Sitecore Commerce Connect 8.2.1

The Commerce Connect Developer's Guide

A Developer's Guide for Using Commerce Connect to build e-commerce solutions

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Table of Contents

Sitecore Commerce Connect 8.2.1 1
Table of Contents 2
Chapter 1 Introduction 9
1.1 The Connect Service Layer 10
1.1.1 ServiceProviderRequest and ServiceProviderResult 10
ServiceProviderRequest 10
ServiceProviderResult 11
The RequestContext 11
The ShopProvider 13
The AuthenticationTokenProvider 14
1.2 Page Events, Goals, Outcomes and Custom Analytics Data 16
1.2.1 Analytics Data Classes 16
1.2.2 Outcome Data Classes 20
1.3 Federated payments 25
1.3.1 Order processing 25
Chapter 2 Using Connect to build e-commerce solutions 28
2.1 Carts 29
2.1.1 Cart Domain Model 29
IReadOnlyCollections and service API 30
Cart Keys 30
Class: Cart Base 30
Class: Cart 31
Class: Cart Line 31
Class: Cart Product 32
Class: Cart Adjustment 33
Class: Cart Option 33
2.1.2 Cart Service Provider 34
GetCarts 34
CreateOrResumeCart 35
LoadCart 36
SaveCart 36
AddCartLines 37
RemoveCartLines 38
UpdateCartLines 39
DeleteCart 40
UpdateCart 41
LockCart 42
UnlockCart 42
MergeCart 43
AddParties 44
RemoveParties 45
UpdateParties 45
AddPaymentInfo 46
RemovePaymentInfo 47
AddShippingInfo 47
RemoveShippingInfo 48
2.1.3 Cart Pipelines 48
GetCarts 49
CreateOrResumeCart 50
CreateCart 52
ResumeCart 54
LoadCart 56
SaveCart 57
AddCartLines 59
RemoveCartLines 61

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

UpdateCartLines ... 63
DeleteCart ... 64
UpdateCart ... 66
LockCart ... 67
UnlockCart ... 69
MergeCart ... 70
2.2 Pricing ... 72
2.2.1 The Pricing Domain Model ... 72
Class: Price ... 72
Class: PriceCondition ... 73
Class: DatePriceCondition ... 74
Class: QuantityPriceCondition ... 74
Class: Total ... 74
Class: TaxTotal ... 75
Class: TaxSubtotal ... 75
Class: Promotion ... 75
2.2.2 Pricing Service Methods ... 75
GetProductPrices ... 76
GetProductBulkPrices ... 77
GetCartTotal ... 77
GetSupportedCurrencies ... 78
CurrencyChosen ... 79
GetEligiblePromotionIds ... 79
GetProductPromotionDescription ... 80
2.2.3 Pricing Pipelines ... 81
GetProductPrices ... 81
GetProductBreakPrices ... 81
GetCartTotals ... 82
GetSupportedCurrencies ... 83
CurrencyChosen ... 83
GetEligiblePromotionIds ... 84
GetProductPromotionDescription ... 84
2.3 Order ... 86
2.3.1 The Order Domain Model ... 86
Class: Order ... 86
Class: OrderHeader ... 86
Class: OrderCancellation ... 87
2.3.2 Order Service Methods ... 87
SubmitVisitorOrder ... 87
GetAvailableCountries ... 88
GetAvailableRegions ... 88
GetVisitorOrder ... 89
GetVisitorOrders ... 89
OrderStatusChanged ... 90
Reorder ... 90
VisitorCancelOrder ... 91
2.3.3 Order Pipelines ... 92
SubmitVisitorOrder ... 92
GetAvailableCountries ... 92
GetAvailableRegions ... 93
GetVisitorOrders ... 93
GetVisitorOrder ... 94
OrderStatusChanged ... 95
Reorder ... 95
VisitorCancelOrder ... 96
2.4 Catalog ... 98
2.4.1 The Catalog Domain Model ... 98
Class: SortDirection ... 99

