Active Commerce Developer's Cookbook

Registering Types in Microsoft Unity

Active Commerce utilizes the Microsoft Unity container to register implementations of various domain objects, services, and other types used throughout the product. This means that if you want to add new integrations or override other logic in Active Commerce, you can register overrides within Unity, either for your whole installation or for specific sites within Sitecore.

Active Commerce makes it easy to register your own types. At Sitecore startup, it will scan for any implementations of *ITypeRegistration* and *ISiteTypeRegistration* and execute them.

ITypeRegistration

By implementing *ITypeRegistration*, you can register types in the Unity container for the entire Sitecore / Active Commerce installation. The *SortOrder* property can be used to ensure the order in which types are registered. In most cases, it is safe to simply return a *0* value. See examples below on how to register specific types within Unity, or review *ActiveCommerce.IoC.RegisterTypes* in your reflector of choice.

```
public class RegisterTypes : ITypeRegistration
{
	public void Process(Microsoft.Practices.Unity.IUnityContainer
container)
	{
		//register your types here
	}

	public int SortOrder
	{
		get { return 0; }
	}
}
```

ISiteTypeRegistration

By implementing *ISiteTypeRegistration*, you can register types in Unity for specific sites within your Sitecore / Active Commerce installation. The *Sites* property can be used to return a collection of site names for which the registrations should apply. You can include the site names directly in your code, or perhaps reference the configured sites (*Sitecore.Sites.SiteContextFactory.Sites*) for those with a particular attribute configured (*SiteInfo.Properties*).

```
public class RegisterTypes : ISiteTypeRegistration
{
	public string[] Sites
	{
		get { //return string[] array }
	}

	public void Process(Microsoft.Practices.Unity.IUnityContainer
container)
	{
		//register your types here
```

```
	}

	public int SortOrder
	{
		get { return 0; }
	}
}
```

Adding a Custom Payment Provider

1. Create the Payment Provider class

- For an integrated payment provider (Authorize.Net, CyberSource), inherit from ActiveCommerce.Payment.IntegratedPaymentProviderBase.
    - Specify additional features that the provider will support by inheriting IReservable, and/or ICreditable.
- For an online payment provider (e.g. PayPal), inherit from ActiveCommerce.Payment.OnlinePaymentProviderBase.
    - Specify additional features that the provider will support by inheriting IReservable, and/or ICreditable.

