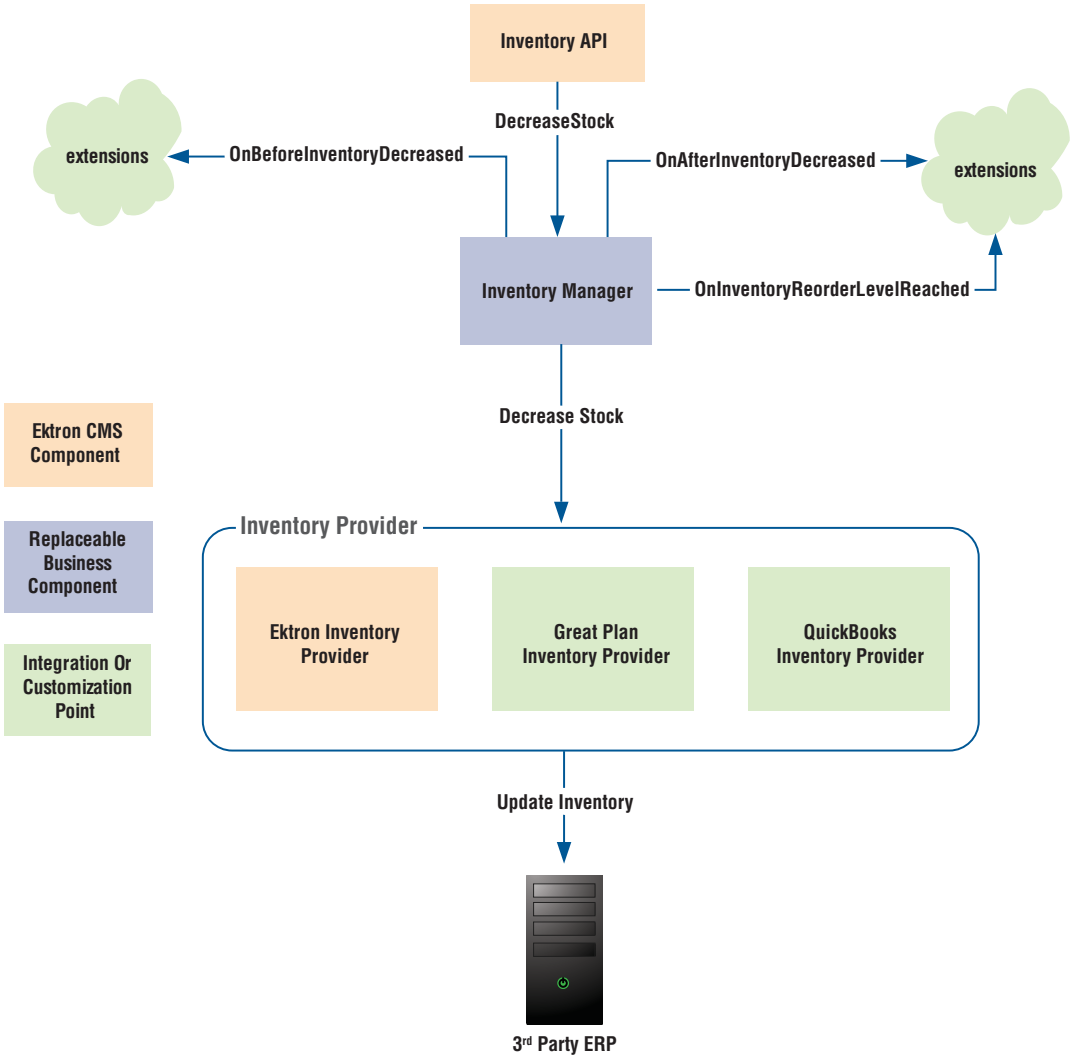
Order Fulfillment

Order processing in Ektron includes the management of custom coupons, shipping providers, tax calculation and payment providers.



Inventory Process

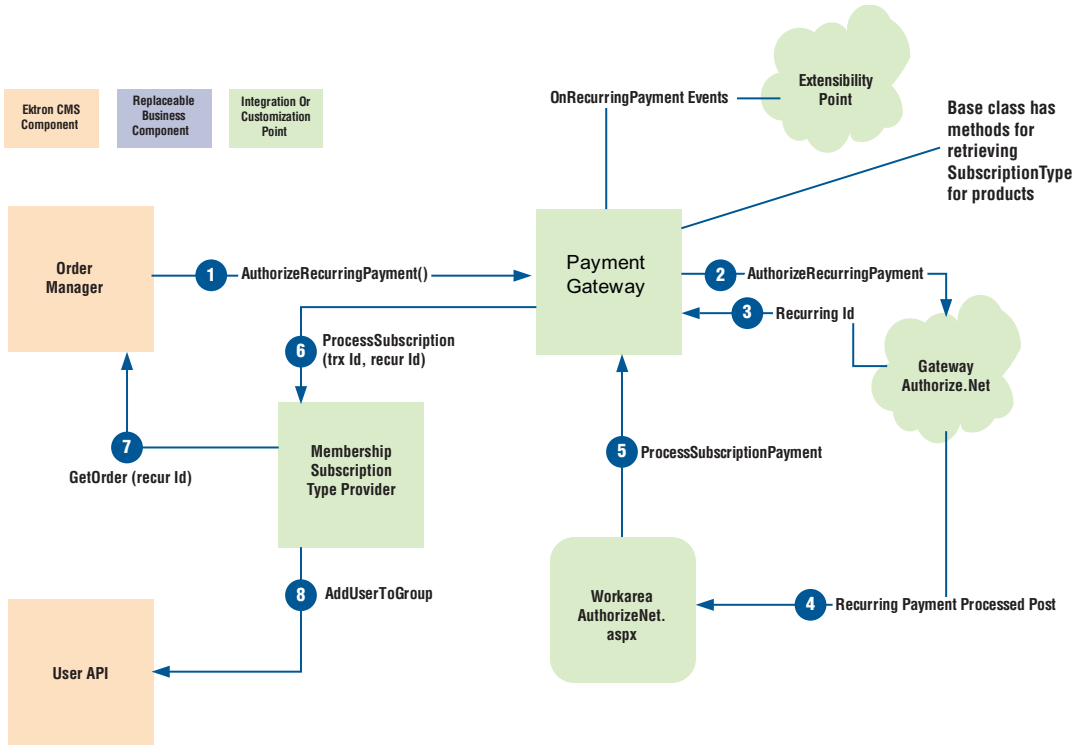
The inventory process uses an inventory provider model. You can use the inventory system included as part of Ektron's eCommerce module, or plug into external systems. It is called from the order workflow and is completely extensible.



Subscription Model

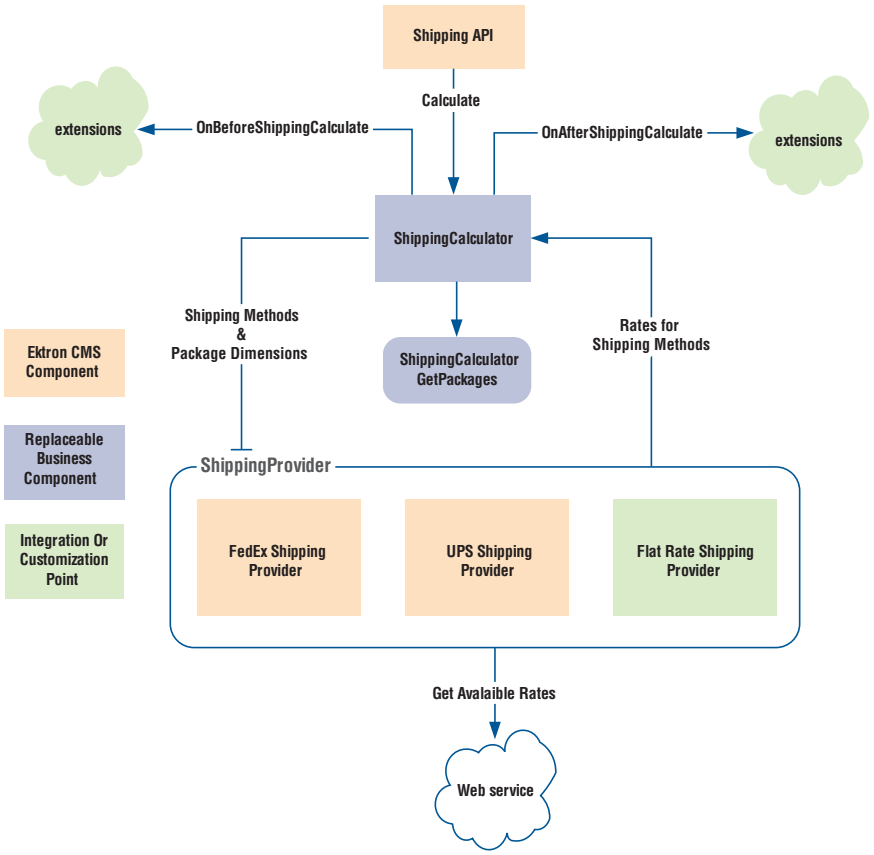
Ektron's eCommerce solution allows you to not only sell durable goods, but also memberships and subscriptions. It uses a provider model, and the default implementation provisions users into groups.

You allow shoppers to buy membership directly on your site (access to assets or content) and configure recurring billing at any interval.


Ektron CMS Component
Replaceable Business Component
Integration Or Customization Point
OnRecurringPayment Events
Extensibility Point
Base class has methods for retrieving SubscriptionType for products
Order Manager
1 AuthorizeRecurringPayment()
Payment Gateway
2 AuthorizeRecurringPayment
3 Recurring Id
Gateway Authorize.Net
6 ProcessSubscription (trx Id, recur Id)
7 GetOrder (recur Id)
Membership Subscription Type Provider
5 ProcessSubscriptionPayment
8 AddUserToGroup
Workarea AuthorizeNet. aspx
4 Recurring Payment Processed Post
User API

Shipping Process

The shipping system calculates real-time shipping rates from FedEx or UPS by default. However, its provider model lets you configure the API to access rates from any provider. It is completely customizable and extensible.



Use Case: EktronStore.com

In this *QuickStart* guide, we will be using EktronStore.com to demonstrate how to set up, build, and deploy an online marketplace using CMS4000.NET. EktronStore.com will sell furniture and accessories and leverage many eCommerce features in CMS4000.NET.

2 Configuring your Store

Building EktronStore.com

This section describes how to get started with the task of setting up your store.

Web.Config

Ektron CMS400.NET's web.config file lets you control many key functions of the eCommerce component of the content management system. When you install Ektron CMS400.NET, web.config is placed into webroot/siteroot.

If your server is currently running another .NET application, you must merge that web.config file with this one. To distinguish Ektron CMS400.NET's tags from others, all Ektron CMS400.NET settings begin with `ek_` and reside within the `<appSettings>` tags of the web.config file.

Security Compliance

To view information about Ektron' eCommerce security standards and implementation, see the Ektron CMS400.NET eCommerce Module Implementation Guide for PCI DSS Compliance.

This guide is required as part of the Payment Application Data Security Standard (PA DSS) certification as defined by the Payment Card Industry Security Standards Council (PCI SSC). The Security Guide is to be used by both Ektron CMS400.NET's version 7.6.5 eCommerce partners and customers to help them implement a secure Web site according to the Payment Card Industry Data Security Standard (PCI DSS).

What is PA DSS?

Note: For more information, see the Ektron CMS400.NET eCommerce Module Implementation Guide for PCI DSS Compliance.

PA DSS is a certification for software applications that store, process or transmit credit card data during a transaction. Most payment card brands encourage merchants to use payment applications that are certified PA DSS Compliant.

Due to Ektron's leadership position in Content Management and its commitment to security, CMS400.NET will be PA DSS certified to ensure our application conforms to payment card industry standards.

It is Ektron's responsibility to become PA DSS certified. In other words, make sure that CMS400.NET is designed in such a way as to meets the standard for applications as set by the PCI Security Standards Council.

As a merchant or eCommerce Web site owner, it will be your responsibility to make sure your Web site is PCI DSS Certified. You will need to work together with your hosting provider to obtain this certification. This means using PCI DSS compliant server architecture, performing proper hardware and port scans, and using the proper software and hardware configurations to meet these standards.

Configuration Tasks

To get started with eCommerce, do the following:

Determine Units of Measure

For each item sold on your Web site, you define a height, length, depth, and weight. If English, the units are inches and pounds. If Metric, the units are centimeters and kilograms. This information is used to calculate shipping costs.

By default, this is set to English. To change to Metric, open the `siteroot/web.config` file, and edit the value of the `ek_measurementsystem` property.

```
<ektroncommerce>
ek_measurementsystem value = “english” or “metric”
```

After you make this decision and build your eCommerce system, you cannot change the units of measure.

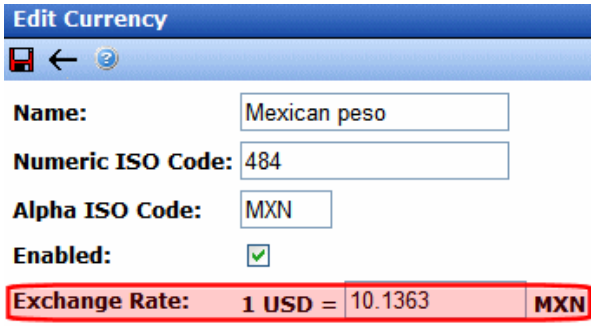
Determine the Default Currency

Decide on a default currency, the one on which all exchange rates are based.

After your eCommerce feature goes live, do not change the `ek_ecom_DefaultCurrencyId` value in web.config.

The Default Currency

The default currency is the reference currency when setting the exchange rate on the Edit Currency screen. For example, if the default is US dollars, USD appears on the left of the exchange rate equation, as shown below.



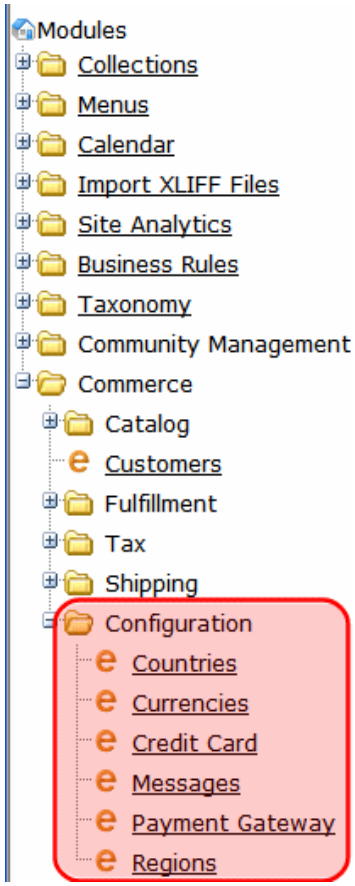
The default currency is also used as the default choice in the Currency Selector server control, and on the Pricing tab of the View/Edit Catalog Entry screen.

Viewing and Changing the Default Currency

When you install Ektron CMS400.NET, the default currency is the Numeric ISO code for U.S. dollars (Id 840). At that time, you can change this in the web.config file's `ek_ecom_DefaultCurrencyId tag`. However, once you set your default currency in web.config and begin to deploy eCommerce, do not change it.

Set Up eCommerce Configuration Screens

The configuration screens collect critical information that is used throughout the eCommerce feature.



Steps to setup the eCommerce configuration.

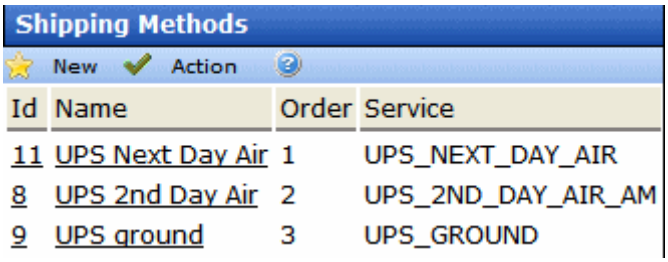
Step	Screen(s)	General Information
1	Countries (enable ones you will support)	This screen allows your to define country tax tables. For example, assume Canada has a national tax of 5% that applies to the sale of certain goods. National tax tables are defined in the Country Tax Tables screen. This screen also determines shipping costs. In the Order checkout process, the purchaser selects a shipping address, which includes country. So, be sure to enable all countries from which site visitors can enter orders. Countries are also part of the location on the Warehouse screen. An order's shipping cost is determined by the distance between the warehouse and the destination.
2	Regions (dependent on Countries)	The Regions screen lets you define geographic regions within countries that your eCommerce site services. For example, in the United States of America, each state is a region.

Step	Screen(s)	General Information
3	Tax classes	Use the Tax Class screen to define all classes of catalog entries that can be taxed. When you install eCommerce, the following tax classes are installed.
4	Postal Code Tax Regional Tax Country Tax	Depends on: • Countries, Regions, Tax Class • Regions, Tax Class • Countries, Tax Class eCommerce provides a set of tax screens in the CMS Workarea that lets you set up tax classes (for example, goods, services, tobacco), define tax rates for these geographic entities (from most to least specific) • postal code • region • country
5	Warehouse	Depends on: • Countries and Regions
6	Credit Card Types, Payment Gateways	The credit card types screen allows you to enter all types of credit cards that your eCommerce system can use. By default, CMS400.NET's developer sample site provides the following card types. • American Express • Diners Club (not accepted by default) • Discover • Master Card • Visa See "Select your Payment Gateway" on page 16 to select gateway. See "Customizing the Payment Gateway" on page 84 for more information about configuring a custom gateway.
7	Currencies (enable ones you will support)	eCommerce supports all currencies. However, site visitors can only choose enabled currencies. The Currency screen lets you enable and edit information for all installed currencies. By default, Ektron CMS400.NET provides all currencies on the ISO 4217 currency names and code elements list. The default currency is the reference currency when setting the exchange rate on the Edit Currency screen. For example, if the default is US dollars, USD appears on the left of the exchange rate equation, as shown below.
8	Shipping Methods and Packages	The Packages screen lets you define standard sizes of packaging. Use it to define every package size that your shipping department uses to ship your products. See "Select a Shipping Provider" on page 15 to select a shipping provider. See "Customizing Shipping" on page 76 for more information about configuring a custom shipping provider.

Step	Screen(s)	General Information
9	Messages	Ektron's eCommerce feature provides a messaging feature that can notify a purchaser when these events occur: • Order is submitted • Order is cancelled • Order is shipped Each message has a unique type that corresponds to an event. The message types are • OrderReceived • OrderCancelled • Order Shipped You can create any number of messages for each event, assigning one as the default. Only the default message is sent when the corresponding event occurs.

Select a Shipping Provider

Use the Shipping Methods to define all choices your site visitors will have for delivering their purchases.



Prerequisites:

- Default warehouse with an address
- A default shipping provider is defined in the `siteroot\shipment.config` file

When you install the CMS, several providers exist in the `siteroot\shipment.config` file:

- FedExShipmentProvider
- FlatRateShipmentProvider
- UPSShipmentProvider

However, the FedEx and UPS information is for testing purposes only. In order to use FedEx, UPS or another shipping provider, obtain the following information from them.

- service URL
- key
- password
- account number
- meter number
- transaction Id

The above list can vary slightly for each provider.

Then, enter that information into `shipment.config`, following the format of providers already in the file.

The `shipment.config` file also contains `name` and `type` properties for each provider (shown in red below).

```
name="FedExShipmentProvider" type="Ektron.Cms.Commerce.Shipment.Services.FedExShipmentProvider, Ektron.Cms.Commerce"
```

The red text that you insert (above) is important. Although the actual name is arbitrary, the two strings must match and be identical to what is in the default provider (if you are using that particular one as the default provider. See "Customizing Shipping" on page 76 for more information).

Select your Payment Gateway

A Payment Gateway Provider is a pluggable component that is integrated into Ektron's CMS400.NET eCommerce module. A Payment provider handles eCommerce customer payments by using third-party payment gateways. CMS400.NET's eCommerce module accepts payments such as credit cards. Then, it passes that information to a third-party service. The third-party service processes the payment and returns a transaction ID that is stored with the customer's order.

Your company needs to setup an account with a third party payment service before using the payment provider. This includes payment providers such as Authorize.NET and PayFlow which are included with CMS400.NET's eCommerce Module.

CMS400.NET comes with several payment providers, including Authorize.NET and PayFlow. You can customize these providers or create your own using the extendable Payment Gateway Provider architecture. See "Customizing the Payment Gateway" on page 84 for more information.

Each type of payment gateway provider will accept configuration parameters. For example, Authorize.NET requires a username and password while PayFlow requires a Username, password, vendor, and partner.

In addition, some payment gateways might support recurring payments, while others might not. Recurring payments provide the ability to create a payment that recurs at a given interval for a specified period of time. For example, you could create a payment for $9.99 that occurs on the first of every month for the next 12 months. This is something to consider if your site relies on a subscription service.

The following steps describe the flow of payment information for a customer purchasing a product from your site.

Typical Payment Flow

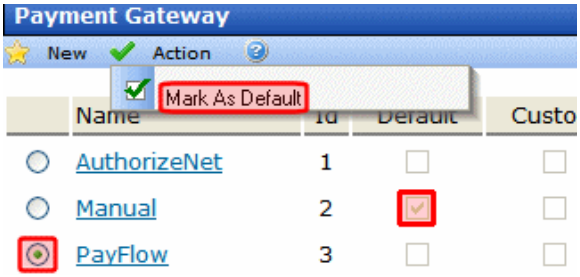
1. A customer purchases a product from your site and submits their payment information.
2. Ektron CMS400.NET's passes the information to your payment gateway.
3. Your payment gateway provider passes the information to your bank's processor.
4. The bank's processor submits the information to a credit card interchange for processing, clearing and settlement.
5. The interchange notifies the customer's credit card company of the transaction's details.
6. The credit card company accepts or declines the transaction based on the customers account information.
7. If the transaction is approved, the funds are transfer to the interchange.
8. The credit card interchange sends information about whether the transaction is approved or declined to your bank's processor.
9. This information is passed to your payment gateway provider.
10. The provider notifies your site of the information. The results of the transaction are displayed on the page the customer is viewing.
11. The credit card interchange sends funds to your merchant account.

The Default Gateway

Only the default gateway is used by the checkout process. Additional gateways are unused until they are made the default. The default gateway cannot be deleted. If you want to delete it, you must first make another gateway the default.

To change the default gateway, follow these steps.

1. Go to Modules > Commerce > Configuration > Payment Gateway.
2. The currently-defined default has a check in the Default column.
3. Click inside the radio button to the left of the gateway you want to make the default.



4. Click Action > Mark as Default.
5. A confirmation message appears. Click OK.

Add a Payment Gateway

To add a new payment gateway, follow these steps.

1. In the CMS, go to Modules > Commerce > Configuration > Payment Gateway.
2. Click New > Payment Gateway.
3. Use the following table to complete the screen.

Field	Description
Name	Enter the name of the gateway.
Default	Check this box if this is the default gateway. If you do, and another gateway is currently the default, it is replaced by this one.
User ID	Enter your User ID. This ID will identify your account with this gateway provider.
Password	Enter the password for your account with this gateway provider.
Expand Custom Values	If this gateway provider needs additional fields of information, enter those values into the Custom 1 and Custom 2 fields.

Select an Inventory Provider

If you are not using CMS400.NET's inventory system, create a custom inventory provider and add it to the web.config file's InventoryProvider area. Ektron's Inventory Provider feature allows you to work with an existing inventory system. This is set in the web.config file.