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

2.4.2 Catalog Service Methods ... 99
ProductSorting ... 99
FacetApplied ... 99
VisitedCategoryPage ... 100
VisitedProductDetailsPage ... 100
SearchInitiated ... 101
2.4.3 Catalog Pipelines ... 101
Product Sorting ... 101
Facet Applied ... 102
Visited Product Details Page ... 103
Visited Category Page ... 103
Search Initiated ... 104
2.5 Globalization ... 105
2.5.1 The Globalization Domain Model ... 105
2.5.2 Globalization Service Methods ... 105
CultureChosen ... 105
2.5.3 Globalization Pipelines ... 105
CultureChosen ... 105
2.6 Inventory ... 107
2.6.1 The Inventory Domain Model ... 107
Class: StockInformation ... 108
Class: OrderableInformation ... 108
Class: IndexStockInformation ... 108
Class: StockInformationUpdate ... 109
Class: StockInformationUpdateLocation ... 109
Class: StockDetailsLevel ... 109
Class: StockStatus ... 110
Class: InventoryProduct ... 111
Class: IndexStockInformation ... 111
Class: StockLocation ... 111
2.6.2 Inventory Service Methods ... 111
GetStockInformation ... 112
GetPreOrderableInformation ... 112
GetBackOrderableInformation ... 113
VisitedProductStockStatus ... 114
ProductsAreBackInStock ... 114
VisitorSignUpForStockNotification ... 115
RemoveVisitorFromStockNotification ... 116
GetBackInStockInformation ... 116
GetStockLocations ... 117
GetProductStockLocations ... 118
2.6.3 Inventory Pipelines ... 118
GetStockInformation ... 118
StockStatusForIndexing ... 119
GetPreOrderableInformation ... 120
GetBackOrderableInformation ... 121
ProductsAreBackInStock ... 121
GetBackInStockInformation ... 122
VisitorSignUpForStockNotification ... 123
RemoveVisitorFromStockNotification ... 124
OrderedProductStockStatus ... 125
GetStockLocations ... 125
GetProductStockLocations ... 126
2.7 Customer ... 127
2.7.1 The Customer Domain Model ... 127
Class: CommerceUser ... 127
Class: CommerceCustomer ... 128
Class: CustomerParty ... 129

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Class: CustomerPartyTypes 130
Class: Party 130
2.7.2 Customer Service Methods 131
CreateUser 131
UpdateUser 131
DeleteUser 132
DisableUser 132
EnableUser 133
GetUser 134
GetUsers 134
CreateCustomer 135
UpdateCustomer 136
DisableCustomer 136
EnableCustomer 137
DeleteCustomer 138
GetCustomer 138
GetCustomers 139
AddCustomers 140
AddUsers 140
RemoveCustomers 141
RemoveUsers 142
AddCustomerParties 142
RemoveCustomerParties 143
UpdateCustomerParties 144
AddParties 144
GetParties 145
RemoveParties 146
UpdateParties 147
UpdatePassword 147
2.7.3 Customer Pipelines 148
CreateUser 148
UpdateUser 149
DeleteUser 150
DisableUser 150
EnableUser 151
GetUsers 152
GetUser 152
CreateCustomer 153
GetCustomers 154
GetCustomer 154
UpdateCustomer 155
DeleteCustomer 155
DisableCustomer 156
EnableCustomer 156
AddCustomerParties 157
RemoveCustomerParties 157
UpdateCustomerParties 158
GetParties 159
AddParties 159
RemoveParties 160
UpdateParties 161
2.8 Product Sync 162
2.8.1 The Product Sync Domain Model 162
Class: Product 164
Class: ProductSpecifications 164
Class: ProductSpecification 165
Class: ProductClassification 165
Class: ProductType 165