- Implement the required method(s). Use the values from the method parameters as needed (PaymentSystem, PaymentArgs).
- Here's an example of our CyberSource plugin:

```
public class PaymentProvider : IntegratedPaymentProviderBase, ICreditable,
IReservable
{
	protected const string SETTING_AUTHORIZE_ONLY = "authorizeOnly";
	protected const string DECISION_ACCEPT = "ACCEPT";
	protected const string DECISION_ERROR = "ERROR";
	protected const string DECISION_REJECT = "REJECT";

	public override void Invoke(PaymentSystem paymentSystem, PaymentArgs
paymentArgs)
	{
	var cart = paymentArgs.ShoppingCart as ShoppingCart;
	if (cart == null)
	{
		throw new Exception("Cart must be of type
ActiveCommerce.Carts.ShoppingCart");
	}
	var settings = GetSettings(paymentSystem);

	var request = new RequestMessage
```

```
      {
            merchantID = paymentSystem.Username,
            merchantReferenceCode = cart.OrderNumber,
            billTo = new BillTo
             {
                firstName = cart.CustomerInfo.BillingAddress.Name,
                lastName = cart.CustomerInfo.BillingAddress.Name2,
                street1 = cart.CustomerInfo.BillingAddress.Address,
                city = cart.CustomerInfo.BillingAddress.City,
                state = cart.CustomerInfo.BillingAddress.State,
                postalCode = cart.CustomerInfo.BillingAddress.Zip,
                country = cart.CustomerInfo.BillingAddress.Country.Code,
                email = cart.CustomerInfo.Email,
                phoneNumber = cart.CustomerInfo.BillingAddress.GetPhoneNumber()
            },
            card = new Card
             {
             accountNumber = cart.CreditCardInfo.CardNumber.ToString(),
                expirationMonth = cart.CreditCardInfo.ExpirationDate.ToString("MM"),
                expirationYear = cart.CreditCardInfo.ExpirationDate.ToString("yyyy"),
                cardType = GetCardTypeId(cart.CreditCardInfo.CardType)
            },
            purchaseTotals = new PurchaseTotals
             {
                currency = cart.Currency.Code,
                grandTotalAmount = cart.Totals.TotalPriceIncVat.ToString("0.00") //
Uses midpoint away from zero rounding
             }
      };

      // For CyberSource (Them: "To help us troubleshoot any problems that you may
encounter, please include the following information about your application.")
        request.clientLibrary = ".NET WCF";
        request.clientLibraryVersion = Environment.Version.ToString();

request.clientEnvironment = Environment.OSVersion.Platform +
Environment.OSVersion.Version.ToString();

      // Turn on Authorize
        request.ccAuthService = new CCAuthService {run = "true"};
      if (settings.IncludeCapture)
      {
            // Turn on Capture
            request.ccCaptureService = new CCCaptureService {run = "true"};
      }

      var reply = MakeRequest(request, paymentSystem, false /* CheckPaymentStatus
will log error */);
```

```
        if (reply.decision != DECISION_ACCEPT) return;

        var transactionData = Context.Entity.Resolve<ITransactionData>();
        if (settings.IncludeCapture)
        {
              PaymentStatus = PaymentStatus.Captured;
        }
        else
        {
               var reservationTicket = new ReservationTicket
                                            {
                                                  InvoiceNumber = cart.OrderNumber,

Amount = Math.Round(cart.Totals.TotalPriceIncVat, 2, MidpointRounding.AwayFromZero),

AuthorizationCode = reply.ccAuthReply.authorizationCode,
                                                  TransactionNumber = reply.requestID
                                            };

transactionData.SavePersistentValue(cart.OrderNumber, "ReservationTicket",
reservationTicket);
              PaymentStatus = PaymentStatus.Reserved;
        }

          transactionData.SaveCallBackValues(cart.OrderNumber,
                                                  PaymentStatus.ToString(),
                                                  reply.requestID,

cart.Totals.TotalPriceIncVat.ToString("0.00"),
                                                  cart.Currency.Code,
                                                  reply.decision, //SEFE bug. pass in here,
get via PaymentStatus
                                                  reply.reasonCode,
                                                  reply.reasonCode,
                                                  cart.CreditCardInfo.CardType);

transactionData.SavePersistentValue(cart.OrderNumber,
TransactionConstants.AuthorizationNumber, reply.ccAuthReply.authorizationCode);

        }

    #region IReservable Members

        public void Capture(PaymentSystem paymentSystem, PaymentArgs paymentArgs,
ReservationTicket reservationTicket, decimal amount)
        {
        var request = new RequestMessage
```

```
        {
              merchantID = paymentSystem.Username,
              merchantReferenceCode = reservationTicket.InvoiceNumber,
              ccCaptureService = new CCCaptureService
               {
                  run = "true",
                  authRequestID = reservationTicket.TransactionNumber
              },
              purchaseTotals = new PurchaseTotals
               {
                  currency = paymentArgs.ShoppingCart.Currency.Code,
                  grandTotalAmount = reservationTicket.Amount.ToString("0.00") // Uses
away from zero rounding
               }
        };

        var reply = MakeRequest(request, paymentSystem);
        if (reply.decision != DECISION_ACCEPT) return;

        var transactionData = Context.Entity.Resolve<ITransactionData>();
         transactionData.SavePersistentValue(reservationTicket.InvoiceNumber,
PaymentConstants.CaptureSuccess);
        }

        public void CancelReservation(PaymentSystem paymentSystem, PaymentArgs
paymentArgs, ReservationTicket reservationTicket)
        {
        var request = new RequestMessage