Add the custom provider between the InventoryProvider area's `<providers>` tags and change the default provider to the name of your custom inventory provider. For example:

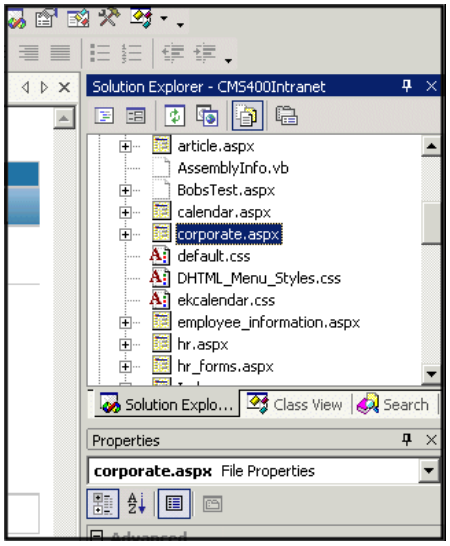
```
<inventoryProvider defaultProvider="MyCustomInventoryProvider">
```

See "Customizing Inventory" on page 82 for more information.

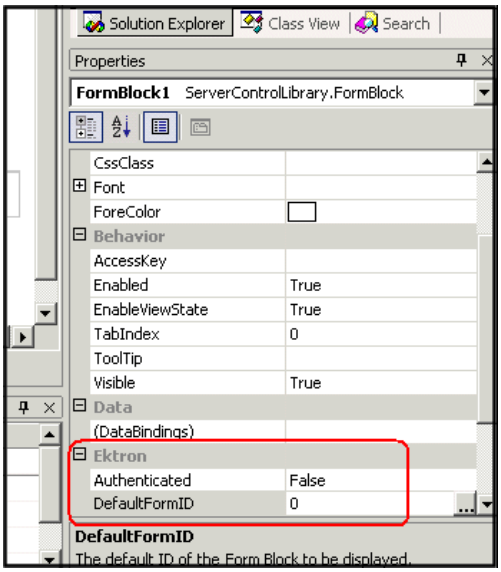
Opening the eCommerce Project

To help get you started, these directions explain how to open the sample site delivered with Ektron CMS400.NET.

1. Browse to and double click Ektron CMS400.NET's solution file, `localhost/siteroot/CMS400Developer.sln`.
2. The sample site project opens.
3. To work on a template page, click it from the Solution Explorer.



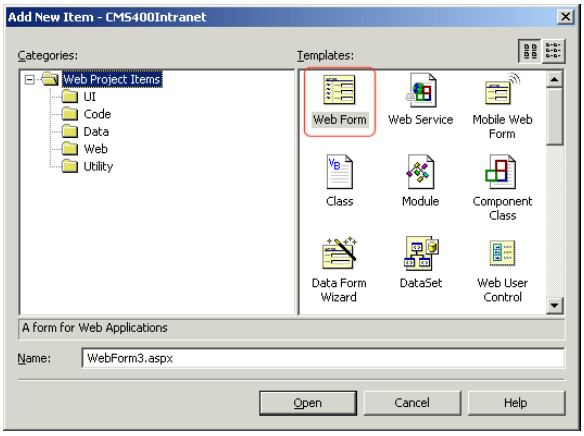
When you select a page, its properties appear in the Properties area (below Solution Explorer), and the page appears in the center of the screen. A control's properties include several standard .NET properties along with Ektron-specific ones. The Ektron properties are labeled as illustrated below.



Setting Up a Template

To create a new template (.aspx) page, follow these steps.

1. Click Project > Add Web Form.
2. On the Add New Item screen, click Web Form and assign a name.

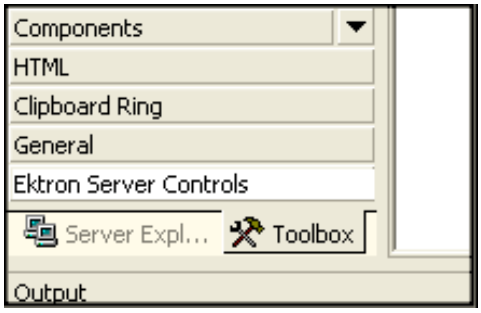


3. Add controls to determine the page content.

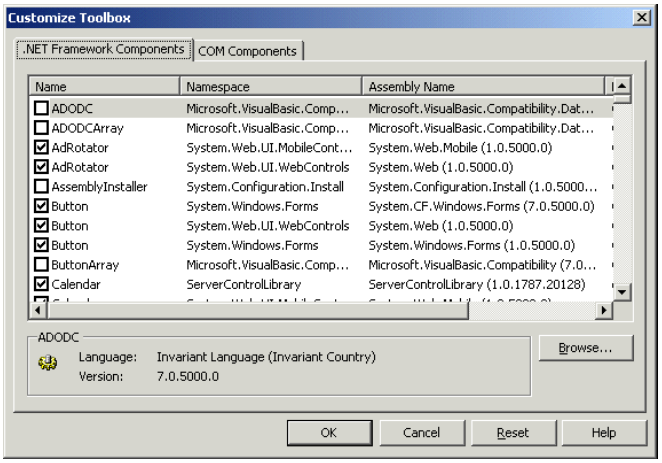
Making the Server Controls Available

You must copy the DLLs to a local drive before installing them. You cannot copy them from a network drive.

1. Display the Visual Studio 2005 toolbox (View > Toolbox).
2. Right click the mouse within the Toolbox.
3. Click Add Tab.
4. Type Ektron Server Controls then press enter.



5. Click the Ektron Server Controls tab.
6. Right click the mouse in the empty area.
7. Click Add/Remove Items. If using Visual Studio 2005, click Choose Items.
8. The Customize Toolbox dialog appears.



9. Select the .NET Framework Components tab.
10. Browse to the directory that stores Ektron CMS400.NET's DLL files, `localhost/CMS400Developer/bin`, and add the `Ektron.Cms.Controls.dll` file. This file provides access to Ektron CMS400.NET's server controls.

    Alternatively, you could use the following location, `C:\Program Files\Ektron\CMS400v7x\bin.` The file is identical in both places.

    Using the bin folder in your site provides better speed when loading Web pages. However, if you use the bin folder located in Program Files, you do not have to worry about deleting the .DLL file if you change or delete your site.
11. Press OK.

For easier viewing once the server controls are installed, you can right click on them and select Sort Items Alphabetically. Note that you can only see the server controls when an aspx template is selected.

Create a Master Page

Before you begin creating eCommerce pages, you should have already created a Wireframe diagram that outlines the structure of the page. This wireframe document can be produced by you or a any number of departments and stakeholders. Based on the wireframe, a visual mockup is created using PhotoShop or other similar application. Then, you can use callouts on the "picture" of the site to finalize the look and structure of all the pages.

A "Master Page" in Ektron eCommerce used to define and control areas and functionality of the store that are shared by most or all pages. For example, the store's navigation system, header, analytics (server control), currency selector (server control) are all candidates for inclusion in Masterpage.aspx.

For complete information about creating store pages, see "Planning and Site Design" on page 23.

Using eSync with eCommerce

When using eSync with an eCommerce site, you need to prevent orders from being processed on your staging server. This is to prevent orders from being processed twice.

For example, a customer on your production site is about to purchase a product. If the staging server and production server are synchronized before the credit card is processed, the potential exists for the customer's credit card to be charged twice — once from the production server and once from the staging server.

To prevent orders from being processed on your staging server, go to your staging server and edit your site's web.config file. Next, set the following property to `true`.

```
<add key="ek_ecom_OrderProcessingDisabled" value="true" />
```

When the property is set to `true`, you cannot process or edit orders from the Workarea's View Order page on the staging server. Also, if you try to create an order from the staging server, you receive the following message.

"We're sorry, an error occurred while processing your request. Please try again later..."

Note that all other eSync functions will still operate properly. For example, when you create catalog entries on a staging server and perform a synchronization, the entries will be moved to the production server. The key shown above only effects the actual processing of orders.

3 Planning and Site Design

Building EktronStore.com

In this *QuickStart* guide, we will be using EktronStore.com to demonstrate how to set up, build, and deploy an online marketplace using CMS4000.NET. EktronStore.com will sell furniture and accessories and leverage many eCommerce features in CMS4000.NET.

Before starting the process of building EktronStore.com, you should first consider the goals of your online marketplace. Do you want to simply sell as many widgets as possible? Or, do you see your store as a value-added component designed to enhance the site visitor experience?

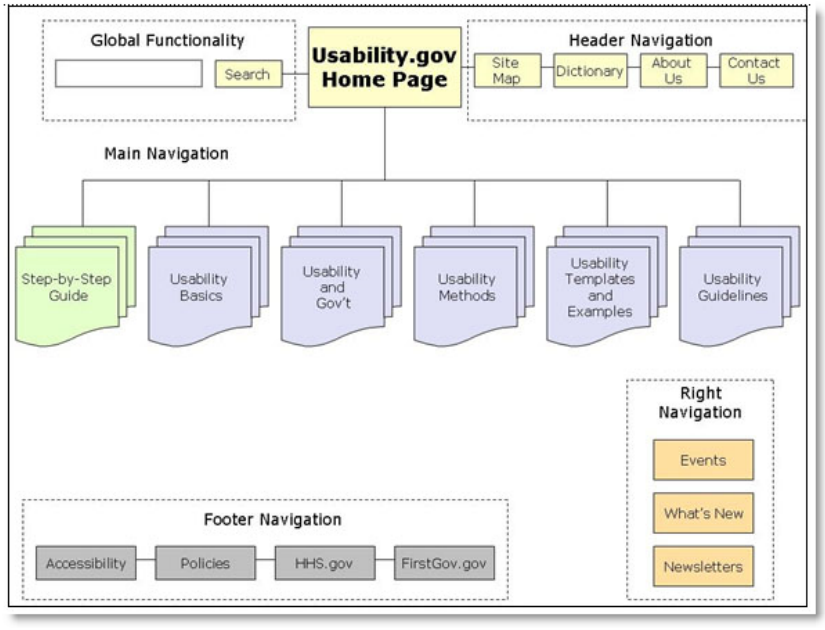
When considering store design and functionality, what you intend to sell plays a significant role in this endeavor. For example, are you selling durable goods? Subscriptions? Access to premium content? All of the above? Make sure you consider this when planning the layout of the store.

Shipping options, payment methods, taxation, and the general look-and-feel of your store also need to be determined. Make sure you address the general (work) flow of your store — that is, how shoppers are guided through the entire shopping experience.

Sitemap

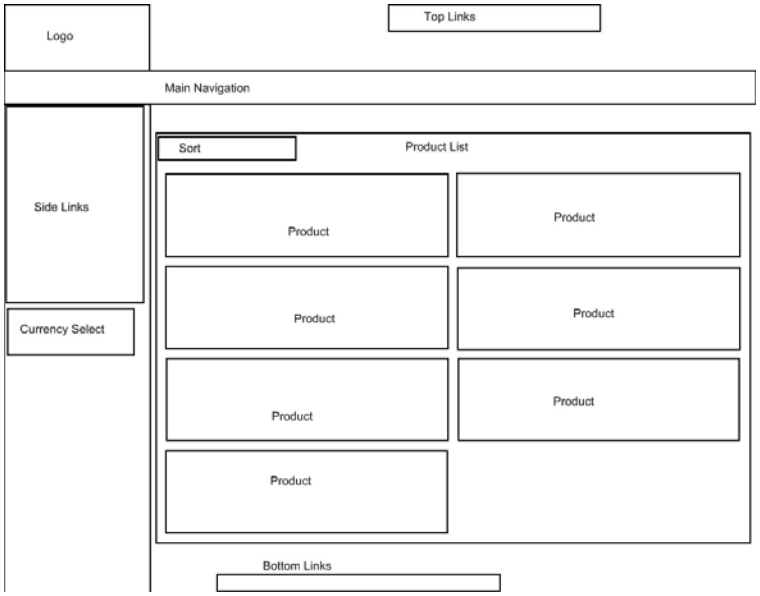
It all starts by planning the basic architecture of the store. This is called a “sitemap” and it is a depiction of the architecture of the web site. It can be either a document (in any form) used as a planning tool for web design, or a web page that lists the pages on a web site, typically organized in hierarchical fashion.

Essentially, the sitemap is the layout of the main site structure. It identifies major pages, sections, sub-sections and shared elements of the site.



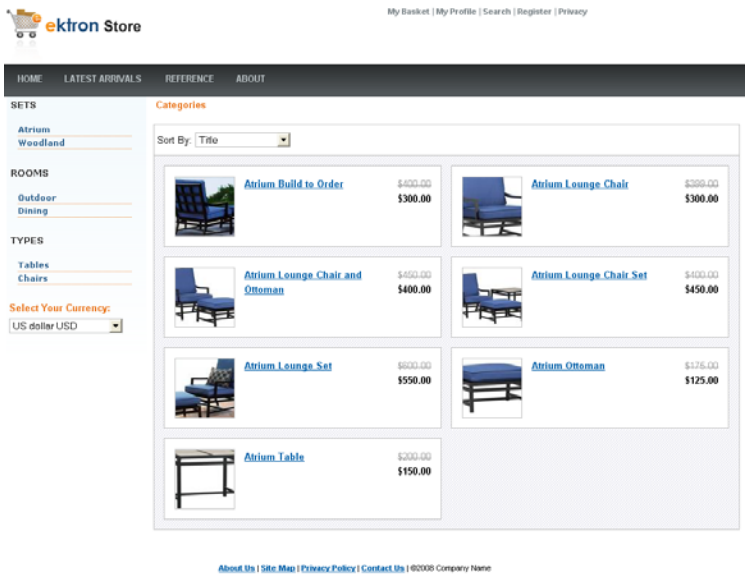
Wireframes

Next, you must plan and scope out all of the major pages in your store. Using pencil and paper or advanced software applications, you sketch the basic architecture and components of individual store pages. This visual guide is referred to as a “wireframe”.



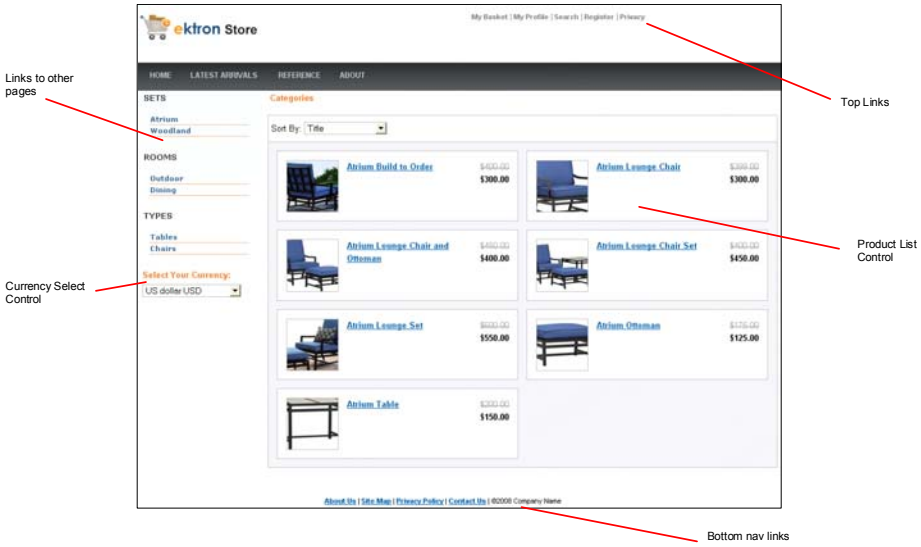
Mockups

The wireframe helps inform visual design. Typically, a graphic artist will create mockups (or visual preview) of store pages in PhotoShop (or a similar application) so that all stakeholders can see how the store will look without actually building Web pages.



Callouts

Callouts help identify areas of functionality on store pages. Using the mockups, you can identify navigation areas, locations for promotional information, header and footer details, and where CMS400.NET controls should be placed.



Building Pages and Using Server Controls

Ektron provides a complete set of eCommerce server controls that allow you to set up an online market place where site visitors can purchase merchandise, services or content. Site visitors interact with these server controls on your site to

- select the type of currency
- search for products or view a list of products
- view the details of a product and add it to their shopping cart
- get recommendations on other products
- view products in a shopping cart, create new carts, and apply discounts
- go through the checkout process
- view a list of processed orders
- view and change their account information

Most of the eCommerce server controls allow site visitors to move from one control to another. In these server controls, there are template properties that allow you to define the location of another server control. For example, if you define the path to the Cart server control in the Product server control's `TemplateCart` property, site visitor are sent to the Cart control when they click the Add to Cart button in Product control.

eCommerce Server Controls

Ektron CMS400.NET's server controls let you insert many standard methods and properties within the Visual Studio 2005 environment. This means that you can see the effect of your changes in real time -- you don't have to modify a page then compile a sample project to see the results.

You can insert server controls using drag and drop or programmatically. You can also use databinding to retrieve and display data from Ektron CMS400.NET. This chapter explains the use of the Ektron CMS400 Server Controls through the following topics.

The following table shows typical actions a site visitor might perform when visiting your eCommerce site and how you can facilitate these actions.

Site Visitor Action	How you accomplish this	When to Use
View account information	Create a link in the master page that leads to the template containing the MyAccount server control.	This link should be available from any page.
View previous and current orders	Create a link in the master page that leads to the template containing the OrderList server control.	This link should be available from any page.
Change the currency type	Add the CurrencySelect server control to a master page.	This server control should be available from any page.
Search for products	Either create a link to the template containing the ProductSearch server control. - OR - On the master page, add a textbox that passes the text to a hidden ProductSearch server control and post the results to a separate template.	This link or textbox should be available from any page.
View a list of products	Create a template that contains the ProductList server control. This template might be the first page site visitors see when they view the eCommerce portion of your site.	When ever you want to display a list of products based on a catalog, taxonomy or a list content IDs. For example, you could create a list of products in a panel on the right side of your Web site.
View a product's details	Create a template that contains the Product server control. Setting the `TemplateProduct` property in the following server controls to the path of this control makes a product's title a clickable link that takes the site visitor to this control. • ProductSearch • ProductList • OrderList • Cart • Recommendation	When ever a site visitor needs to view details of a product.
View additional products associated with another product	Add the Recommendation server control to a template that contains a Product server control and set the Recommendation control's DynamicProductParameter property to the QueryString parameter that's used to pass a products ID.	When ever you want to present a site visitor with cross-sell and upsell opportunities associated with a given product.

Site Visitor Action	How you accomplish this	When to Use
Add a product to their shopping cart	Set the `TemplateCart` property in the following server controls to the path of the Cart server control. Setting this property is one of the requirements for the Add to Cart button or link to appear in these controls. • ProductSearch • ProductList • Product • Recommendation Note: In addition to setting the above property, a product must be in-stock, not archived and buyable; otherwise, the Add to Cart button or link does not show for a product.	When you want to allow site visitors to add products to their shopping cart.
View products in their shopping cart	When a site visitor adds a product to their cart, they are sent to the Cart server control. You should also create a link in the master page that leads to a template containing this control.	The link should be available from any page.
Checking out and paying for their selected product.	Add the Checkout server control to a template and add that template's path to the Cart server control's TemplateCheckout property. When you do, the Checkout button in the Cart server control becomes active.	You must set this up in order for site visitors to checkout.

Building the Store

This section describes how to set up a basic eCommerce Web site. This walk through explains everything you need to set in the web.config file and the Workarea. It also includes a list of templates needed and how to set up Ektron's eCommerce Server Controls on them.

Create Store Pages

After clicking the Save button from the previous step, a content editor screen appears and allows you to enter XML Smart Form information. This Smart Form is what a user fills out when creating a Catalog Entry (Product).

The information added by a user appears on a product's details page on your Web site.

Here are some fields you might want to create in your smart form.

- Title
- Description
- Image

Once you have the Smart Form complete, click the Save button.

More Information

To learn about defining Product Types, see the Administrator Manual section "eCommerce" > "eCommerce Products" > "Product Types."

To learn about Smart Forms, see the Administrator Manual section "Managing Content" > "Working with Smart Forms."

Create Catalogs and Assign Product Types

A catalog folder is a CMS400.NET folder designed to hold eCommerce entries (products). This is similar to the way content folders

hold HTML or Smart Form content. By assigning a product type to the folder, you can control the way products are added to the catalog.

Catalogs are created in the Workarea.

1. Click the **Content** folder bar to display the list of content folders.
2. Click **New** > **Catalog**.
3. Set the catalog's Properties, Metadata, Web Alerts and Breadcrumb information. (Similar to creating a Content Folder.)
4. On the **Product Types** tab, select a Product Type from the drop down list.
5. Click the **Add** link.
6. Click **Save**.

More Information

To learn about Catalogs and assigning Product Types, see the Administrator Manual section "eCommerce" > "eCommerce Products" > "Creating a Catalog Folder" and in that section, see "Assigning a Catalog Folder's Product Type."

Create Templates

Create the Web site templates your site visitors will use to interact with your eCommerce site.

Here is a list of the templates needed to create a basic eCommerce site.

- Master page - recommended, but not absolutely necessary. This template could contain any of the following:
  - CurrencySelect server control - allow site visitors to choose a currency.
  - View Cart link - links to the template containing the Cart server control.
  - View My Account / Orders link - links to the template containing the MyAccount and OrderList server control.
  - Login server control - allows site visitors and users to log in from any page.
- Landing page - this page should have a way for site visitors to start the shopping process and could contain a ProductList, ProductSearch server control.
- Product Display page - use the Product server control on a template to display the details of a catalog entry (product). If you are using the Cross Sell or Up Sell functionality, add a recommendation server control to this template.
- Product Search page - use the ProductSearch server control on a template to allow site visitors to search for product.
- Cart page - use the Cart server control on a template to allow a site visitor to work with the items they have selected to purchase.
- Checkout page - use the Checkout server control on a template to facilitate the check out process.
- My Account / Order History page - use a MyAccount server control and an OrderList server control to display a site visitor's account information and a list their order history.

The steps below are an example of creating a Web site using Ektron's eCommerce Server Controls.

Create a Master Page

Create a master page and add the following items to a header area or left side column.

CurrencySelect server control - allows site visitors to select from available monetary types. This control displays currencies that have been enable in the Workarea during the "Create a Master Page" on page 28 task. A My Cart link that leads to a template containing the Cart server control.

- My Account / Order History link that leads to a template containing the MyAccount and OrderList server controls.
- Product Search link - (optional) add a link that leads to a template containing a ProductSearch server control.

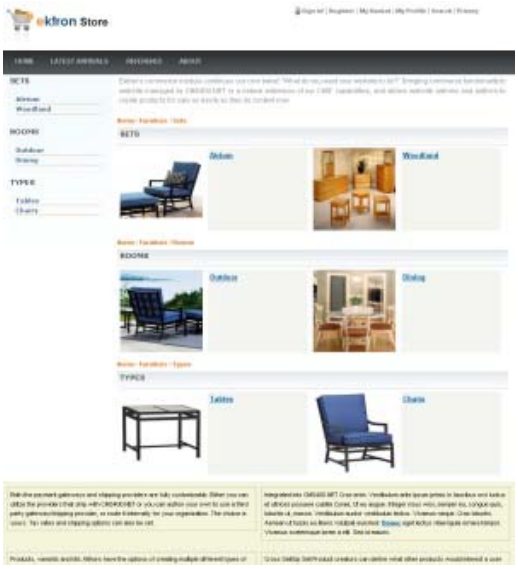
- Product Search server control - (optional) allows a user to search for a product from anywhere on the site. Note: Adding this option is involves more than just dragging and dropping a server control to the header or the left side column, it includes some advanced customizations and coding that allows the search term to be passed from one form to another. Ektron's Developer Sample site shows an example of doing this with the WebSearch server control.
- Login server control - (optional) this allows existing customers to login once they arrive at your site. If you only want site visitors logging in through this control, set the OnlyAllowMemberLogin property to True.

More Information

See the eCommerce server control documentation in the *Developers Guide*.