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Class: ProductManufacturer 165
Class: ProductClassificationGroup 166
Class: ProductVariantSpecificaions 166
Class: ProductResource 166
Class: Division 166
Class: ProductRelation 167
Class: ProductRelationType 167
2.8.2 Product Sync Service Methods 167
SynchronizeProducts 167
SynchronizeProductList 168
SynchronizeProduct 168
SynchronizeArtifacts 169
2.8.3 Product Sync Pipelines 169
SynchronizeProducts 169
SynchronizeProductList 170
GetExternalCommerceSystemProductList 170
GetSitecoreProductList 171
SynchronizeArtifacts 171
SynchronizeManufacturers 171
SynchronizeClassifications 172
SynchronizeTypes 172
SynchronizeDivisions 173
SynchronizeResources 173
SynchronizeSpecifications 174
SynchronizeGlobalSpecifications 174
SynchronizeTypeSpecifications 175
SynchronizeClassificationSpecifications 175
SynchronizeProduct 177
SynchronizeProductManufacturers 177
SynchronizeProductType 178
SyncronizeProductClassifications 178
SynchronizeProductEntity 179
SynchronizeProductDivisions 179
SynchronizeProductResources 180
SynchronizeProductRelations 180
SynchronizeProductSpecifications 181
2.9 Gift Cards 182
2.9.1 The Gift Cards Domain Model 182
Class: GiftCard 183
Class: GiftCardPaymentInfo 183
2.9.2 Gift Cards Service Provider 183
GetGiftCard 184
2.9.3 Gift Cards Pipelines 184
GetGiftCard 184
2.10 Loyalty Programs & Cards 186
2.10.1 The Loyalty Programs & Cards Domain Model 186
Class: LoyaltyCard 188
Class: LoyaltyCardPaymentInfo 188
Class: LoyaltyCardTier 188
Class: LoyaltyCardTransaction 188
Class: LoyaltyProgram 189
Class: LoyaltyProgramStatus 189
Class: LoyaltyProgramSummary 190
Class: LoyaltyRewardPoint 190
Class: LoyaltyRewardPointEntryType 190
Class: LoyaltyTier 191
Class: PointBasedLoyaltyTier 191
Class: RewardPointType 191

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

2.10.2 Loyalty Programs & Cards Service Provider ... 191
GetLoyaltyPrograms ... 191
GetLoyaltyProgram ... 192
GetLoyaltyProgramStatus ... 193
JoinLoyaltyProgram ... 194
GetLoyaltyCards ... 194
GetLoyaltyCardTransactions ... 195
2.10.3 Loyalty Programs & Cards Pipelines ... 196
GetLoyaltyPrograms ... 196
GetLoyaltyProgram ... 196
GetLoyaltyProgramStatus ... 196
JoinLoyaltyProgram ... 197
GetLoyaltyCards ... 197
GetLoyaltyCardTransactions ... 198
2.11 Payments ... 199
2.11.1 Payments Domain Model ... 199
Class: PaymentMethod ... 201
Class: PaymentOption ... 201
Class: PaymentOptionType ... 201
Class: PaymentLookup ... 202
Class: PaymentPrice ... 202
Class: CardType ... 202
Class: PurchaseLevel ... 202
Class: TransactionType ... 203
2.11.2 Payments Service Provider ... 203
GetPaymentOptions ... 203
GetPaymentMethods ... 204
GetPricesForPayments ... 205
GetPaymentServiceUrl ... 206
GetPaymentServiceActionResult ... 207
2.11.3 Payments Pipelines ... 208
GetPaymentOptions ... 208
GetPaymentMethods ... 209
GetPricesForPayments ... 209
GetPaymentServiceUrl ... 209
GetPaymentServiceActionResult ... 210
2.12 Shipping ... 211
2.12.1 Shipping Domain Model ... 211
Class: ShippingMethod ... 212
Class: ShippingMethodPerItem ... 212
Class: ShippingOption ... 212
Class: ShippingOptionType ... 212
Class: LineShippingOption ... 213
Class: ShippingLookup ... 213
Class: ShippingPrice ... 213
2.12.2 Shipping Service Provider ... 214
GetShippingOptions ... 214
GetShippingMethods ... 214
GetShippingMethod ... 215
GetPricesForShipments ... 216
2.12.3 Shipping Pipelines ... 217
GetShippingOptions ... 217
GetShippingMethods ... 217
GetShippingMethod ... 217
GetPricesForShipments ... 218
2.13 Wish Lists ... 219
2.13.1 Wish Lists Domain Model ... 219
Class: WishList ... 220

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Class: WishListLine ... 220
Class: WishListHeader ... 220
2.13.2 Wish Lists Service Provider ... 221
CreateWishList ... 221
DeleteWishList ... 222
EmailWishLists ... 222
GetWishList ... 223
GetWishLists ... 224
UpdateWishList ... 224
AddLinesToWishList ... 225
UpdateWishListLines ... 226
RemoveWishListLines ... 227
PrintWishList ... 227
2.13.3 Wish Lists Pipelines ... 228
CreateWishList ... 228
DeleteWishList ... 229
EmailWishLists ... 229
GetWishList ... 230
GetWishLists ... 230
UpdateWishList ... 230
AddLinesToWishList ... 231
UpdateWishListLines ... 231
RemoveWishListLines ... 231
PrintWishList ... 232
2.14 Connect Configuration ... 233
2.14.1 Factories and entities ... 233
2.14.2 Pipelines for Methods ... 233

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Chapter 1

Introduction

Sitecore Commerce Connect is an e-commerce framework designed to integrate Sitecore with different external commerce systems and at the same time integrate customer engagement functionality provided in the Sitecore Experience Platform.