        {
              merchantID = paymentSystem.Username,
              merchantReferenceCode = reservationTicket.InvoiceNumber,
              voidService = new VoidService
               {
                  run = "true",
                  voidRequestID = reservationTicket.TransactionNumber
               }
        };

        var reply = MakeRequest(request, paymentSystem);
        if (reply.decision != DECISION_ACCEPT) return;

        var transactionData = Context.Entity.Resolve<ITransactionData>();
         transactionData.SavePersistentValue(reservationTicket.InvoiceNumber,
PaymentConstants.CancelSuccess);
        }

    #endregion
```

```
    #region ICreditable Members

       public void Credit(PaymentSystem paymentSystem, PaymentArgs paymentArgs,
ReservationTicket reservationTicket)
       {
       var request = new RequestMessage
       {
             merchantID = paymentSystem.Username,
             merchantReferenceCode = reservationTicket.InvoiceNumber,
             ccCreditService = new CCCreditService
              {
                 run = "true",
                 captureRequestID = reservationTicket.TransactionNumber
             },
             purchaseTotals = new PurchaseTotals
              {
                 currency = paymentArgs.ShoppingCart.Currency.Code,
                 grandTotalAmount = reservationTicket.Amount.ToString("0.00") // Uses
away from zero rounding
              }
       };

       var reply = MakeRequest(request, paymentSystem);
       if (reply.decision != DECISION_ACCEPT) return;

       var transactionData = Context.Entity.Resolve<ITransactionData>();
        transactionData.SavePersistentValue(reservationTicket.InvoiceNumber,
PaymentConstants.CreditSuccess);
       }

    #endregion

       protected ReplyMessage MakeRequest(RequestMessage request, PaymentSystem
paymentSystem, bool logError = true)
       {
       try
       {
              var proc = new TransactionProcessorClient();

             proc.ChannelFactory.Credentials.UserName.UserName = request.merchantID;

proc.ChannelFactory.Credentials.UserName.Password = paymentSystem.Password;

              var reply = proc.runTransaction(request);

              if (reply.decision != DECISION_ACCEPT)
```

```
            {
                var message = new StringBuilder();

if (reply.ccAuthReply != null && !
String.IsNullOrWhiteSpace(reply.ccAuthReply.reasonCode))
                {
                    message.AppendFormat("[Auth]:{0}", reply.ccAuthReply.reasonCode);
                }

if (reply.ccCaptureReply != null && !
String.IsNullOrWhiteSpace(reply.ccCaptureReply.reasonCode))
                {

message.AppendFormat("[Capture]:{0}", reply.ccCaptureReply.reasonCode);
                }

if (reply.ccCreditReply != null && !
String.IsNullOrWhiteSpace(reply.ccCreditReply.reasonCode))
                {

message.AppendFormat("[Credit]:{0}", reply.ccCreditReply.reasonCode);
                }
                var transactionData = Context.Entity.Resolve<ITransactionData>();
                transactionData.SaveCallBackValues(request.merchantReferenceCode, //
OrderNumber
                                                   PaymentStatus.Failure.ToString(),
                                                   reply.requestID,
                                                   reply.purchaseTotals != null ?
reply.purchaseTotals.grandTotalAmount : null,
                                                   reply.purchaseTotals != null ?
reply.purchaseTotals.currency : null,
                                                   reply.decision, //SEFE bug. pass
in here, get via PaymentStatus
                                                   reply.reasonCode,
                                                   message.ToString(),
                                                   request.card != null ?
request.card.cardType : null);

                PaymentStatus = PaymentStatus.Failure;

                if (logError)
                {
                    Log.Error(string.Format(
                          "CyberSource transaction failure: {0}\nProvider Message =
{1}\nProvider Error Code = {2}\nTransaction Number = {3}",
                          reply.decision, message, reply.reasonCode,
reply.requestID), this);
```

```
                }
            }
            return reply;
        }
            catch (Exception e)
        {
            Log.Error("CyberSource transaction failure", e, this);
           PaymentStatus = PaymentStatus.Failure;
            return new ReplyMessage {decision = DECISION_ERROR};
        }
        }

        protected static string GetCardTypeId(string cardType)
        {
        switch (cardType.ToLowerInvariant())
        {
            case "visa":
                return "001";
            case "mastercard":
                return "002";
            case "american express":
                  return "003";
            case "discover":
                return "004";
            case "diners club":
                return "005";
            case "carte blanche":
                return "006";
            case "jcb":
                return "007";
            default:
                return null;
        }
        }

        protected Settings GetSettings(PaymentSystem paymentSystem)
        {
        var settings = new Settings();
        var settingsReader = new
Sitecore.Ecommerce.Payments.PaymentSettingsReader(paymentSystem);

        //a little backwards, but we want to configure for authorize only, not for
include capture. capture unless auth only explicitly set to "true"
        var authorizeSetting = settingsReader.GetSetting(SETTING_AUTHORIZE_ONLY);

settings.IncludeCapture = authorizeSetting == null || (!
Boolean.TrueString.ToLower().Equals(authorizeSetting.ToLower()));
```

```
        return settings;
        }

        protected struct Settings
        {
        public bool IncludeCapture;
        }
}
```