Create a Landing Page

This template should be the first page a site visitor sees when they arrive at your site and it should have a way for site visitors to start shopping for product. The landing/home page should look similar to this:



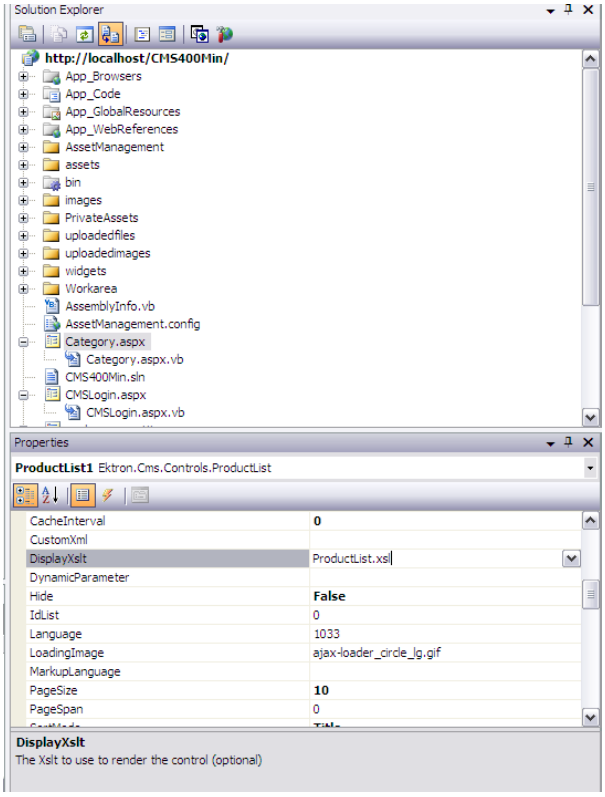
Make sure this page has one of the following:

- ProductList server control - use the this control to display products by Taxonomy, Catalog or ID.
  - To display a single taxonomy, set the SourceType property to Taxonomy and enter a single Taxonomy ID in the SourceId property.
  - To display multiple taxonomies, set the SourceType property to TaxonomyList and enter a comma separated list of Taxonomy IDs in the IdList property.
  - To display a single catalog, set the SourceType property to Catalog and enter a single catalog ID in the SourceId property. If you want to display sub catalogs for a given ID, set the Recursive Property to True.
  - To display multiple catalogs, set the SourceType property to CatalogList and enter a comma separated list of catalog IDs in the IdList property.
  - To display products by their ID, set the SourceType property to IdList and enter a comma separated list of product IDs in the IdList property.
  - Set the TemplateProduct property to the template containing the Product server control.
- ProductSearch server control - this control provides the means for site visitors to search your Web site for products. If this control is not on your landing page or part of your master page, you should create a separate template containing this control.
  - Set the CatalogId property to the ID of the catalog to search.
  - Set the TemplateCart property to the template containing the Cart server control.
  - Set the TemplateProduct property to the template containing the Product server control.

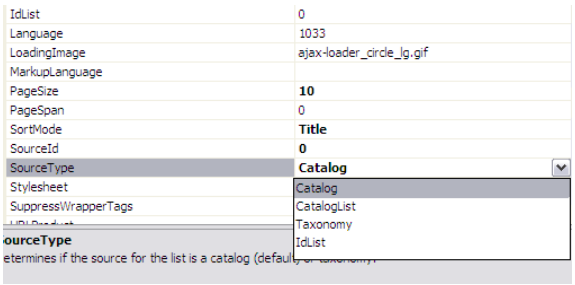
Create a Category Landing Page

When you open the category.aspx page, the template page is blank. Now you will start placing controls into this page.

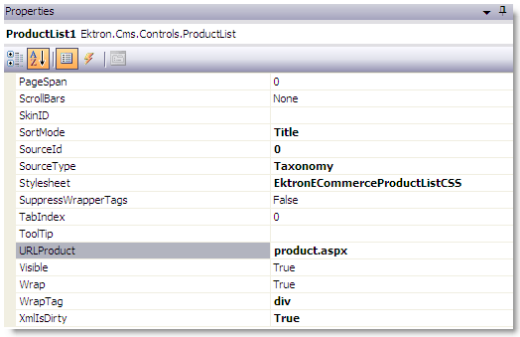
1. Select the Product List control and place it on the page. Display and styling of the product list is controlled through xslt files that you specify in the CMS Workarea > XSLT > Commerce Folder.



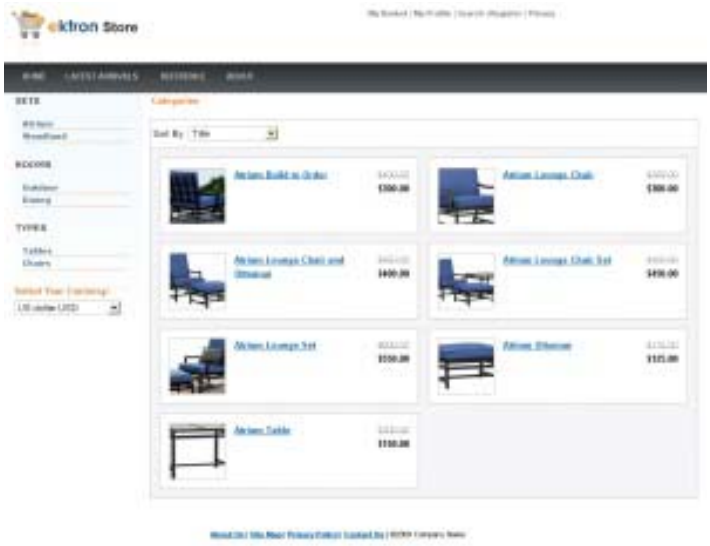
2. In the list of properties, select Catalog for the Sourcetype. This specifies the location from which you are getting your products. The other options are catalog list, taxonomy, taxonomy ids, idlist,



3. Select _product.aspx for URLProduct. This specifies the page to navigate to when a shopper clicks on a product.



When complete, the page should look similar to this:

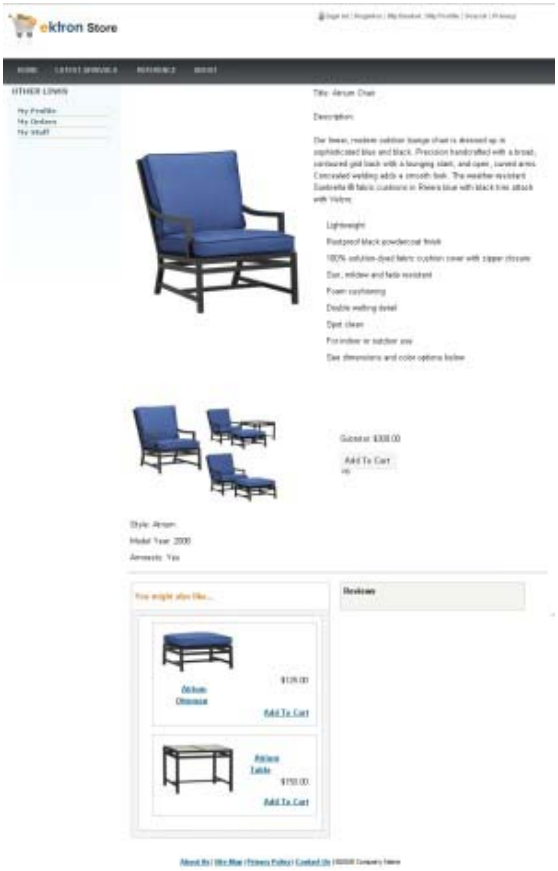


Create a Product Page

This is the template where a site visitor views a product's details. It contains a Product server control and optionally a Recommendation server control.

- Product server control - this control displays a product's details.
  - Make sure the DynamicParameter property is set to the parameter name used to pass product IDs to the QueryString.
  - If you want a default product to display when no ID is passed, set the DefaultProductID property to the ID of a product.
  - Set the TemplateCart property to the template containing the Cart server control.

- Recommendation server control- this control displays Cross Sell and Up Sell opportunities associated with a product. These are set in Workarea, under the View menu's Cross Sell and Up Sell selections for a catalog entry
  - Set the RecommendationType property to CrossSell or UpSell.
  - Make sure the DynamicProductParameter property is set to the parameter name used to pass product IDs to the QueryString.
  - If you want a product's default Cross Sell or Up Sell items to display when no ID is passed, set the DefaultProductID property to the ID of a product.
  - Set the TemplateCart property to the template containing the Cart server control.
  - Set the TemplateProduct property to the template containing the Product server control.



More Information

See the eCommerce server control documentation in the *Developers Guide*.

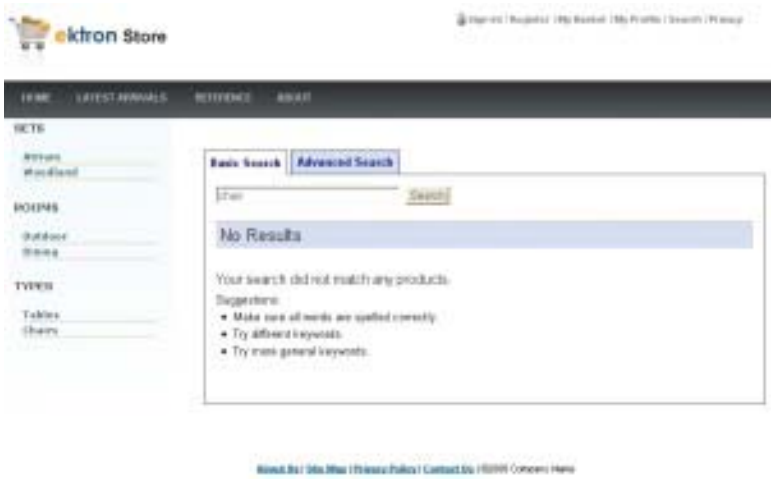
Create a Product Search Page

This page allows site visitors to search for products on your Web site.

- ProductSearch server control - this control provides the means for site visitors to search your Web site for products. If this control is not on your landing page or part of your master page, you should create a separate template containing this control.
  - Set the CatalogId property to the ID of the catalog to search.

- Set the TemplateCart property to the template containing the Cart server control.
- Set the TemplateProduct property to the template containing the Product server control.

When complete, the page should look similar to this:



More Information

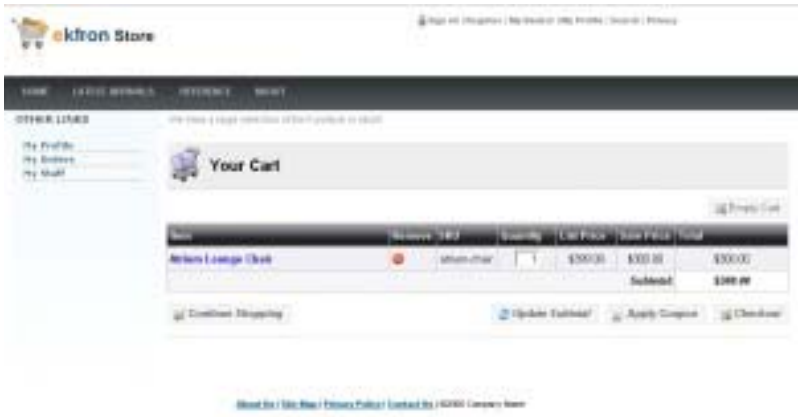
See the eCommerce server control documentation in the *Developers Guide*.

Create a Cart Page

This template contains a Cart server control.

- Cart server control - this control allows a site visitor to work with products they have selected to purchase. As a site visitor navigates around your site selecting products to purchase, they are added to a cart.
  - Set the TemplateCheckout property to the template containing the Checkout server control.
  - Set the TemplateProduct property to the template containing the Product server control.
  - Set the TemplateShopping property to the Landing page template or a template containing a ProductList or ProductSearch server control.
  - If you are using coupons make sure the EnableCoupons property is set to True.

When complete, the page should look similar to this:



More Information

See the eCommerce server control documentation in the *Developers Guide*.

Create a Checkout Page

This template contains a Checkout server control.

Note: Because Credit Card information is entered into the Checkout server control, Ektron strongly recommends using SSL Encryption for this page.

- Checkout server control - this control allows a site visitor to navigate through the checkout process.
  - Set the DefaultCountryID property to the country you want to be the default selection in the Billing and Shipping address sections.
  - Set the TemplateCart property to the template containing the Cart server control.
  - Set the TemplateOrderHistory property to the template containing the OrderList server control.
  - Set the TemplateShopping property to the landing page template or a template containing a ProductList or ProductSearch server control.
  - If you are using SSL Encryption, set the IsSSLRequired property to True.

When complete, the page should look similar to this:

More Information

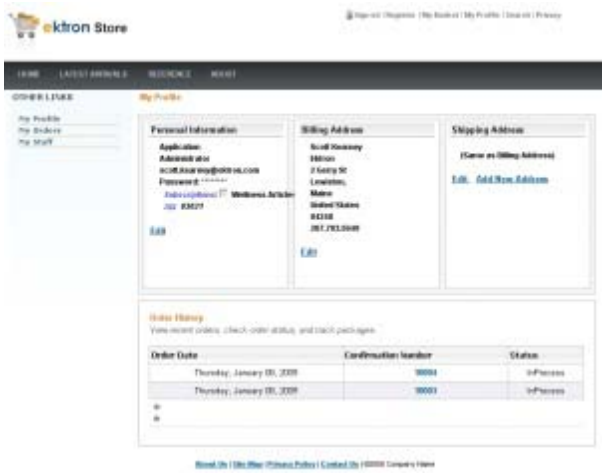
See the eCommerce server control documentation in the *Developers Guide*.

Create a “My Account” Page for Shoppers

This template contains a MyAccount and an OrderList server control.

- MyAccount server control - this server control allows site visitors to view billing, shipping and alternative shipping information associated with their account.
  - Set the DefaultCountryID property to the country you want to be the default selection in the Billing and Shipping address sections.
  - (Optional) Set the CustomPropertyID property to the ID or list of comma separated IDs that represent custom properties associated with users. Custom Properties are set up in CMS400.NET’s Workarea > Settings > Configuration > User Properties.
- OrderList server control - this server control allows site visitors to view a list of their processed orders.
  - Make sure the DynamicOrderParameter is set to the parameter name used to pass order IDs to the QueryString.
  - Make sure the DynamicProductParameter is set to the parameter name used to pass product IDs to the QueryString.

When complete, the page should look similar to this:



More Information

See the eCommerce server control documentation in the *Developers Guide*.

4 Product Management

Building EktronStore.com

Adding Product Type Definitions

Create a separate *product type* definition for each category of catalog entries that you sell. For example, you might have a product type for movies, another for books, a third for electronics, etc.

When thinking about creating new product types, the most significant differences are the Class field (described below) and the content page, which defines the XML Smart Form for products you will create of a Product Type.

A product type determines the following aspects of the catalog entries based on it.

Catalog Entry Property	Description
Product class	A catalog entry's product class affects site visitors' purchasing options. Class can be either: • Kit • Bundle • Product • Subscription
Content page	After you complete and save the Add Product Type screen, a new screen allows you to enter XML Smart Form information. On this screen, you set up fields to collect information for the *content page* of catalog entries that will use this product type.
Attributes	Define additional information about catalog entries based on this product type. Attributes typically appear on the Product server control.
Media defaults (that is, size of thumbnail images added to a catalog entry)	If desired, enter sizes of thumbnail images. If you do, thumbnails are automatically generated for images (added on the Media tab) applied to catalog entries based on this product type.

Product Classes

A catalog entry's product class affects site visitors' purchasing options. Classes are explained below.

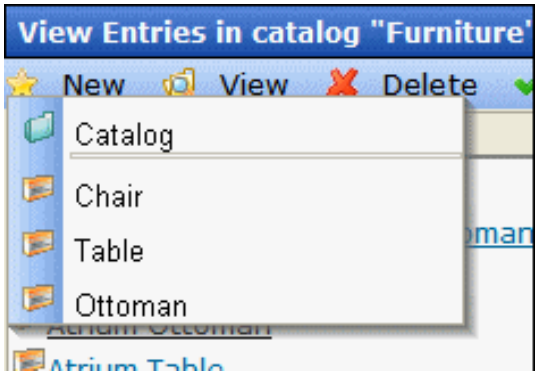
Class	Description	Example
Kit	Let a site visitor select from any number of free-text options. Options can increase or decrease the overall price.	A computer whose price changes as a site visitor selects RAM, hard drive, monitor, etc.
Bundle	A catalog entry consisting of several other catalog entries bundled together. Its pricing, shipping, images, etc. are independent of the individual items.	A living room set: couch, end tables and lamps. Instead of buying each item separately, the site visitor buys all for a "package" price.
Product	Simple or complex catalog entries. A complex product is a "wrapper" that provides links to related simple products. Each simple product has its own SKU, price, inventory data, etc.	A movie in three formats and prices: VHS ($12.95) DVD ($14.95) Blu-ray ($17.95)
Subscription	A catalog entry which can provide access to designated site pages, and may billed on a recurring basis.	Web site content that is only available to subscribed members

Adding a Product Type

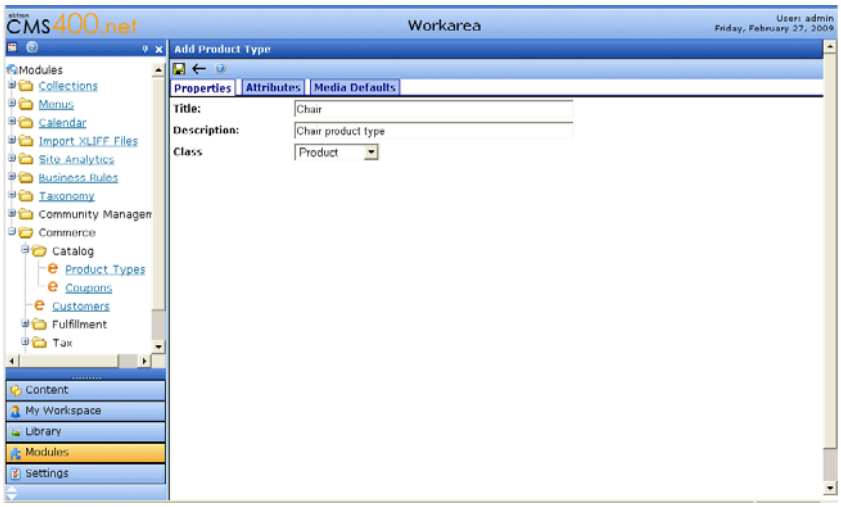
In this example, you are creating a “Chair” product type.

1. Go to **Workarea** > **Modules** > **Commerce** > **Catalog** > **Product Types** > **New** > **Product Type**.
2. In this example, add “Chair” as the Title. The Title identifies this entry on the View Product Types page.

   When creating a new catalog entry, users must select a product type from the catalog’s New menu (shown below).

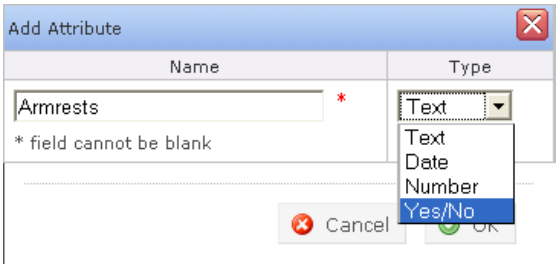


3. Enter as “Chair Product Type” as the Description, and select “Product” as the Class.

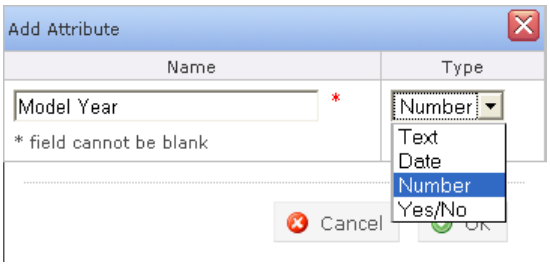


4. Click the Attributes tab. Attributes define additional information about catalog entries based on the product type.

   By default, Attributes appear on the Product server control.
5. Add a new attribute called “Armrests” as a Yes/No (Boolean) type.

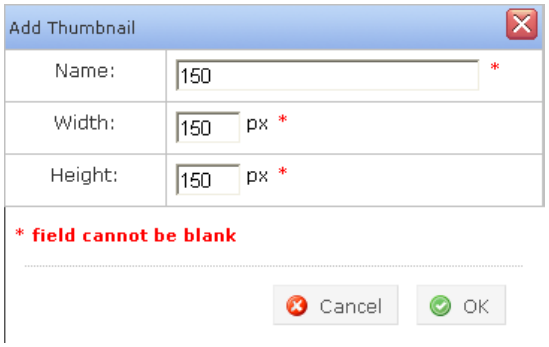


6. Add a second attribute called “Model Year” as a Number type.

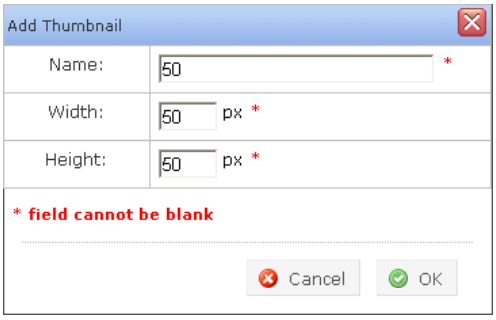


7. Add a third attribute called “Style” as a Text type.
8. Click the Media Defaults tab.

   Here you are instructing the CMS to generate both 150px x 150px and 50px x 50px thumbnails when a new image is added for a product. To do this, add two thumbnail definitions to the Media Defaults tab (step 9 and 10). Then, whenever you add a new image to a chair through the Media tab on the Product Editing page, both a 100px x 100px and 50px x 50px thumbnail is generated from that image.
9. Add a 150px by 150px media default thumbnail.



10. Add a 50px by 50px media default thumbnail and click **Save**. The product type design window displays.

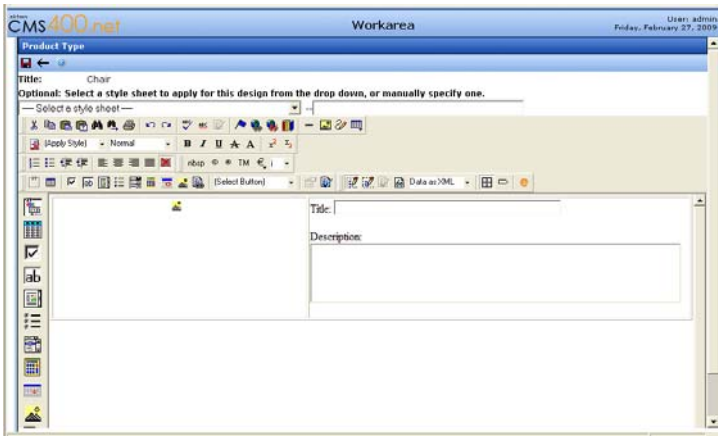


Attribute Statuses:

Note: You can delete an attribute only if it is *not* applied to a catalog entry. If you try to delete an attribute applied to a catalog entry, it is marked "Inactive." Inactive attributes can no longer be assigned.

- **Active** - Attribute can be assigned to catalog entries
- **Inactive** - Attribute already assigned to one or more catalog entries, but can no longer be assigned. To make an attribute inactive, click the corresponding Mark for Delete button ( ).
- **Not Published** - Attribute has been created but Product Type not saved yet
- **Active** - Marked for delete. Attribute will be deleted when Product Type is saved.

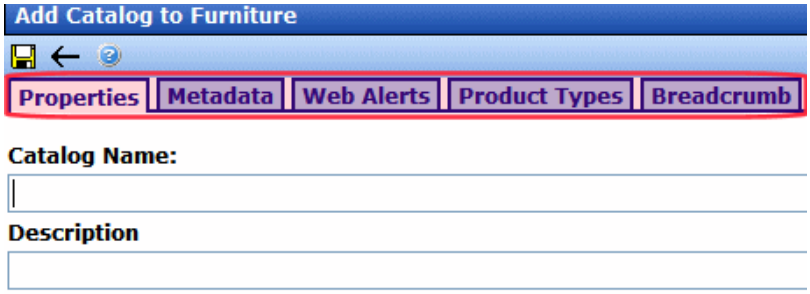
11. Add a two column table with an image field in the left column and a text box and rich field in the right column, as shown above. Click Save, and continue on until you reach the main Product Types page.