Note
In the following, Connect is used as an abbreviation for Sitecore Commerce Connect and ECS is used for External Commerce System.

Sitecore Commerce Connect 8.2.1 extends the 8.2 release by adding support for the new Sitecore Commerce product. Connect is embedded as a component in the Sitecore Commerce 8.2.1 product. For release 8.2.1, this document adds:

- Pricing section: adds classes and service methods associated with promotions.
- Order section: adds a cancellation class and a reorder pipeline.

For a general introduction and overview of the components in Connect, see the Sitecore Commerce Connect Components Overview.

This guide describes the API and configuration of Connect for frontend developers who create Sitecore solutions and are looking for information about how to use Connect

If you are a developer looking for information about how to integrate Connect with an external commerce system, see the Connect Integration Guide

- **1 — Introduction**
  This chapter contains an introduction for this guide as well as conceptual overviews.

- **2 — Using Connect to build e-commerce solutions**
  This chapter describes how to use the Connect API as a solution developer.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

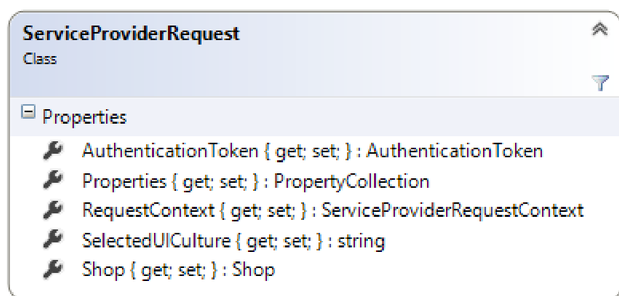
1.1 The Connect Service Layer

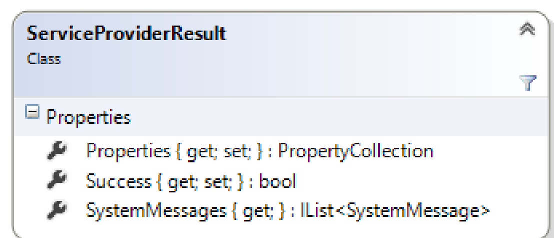
The Connect service layer is the means by which a storefront application communicates with the ECS. The service layer is broken down into functional areas each exposing their own sets of methods. The following functional areas exist:

1. Carts
2. Pricing
3. Order
4. Catalog
5. Globalization
6. Inventory
7. Customer
8. Product Sync
9. Gift Cards
10. Loyalty Programs & Cards
11. Payments
12. Shipping
13. Wish Lists

1.1.1 ServiceProviderRequest and ServiceProviderResult

Every service method takes an instance of a class derived from the ServiceProviderRequest and returns an instance of a class derived from the base ServiceProviderResult. Each functional area defines its own base class which contain the common properties of each request and response they expose.





ServiceProviderRequest

Name	Description
AuthenticationToken	The AuthenticationToken is automatically populated by the framework on every call, but can be explicitly set on every call if needed, which will take precedence. See the AuthenticationToken section below.
Properties	Key/value property collection. Can be used as an extensibility method for passing extra request parameters to the Connect service layer. If these are required, they would be defined by the ECS implementation.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

	NOTE: It's best practice to use strongly typed parameters by extending the ServiceProviderRequest parameter on every call as opposed to using the weakly typed approach of using the Properties collection.
RequestContext	This parameter allows to pass in additional request context that maybe required by an ECS specific implementation. Also serves as a means for processors to exchange information between them without affecting the original request. The RequestContext is passed along to each processor and any additional pipeline that may be called along the way. An example of how to use the RequestContext is provided later in this document.
SelectedUICulture	On each call, this property is automatically populated with the currently selected Sitecore language name.
Shop	The Shop information is automatically populated by the framework on every call, but can be explicitly set on every call if needed, which will take precedence. See the ShopProvider section below.