- You could also check out the SES ones: https://sdn.sitecore.net/Products/SEFE/SES12/Payment%20Providers/120_111101.aspx

2. Register with Unity

- Make sure to register it as a named instance (e.g. "CyberSource").
- For example, here is the registration for CyberSource plugin:

```
container.RegisterType(typeof(Sitecore.Ecommerce.DomainModel.Payments.PaymentProvider
),
                                    typeof(PaymentProvider), "CyberSource");
```

3. Add to Payment Options

- In Sitecore, add a new Payment to Webshop Business Settings/Payment Options
- Set the "Code" field to the same name you registered with Unity (e.g. "CyberSource"), and make sure "Enabled" is checked.
- Those are the only fields technically required. The rest are there for you to use to make the provider configurable in Sitecore (e.g. Username and Password typically correspond to auth for your 3rd party service).

Adding a Custom Tax Calculator

Active Commerce comes with 2 tax calculators. One for VAT (ActiveCommerce.Taxes.Vat.VatTaxCalculator), and one for North America (ActiveCommerce.Taxes.NorthAmerica.ShippingAddressTaxCalculator). The following instructions are for creating your own calculator.

1. Create the Tax Calculator class

- Inherit from ActiveCommerce.Taxes.TaxCalculatorBase or ActiveCommerce.Taxes.ITaxCalculator.
- TaxCalculatorBase provides access to the Sitecore fields related to this calculator via TaxConfiguration class. There are also base implementations and helpers which may be useful. If you don't need any of this, simply use ITaxCalculator.

- Implement the required properties and methods.
    - **TaxConfiguration** - This will automatically be set if using the TaxCalculatorBase. If using ITaxCalculator, you must define, but it's up to you'd like to use or not.
    - **Source** - This will be populated with the address of the company (as defined on / Webshop Business Settings/Company Master Data)
    - **Billing** - This will be populated with the customer billing address
    - **Shipping** - This will be populated with the customer shipping address
    - **Currency** - This will be populated with the current currency
    - **WillHandle()** - return boolean whether this tax calculator will handle calculation for the addresses. If using TaxCalculatorBase, the MatchesLocation helper method is useful here if based on configured location (see step 3). If this will be your only tax calculator, then you could just return true.
    - **GetTaxes(IEnumerable<TaxInquiry> order)** - return your TaxTotals
- Here's an example of our VatTaxCalculator:

```
public class VatTaxCalculator : TaxCalculatorBase<TaxConfiguration>
{
	public const string JURISDICTION_TYPE = "VAT";

	public override TaxTotals GetTaxes(IEnumerable<TaxInquiry> order)
	{
	var rounder = Sitecore.Ecommerce.Context.Entity.Resolve<ICurrencyRounder>();
	order = order.ToList();
	return new TaxTotals
	{
		ProductTax = order.Where(x => !x.IsShipping && !x.IsHandling)
				.Select(x => new { TaxInquiry = x, Product =
GetProduct(x.ProductCode) })
				.Select(x => new { TaxInquiry = x.TaxInquiry,
Product = x.Product, Rate = GetRate(x.Product.TaxType) })
				.Select(x => new TaxLine
						{
						    ProductCode = x.Product.Code,
						    Type = x.Product.TaxType !=
null ? x.Product.TaxType.Code : string.Empty,
						    TaxedAmount = x.TaxInquiry.Total,

Jurisdictions = new TaxJurisdiction
                                                                {
                                                                    Name =
Config.Name,
                                                                    Type =
TaxJurisdiction.Types.VAT,
                                                                    Rate =
x.Rate,
                                                                    Tax =
rounder.Round(x.Rate * x.TaxInquiry.Total, Currency)
```

```
 }.ToEnumerable()
                                                 }),
            ShippingTax = order.Where(x => x.IsShipping)
                                .Select(x => new { TaxInquiry = x, Rate =
GetRate(Config.ShippingVatType)})
                                .Where(x => x.Rate > 0)
                              .Select(x => new TaxLine
                                                 {

Type = Config.ShippingVatType.Code,
                                                      TaxedAmount = x.TaxInquiry.Total,
                                                       Jurisdictions = new
TaxJurisdiction
                                                                                {
                                                                                    Name =
Config.Name,
                                                                                    Type =
JURISDICTION_TYPE,
                                                                                    Rate =
x.Rate,
                                                                                    Tax =
rounder.Round(x.Rate * x.TaxInquiry.Total, Currency)

 }.ToEnumerable()
                                                 }).FirstOrDefault(),
            HandlingTax = order.Where(x => x.IsHandling)
                                .Select(x => new { TaxInquiry = x, Rate =
GetRate(Config.HandlingVatType)})
                                .Where(x => x.Rate > 0)
                                .Select(x => new TaxLine
                                                 {

Type = Config.ShippingVatType.Code,
                                                      TaxedAmount = x.TaxInquiry.Total,
                                                       Jurisdictions = new
TaxJurisdiction
                                                                                {
                                                                                    Name =
Config.Name,
                                                                                   Type =
JURISDICTION_TYPE,
                                                                                    Rate =
x.Rate,
                                                                                    Tax =
rounder.Round(x.Rate * x.TaxInquiry.Total, Currency)

 }.ToEnumerable()
```

```
                                                    }).FirstOrDefault()
        };
        }

        public override bool WillHandle()
    {
        return MatchesLocation(Shipping);
        }


        protected virtual Product GetProduct(string productCode)
        {
        var repository =
Sitecore.Ecommerce.Context.Entity.Resolve<IProductRepository>();
        var product = repository.Get<Product>(productCode);
        return product;
        }

        protected virtual decimal GetRate(TaxType taxType)
        {
        if (taxType == null || !taxType.Values.Any())
        {
                return 0m;
        }
        var taxValue = taxType.Values.FirstOrDefault(x => x.TaxConfiguration != null
&& x.TaxConfiguration.Code == Config.Code);
        if (taxValue == null)
        {
                return 0m;
        }
        return taxValue.Rate;
        }
}
```