Creating a Catalog Folder

A catalog folder is a special kind of Ektron CMS400.NET folder designed to contain eCommerce products. Its unique icon ( ) distinguishes it from other folder types. If you are familiar with CMS content folders, much of that information applies to catalog folders (see "Managing Content Folders" in the *Administration Guide* for more information).

Catalog folders have several tabs/screens, as shown below.



These tabs work the same as regular content folders. The Product Types tab is unique to eCommerce. See "Associating a Product Type with a Catalog" on page 42.

Like content folders, the following catalog folder information can be inherited from its parent or uniquely set for each catalog. You can only make changes after creating the catalog.

- Permissions
- Approvals
- Purge History
- Restore Web Alert Inheritance

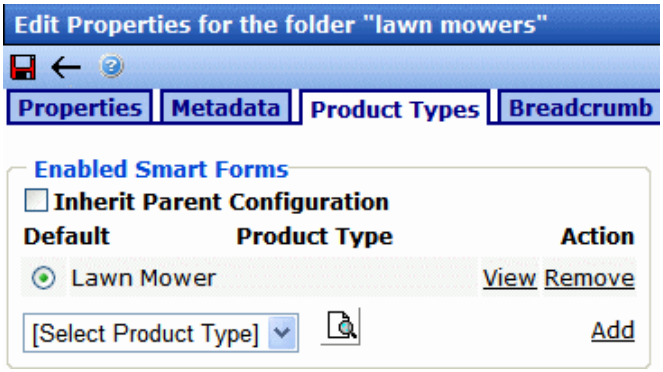
Associating a Product Type with a Catalog

A catalog folder's Product Types screen lets you identify product types upon which catalog entries can be based.

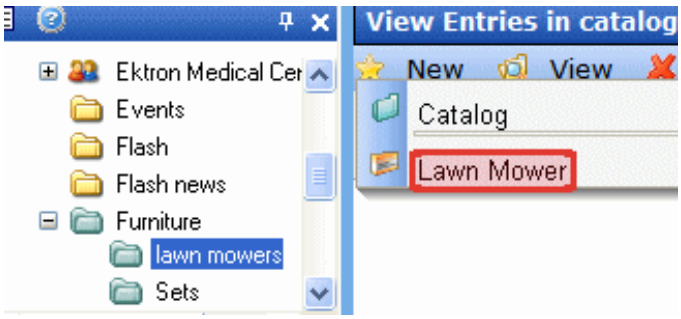
Catalog folders generally contain the same kind of catalog entries, such as DVDs or digital cameras. Some information applied to a catalog folder is inherited by all entries in the folder.

For example, you created a Sofa Product Type, and want only sofas to be entered in a catalog folder. In this case, open a catalog folder's Product Types tab, break inheritance if necessary, and select Sofa as the catalog's only product type.

After you create product types, apply appropriate ones to catalog folders using the Catalog Properties screen's Product Types tab (shown below).



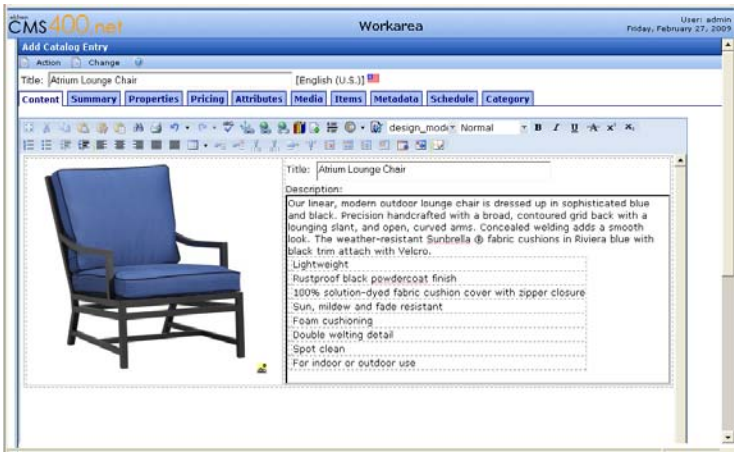
Then, when a Ektron CMS400.NET user creates an entry in this catalog, the user must use the Sofa product type.



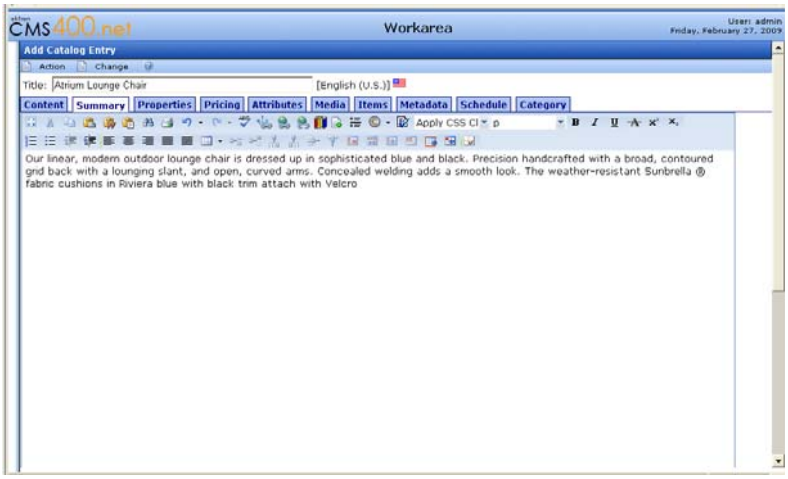
Adding a Product

To create a new catalog entry, you first navigate to a catalog folder. Then, hover the mouse cursor over New and choose a product type assigned to the folder (shown below).

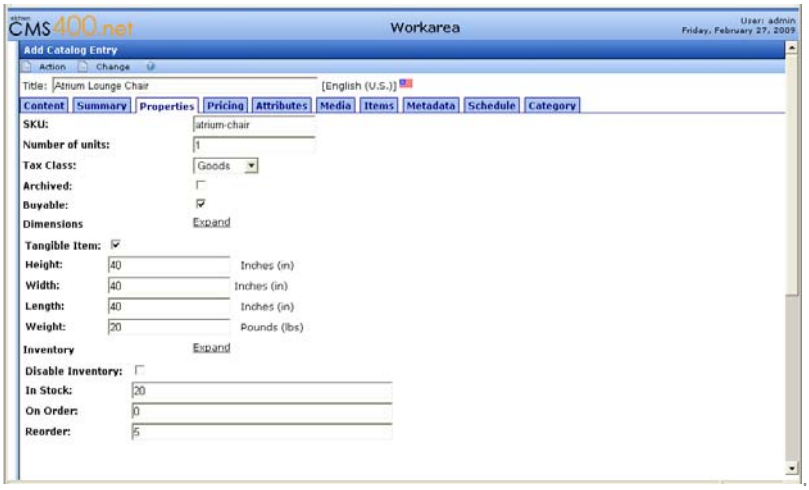
For this example, you select Chair. This launches up the Add Product page where you set information about the product. You first fill-in the Smart Form on the Content tab (shown below). Enter “Atrium Lounge Chair” into the Title field, some text into the Description, and select an image from the Library by clicking the image field on the left.



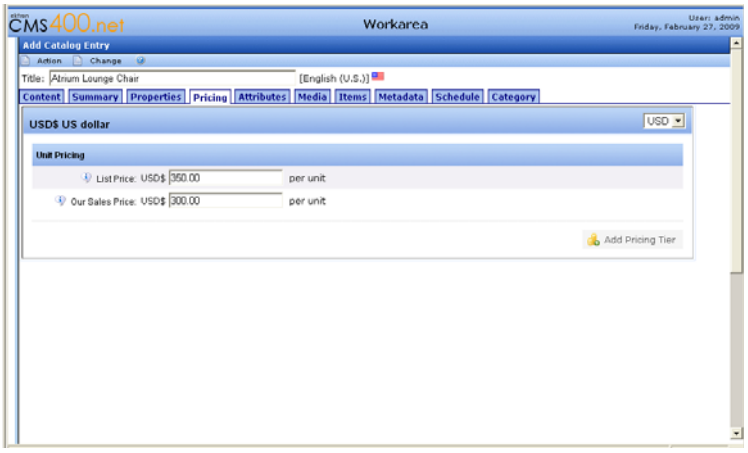
On the Summary tab, enter a short description of the product.



The Properties tab is where you define aspects of the product such as SKU number, tax class, dimensions and inventory information. Enter “atrium-chair” for the SKU, select “Goods” as the tax class, and check “Tangible Item”. Set the Height, Width and Length fields to 40 inches, and set the weight to 20 pounds. Clear “Disable Inventory”, set the In Stock inventory to 20, and the reorder level to 5.



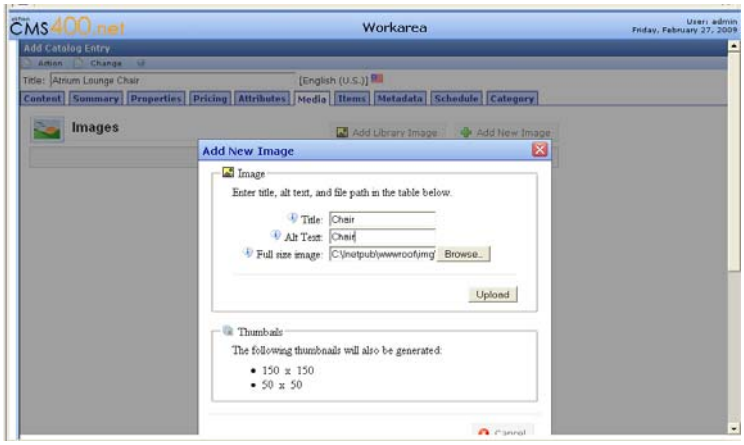
The Pricing tab is where you set the cost of this product. List Price is the price that the item retails for in the catalog. Sales Price is the price it actually sells for. It is the difference between these two numbers which can create the perception of value. For example, a product with a $100 list price is on sale for $30, which can then be retailed as 70% off. In this example, set the List Price as $350.00 and the Sales Price as $300.00.



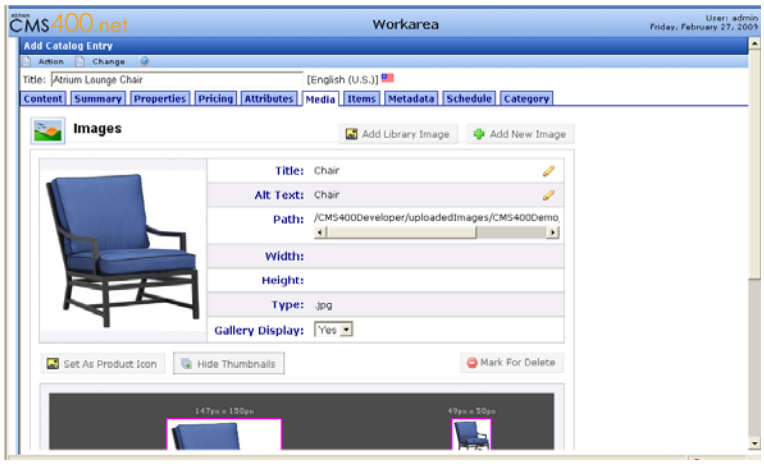
In the Attributes tab, set the product's attributes. For this example use these properties:

- Arm Rests
- Model Year
- Style

In the Media tab, you can add new product images which provides an image gallery effect on the site with the automatic thumbnail generation feature. In this tab, click "Add New Image", and in the model select a local image. Add the Title and Alt Text values, then click Upload.



This uploads the image, generates the thumbnails and then inserts the image into the Media tab's list. Clicking View Thumbnails allows you to see the thumbnails that were generated.



In the Metadata tab, you can set the SEO properties of the product, such as Title, Keywords and any other metadata definitions you have applied.

In the Category Tab you can select the taxonomy categories that apply to the product.

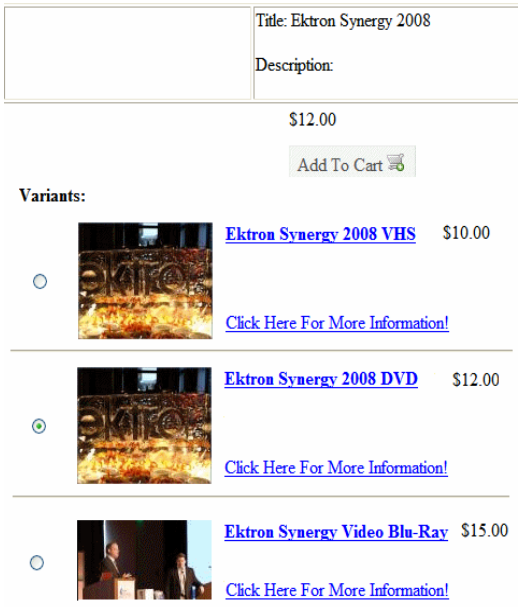
Complex Product

A Complex Product is similar to a bundle in that one catalog entry serves as an "umbrella" for other entries. However, unlike a bundle, a site visitor must select one of the assigned catalog entries. So, the price is that of the selected entry, not the umbrella item.

The advantage of a Complex Product is its ability to consolidate several related items under one title, yet allow site visitors to choose the item they want. Because each item under the umbrella is its own catalog entry, it is priced and tracked separately.

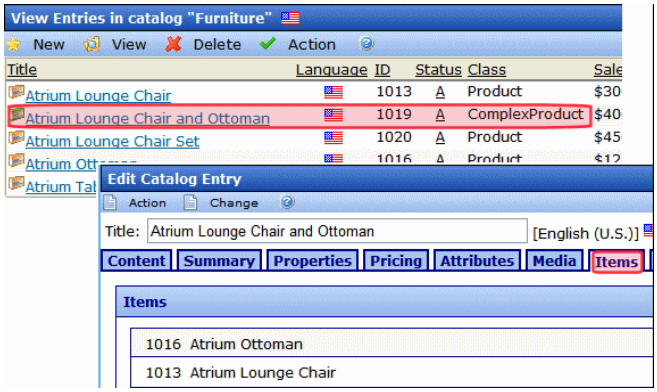
For example, a site sells movies in three formats: VHS, DVD, and Blu-ray. Each is priced differently. To accommodate this, create catalog entries for the movie in each format, assigning to each one images, SKU number, pricing, dimensions, etc. Then, create a Complex Product, make its title the title of the movie, and use the Items tab to select the three catalog entries of the individual formats.

The Product List or Product Search control displays the Complex Product, which is the movie title. When a site visitor selects that, they see the product page, which lists the format and price of each version (that is, each assigned catalog entry). They can only choose one version. See example below.



Creating a Complex Product

Unlike the Kit or Bundle, a complex product does not have its own product class. Instead, you create a catalog entry whose product class is product, then assign other catalog entries to it via the Items tab. As soon as you assign one catalog entry via the Items tab, the original catalog entry is converted to a Complex Product.



Note that the price you assign to the “umbrella” product appears on the Product List and Product Search server controls. This can be misleading, since the price of individual products within the complex product can vary.

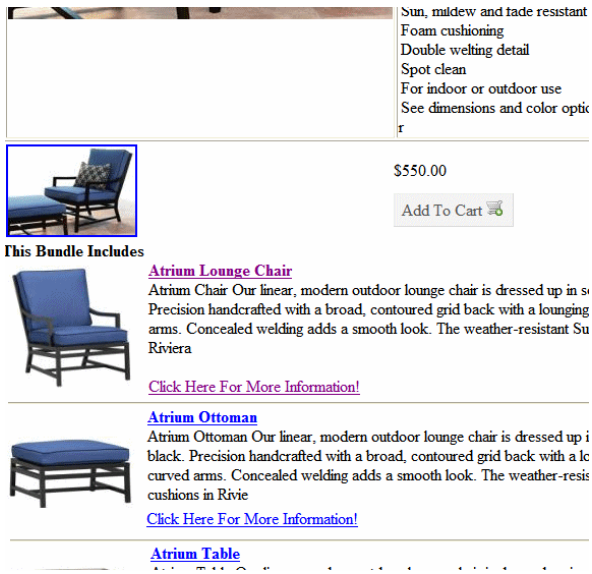
Bundles

A bundle consists of several simple products. You would use it to combine several existing products into one “umbrella” catalog entry. That entry is used to store and update pricing, SKU number, shipping, and inventory information for the bundle. The information for the individual entries that make up the bundle are not affected when one is ordered.

As an example of a bundle using furniture, assume your eCommerce site sells a chair, a table, and an ottoman separately. You also sell them together as a set.

Use a bundle to define the set, add pictures, pricing, etc. On the bundle's Items tab, you select the individual catalog entries that make up the bundle, as shown below.

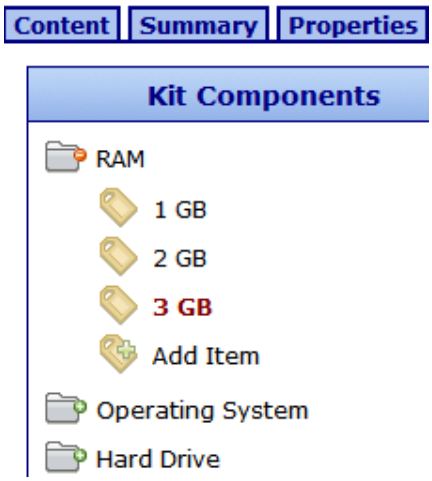
Here is how this bundle looks on a product page.



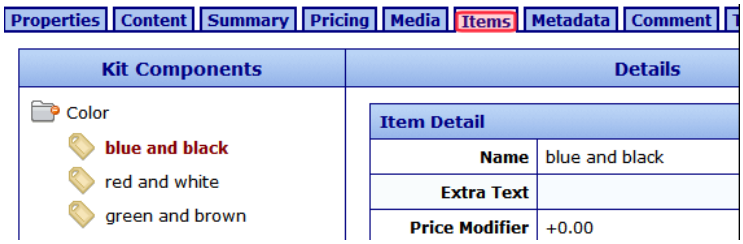
Kits

A kit is a type of catalog entry that allows a site visitor to select from free-text options, which can affect the item's price. The options can also be placed into logical groups.

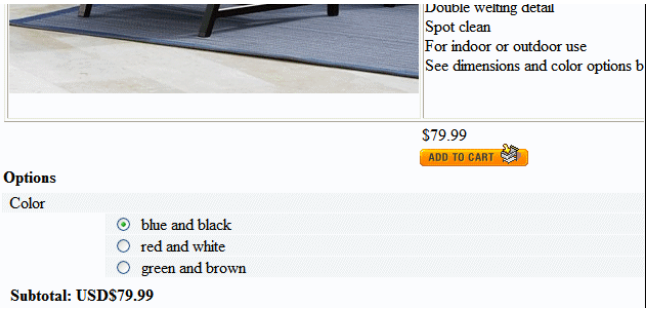
For example, if a catalog entry is a laptop computer, one group may let you enter options for RAM, another for operating system version, and a third could present hard drive options. This is illustrated below.



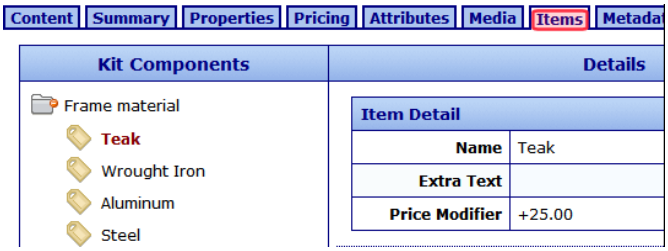
There is no limit to the number of groups you can add, nor to the number of options in a group. All groups and options are simple text, created on a kit's Items tab as needed, and can accommodate a price modifier. The example below shows how to set up a group of chair color combinations.



In this example, the options do *not* affect the price. Here is how the options appear on a product page.



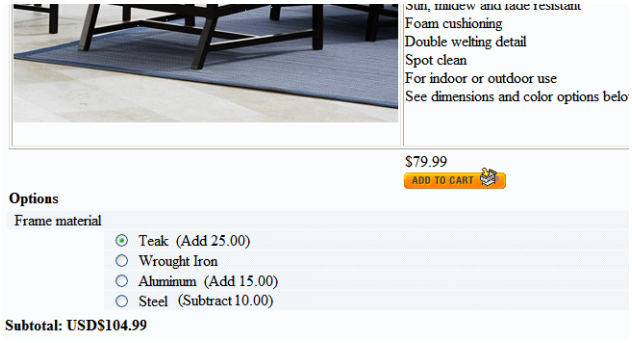
Consider another example in which the price does change. In this case, use the Item screen's Price Modifier field to adjust the price based on the chair frame. As shown below, if the chair frame is teak, the sale price increases by $25.00.



Other chair frame materials might adjust the price as shown below.

Frame material	Price Modifier
Wrought Iron	$0.00
Aluminum	+$15.00
Steel	-$10.00

Note how the Price Modifier can either increase or decrease the sale price. Here is how these options look on the product page.



The chair's base price is $79.99, and the teak frame increases it to $104.99.

Subscriptions

A subscription is a good or service which

- places users who purchase it into a designated user group, letting you grant them access to private content
- can be billed on a one-time or a recurring basis

Examples include

- The CMS content that is only available to members who purchase a subscription
- a gym membership
- a book club, in which a customer commits to the same payment and receives a different book each month
- anti-virus software that expires after 12 months
- a 3-year maintenance contract on a digital television, payable yearly

If you use the recurring billing feature, the bill can be generated for any number of months or years, but no other time increments. Each payment amount must be the same for the term of the subscription. The recurring billing term begins when a customer submits the order.

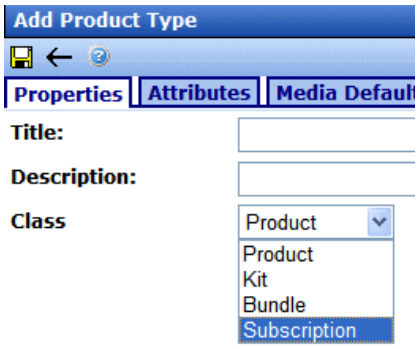
Site visitors cannot apply a cart-level coupon to a subscription-based catalog entry.

Creating a Subscription Based Catalog Entry

Creating a subscription-based catalog entry is the similar to creating a regular catalog entry. The table below lists everything you need to do, highlighting tasks that are unique to subscription-based entries.

Create a subscription-based Product Type:

1. On the Product Type screen's **Product Class** drop-down, select **Subscription** (illustrated below).



2. Select the default Subscription Provider (MembershipSubscriptionProvider).
3. Assign the **Product Type** to a catalog folder.
4. In that folder, create a catalog entry for each subscription.
5. On the **Pricing** tab, enter the cost. If using recurring billing, enter the term of the subscription.

   If this product does not use recurring billing, click **No** at the **Use Recurring Billing** drop-down.
6. On the **Items** tab, assign membership and CMS user groups.
7. If a subscription provides access to CMS content, use its folder Permission Table to grant permission to the membership and CMS user groups you defined in the previous step.

Entering Recurring Billing Information:

The Pricing tab's Recurring Billing area of a subscription-based catalog entry (highlighted below) lets you determine if a subscription will be billed on a one-time or a recurring basis. If recurring, it helps you define the terms.