ServiceProviderResult

Name	Description
Properties	Key/value property collection. Can be used as an extensibility method for returning extra request parameters from the Connect service layer. If these are to be returned, they would be defined by the ECS implementation. NOTE: It's best practice to use strongly typed parameters by extending the ServiceProviderRequest parameter on every call as opposed to using the weakly typed approach of using the Properties collection
Success	Returns True if the call was successful; Otherwise False is returned. When False is returned, a System Message should provide context as to what has failed.
SystemMessages	System messages are to be returned when the call to the service layer fails. The ECS and base Connect implementations are the ones responsible for populating system messages. Sitecore specific errors may end up resulting in an uncaught exception depending on the how the ECS implementation was done.

The RequestContext

As mentioned earlier, the RequestContext of the ServiceProviderRequest can be used to exchange information between pipeline processors and pipelines.

Let's take an example from the CommerceServer Connect ECS to demonstrate how it uses the RequestContext to share the Basket amongst processors, therefore promoting modulariry of the processors and performance by making sure the Basket is loaded only once during the execution of a pipeline.

Let's take a look at a simplified version of the AddCartLines pipeline:

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

```
<commerce.carts.addCartLines>
  <!--  Setup the commerce server context  -->
  <processor type="Sitecore.Commerce.Connect.CommerceServer.Orders.Pipelines.ResolveBasket, Sitecore.Commerce.Connect.CommerceServer" />
  <processor type="Sitecore.Commerce.Connect.CommerceServer.Orders.Pipelines.AddLinesToCart, Sitecore.Commerce.Connect.CommerceServer" >
            <param desc="Rollup">True</param>
            <param ref="eaPlanProvider"/>
            <param ref="eaStateCartRepository"/>
  </processor>
  <processor type="Sitecore.Reference.Storefront.Connect.Pipelines.Carts.RunPipeline, Sitecore.Reference.Storefront.Powered.by.CommerceServer">
            <param desc="pipelineName">Basket</param>
  </processor>
  <processor type="Sitecore.Reference.Storefront.Connect.Pipelines.Carts.RunTotalPipeline, Sitecore.Reference.Storefront.Powered.by.CommerceServer">
            <param desc="pipelineName">Total</param>
  </processor>
  <processor type="Sitecore.Commerce.Pipelines.Carts.Common.RunSaveCart, Sitecore.Commerce"/>
</commerce.carts.addCartLines>
```

In the above example, the first processor of every cart related operation makes use of a ResolveBasket processor. This processor is responsible for loading the Commerce Server basket and setting up the request context. All of the other processors will fetch the Commerce Server basket from the request context.

The last processor RunSaveCart calls another pipeline to save the basket. The request context will automatically flow to this pipeline and it will be able to fetch the basket in order to save its content.

Let's start by looking at the CartPipelineContext utility class:

```
        public class CartPipelineContext : PipelineContext
        {
           protected CartPipelineContext()
              : base(PipelineContextNames.CartPipelineContext)
           {
              this.HasBasketErrors = false;
           }

           public Basket Basket { get; set; }

           public static CartPipelineContext Get(ServiceProviderRequestContext requestContext)
           {
              CartPipelineContext pipelineContext = 
requestContext.Properties[PipelineContextNames.CartPipelineContext] as CartPipelineContext;

              if (pipelineContext == null)
              {
                 pipelineContext = new CartPipelineContext();

                 requestContext.Properties[PipelineContextNames.CartPipelineContext] = 
pipelineContext;
              }

              return pipelineContext;
            }
         }
```

In the above code, the static Get() method provides a useful way for creating and retrieving the CartPipelineContext instance from the RequestContext. Notice that a key/value pair is added to the Properties collection of the request context. It is important to make this name unique to make sure it does not enter in conflict with other request context items that could be added by other processors.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Now let's look at snipet of code from the ResolveBasket processor:

```
IOrderRepository repository = CommerceTypeLoader.CreateInstance<IOrderRepository>();

var cartContext = CartPipelineContext.Get(args.Request.RequestContext);

cartContext.Basket = repository.GetBasket(Guid.Parse(userId), cartName);
```

Since the call to the CartPipelineContext.Get() is issued from the 1st processor, the context class is created and added to the RequestContext collection. Next, the code simply sets the cartContext.Basket to the actual Commerce Server basket.

All other processors, including the RunSaveCart pipeline simply perform the following to retrieve the Basket from the RequestContext:

```
var cartContext = CartPipelineContext.Get(request.RequestContext);
Assert.IsNotNull(cartContext, "cartContext");
Assert.IsNotNull(cartContext.Basket, "cartContext.Basket");
```

At this point, the cartContext.Basket can be used to perform the necessary operation (such as adding a line item) on the Commerce Server basket.