2. Register with Unity

- Make sure to register it as a named instance (e.g. "MyTaxCalc").

```
container.RegisterType(typeof(ITaxCalculator), typeof(Foo.Bar.MyTaxCalculator),
"MyTaxCalc")
```

3. Add to Tax Calculations

- In Sitecore, under Webshop Business Settings/Tax Calculation, add a new item. Insert from Template, and choose /sitecore/templates/ActiveCommerce/Business Catalog/Tax Calculator

- Set the "Tax Calculator" field to the same name you registered with Unity (e.g. "MyTaxCalc").
- That is the only field technically required. You can use the Locations field if you'd like to restrict the use of this calculator to specific countries/regions (in conjunction with the MatchesLocation helper method). You can also extend this template with your own fields, and create a new TaxConfiguration to access those values.

Adding a Custom Tax Rate Provider

The built-in North America tax calculator makes use of a Rate Provider. See the *Active Commerce Configuration Guide* for a list of Rate Provider plugins already offered. The following instructions are for creating your own provider.

1. Create the Tax Rate Provider class

- Inherit from ActiveCommerce.Taxes.Rates.ISalesTaxRateProvider.
- Implement the required method(s).
- Here's an example of our FastTax plugin:

```
public class SalesTaxRateProvider : ISalesTaxRateProvider
{
	protected static Cache _cache;
	protected static object _cacheLock = new object();

	protected const string TAX_TYPE = "sales";
	protected const string UNITED_STATES = "US";
	protected const string CANADA = "CA";
	protected const string LICENSE_KEY_SETTING
= "ActiveCommerce.FastTax.LicenseKey";
	protected const string FAST_TAX_CACHE_NAME = "ActiveCommerce.FastTax.Rates";
	protected const string FAST_TAX_CACHE_SIZE_SETTING
= "ActiveCommerce.FastTax.Cache.Size";

	protected string _licenseKey;

	protected Cache Cache
	{
	get
	{
		lock (_cacheLock)
		{
		   if (_cache == null)
		   {

_cache = Sitecore.Caching.Cache.GetNamedInstance(FAST_TAX_CACHE_NAME,

	Sitecore.StringUtil.ParseSizeString(Sitecore.Configuration.Settings.GetSetting
(FAST_TAX_CACHE_SIZE_SETTING)));
		   }
```

```
                }
                return _cache;
        }
        }

        public SalesTaxRateProvider()
        {

_licenseKey = Sitecore.Configuration.Settings.GetSetting(LICENSE_KEY_SETTING);
        }

    #region ISalesTaxRateProvider Members

        public IEnumerable<TaxRate> GetRates(AddressInfo address)
        {
        if (address.Country == null)
        {
                throw new Exception("No country on the provided address");
        }

        string cacheKey = GetCacheKey(address);
        var cachedRates = Cache.GetEntry(cacheKey, true);
        if (cachedRates != null)
        {
                return cachedRates.Data as IEnumerable<TaxRate>;
        }

        IEnumerable<TaxRate> rates = null;
        if (address.Country.Code == UNITED_STATES)
        {
              rates = GetUnitedStatesTaxRates(address);
        }
        else if (address.Country.Code == CANADA)
        {
              rates = GetCanadianTaxRates(address);
        }
        else
        {
                throw new Exception("This rate provider only supports the United States
and Canada");
        }

        if (rates != null)
        {
              Cache.Add(cacheKey, rates, GetDataSize(rates), DateTime.UtcNow.AddDays(1)
);
        }
```

```
        return rates;
        }

    #endregion

        public IEnumerable<TaxRate> GetUnitedStatesTaxRates(AddressInfo address)
        {
        var service = new DOTSFastTaxSoapClient();

        if (address.Zip.IsNullOrEmpty())
        {
               throw new Exception("U.S. sales tax lookup requires zip code");
        }

        TaxInfo taxInfo;
        if (address.Address.IsNullOrEmpty() || address.City.IsNullOrEmpty())
        {
             taxInfo = GetTaxInfoForZip(address.Zip);
        }
        else
        {
               try
               {
                  taxInfo = GetTaxInfoForAddress(address);
               }
               catch (FastTaxException exception)
               {
                  Sitecore.Diagnostics.Log.Warn("Exception getting sales tax for
address {0}, will attempt by zip alone".FormatWith(GetCacheKey(address)), exception,
this);
                  taxInfo = GetTaxInfoForZip(address.Zip);
               }
        }
        return GetRatesForTaxInfo(taxInfo);
        }