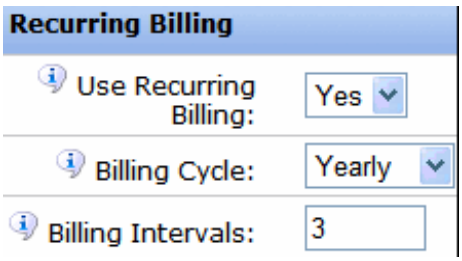
To set up recurring billing (that is, a series of scheduled payments), click Yes at the Use Recurring Billing drop-down.

The recurring billing term begins when a customer submits an order, and is based on a number of months or years. You use the Billing Cycle drop-down to choose a time interval.

Next, enter a number of Billing Intervals for which a customer is charged. For example, if a customer should be charged once a month for 12 months, complete the screen as shown above. If a customer purchases the item on June 1, 2008, his credit card will be billed on these dates.

- June 1, 2008 (purchase date)
- July 1, 2008
- August 1, 2008
- September 1, 2008
- October 1, 2008
- November 1, 2008
- December 1, 2008
- January 1, 2009
- February 1, 2009
- March 1, 2009
- April 1, 2009
- May 1, 2009

As another example, if a customer should be charged once a year for 3 years, complete the screen as shown below.



If a customer purchases the item on June 1, 2008, his credit card will be charged on the following dates.

- June 1, 2008 (purchase date)
- June 1, 2009
- June 1, 2010

Assigning Membership and User Group to a Subscription-Based Catalog Entry

You *must* assign a membership user group to a subscription-based catalog entry. Optionally, you *can* assign a CMS user group. You assign groups through the subscription's Items tab.

So, as part of setting up a subscription, you must either create new or use existing groups.

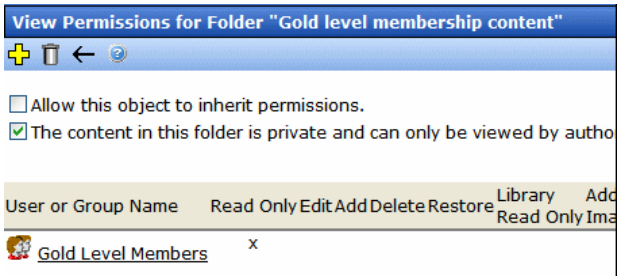
All users who purchase the product are automatically added to one of the groups.

- CMS users are added to the group defined in the CMS Author Group field
- Non-CMS users are added to the membership group defined at the Member Group field

When a user is provisioned into the membership or CMS author group, you can use the security permissions of the CMS to enable private content.

First, create a regular content folder in which to create the confidential content.

Next, use the content folder's permission table to grant the subscription user groups access to that folder.



Upselling and Cross-selling

Upselling and cross-selling are designed to increase the amount of cart sales.

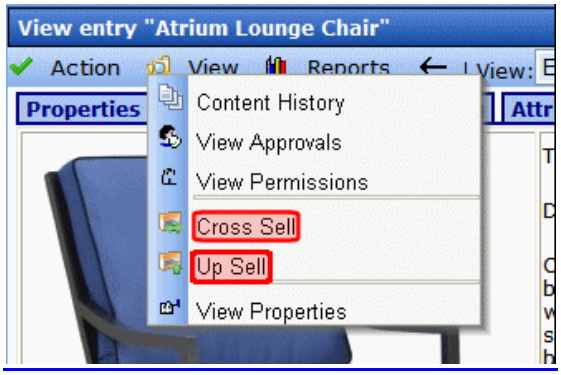
Cross-sell offers items related to a catalog entry. So, for example, if someone is purchasing a digital camera, cross-sell items might be a carrying bag, digital photo printer, larger memory cards, additional warranty, etc.

Upselling refers to similar but more expensive items. So, for example, if someone is viewing a 7 megapixel digital camera, upsell items might be a 10 megapixel camera, or a digital SLR camera.

Assigning Cross Sell and Upsell Items to a Catalog Entry

1. Open the catalog folder containing the catalog entry to which you want to assign cross sell items.
2. Click that entry.

3. Click **View** > **Cross Sell** or **View Upsell**.



4. Click the **Add** button ( ).
5. A screen displays all catalog folders.
6. Open the folder that contains the cross sell or upsell catalog entry.
7. Click that catalog entry.
8. Press **OK**.
9. Enter additional cross sell entries as needed.
10. Click **Action** > **Save**.

Displaying Cross Sell and Upsell Items on Your Web Site

Use the Recommendation server control to place cross-sell or upsell items anywhere on a Web page. A single control can only show *either* cross sell *or* upsell items.

5 Orders and Analytics

Running your store

The fulfillment aspect of eCommerce involves:

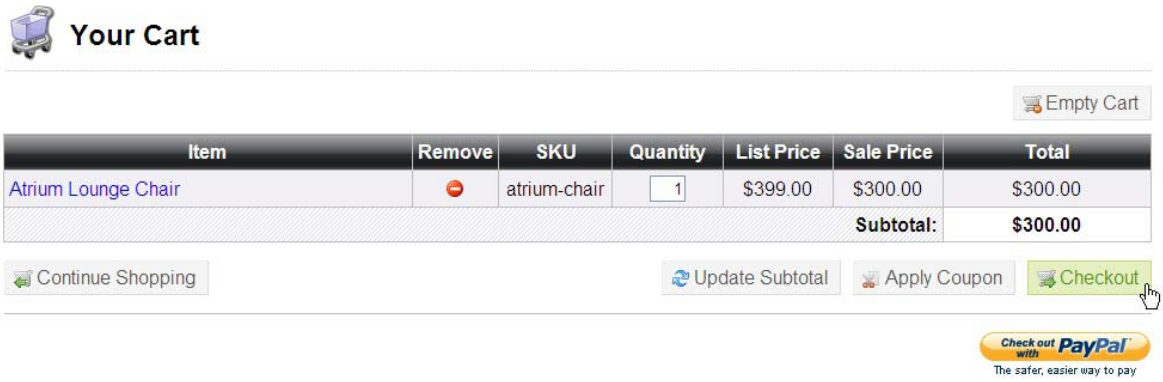
- **Order placement** — Customers adding items to their carts, navigating through the checkout process, and submitting orders.
- **Tracking submitted orders** — This includes order status, charge capture, shipping, and cumulative sales information on customers.
- **Customer information** — This includes orders submitted, addresses, and current shopping carts.

The Order Process

Shoppers adds items (physical good, subscriptions, or soft goods) to their cart by clicking the “Add to Cart” button related to that item.

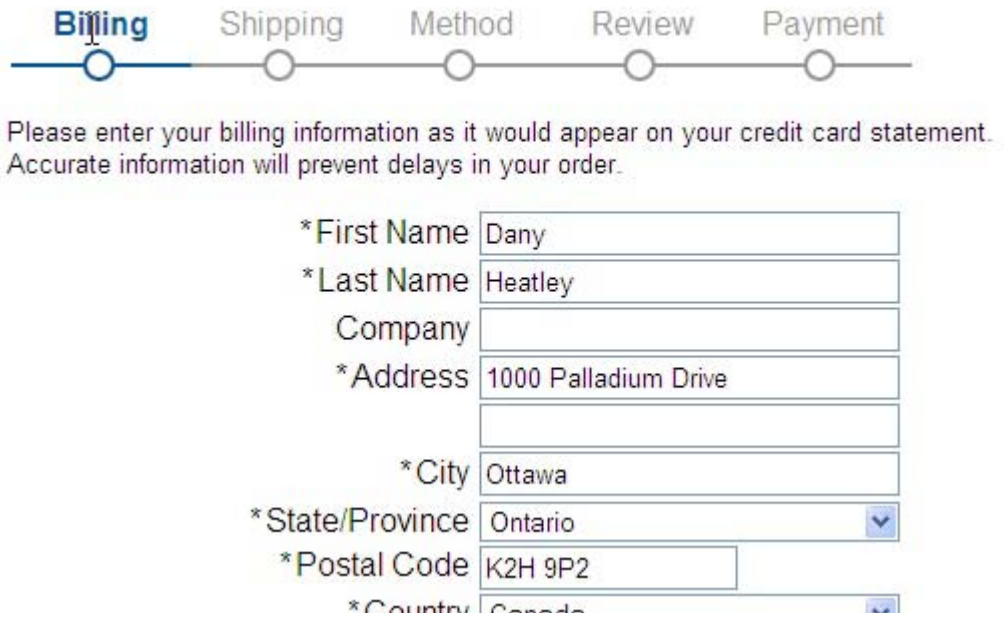
When a shopper is ready to pay for their items, they click the Checkout button and begin the checkout process.

Note: Shoppers have the option of checking out using PayPal’s checkout system. All order information is exported and a shopper can use their PayPal account to fund the purchase. See the *Administration Guide* for more information about using PayPal Checkout in your online store.

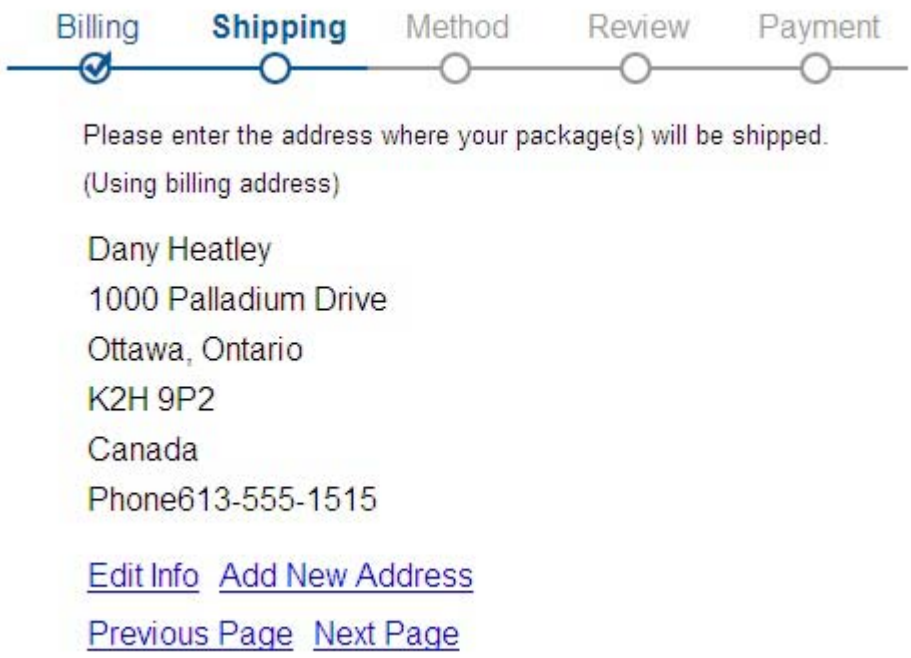


After clicking Checkout, a returning shopper can log into their account and con ti nu re the checkout process. The shopper’s shipping and billing information appears automatically.

New shoppers click the Next Page link and enter their information in the Billing Page.

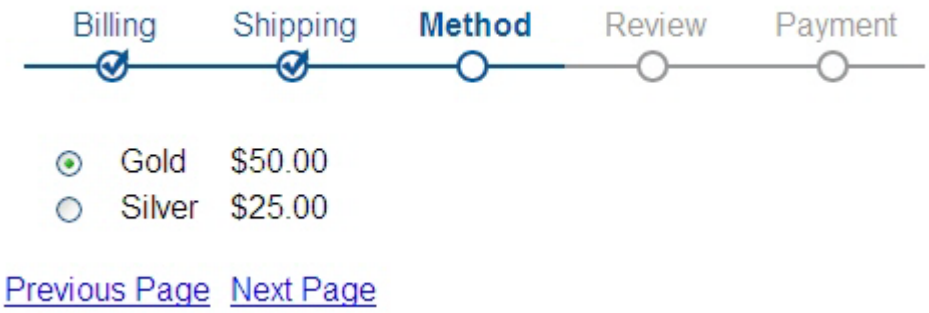


Clicking **Next Page** creates an account for the shopper. After confirming the information they entered, the shopper can then enter a different shipping address.

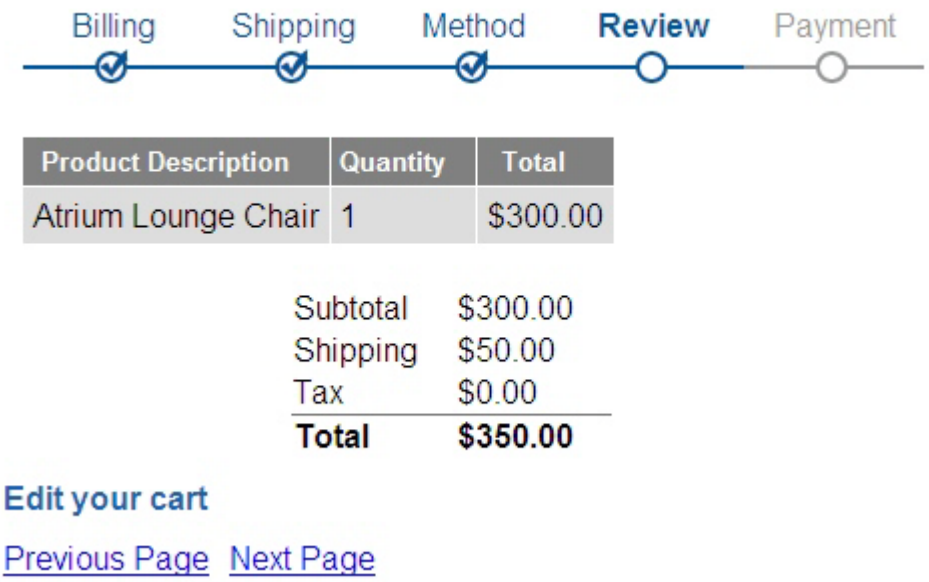


When the shopper clicks the **Next Page** link, they can then select the method by which they want their item delivered.

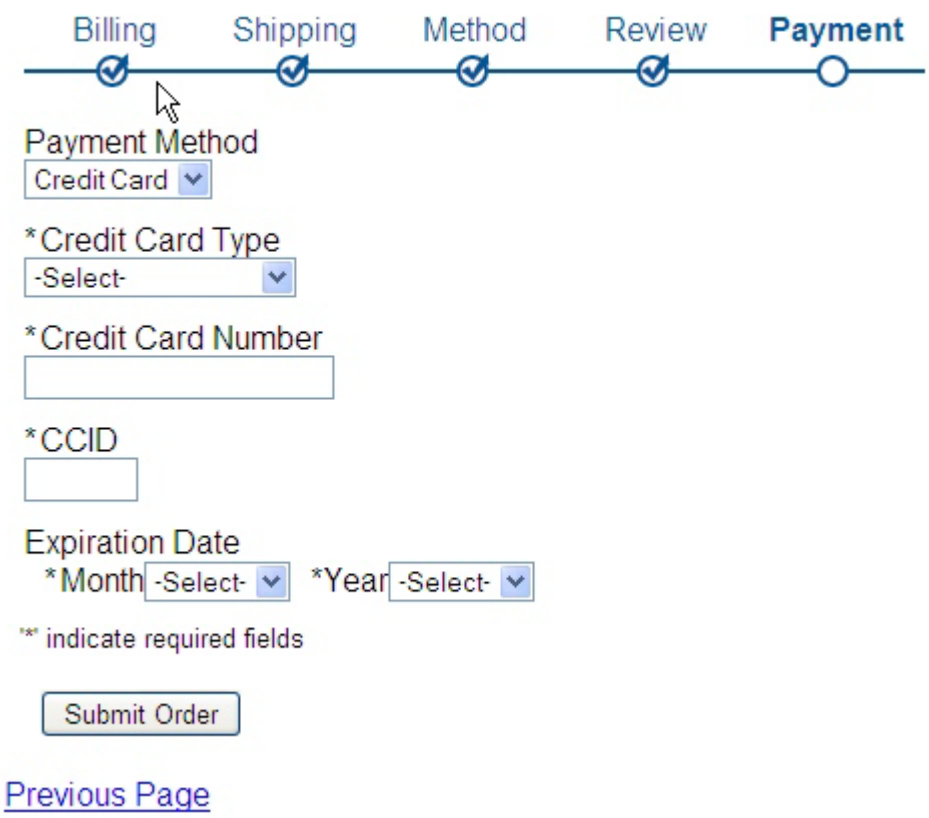
Note: In most cases, if the cart contains a subscription or soft good (like downloadable software), this page is bypassed.



The shopper can modify the order in the Review Page by clicking the **Edit your cart** link. Otherwise, they can move forward to the Payment options page.



Here, the shopper specifies payment information and submits the order.

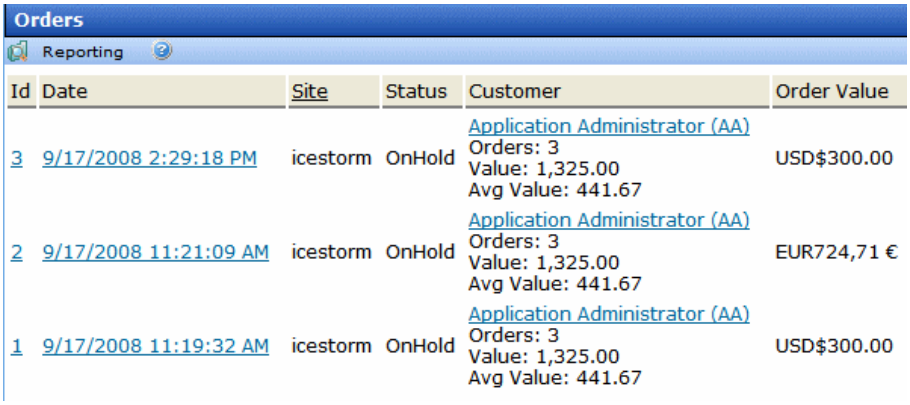


An order invoice page displays to confirm the order. If the order was for a subscription service to content or downloadable software, additional information about accessing these items will also appear on the page.

Processing Orders

The orders screen provides information about all orders in your eCommerce system. To access it, go to Ektron CMS400.NET **Workarea** > **Modules** > **Commerce** > **Fulfillment** > **Orders**.

Note: In order to process orders, the Distributed Transaction Coordinator (DTC) Windows service must be running. It is started by default when you install Ektron CMS400.NET.



Initially, orders are sorted by submission date and time, with the most recent at the top.

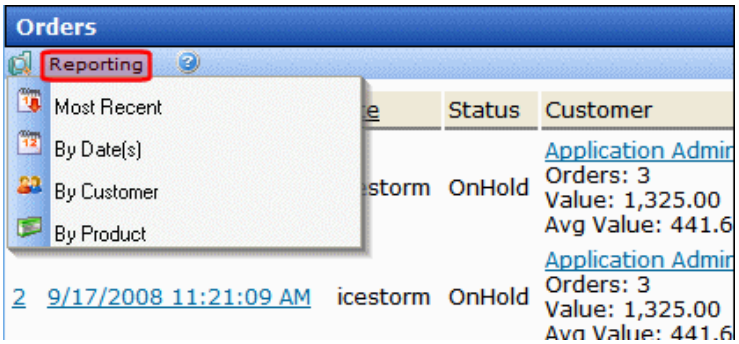
Selecting Orders by Criteria

You can select orders by any of the following criteria.

- entry date range
- customer
- catalog entry

To use any filter, click Reporting (see below) then the appropriate filter option.

Note: The Most Recent menu option is the default sort order.



Order Statuses

The following table assumes you use the default workflow installed with eCommerce. If you customize the workflow, the following table might not represent your custom workflow.

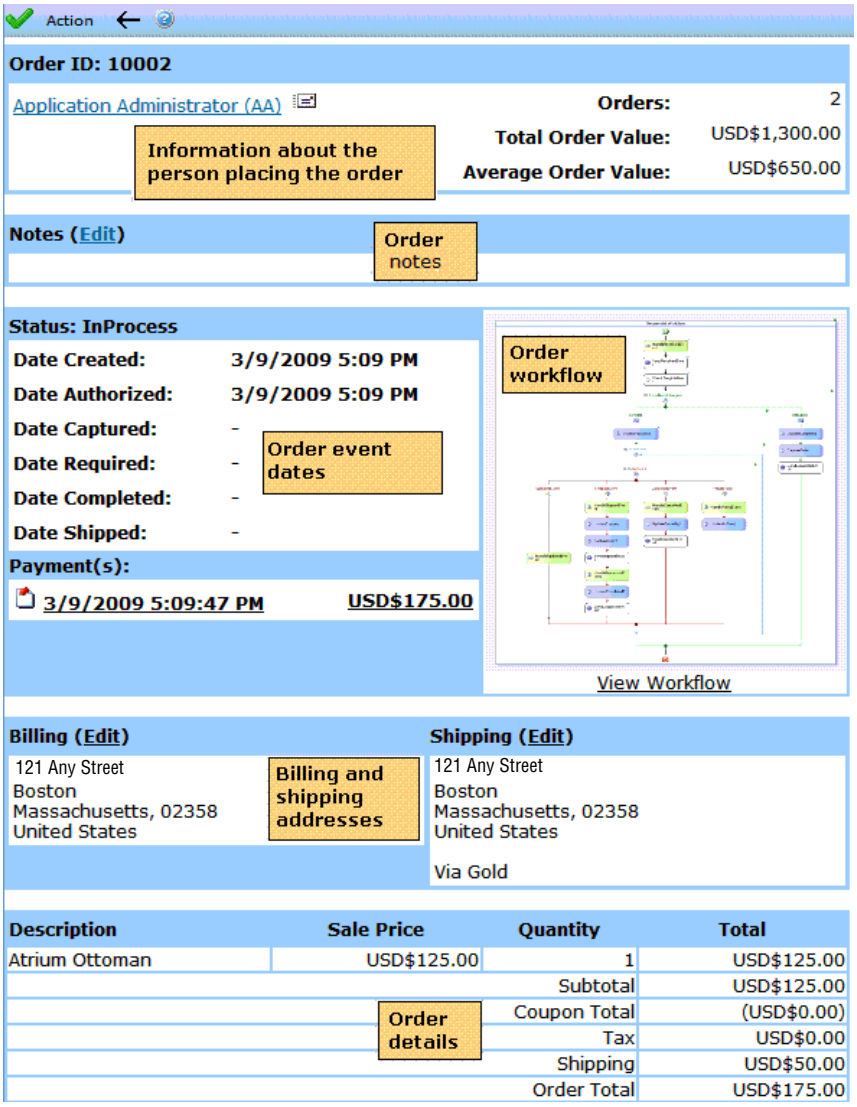
Status	Description	How Set
Received	Order has been submitted.	Site visitor submits order
Fraud	Your organization has determined that the order is fraudulent.	View Order Screen > Action Menu > Mark as Fraud

Status	Description	How Set
Shipped	Order has been shipped Note: This status can be the workflow's end point. You do not need to use the Completed status.	View Order Screen > Action Menu > Ship Order or View Order Screen > Action Menu > Enter/Edit Tracking Number > Check Mark as Shipped checkbox and save
Completed	All order activities are finalized. No events can occur to a Completed order.	View Order Screen > Action Menu > Process Order
Cancelled	Order has been cancelled.	View Order Screen > Action Menu > Cancel Order
InProcess	Automatically assigned when order is submitted.	Order placed but not yet captured

The View Order Screen

To view the details of any order, click it. When you do, the View Order screen appears.

Note: If the View Workflow option does not display a flowchart, verify that the server's Distributed Transaction Coordinator (DTC) Windows service is running. It is started by default when you install Ektron CMS400.NET, but if it stops for some reason, the Workflow does not appear on this screen.



The View Order screen's Action menu has the following options.

- Capture - See "Capturing the Order" on page 64
- Mark as Fraud - "Marking the Order as Fraud" on page 66
- Enter Tracking Number - "Entering a Tracking Number" on page 65
- Cancel Order - "Cancelling the Order" on page 65

You can also edit the order's billing and shipping addresses. See "Editing an Order's Billing and Shipping Addresses" on page 66.