        public IEnumerable<TaxRate> GetCanadianTaxRates(AddressInfo address)
        {
        var service = new DOTSFastTaxSoapClient();

        if (address.State.IsNullOrEmpty())
        {
               throw new Exception("Canadian sales tax lookup requires province.");
        }
```

```
	var taxResponse = service.GetCanadianTaxInfoByProvince(address.State,
_licenseKey);
	if (taxResponse.Error.IsError())
	{
		throw new FastTaxException(taxResponse.Error);
	}

	var rates = new List<TaxRate>();
	if (taxResponse.HarmonizedSalesTax > 0)
	{
		rates.Add(new TaxRate
				{
					Name = taxResponse.ProvinceAbbreviation,
					Type = TaxJurisdiction.Types.CANADA_HST,
					Rate = taxResponse.HarmonizedSalesTax
				});
	}
	else if (taxResponse.ProvinceSalesTax > 0)
	{
		rates.Add(new TaxRate
				{
					Name = CANADA,
					 Type = TaxJurisdiction.Types.CANADA_GST,
					Rate = taxResponse.GoodsSalesTax
				});
		rates.Add(new TaxRate
				{
					Name = taxResponse.ProvinceAbbreviation,
					Type = TaxJurisdiction.Types.CANADA_PST,
					Rate = taxResponse.ProvinceSalesTax,
					Compounds =
Boolean.TrueString.Equals(taxResponse.ApplyGSTFirst,
StringComparison.InvariantCultureIgnoreCase)
				});
	}
	return rates.AsEnumerable();
	}

	protected TaxInfo GetTaxInfoForZip(string zipCode)
	{
	var service = new DOTSFastTaxSoapClient();
	var taxResponse = service.GetTaxInfoByZip_V2(zipCode, TAX_TYPE, _licenseKey);
	if (taxResponse.Error.IsError())
	{
		throw new FastTaxException(taxResponse.Error);
	}
	if (taxResponse.TaxInfo == null || taxResponse.TaxInfo.Length == 0)
	{
```

```
            throw new FastTaxException(string.Format("Fast Tax returned empty tax
info for {0}", zipCode));
      }
      return taxResponse.TaxInfo[0];
      }

      protected TaxInfo GetTaxInfoForAddress(AddressInfo address)
      {
      var service = new DOTSFastTaxSoapClient();
      var taxInfo = service.GetTaxInfoByAddress(address.Address, address.City,
address.State, address.Zip, TAX_TYPE, _licenseKey);
      if (taxInfo.Error.IsError())
      {
            throw new FastTaxException(taxInfo.Error);
      }
      return taxInfo;
      }

      protected IEnumerable<TaxRate> GetRatesForTaxInfo(TaxInfo taxInfo)
      {
      var rates = new List<TaxRate>();
      if (taxInfo.StateRate > 0)
      {
            rates.Add(new TaxRate
                        {
                              Name = taxInfo.StateAbbreviation,
                              Type = TaxJurisdiction.Types.STATE,
                              Rate = taxInfo.StateRate
                        });
      }
      if (taxInfo.CountyRate > 0)
      {
            rates.Add(new TaxRate
                        {
                              Name = taxInfo.County,
                              Type = TaxJurisdiction.Types.COUNTY,
                              Rate = taxInfo.CountyRate
                        });
      }
      if (taxInfo.CountyDistrictRate > 0)
      {
            rates.Add(new TaxRate
                        {
                            Name = taxInfo.County,
                              Type = TaxJurisdiction.Types.COUNTY_DISTRICT,
                              Rate = taxInfo.CountyDistrictRate
                        });
      }
```

```
        if (taxInfo.CityRate > 0)
        {
              rates.Add(new TaxRate
                            {
                                  Name = taxInfo.City,
                                  Type = TaxJurisdiction.Types.CITY,
                                  Rate = taxInfo.CityRate
                            });
        }
        if (taxInfo.CityDistrictRate > 0)
        {
              rates.Add(new TaxRate
                            {
                                  Name = taxInfo.City,
                                  Type = TaxJurisdiction.Types.CITY_DISTRICT,
                                  Rate = taxInfo.CityDistrictRate
                            });
        }
        return rates.AsEnumerable();
        }

        public string GetCacheKey(AddressInfo address)
        {
        return address.Country.Code + "~" + address.Address + "~" + address.City
+ "~" + address.State + "~" + address.Zip;
        }

        public long GetDataSize(IEnumerable<TaxRate> rates)
        {
        rates = rates.ToList();
        //decimal is 128-bit / 16 bytes
        return rates.Sum(x => x.Name.Length) + rates.Sum(x => x.Type.Length) +
(rates.Count()*16);
        }

}
```

2. Register with Unity

- Make sure to register it as a named instance (e.g. "FastTax").

```
        container.RegisterType(typeof(ISalesTaxRateProvider),

        typeof(ActiveCommerce.FastTax.SalesTaxRateProvider), "FastTax");
```

3. Update Tax Calculations to use

- Follow the Active Commerce Configuration Guide - Rate Provider section, except we'll set the "Rate Provider" field to the name you registered with Unity (e.g. "FastTax").

Adding a Custom Address Validator

1. Create the Address Validator class

- Inherit from ActiveCommerce.Validation.IAddressValidator.
- Implement the required method(s).
- Here's an example of our default SimpleAddressValidator:

```
public class SimpleAddressValidator : IAddressValidator
{
	public const string US_ZIP_REGEX = @"^\d{5}(-\d{4})?$";
	public const string CANADA_ZIP_REGEX = @"^[ABCEGHJKLMNPRSTVXY]{1}\d{1}[A-Z]{1}
*\d{1}[A-Z]{1}\d{1}$";

	public Sitecore.Ecommerce.DomainModel.Addresses.AddressInfo
Validate(Sitecore.Ecommerce.DomainModel.Addresses.AddressInfo address)
	{
	if (String.IsNullOrWhiteSpace(address.Address) ||
		String.IsNullOrWhiteSpace(address.City) ||
		address.Country == null)
	{
		throw new AddressValidationException("Street, City, and Country are
required.");
	}

	// Perform some basic housekeeping
	  address.Address = address.Address.Trim();
	  address.Address2 = address.Address2 != null ? address.Address2.Trim() : null;
	  address.City = address.City.Trim();
	  address.State = address.State.Trim();
	  address.Zip = address.Zip.Trim();

	// US validation
	if (address.Country.Code.Equals("US",
StringComparison.InvariantCultureIgnoreCase))
	 {
		if (!(new Regex(US_ZIP_REGEX)).IsMatch(address.Zip))
		{
		   throw new AddressValidationException("Not a valid US zip code");
		}

		if (String.IsNullOrWhiteSpace(address.State))
		{
		    throw new AddressValidationException("State is required");
		}
	}
```

```
        // Canada validation
        if (address.Country.Code.Equals("CA",
StringComparison.InvariantCultureIgnoreCase))
        {
                if (!(new Regex(CANADA_ZIP_REGEX)).IsMatch(address.Zip))
                {
                   throw new AddressValidationException("Not a valid Canadadian zip
code");
                }

                if (String.IsNullOrWhiteSpace(address.State))
                {
                   throw new AddressValidationException("Province is required");
                }
        }

        return address;
        }
}
```