These options are explained below.

Capturing the Order

Authorization versus Capture

Note: In many states, you must ship an item before you can charge a customer's credit card for it.

Authorization occurs while a customer is using the Checkout server controls. It submits a customer's payment information to a payment gateway, bank, or PayPal, and returns either an approval with an authorization number (transaction ID) or a decline along with an explanation.

Capture is the process of submitting encrypted order information (including the transaction ID) to a payment gateway account, bank, or PayPal. At this time, the account is charged for the order amount.

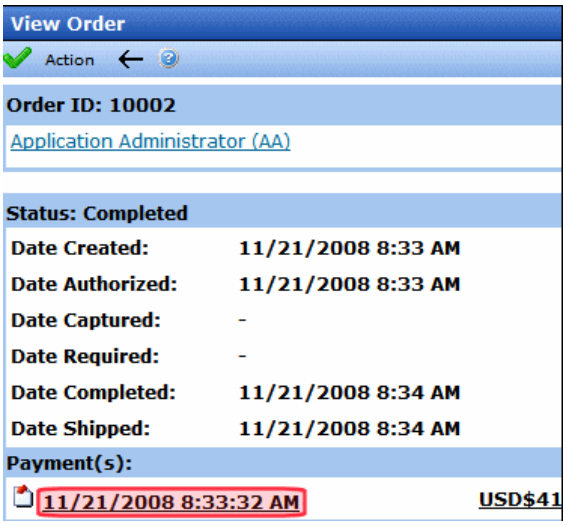
Next, you receive confirmation from the paying agency that the money has been deposited to your account and is available. At this time, you use the Mark as Settled option to inform the eCommerce feature that payment has been received. This changes the order status to Complete.

Capture

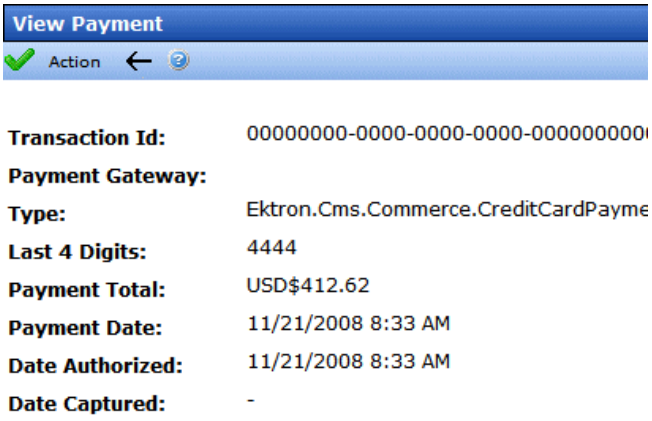
To capture an order, follow these steps.

Note: Items that are not tangible are automatically captured.

1. Navigate to **Workarea** > **Modules** > **Commerce** > **Fulfillment** > **Orders**.
2. Click the order that you want to capture.
3. Click the **Payment** link (see illustration below).



4. The View Payment screen appears, showing information about the transaction captured when the site visitor submitted their payment information.



5. Click **Action** > **Capture**.

   A confirmation prompt appears.

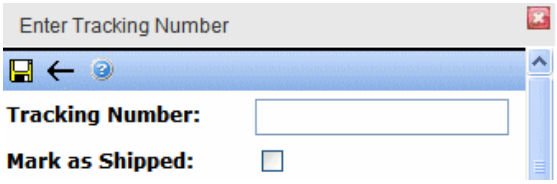
6. Click **OK**.

Cancelling the Order

Click **Action** > **Cancel Order** to stop all processing on the order. If you do, the order's status changes to Cancelled.

Entering a Tracking Number

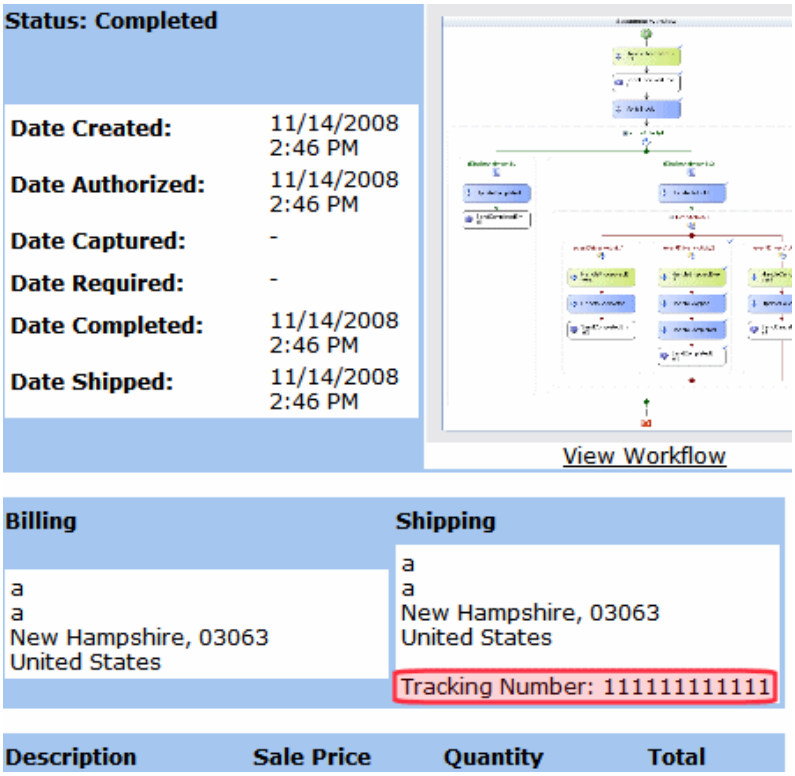
When an order is shipped, the shipping provider typically provides a tracking number to help locate the shipment within the delivery chain. Use the "Enter Tracking Number" option to enter this part of the order information.



After entering the tracking number, you can indicate whether the order has shipped.

Viewing the Tracking Number

You can see an order's tracking number on the View Order screen (under the Shipping Address, as illustrated below).



Marking the Order as Fraud

Use this option if your organization determines that the order is fraudulent. These orders cannot be processed any further.

Marking the Order as Shipped

Note: If an order has not yet been captured, it will be captured as part of the shipping of the order.

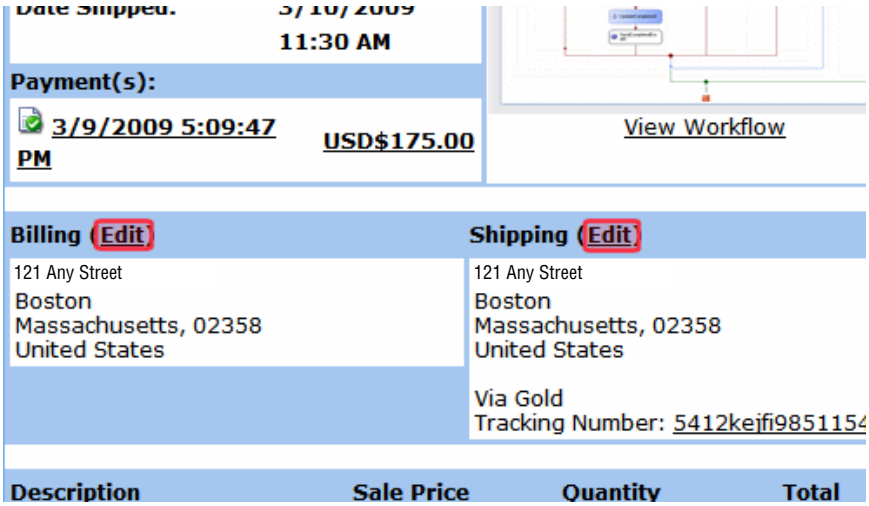
To denote that an order has shipped, follow these steps.

1. Go to **Modules** > **Commerce** > **Fulfillment** > **Orders**.
2. Click the order that is being shipped.
3. Click **Action** > **Ship Order**.

Alternatively, you can mark the order as shipped while entering its tracking number. "Entering a Tracking Number" on page 65.

Editing an Order's Billing and Shipping Addresses

To edit an order's billing or shipping address, click Edit as circled below.



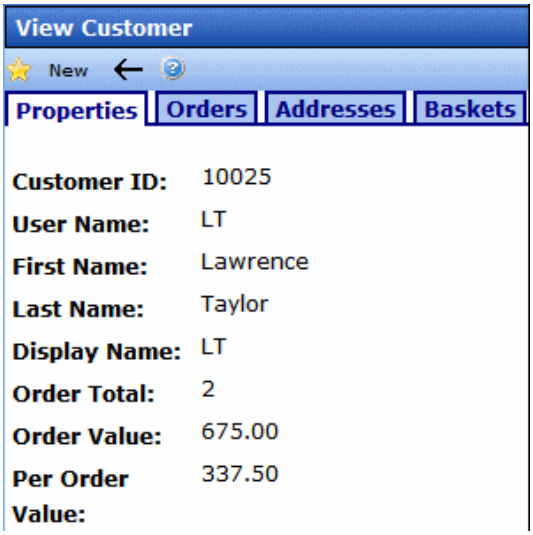
Customer Information

Customers create an account the first time they complete the checkout procedure. The Customers screen provides information about all customers who order from your Web site. The first screen displays the following:

- Customer ID number
- User name
- Number of orders
- Amount of all orders
- Average order amount

You can click any customer's ID number or name to see more details about their order history. The View Customer screen provides the following customer information:

- Summary information (name, number of orders, total of all orders, average order amount)
- Orders (see "Processing Orders" on page 60)
- Addresses;
  - You can edit any address or enter a new one (see "Entering a New Customer Address" on page 68).
  - Whenever a user creates a new order and enters a new or selects an existing billing/shipping address, that becomes the new default.
- Shopping carts (see "Viewing a Customer's Shopping Cart" on page 69).



Entering a New Customer Address

Use this screen to enter a new address for a customer or edit an existing one. It may be used as the default billing address, shipping address, or both.

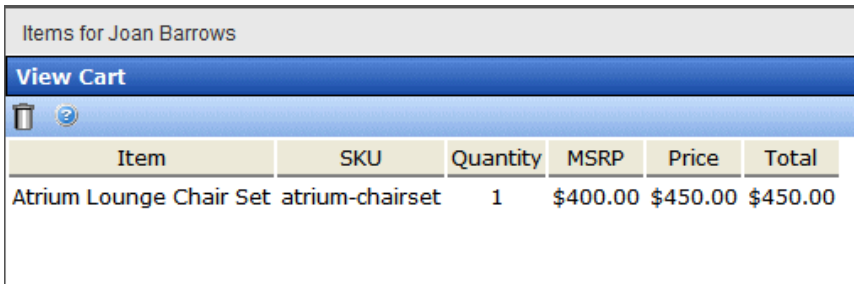
Use the following table to complete the screen.

Field	Description
Name	Enter the customer's name.
Company	If available, enter the name of the customer's company.
Street Address	Enter the customer's street address.
City	Enter the customer's city.
Postal Code	Enter the customer's postal code.
Country	Select the customer's country from the pull-down list.
State/Province	Select the customer's state or province from the pull-down list.
Phone	Enter the customer's phone number. You can add characters or spaces between groupings if you wish. For example: 603-555-0249.
Default Billing	If you want this address to appear by default as the billing address whenever this customer makes a purchase, check this box.
Default Shipping	If you want this address to appear by default as the shipping address whenever this customer makes a purchase, check this box.

Viewing a Customer's Shopping Cart

To view active shopping carts for a customer, follow these steps. After viewing the carts, you can delete any of them.

1. **Modules** > **Commerce** > **Customers**.
2. Click a customer ID number.
3. Click the Baskets tab.
4. Click the ID number of the shopping cart that you want to view.
5. A View Cart screen shows information for items in the cart.



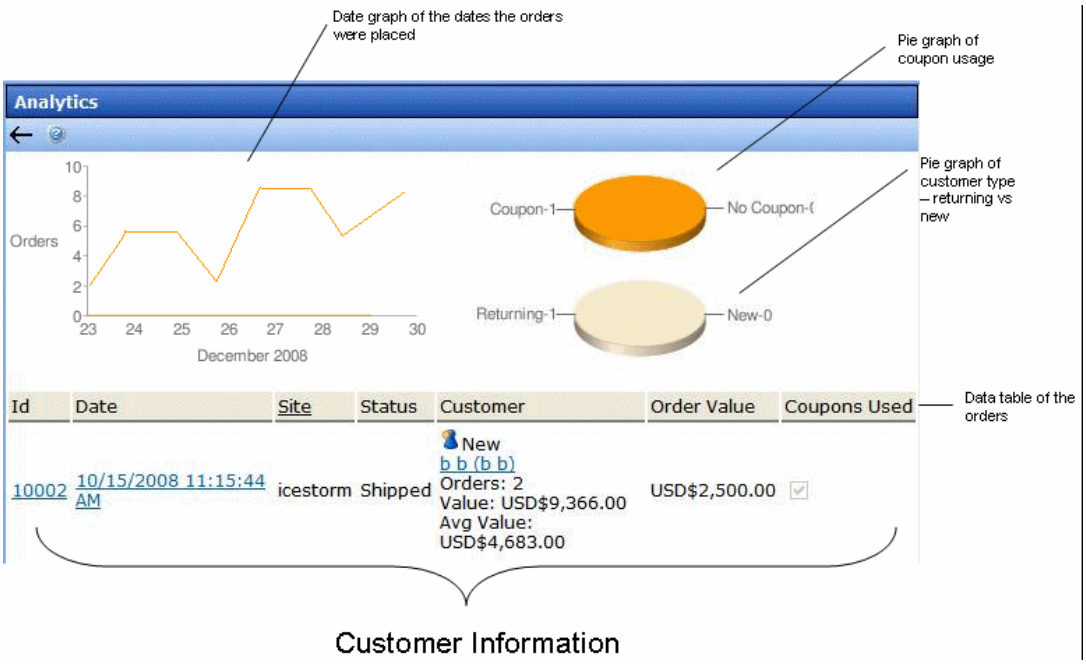
Deleting a Shopping Cart from the Workarea

Follow these steps to delete a shopping cart from the Ektron CMS400.NET Workarea.

1. Access the View Cart screen by following the steps in "Viewing a Customer's Shopping Cart" on page 69.
2. Click the delete button ( ).
3. A confirmation message appears. Click **OK**.

eCommerce Analytics

The eCommerce Analytics screen provides a visual display of order information for a selected catalog entry.



- The top left section graphs the quantity of this catalog entry ordered over the past seven days
- The top right charts the number of orders that used a coupon versus the number that did not
- Below that pie chart is another that shows the number of returning customers versus the number of new ones
- The lower part of the screen shows the following order information
  - order number
  - date/time when order was entered
  - site from which order was entered
  - order status
  - customer information: new or returning, name, number of orders entered, value of all orders entered, average order monetary value
  - order monetary value
  - whether or not a coupon was used

6 Customizing EktronStore.com

Tailoring your store to meet your needs

Customizing your Ektron Store

There are many ways to customize your EktronStore.com site. Here are some examples.

Creating a Workflow

Ektron utilizes Microsoft's Windows Workflow Foundation to implement workflows in CMS400.NET.

Windows Workflow Foundation is a development tool that allows you to automate a business process. Released with Microsoft's .NET 3.0, it allows you to create an activities based-workflow that has the ability to persist over a given length of time or be paused and restarted depending on events. The text below is from Microsoft's MSDN:

> Windows Workflow Foundation (WF), originally introduced as part of the .NET Framework 3.0 with extensions for Visual Studio 2005's designers, has continued to be enhanced for the .NET Framework 3.5 and Visual Studio 2008. WF makes it possible, and for many workflow scenarios, even easy to create robust, manageable workflow-based applications. WF is actually many things: It's a programming model, and runtime engine, and a set of tools that help you create workflow-enabled applications hosted by Windows.

Workflows are comprised of Activities, and each activity represents a portion of your Business Process. Once all activities have finished within a workflow, the workflow terminates. There are two types of activities:

- **Activities** — executed inside the workflow once the activity is reached. For example, an email might be sent to a customer once their order is received by using the `AdvancedEmailActivity` placed after the `OrderReceivedEventActivity`. This email would be automatically sent with no human interaction needed and the workflow would continue on.
- **Event Activities** — cause the workflow to pause until the event has taken place. Once the event takes place, the activity associated with it is executed. Think of events as road blocks in the workflow that don't open unless a matching action happens. Once the event happens, the workflow follows the path associated with that event. For example, an `OrderFraudEventActivity` in a workflow would keep the workflow from going through the Fraud Event portion of the workflow unless the order is marked as fraud.

Ektron supports Sequential Workflows and State Machine Workflows. Sequential Workflows are structured; a step based process where one activity leads to the next. State Machine Workflows typically move from one activity to another when their state has changed.

Only one workflow project can be run at a time. You can have multiple workflows projects associated with your eCommerce site and you can change workflow projects at anytime. However, once an order process is started with a specified workflow, it will continue through that Workflow.

Installing Ektron's Sample Workflow Template

Ektron provides a sample Workflow template which allows you to create new workflows based on the default workflow included with Ektron's Developer Starter Site.

If you are using Visual Studio 2005 to work with Microsoft Windows Workflow Foundation, install Microsoft's Visual Studio 2005 Extensions for Windows Workflow Foundation (http://www.microsoft.com/downloads/details.aspx?FamilyId=5D61409E-1FA3-48CF-8023-E8F38E709BA6&displaylang=en).

Before you install the sample workflow template, you must install Ektron's CMS400.NET SDK. See the *Ektron CMS400.NET Setup Manual* for more information. Note that this release only supports the Visual C# development environment.

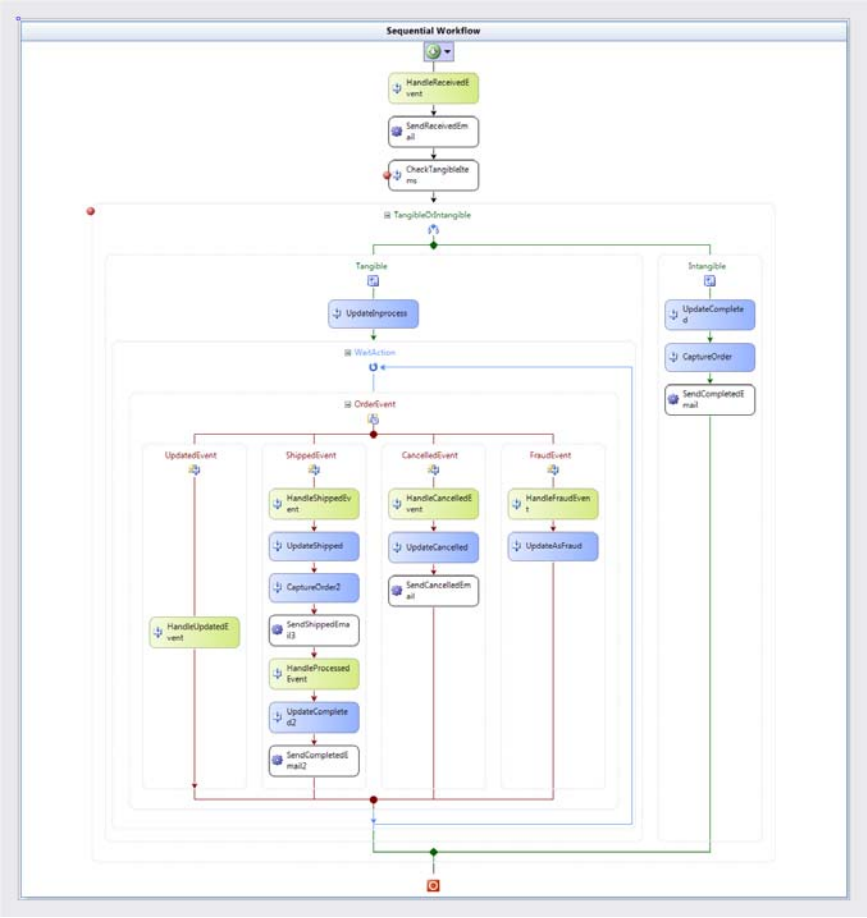
Copy the "Ektron Ordering Sequential Flow.zip" file from `C:\Program Files\Ektron\CMS400SDK\Commerce\Workflow\Templates\VS2008\` and paste it to `C:\Documents and Settings\~user name~\ My Documents\Visual Studio 2008\Templates\ProjectTemplates\ Visual C#\Workflow.`

If you are using Visual Studio 2005, replace *2008* in the above paths with *2005*.

Working with the Sample Workflow

Follow the steps below to open and work with Ektron's sample workflow.

1. Open Visual Studio.
2. Click **File** > **New** > **Project**.
3. If you are using C#, expand the Visual C# project type. Otherwise, expand the Visual Basic project type.
4. Click **Workflow**.
5. Select **Ektron Ordering Sequential Flow** from the My Templates area.
6. Select a new Name, and if necessary, change the Location, Solution and Solution Name.
7. Click **OK**.
8. Copy Ektron.Cms.Common.dll and Ektron.Workflow.dll from the Bin folder of your Ektron site to the Bin folder for newly created project. For example, copy the files to `C:\Documents and Settings\~user name~\My Documents\Visual Studio 2008\Projects\MyWorkFlow\MyWorkFlow\bin.`
9. Open the Workflow.cs file in the Solution Explorer.

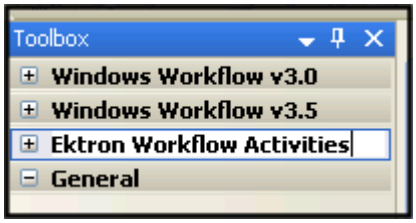


10. Make any necessary changes to the workflow.
11. Debug as needed.
12. Build the workflow. This creates a DLL of the workflow and places it in the project's bin/debug folder.
13. Navigate to the folder where the DLL is stored. For example, `C:\Documents and Settings\~user name~\My Documents\Visual Studio 2008\Projects\MyWorkFlow\MyWorkFlow\bin\Debug.`
14. Move or copy the newly created Workflow DLL file to your Web site's bin folder.

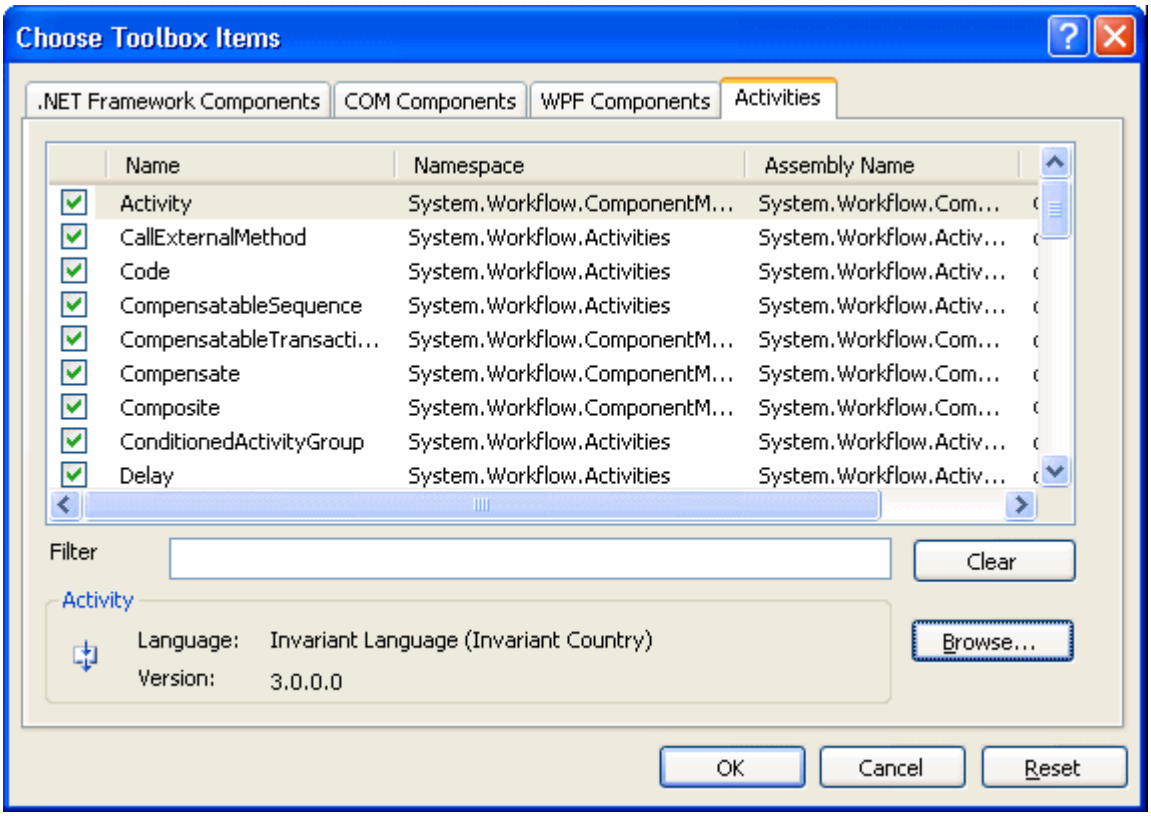
Making Ektron's Workflow Activities Available

To add Ektron's Workflow Activities to your Toolbox, follow these steps.