2. Register with Unity

```
        container.RegisterType(typeof(ActiveCommerce.Validation.IAddressValidator),
typeof(Foo.Bar.AddressValidator));
```

Adding a Product Detail Tab

1. Create a user control

- Inherit from ActiveCommerce.Web.skins.ProductDetails_Base and ActiveCommerce.Web.skins.ITabControl
- Implement the required properties.
    - “TitleKey” should return a key corresponding to a value in the Translation Dictionary (Add a new Dictionary entry if necessary)
    - “Id” should return a unique (DOM) identifier (this will be the id of the tab element)
- Here is an example:

```
public partial class ProductTab_Info : ProductDetails_Base, ITabControl
{
        public string TitleKey
        {
        get { return "Product-Tab-Info"; }
        }

        public string Id
```

```
        {
        get { return "Info"; }
        }
}
```

- Use the following scaffolding for you .ascx code:

```
<% if (Sitecore.Context.PageMode.IsPageEditor) { %>
<ul class="ui-tabs-nav">
        <li class="ui-state-active"><a
href="#<%=Id %>"><%=Translator.Render((TitleKey)) %></a></li>
</ul>
<% } %>

<div id="<%=Id %>" class="ui-tabs-panel">
        <div class="body">
                <!--YOUR TAB CODE HERE-->
        </div>
</div>
```

- This ensures your new tab is usable in the Page Editor

2. Create a sublayout

- Set your user control as the Ascx file

3. Add sublayout to presentation details

- Add your sublayout to the /sitecore/templates/ActiveCommerce/Product/Product/__Standard Values presentation details
- Use sorting to define where your tab appears (look for “Product Tab - “, these are the default Active Commerce tabs)

Adding an Order Pipeline Processor

1. Create a class

- Inherit from ActiveCommerce.OrderProcessing.IOrderPipelineProcessor
- Implement the required “Process” method
    - Use OrderPipelineArgs argument as necessary
        - Cart contains the state of the Shopping Cart when ordered
        - Order contains the order information (if after the “CreateOrder” step)
        - To abort the order process, set the Status and optionally AddMessage appropriately, and then call AbortPipeline()

2. Inject the processor into the orderProcessing Pipeline

- An example config file:

```
<configuration xmlns:patch="http://www.sitecore.net/xmlconfig/">
    <sitecore>
        <processors>
            <orderProcessing>
```

```
            <processor type="MySite.OrderProcessing.SomeAdditionalCheck,
MySite.Classes"

      patch:after="processor[@type='ActiveCommerce.OrderProcessing.CreateOrder,
ActiveCommerce.Kernel']"/>
            </orderProcessing>
        </processors>
    </sitecore>
</configuration>
```

Adding a Custom Order Status

1. Create the order status class

- Inherit from Sitecore.Ecommerce.Orders.Statuses.OrderStatusBase.
- Implement the required method(s).
- Here's an example:

```
public class Shipped : Sitecore.Ecommerce.Orders.Statuses.OrderStatusBase
{
      protected override void Process<T>(T order)
      {
            // Do nothing.
      }
}
```

2. Register with Unity

```
      container.RegisterType(typeof(Sitecore.Ecommerce.DomainModel.Orders.OrderStatu
s), typeof(Foo.Bar.Orders.Statuses.Shipped), "Shipped");
```

3. Add to Order Statuses

- In Sitecore, add a new Order Status to Webshop Business Settings/Order Status
- Set the "Code" field to the same name you registered with Unity (e.g. "Shipped"). Optionally, assign any next available statuses using the “Available List” field.