1. Open or create a Workflow in Visual Studio 2005/2008.
2. Display the Toolbox (**View** > **Toolbox**).
3. Right click the mouse within the Toolbox.
4. Click **Add** tab.
5. Type **Ektron Workflow Activities** then press enter.



6. Click the **Ektron Workflow Activities** tab.
7. Right click the mouse in the empty area.
8. Click **Choose Items**.
9. The **Choose Toolbox Items** dialog appears.



10. Select the **Activities** tab.
11. Browse to the directory that stores Ektron CMS400.NET's DLL files, `[Web root]\[Site Root]\bin`, and add the Ektron.Workfloww.dll file. This file provides access to Ektron CMS400.NET's workflow activities.

    Alternatively, you could use the following location, `C:\Program Files\Ektron\CMS400v7x\bin`. The file is identical in both places.

    Using the bin folder in your site provides better speed. However, if you use the bin folder located in Program Files, you do not have to worry about deleting the .DLL file if you change or delete your site.

12. Click **OK**.

For easier viewing once the workflow activities are installed, you can right click on them and select Sort Items Alphabetically.

Ektron's workflow activities appear only when the workflow is opened in design mode.

Removing Ektron's Workflow Activities

1. Display the Visual Studio toolbox (**View** > **Toolbox**).
2. Right click the mouse within the Toolbox.
3. Click **Choose** Items.
4. Click the **Activities** tab.
5. Click **Namespace** or **Assembly Name** to sort the **Workflow Activities** by manufacturer.
6. Clear all boxes that are `Ektron.Workflow.Activities`.
7. Click **OK**.
8. Right click the **Ektron Workflow Activities** tab.
9. Click **Delete** tab to remove it.

Updating Ektron's Workflow Activities

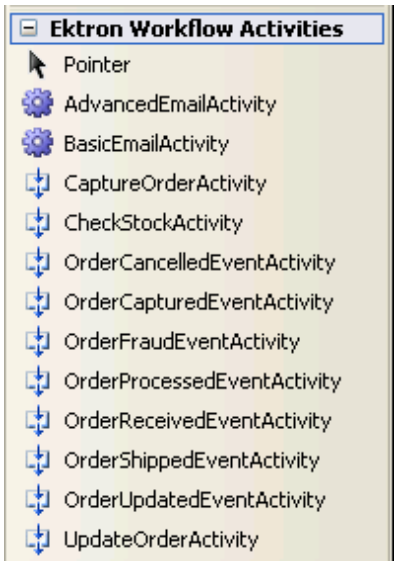
To update the workflow activities, you must first remove the existing ones in Visual Studio, then add the new workflow activities.

Inserting Workflow Activities Using-Drag-and-Drop

Because Visual Studio is a visual environment, you can see the workflow change as you add or remove activities.

Adding an Activity to a Workflow

1. Display the Visual Studio toolbox (**View** > **Toolbox**).
2. Click the **Ektron Workflow Activities** tab. Ektron's Workflow Activities appear.



3. Drag an activity and drop it into the desired location in the Workflow.
4. As desired, modify the activity's properties using the Properties area of Visual Studio.

List of Workflow Activities

You can customize your workflow using the following activities.

Activity	Description
AdvancedEmailActivity	Send an email based on the order ID and a specified message type.
BasicEmailActivity	Send a generic email message when this activity is reached in the workflow.
CaptureOrderActivity	Submits order information to a payment gateway.
CheckStockActivity	Verifies whether the item is an in-stock item or not.
OrderCancelledEventActivity	Customize the handling of an order when it's cancel.
OrderCapturedEventActivity	Customize the handling of an order when it's captured.
OrderFraudEventActivity	Customize the handling of an order when it's marked as fraud.
OrderProcessedEventActivity	Customize the handling of an order when it's being processed.
OrderReceivedEventActivity	Customize the handling of an order when it's marked as received.
OrderShippedEventActivity	Customize the handling of an order when it's marked as shipped.
OrderUpdatedEventActivity	Customize the handling of an order when it's updated.
UpdateOrderActivity	Updates the order when once this activity is reached.

Customizing Shipping

Shipping Calculator

The shipping calculator has two events that customers can use to extend the shipping calculation logic:

```
void OnBeforeCalculate(BasketCalculatorData basketData, ShippingMethodData shippingMethod, CancellableEventArgs eventArgs)
```

This method can be used to perform processing before the `ShippingCalculator` executes. Using the supplied `BasketCalculatorData` you can set shipping rates directly. If `eventArgs.IsCancelled` is set to `true`, the `ShippingCalculatorexecution` does not continue and the settings you specify are used.

```
void OnAfterCalculate(BasketCalculatorData basketData, CmsEventArgs eventArgs)
```

This method is similar to `OnBeforeCalculate` except it occurs after the `ShippingCalculator` has already executed. You can view and change the shipping rates that the `ShippingCalculator` calculated directly on the `BasketCalculatorData` object.

IShippingCalculator

If you wish to have complete control over the Shipping Calculation process, you can implement your own `IShippingCalculator`.

1. Create a new Class Library Project in Visual Studio.
2. Create a class that implements `IShippingCalculator`.

a. `IShippingCalculator` has one method:

```
void Calculate(BasketCalculatorData basketData)
```

3. Compile your DLL and drop it into the site's bin\ directory.
4. Open the site's ObjectFactory.Config and add the following entry:

```
<objectImplementations>
<add name="IShippingCalculator"type="Ektron.Cms.Commerce.IShippingCalculator, Ektron.Cms.ObjectFactory"
implementation="ComanyABC.MyShippingCalculator, ComanyAbc.dll"/>
</objectImplementations>
```

`Calculate` should look at the basket items in `BasketCalcualtorData`, calculate the shipping costs for each option, and add the `shippingOption` to `BasketCalcualtorData.ShippingRates`. It might be easier to extend the existing `Ektron.Cms.Commerce.ShippingCalculator` and overwrite `Calculate` instead of creating a new shipping calculator completely from scratch. This gives you access to many helper functions used by the existing shipping calculator.

Creating a Custom CMS400.NET Shipping Provider

The Ektron CMS400.NET shipping provider allows developers to obtain shipping information from any location. These options can be real-time rates from a service such as UPS or FedEx, flat rates, or any other type of logic deemed necessary. This article details how to create a custom shipping provider for CMS400.NET.

Creating the Provider Code

First, create a class library project in Visual Studio and import the namespaces you need. Add references to:

- `Ektron.Cms.Commerce`
- `Ektron.Cms.Common`
- `Ektron.Cms.ObjectFactory`
- `Ektron.Cms.Instrumentation`
- `System.Configuration`

Next add the following statements:

```
using System.Configuration.Provider;
using Ektron.Cms.Commerce.Shipment.Provider;
using Ektron.Cms.Commerce;
using Ektron.Cms.Instrumentation;
```

Change the namespace to `Ektron.Cms.Extensibility.Commerce.Samples`, rename your class to `CustomShipmentProvider`, and inherit from the `Ektron.Cms.Commerce.Shipment.Provider.ShipmentProvider` class.

```
namespace Ektron.Cms.Extensibility.Commerce.Samples
    {
    public class CustomShipmentProvider : ShipmentProvider
```

Next, add in the following constructor, private variables, and properties. The items added to the `shippingOptionList` will be

exposed as service types in the management page within the workarea:

```
#region constructor, member variables
public CustomShipmentProvider()
{
    IsTrackingSupported = false;
    _shippingOptionList = new List<string>();
    _shippingOptionList.Add("CustomOption_1");
    _shippingOptionList.Add("CustomOption_2");
}
private List<string> _shippingOptionList;
#endregion
```

Now, add the methods required by the `ShipmentProvider` base class. The `Initialize` method reads configuration information, `GetServiceTypes` returns a `List<string>` of the shipping options available, and `GetTrackingUrl` exposes tracking support. Leave `GetRates` blank (this is explained later).

```
#region ShipmentProvider Implementation
public override void Initialize(string name, System.Collections.Specialized.NameValueCollection config)
{
    if (config == null)
    throw new ArgumentNullException("config");
    // Assign the provider a default name if it doesn't have one
    if (String.IsNullOrEmpty(name))
    name = "CustomShipmentProvider";
    if (string.IsNullOrEmpty(config["description"]))
    config.Remove("description");
    config.Add("description", "CustomShipmentProvider Provider");
    // Call the base class's Initialize method
    base.Initialize(name, config);
    // Throw an exception if unrecognized attributes remain
    if (config.Count == 0)
    throw new ProviderException("Shipment provider attribute missing.");
    else
    //read all config attributes.
    ServiceUrl = config["serviceUrl"];
    Key = config["key"];
    Password = config["password"];
    AccountNumber = config["accountNumber"];
    MeterNumber = config["meterNumber"];
    TransactionId = config["transactionId"];
    }
}
public override List<string> GetServiceTypes()
{
    return _shippingOptionList;
}
public override string GetTrackingUrl(string trackingId)
{
    return "";
}
public override List<ShippingOptionData> GetRates(IEnumerable<ShippingMethodData> expectedOptions, AddressData origin,
AddressData destination, Weight weight, Dimensions dimensions)
{
}
#endregion
```

Now, implement the `GetRates` method. This method returns rates for the expected shipping when called. In this example:

```
public override List<ShippingOptionData> GetRates(IEnumerable<ShippingMethodData> expectedOptions, AddressData origin,
AddressData destination, Weight weight, Dimensions dimensions)
{
    List<ShippingOptionData> availableOptions = new List<ShippingOptionData>();
    try
    {
    foreach (ShippingMethodData expectedOption in expectedOptions)
    {
    Log.WriteInfo("Custom Shipping Provider.ExpectedOption:" + expectedOption.Name);
    switch (expectedOption.ProviderService.ToLower())
```

```
    {
    case "customoption_1" :
    ShippingOptionData customOption1 = new ShippingOptionData();
    customOption1.Id = expectedOption.Id;
    customOption1.Name = expectedOption.Name;
    customOption1.ShippingFee = 25.00M;
    customOption1.ProviderService = "CustomOption_1";
    availableOptions.Add(customOption1);
    break;
    case "customoption_2" :
    ShippingOptionData customOption2 = new ShippingOptionData();
    customOption2.Id = expectedOption.Id;
    customOption2.Name = expectedOption.Name;
    customOption2.ShippingFee = 50.00M;
    customOption2.ProviderService = "CustomOption_2";
    availableOptions.Add(customOption2);
    break;
    }
    }
    }
    catch (Exception e)
    {
    Log.WriteError("Custom Shipping Provider: Error retrieving shipping rates." + e.Message);
    throw;
    }
    return availableOptions;
}
```

In this example, you are using flat rates for your shipping options. In a real-world scenario, the shipping options can be obtained through FedEx, UPS, a third-party application, or a Web service.

If you want to disallow shipping to all countries except the U.S.A. and Canada, add these lines to your shipping provider's `GetRates` method:

```
if (destination.Country.Id != 124)
    throw new Ektron.Cms.Commerce.Exceptions.Shipping.InvalidAddressException("We only ship to Canada.");
```

Now, build the project and copy the assembly into the CMS400.NET site's \bin directory.

Registering the Provider

When the assembly is created and placed inside the CMS400.NET site's \bin directory, you need to register the provider and instruct the CMS to use it. The shipment.config file provides the means to manage shipment providers within the CMS. Locate the `shipmentProvider` section in the configuration file. Add a reference to the class you created earlier in the `<providers>` key, then change the `defaultProvider` attribute, as shown below:

```
<shipmentProvider defaultProvider="CustomShipmentProvider">
    <providers>
    <add name="CustomShipmentProvider" type="Ektron.Cms.Extensibility.Commerce.Samples.CustomShipmentProvider,
    CustomShipmentProvider" serviceUrl="" key="" password="" accountNumber="" meterNumber=""
    transactionId="CustomShipmentProvider based transaction" />
    </providers>
</inventoryProvider>
```

When the shipment manager is queried, the calls route through the custom shipment provider.

Adding the Options

The final step is to add the service types so they display as valid shipping options. This is done in Workarea > Modules > Commerce > Shipping > Methods.

1. Select New > Shipping Method.

   The name is what a shopper sees during the checkout process. For example, “Ground” or “Next Day”. For the purposes of this example, use “Custom1”.

2. Select the Active checkbox.
3. View the Provider Service Options, and select “CustomOption_1” from the list.
4. Save.
5. Repeat this process for the CustomOption2 service type.

The options are now available in the checkout process, and are displayed on the shipping method selection page.

Customizing Events

Hooking a CMS400.NET Commerce Event

Ektron CMS400.NET provides events that developers can access. These events run in process with the CMS, and because of this developers have access to the HTTPContext, among other items. In this example, you will be accessing the `Reorder` event to send an email to the administrator.

This example is also included in the SDK to help get you started.

Creating the Event Code

First, create a class library project in Visual Studio and import the namespaces you need. Add references to:

- `Ektron.Cms.Api`
- `Ektron.Cms.Commerce`
- `Ektron.Cms.Common`
- `Ektron.Cms.Instrumentation`
- `Ektron.Cms.ObjectFactory`

Next add the following using statements:

```
using Ektron.Cms.Instrumentation;
using Ektron.Cms.Extensibility;
using Ektron.Cms.Extensibility.Commerce;
using Ektron.Cms.Commerce;
using Ektron.Cms.Common;
```

Change the namespace to `Ektron.Cms.Extensibility.Commerce.Samples`, rename your class to `InventoryReorderLevelStrategy`, and inherit from the `InventoryStrategy` class.

```
namespace Ektron.Cms.Extensibility.Commerce.Samples
{
    public class InventoryReorderLevelStrategy : InventoryStrategy
```

Next, add the following constants, which will be used in the email message:

```
private const string Subject = "Inventory Reorder Level Reached";
private const string Body = @"Inventory reorder level for item has been exceeded.
    Item Id:          {0}
    Item SKU:         {1}
    Reorder Level:    {2}
    Current Inventory {3}";
```

Note that you can also use the messaging APIs of the CMS for the contents of the email.

Now, override the `OnInventoryReorderLevelReached` method, which is called when the inventory in stock falls below the `reorder` level for the product.

```
public override void OnInventoryReorderLevelReached
(InventoryData inventoryData, CmsEventArgs eventArgs)
{
}
```

In this method, you want to add code so that an email is sent to the administrator. In order to accomplish this, you are leveraging some core services of the CMS, such as the Instrumentation logging and the mail services.

```
public override void OnInventoryReorderLevelReached
(InventoryData inventoryData, CmsEventArgs eventArgs)
{
    try
    {
    CatalogEntryApi entryApi = new CatalogEntryApi();
    EntryData entry = entryApi.GetItem(inventoryData.EntryId);
    Log.WriteInfo(string.Format("Inventory Reorder Level Reached: Item Id {0}, SKU {1}, stock level {2}", inventoryData.EntryId,
    entry.Sku, inventoryData.UnitsInStock));
    string emailAddress = GetCommerceEmailAddress();
    if (emailAddress.Length > 0)
    {
    EkMailService mail = new EkMailService
    (eventArgs.RequestInformation);
    mail.MailFrom = emailAddress;
    mail.MailTo = emailAddress;
    mail.MailBodyText = string.Format(Body, entry.Id, entry.Sku, inventoryData.ReorderLevel, inventoryData.UnitsInStock);
    mail.MailSubject = Subject;
    mail.SendMail();
    }
    }
    catch (Exception e)
    {
    Log.WriteError("Commerce.Inventory.ReorderLevel Event. Error occurred. \r\n" + e.Message);
    }
}
```

A real world scenario for this would be to contact a web service for a supplier so that you can reorder additional products.

Finally, add the private function `GetCommerceEmailAddress`, which reads the a key inside the web.config file for the email address to send from (and to).

```
private string GetCommerceEmailAddress()
{
    NameValueCollection commerceConfig = ConfigurationManager.GetSection("ektronCommerce") as NameValueCollection;
    if (commerceConfig != null && commerceConfig["adminEmail"] != null)
    {
    return commerceConfig["adminEmail"];
    }
    return string.Empty;
}
```

Now, build the project and copy the assembly into the CMS400.NET site's \bin directory.

Registering the Event with the System

When the event assembly is created and placed inside the CMS400.NET site's \bin directory, you need to register the event. The objectfactory.config file provides the means to register custom events within the CMS. Locate or add a key `noninflammatory`" in the `objectStrategies` section of the configuration file. Add a reference to the class you created earlier in the `<strategies>` key, as shown below:

```
<objectStrategies>
    <add name="Inventory">
    <strategies>
    <add name="InventoryReorderLevel" type="Ektron.Cms.Extensibility.Commerce.Samples.InventoryReorderLevelStrategy,
    Commerce.Inventory.ReorderLevel"/>
    </strategies>
    </add>
</objectStrategies>
```

When the reorder level is reached, the event executes, triggering the method in our custom class.

There is a large number of Commerce events that you can hook. See the eCommerce developer documentation for more information.

Customizing Inventory

An Inventory Provider is a pluggable component that is integrated into Ektron's CMS400.NET eCommerce module. An Inventory Provider manages inventory information for products within the CMS. Out-of-the-Box, the CMS comes with an Ektron Inventory Provider that tracks inventory within the CMS database.

If your business already has an Accounting or Enterprise Resource Management solution that manages inventory, you can create a custom Inventory Provider that retrieves and updates inventory information directly from your existing system.

Implementing Your Own Inventory Provider

The image below displays the Inventory Object Model.

InventoryProvider

The `InventoryProvider` is the abstract base class you must extend to implement your own Inventory Provider.

The following table shows the `InventoryData` data class. This class represents inventory information that is either provided to you for an update or that you are required to return.

Property	Description
`long EntryId`	The ID of entry that this inventory data represents.
`int UnitsInStock`	The number of units currently in stock for this product.
`int UnitsOnOrder`	The number of units currently on order for this product. This is the number of units that the store has ordered to replenish inventory and NOT the number of units that are currently on order by customers.
`int ReorderLevel`	The inventory level at which stock should be reordered.

The following describes the methods that must be implemented for an Inventory Provider.

Property	Description
`InventoryData GetInventory(long entryId)`	Returns inventory data for the supplied catalog entry.
`void SaveInventory (InventoryData inventory)`	Saves inventory information for a catalog entry.

The following describes the methods that can be overridden if desired.

Property	Description
`int GetUnitsInStock (long entryId)`	Returns the number of units in stock for a given catalog entry. The default implementation calls the `GetInventory` method.
`int GetUnitsOnOrder (long entryId)`	Returns the number of units on order for a given catalog entry. The default implementation calls the `GetInventory` method.
`int GetReorderLevel (long entryId)`	Returns the reorder level for a given catalog entry. The reorder level is the inventory level at which new stock should be ordered. The default implementation calls the `GetInventory` method.
`void IncreaseStockLevel (long entryId, int quantity)`	Increases the stock level for a catalog entry. The default implementation calls the `SaveInventory` method.
`void DecreaseStockLevel (long entryId, int quantity)`	Decreases the stock level for a catalog entry. The default implementation calls the `SaveInventory` method.

Creating a Custom CMS400.NET Inventory Provider

The Ektron CMS400.NET inventory provider allows developers to use existing inventory systems with the CMS. You can store inventory data in any number of locations, including databases, ERP or CRM systems, or XML files. The following information describes how to create a custom inventory provider for CMS400.NET.

Creating the Provider Code

First, create a class library project in Visual Studio and import the namespaces you need. Add references to:

- `Ektron.Cms.Commerce`
- `Ektron.Cms.Common`
- `Ektron.Cms.ObjectFactory`
- `System.Configuration`

Next add the following using statements:

```
using Ektron.Cms.Commerce.Inventory.Provider;
using Ektron.Cms.Commerce.Data;
using Ektron.Cms.Commerce;
using Ektron.Cms.Common;
using Ektron.Cms.Extensibility;
using Ektron.Cms.Extensibility.Commerce;
```

Change the namespace to `Ektron.Cms.Extensibility.Commerce.Samples`, rename your class to `CustomInventoryProvider`, and inherit from the `InventoryProvider` class.

```
namespace Ektron.Cms.Extensibility.Commerce.Samples
```

{

```
    public class CustomInventoryProvider : InventoryProvider
```

Next, add the following constructor, private variables, and properties:

```
public CustomInventoryProvider() { }
private CmsInventory _inventory;
protected CmsInventory Inventory
```

```
{
    get
    {
    if (_inventory == null)
    {
    _inventory = new CmsInventory(RequestInformation);
    }
    return _inventory;
    }
}
```

Now, you must override the `SaveInventory` and `GetInventory` methods, which are called when the inventory system is queried.

```
public override InventoryData GetInventory(long entryId)
{
}
public override void SaveInventory(InventoryData inventory)
{
}
```

The method `GetInventory` returns an `InventoryData` class for a given product ID. `SaveInventory` persists updates to the stock levels inside the inventory system.

```
public override InventoryData GetInventory(long entryId)
{
    return Inventory.GetInventory(entryId);
}
public override void SaveInventory(InventoryData inventory)
{
    OnBeforeInventorySave(inventory);
    Inventory.SaveInventory(inventory);
    OnAfterInventorySave(inventory);
    if (inventory.UnitsInStock < inventory.ReorderLevel)
    {
    OnInventoryReorderLevelReached(inventory);
    }
}
```

Note that in this example, you execute CMS events. In a real-world scenario, the inventory system might already be using inventory events.

Now, build the project and copy the assembly into the CMS400.NET site's \bin directory.

Registering and Using the Provider

When the assembly is created and placed inside the CMS400.NET site's \bin directory, you we need to register the provider and instruct CMS400.NET to use it. The web.config file provides the means to manage inventory providers within the CMS. Locate the `inventoryProvider` section in the configuration file. Add a reference to the class you created earlier in the `<providers>` key, then change the `defaultProvider` attribute, as shown below:

```
<inventoryProvider defaultProvider="CustomInventoryProvider">
    <providers>
    <add name="EktronInventoryProvider" type="Ektron.Cms.Commerce.Providers.Inventory.EktronInventoryProvider,
    Ektron.Cms.Commerce.Providers"/>
    <add name="CustomInventoryProvider" type="Ektron.Cms.Extensibility.Commerce.Samples.CustomInventoryProvider,
    CustomInventoryProvider"/>
    </providers>
</inventoryProvider>
```

When the inventory is queried, the calls will now route through the custom inventory provider.

Customizing the Payment Gateway

A Payment Gateway provider is a pluggable component that is integrated into the Ektron CMS 400.NET eCommerce module. The Payment provider handles eCommerce customer payments by utilizing third party payment gateways. The Ektron CMS 400.NET

eCommerce module accepts payments such as credit cards and then passes the payment information to a third-party service. The third party service processes the payment, and returns a transaction ID that is stored with the customer's order. Ektron CMS 400.NET eCommerce users need to setup an account with a third-party payment service before using the payment provider.

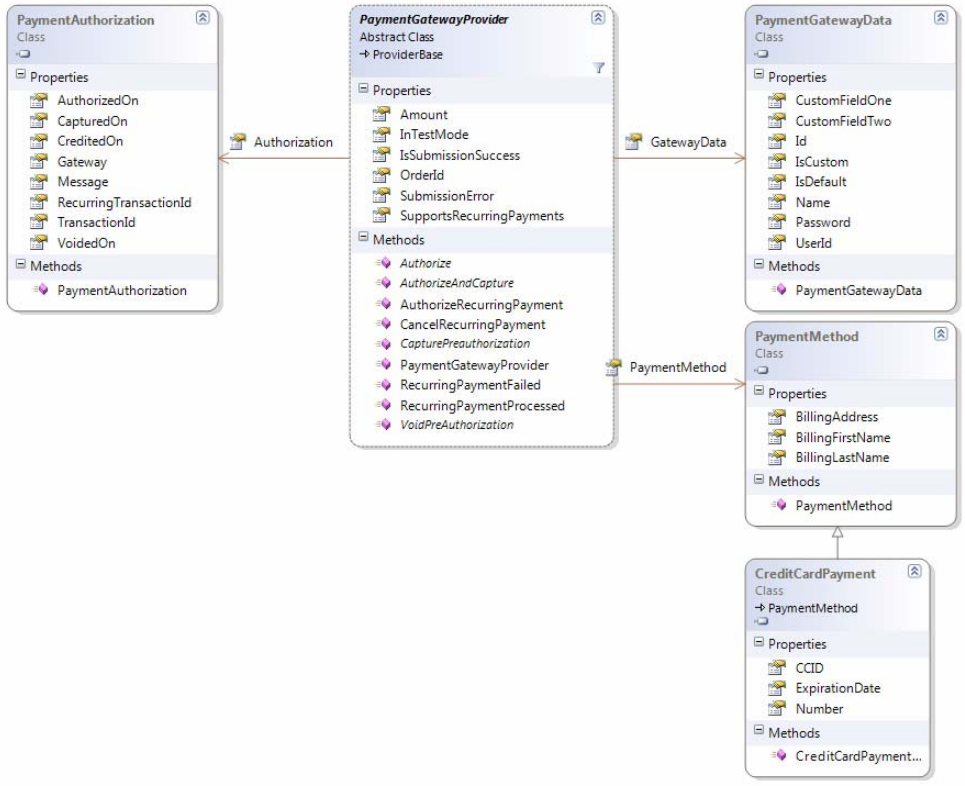
CMS 400.NET comes with several payment providers out-of-the-box, including Authorize.NET and PayFlow. You can customize these out-of-the-box providers or create your own using the extendable Payment Gateway Provider architecture.

Each type of payment gateway provider accepts configuration parameters. For example, Authorize.NET requires a username and password while PayFlow requires a username, password, vendor, and partner. In addition, some gateways might support recurring payments, while others do not. Recurring payments provides the ability to create a payment that recurs at a given interval for a specified period of time. For example, you could create a subscription-based payment of $9.99 that occurs on the first of every month for the next 12 months.

The information in this section explains how you can extend the object model to build your own customized Payment Gateway Provider.

Implementing Your Own Payment Gateway Provider

The image below displays the Payment Gateway Object Model.



PaymentGatewayProvider Class

The `PaymentGatewayProvider` is the abstract base class you must extend to implement your own payment gateway. Details on the class itself, the properties available to you, and the methods you are required to implement are described below. The following

represents the properties that are provided to you.

Property	Description
`long OrderId`	The ID of the order associated with this payment submittal.
`PaymentMethod PaymentMethod`	The payment method for this payment submittal. Often, this would be the credit card information.
`decimal Amount`	The payment amount being submitted.
`PaymentGatewayData GatewayData`	The data associated with this payment gateway provider including the user ID and password used for the gateway.
`bool InTestMode`	Indicates whether or not the application is configured for testing. Many third-party gateways have different modes for testing.

The following represents the properties you are expected to populate when processing payments.

Property	Description
`bool IsSubmissionSuccess`	This must be set to true if authorization was successful
`string SubmissionError`	You should set this property to report any errors that occurred during submission.
`PaymentAuthorization Authorization`	Payment submission results should be set on the Authorization including transaction ID, authorization, captured dates, and so on.
`bool SupportsRecurringPayments`	Return true if your provider supports recurring billing.

The following describes the methods that must be implemented for a payment gateway provider.

Property	Description
`virtual void Initialize (string name, NameValueCollection config);`	The `initialize` method comes from the `ProviderBase` base class. This method supplies all the name value pairs that are specified in the `Providers` web.config section. If you have custom settings that need to be provided to your gateway, you can add them to your provider's web.config key and they will be provided to you at runtime in the `Initialize` method. If you do not have custom settings, you do not need to override.
`abstract string AuthorizeAndCapture();`	This method should authorize and capture a payment. "Capture" is a term meaning that the actual payment is captured at that moment. In the case of credit cards, the credit has now been charged. It should return the transaction ID of the payment.
`abstract string Authorize();`	This method should authorize a payment that will be captured at a later time. It should return the transaction ID of the payment. This ID will be later used to capture the actual payment.

Property	Description
`abstract string CapturePreauthorization (string transactionId)`	This method should capture a previously authorized payment. It should return the transaction ID of the payment. This might be the same ID as the authorization.
`abstract string VoidPreAuthorization (string transactionId)`	This method should void a previously authorized transaction. It should return the transaction ID of the void.

The following methods only need to be implemented if the gateway is to support recurring payments.

Property	Description
`virtual string AuthorizeRecurringPayment(RecurrenceData recurrence)`	This method should authorize a recurring payment. The `recurrenceData` parameter represents the occurrence pattern. The method should return the ID of the recurring transaction.
`virtual void CancelRecurringPayment (string recurringTransactionId)`	This method should cancel a recurring payment. The `recurringTransdactionId` is the previously returned ID of the recurring transaction.

The following methods are related to the processing of recurring payments. For example, PayFlow has the concept of Instant Payment Notifications (IPN). These are notifications that PayFlow sends out when a recurring payment has been processed. The CMS collects these notifications and then calls certain methods on the gateway to notify it that the payment has occurred. The base `PaymentGatewayProvider` already implements these methods and records the payment occurrences. If you override these methods for custom handling, you should still call the base methods to be certain that the CMS is notified of the payment.

Property	Description
`virtual void RecurringPaymentProcessed (string recurringTransactionId, string paymentTransactionId)`	This method is called when the CMS is notified of a recurring payment being processed. The `recurringTransactionId` parameter is the ID of the recurring subscription. The `paymentTransactionId` is the ID for this particular payment occurrence.
`virtual void RecurringPaymentFailed (string recurringTransactionId)`	This method is called when the CMS is notified of a recurring payment that failed. The `recurringTransactionId` parameter is the ID of the recurring subscription.

Web.config settings

Note: If you change `defaultProvider` from `Automatic` to another provider, it overrides any Workarea settings.

After creating your new provider, you need to register it in the web.config so the CMS can use it. You do this in the `EktronPaymentGateway` section of the web.config file. You need to provide a name for it and the fully-qualified URL. You can also

provide other key-value pairs that would be passed to your Provider's `Initialize` method.

```
<EktronPaymentGateway defaultProvider="Automatic">
   <providers>
      <add name="Manual" type="Ektron.Cms.Commerce.ManualPayment" />
      <add name="PayFlow" type="Ektron.Cms.Commerce.PayFlowPayment" />
      <add name="AuthorizeNet" type="Ektron.Cms.Commerce.AuthorizeNetPayment" />
   </providers>
</EktronPaymentGateway>
```

Creating a Custom CMS400.NET Payment Gateway

The Ektron CMS400.NET payment gateway allows developers to create custom payment processors for CMS400.NET. These payment processors can interface with payment gateways to enable support for gateways that Ektron does not support out-of- the-box. This section describes how to create a custom payment gateway for CMS400.NET.

Creating the Provider Code

First, create a class library project in Visual Studio and import the namespaces you need. Add references to:

- `Ektron.Cms.Commerce`
- `Ektron.Cms.Common`
- `Ektron.Cms.ObjectFactory`
- `Ektron.Cms.Instrumentation`
- `System.Configuration`

Then add the following using statements:

```
using System;
using System.Collections;
using System.Collections.Specialized;
using System.Configuration.Provider;
using Ektron.Cms;
using Ektron.Cms.Common;
using Ektron.Cms.Commerce;
```

Change the namespace to `Ektron.Cms.Extensibility.Commerce.Samples`, rename your class to `CustomPaymentProvider`, and inherit from the `Ektron.Cms.Commerce.PaymentGatewayProvider` class.

```
namespace Ektron.Cms.Extensibility.Commerce.Samples
{
    public class CustomPaymentProvider : Ektron.Cms.Commerce.PaymentGatewayProvider
```

Next, add the following constructor.

```
#region constructor, member variables
public CustomPaymentProvider() { }
#endregion
```

Now, you must add the methods required by the `PaymentGatewayProvider` base class. The `Initialize` method reads configuration information, and the other methods are related to the payments. Leave them blank. They will be explained later.

```
#region PaymentProvider Implementation
public override void Initialize(string name, System.Collections.Specialized.NameValueCollection config)
{
    if (config == null)
    throw new ArgumentNullException("config");
    // Assign the provider a default name if it doesn't have one
    if (String.IsNullOrEmpty(name))
```

```
    name = "CustomPaymentProvider";
    if (string.IsNullOrEmpty(config["description"]))
    {
    config.Remove("description");
    config.Add("description", "CustomPaymentProvider");
    }
    // Call the base class's Initialize method
    base.Initialize(name, config);
    // Throw an exception if unrecognized attributes remain
    if (config.Count > 0)
    {
    foreach (string key in config.AllKeys)
    {
    EkException.WriteToEventLog("Unrecognized Payment Gateway Provider attribute: " + key,
    System.Diagnostics.EventLogEntryType.Warning);
    }
    }
}
public override string Authorize()
{
}
public override string AuthorizeAndCapture()
{
}
public override string CapturePreauthorization(string transactionId)
{
}
public override string VoidPreAuthorization(string transactionId)
{
}
#endregion
```

You then implement the `Authorize` method. This method is called when an order is submitted to authorize a certain amount of money. In this example, you check to see if the payment is a credit card, then check whether it is expired. If no, you authorize the transaction by setting `IsSubmissionSuccess` to `true`, and return a `GUID` as the transaction ID.

```
public override string Authorize()
    {
    if (PaymentMethod.GetType() != typeof(CreditCardPayment))
    throw new Ektron.Cms.Commerce.Exceptions.AuthorizationException("Invalid Payment Type");
    CreditCardPayment creditCard = (CreditCardPayment)this.PaymentMethod;
    if (creditCard.ExpirationDate.IsExpired())
    throw new Ektron.Cms.Commerce.Exceptions.Payment.CreditCard.CardExpiredException("Card Is Expired");
    IsSubmissionSuccess = true;
    Authorization.AuthorizedOn = DateTime.Now;
    Authorization.TransactionId = new Guid().ToString();
    return Authorization.TransactionId;
}
```

In this example, you do not check the card number or the card holder name. In a real-world scenario, there would be additional validation (for example, through a checksum) and the authorization might be obtained through a Web service or HTTP post.

If the authorization fails, you can either throw an exception or you can manually set the failure. For example:

```
SubmissionError = “Not enough Funds”;

IsSubmissionSuccess = false;
```

You now implement the `AuthorizeAndCapture`, `CapturePreauthorization` and `VoidPreAuthorization` methods. `AuthorizeAndCapture` requires that `Authorization.CapturedOn` be configured, as the `Capture` occurs at the same time as the `Authorization`. Likewise, for `CapturePreauthorization` and `VoidPreAuthorization` you need to set the appropriate dates for these actions so they are recorded in the system properly.

```
public override string AuthorizeAndCapture()
{
    IsSubmissionSuccess = true;
    Authorization.AuthorizedOn = DateTime.Now;
```

```
    Authorization.CapturedOn = DateTime.Now;
    Authorization.TransactionId = new Guid().ToString();
    return Authorization.TransactionId;
}
public override string CapturePreauthorization(string transactionId)
{
    IsSubmissionSuccess = true;
    Authorization.CapturedOn = DateTime.Now;
    return Authorization.TransactionId;
}
public override string VoidPreAuthorization(string transactionId)
{
    IsSubmissionSuccess = true;
    Authorization.VoidedOn = DateTime.Now;
    return Authorization.TransactionId;
}
```

Now, build the project, and copy the assembly into the CMS400.NET site's \bin directory.

Registering the Provider

When the assembly is created and placed inside the CMS400.NET site's \bin directory, you need to register the provider and instruct CMS400.NET to use it. The web.config file provides the means to manage payment gateway providers within the CMS. Locate the `EktronPaymentGateway` section in the configuration file. Add a reference to the class you created earlier in the `<providers>` key, then change the `defaultProvider` attribute, as shown below:

```
<EktronPaymentGateway defaultProvider="CustomPaymentProvider">
    <providers>
    <add name="CustomPaymentProvider" type="Ektron.Cms.Extensibility.Commerce.Samples.CustomPaymentProvider,
    CustomPaymentProvider" />
    </providers>
</EktronPaymentGateway>
```

Add the gateway into the system

Now register the gateway with the system by doing the following.

1. In the workarea, go to Modules > Commerce > Configuration > Payment Options
2. Select New > Payment Gateway.
3. In the Name dropdown, select customPaymentProvider.
4. Check both **Cards** and **Checks** checkboxes.
5. Save these changes.

When the payment gateway is contacted, the calls will route through your new custom payment gateway provider.

Customizing Tax Calculations

Ektron CMS400.NET provides a flexible and extensible tax calculation system that can be customized as needs dictate.

In this example, we will add a flat recycling fee for a given tax class. We do this first by creating the Event Code and then registering the Event with the System.

Create the Event Code

First, create a class library project in Visual Studio named **Commerce.Tax.RecyclingFee** and then Import the Ektron namespaces you need by adding references to:

- Ektron.Cms.Api
- Ektron.Cms.Commerce
- Ektron.Cms.Common
- Ektron.Cms.Instrumentation
- Ektron.Cms.ObjectFactory

Next add the following using statements.

```
using Ektron.Cms.Instrumentation;
using Ektron.Cms.Extensibility;
using Ektron.Cms.Extensibility.Commerce;
using Ektron.Cms.Commerce;
using Ektron.Cms.Common;
```

Change the namespace to **Ektron.Cms.Extensibility.Commerce.Samples**, rename your class to **RecyclingFee**, and inherit from the **TaxCalculatorStrategy** class.

```
namespace Ektron.Cms.Extensibilitiy.Commerce.Samples
{
    public class RecyclingFee : TaxCalculatorStrategy
```

Now override the **OnBeforeCalculate** method, which is called when the tax is calculated for the basket.

```
public override void OnBeforeCalculate(BasketCalculatorData basketData, CancellableEventArgs eventArgs){ }
```

In the `OnBeforeCalculate` method, add in code to set a recycling fee of 16 dollars for any product with a tax class ID of 12.

```
    public override void OnBeforeCalculate(BasketCalculatorData basketData, CancellableEventArgs eventArgs)
    {

    ITaxClass classService = ObjectFactory.GetTaxClass();
    List classList = classService.GetList(new Criteria());

    //calculate tax for each item
    foreach (AdjustedBasketItem item in basketData.AdjustedItems)
    {

    long electronicItemClass = 12;
    decimal recyclingFee = 16.00M;

    TaxClassData itemTaxClass = classList.Find(delegate(TaxClassData match)
    {
    return match.Id == item.TaxClassId;
    });

    if (item.TaxClassId == electronicItemClass)
    {

    //calculate item taxes.
    item.TaxesApplied = recyclingFee;

    //update or add entry for taxclass sub total
    if (basketData.TaxClassSubtotals.ContainsKey(itemTaxClass))
    basketData.TaxClassSubtotals[itemTaxClass] += item.TaxesApplied;
    else
    basketData.TaxClassSubtotals.Add(itemTaxClass, item.TaxesApplied);

    //update basket total.
    basketData.TotalTaxes += item.TaxesApplied;

    }

    }

    eventArgs.IsCancelled = true;
    base.OnBeforeCalculate(basketData, eventArgs);
```

```
}
```

Build the project, and copy the assembly into the CMS400.NET siteroot\bin directory.

Register the Event with the System

Once the event assembly has been created, place it inside the CMS400.NET `siteroot/bin` directory.

To register this custom event with the CMS, edit the siteroot/**objectfactory.config** file located in the siteroot. In this file, locate or add a key name="`TaxCalculator`" in the `objectStrategies` section of the config file. Add a reference to the class we created earlier in the `<strategies>` key.

```
<objectStrategies>
<add name="TaxCalculator">
      <strategies>
        <add name="RecyclingFee" type="Ektron.Cms.Extensibility.Commerce.Samples.RecyclingFee, Commerce.Tax.RecyclingFee"/>
      </strategies>
      </add>
</objectStrategies>
```

When taxes are calculated for the basket, this function runs first. Setting the property IsCancelled to true on `eventArgs` (located on the next to the last line of the `OnBeforeCalculate` method) tells the CMS to stop further execution of the tax rate calculations, effectively overriding the tax functionality.

7 Glossary

Ektron eCommerce Terms

Term	Description
Electronic Data Interchange (EDI)	The process of sending messages across a network in order to perform financial transactions. EDI systems evolved in the 1970s in order to handle e-commerce--like messages between a large corporation (such as an automobile manufacturer) and its suppliers. While the EDI standards established by the U.N. (via the UN/EDIFAC organization) involved binary formatted data, most EDI in the 2000s use XML messages and SOAP-based Web services to accomplish the same thing.
HTTPS/SSL	Secure HTTP (HTTPS) combines the venerable HTTP standard for Web communication with a security system built using the SSL (Secure Sockets Layer). A Web site or Web service that uses HTTPS provides a "certificate"--an electronic document--that is issued by a trust authority indicating that the site is indeed who they claim to be and that they are not involved in fraudulent activity. Once the browser receives such a certificate, then it uses the information in the certificate to encrypt the contents being sent back to the server. This provides a reasonable degree of security for handling sensitive electronic information, such as credit card numbers.
PCI/DSS	A set of standards that provide for secure communication for financial transactions over the Internet. The PCI Data Security Standard was a concerted effort in 2006 by a consortium made up of members including Visa, Master Card, American Express and Discover in order to minimize Internet-based credit card abuse, though the DSS standard is finding its way into other secure financial transactions as well.
SEO (Search Engine Optimization)	The process of configuring Web content in order to gain the highest potential rankings for a given search engine. While early SEO systems involved simple keyword matches, SEO has evolved considerably to the level of performing semantic searches on content, optimizing the specific layout of a page to make its terms more indexable and using complex mathematical algorithms to better match anticipated search engine behaviors.
Personalization	The process of tailoring pages to individual users' characteristics or preferences. Commonly used to enhance customer service or e-commerce sales, personalization is sometimes referred to as one-to-one marketing, because the enterprise's Web page is tailored to specifically target each individual consumer. Personalization is a means of meeting the customer's needs more effectively and efficiently, making interactions faster and easier and, consequently, increasing customer satisfaction and the likelihood of repeat visits.
Authentication	The process of determining whether the individual signing in under a specific user name actually has an account with permissions to perform specific actions. Authentication can be as simple as providing a user name and password, but especially in high dollar e-commerce settings authentication is usually done in conjunction with access across secure channels and sometimes alternative authentication mechanisms (from thumb prints to retinal scans).
Access Control	The process of determining whether an individual has access to a specific function or piece of information. Authentication and Access Control are related processes with Access Control usually following Authentication.
Affiliate	Any web site or business that provides links or sales to your site through their own marketing efforts.
Affiliate Program	Where a company provides a tangible benefit (typically a fee or portion of sales) to the affiliate site for directing traffic toward the site or for providing additional promotion to secure a sale. Affiliate programs have been used for everything from book sales to political campaigns to blogging, and represents an active form of advertisement.
Banner	An advertisement or image displayed on one or more web sites to attract visitors to your site.
Caching	The storage of web files on a computer or server so they can be accessed quicker by the end
Click-through	The act of clicking on an online advertisement (banner) to the advertiser's web site.

Glossary 7

Term	Description
Clickstream	A virtual trail that a user leaves behind while surfing the Internet. A clickstream is a record of a user's activity on the Internet, including every Web site and every page of every Web site that the user visits, how long the user was on a page or site, in what order the pages were visited, any newsgroups that the user participates in and even the e-mail addresses of mail that the user sends and receives. Both ISPs and individual Web sites are capable of tracking a user's clickstream. Clickstream data is becoming increasingly valuable to Internet marketers and advertisers.
E-commerce	Generally refers to the exchange of goods or services via the Internet. Also e-business... both electronic sales via the Internet and business to business transactions via a dedicated connection (modem or broadband).
Encryption	Process of transforming data into a type that prevents casual observers from deciphering.
ERP	Short for enterprise resource planning, a business management system that integrates all facets of the business, including planning, manufacturing, sales, and marketing. As the ERP methodology has become more popular, software applications have emerged to help business managers implement ERP in business activities such as inventory control, order tracking, customer service, finance and human resources.
Merchant Account	A bank service account that allows you to accept credit card transactions. Just one part of the process of accepting online credit card orders. (See Payment Gateway on page 43)
Payment Gateway	An internet service that connects your e-commerce site with your Merchant Account. A gateway accepts your order information and connects to your Merchant Account to authorize and transfer funds.
shopping cart	A shopping cart is a piece of software that acts as an online store's catalog and ordering process. Typically, a shopping cart is the interface between a company's Web site and its deeper infrastructure, allowing consumers to select merchandise; review what they have selected; make necessary modifications or additions; and purchase the merchandise. Shopping carts can be sold as independent pieces of software so companies can integrate them into their own unique online solution, or they can be offered as a feature from a service that will create and host a company's e-commerce site.
SSL	Short for Secure Sockets Layer, a protocol developed by Netscape for transmitting private documents via the Internet. SSL uses a cryptographic system that uses two keys to encrypt data − a public key known to everyone and a private or secret key known only to the recipient of the message. Most browsers support SSL, and many Web sites use the protocol to obtain confidential user information, such as credit card numbers. By convention, URLs that require an SSL connection start with https: instead of http:. Another protocol for transmitting data securely over the World Wide Web is Secure HTTP (S-HTTP). Whereas SSL creates a secure connection between a client and a server, over which any amount of data can be sent securely, S-HTTP is designed to transmit individual messages securely. SSL and S-HTTP, therefore, can be seen as complementary rather than competing technologies. Both protocols have been approved by the Internet Engineering Task Force (IETF) as a standard.

Term	Description
Web analytics	A generic term meaning the study of the impact of a Web site on its users. E-commerce companies often use Web analytics software to measure such concrete details as how many people visited their site, how many of those visitors were unique visitors, how they came to the site (i.e., if they followed a link to get to the site or came there directly), what keywords they searched with on the site's search engine, how long they stayed on a given page or on the entire site, what links they clicked on and when they left the site. Web analytic software can also be used to monitor whether or not a site's pages are working properly. With this information, Web site administrators can determine which areas of the site are popular and which areas of the site do not get traffic and can then use this data to streamline a site to create a better user experience.