The ShopProvider

Connect introduces the concept of a shop provider. The role of the shop provider is to provide shop information to the service layer on every call.

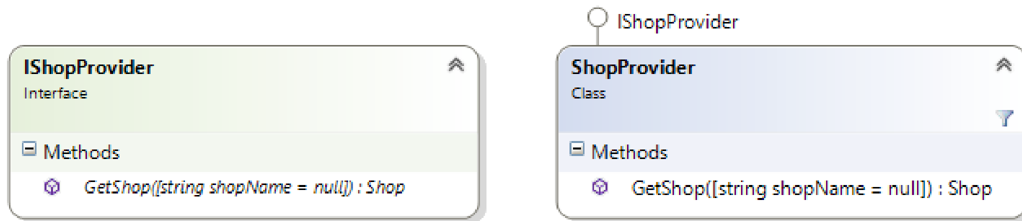
The shop provider is configurable and a specific storefront implementations can be introduced when necessary.

Sitecore.Commerce.config defines the following provider and shop entity configuration:

```
<shopProvider type="Sitecore.Commerce.Providers.ShopProvider, Sitecore.Commerce"
singleInstance="true"/>

<commerce.Entities>
    <Shop type="Sitecore.Commerce.Entities.Shop, Sitecore.Commerce" />
</commerce.Entities>
```

A shop provider implements the *IShopProvider* interface. A single method needs to be implemented called: GetShop()

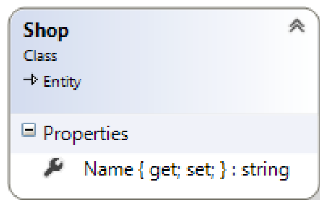
Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.



Name	Description
shopName	When specified, the shop provider must return information for the given *shopName.* When <null>, it must return information about the current shop.

The default implementation returns an instance of the Shop class and populates the *ShopName* property. The ShopName is by default taken from the Sitecore.Context.Site.Name value or an empty string is if it is not specified. When given a *shopName*, it simply populates the returned shop information with this information. This should be adequate for most storefront implementations.

The base *ServiceProviderRequest* class exposes a *Shop* property and is populated by virtue of calling any service provider method.



Name	Description
Name	The name of the shop.

Since every service request now populates the Shop information, the ShopName of existing requests have been marked as obsolete. In order not to break any existing implementation of Connect the ShopName properties of existing requests will keep working until we remove it in a future version. This means that all existing code that set the ShopName explicitely will continue working until the property is removed from all requests.

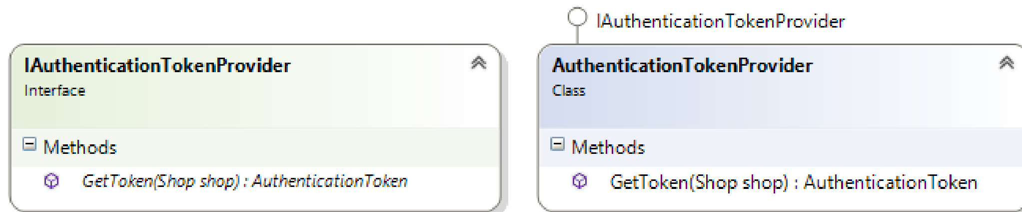
The AuthenticationTokenProvider

To support authentication by using tokens in all of the Commerce Connect APIs, an AuthenticationTokenProvider has been introduced. It consists of an interface defined in Sitecore.Commerce.Config and a default implementation which always returns NULL, meaning no token is provided.

Sitecore.Commerce.config defines the following authentication provider and AuthenticationToken entity configuration:

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

```
<authenticationTokenProvider type="Sitecore.Commerce.Providers.AuthenticationTokenProvider,
Sitecore.Commerce" singleInstance="true"/>
<commerce.Entities>
        <AuthenticationToken type="Sitecore.Commerce.Entities.AuthenticationToken,
Sitecore.Commerce" />
</commerce.Entities>
```



Name	Description
GetToken	Must return the token if one is required.

The introduction of the provider centralizes the logic of providing the Token. The base class ServiceProviderRequest that is used in all service methods as input parameter has been extended with an AuthenticationToken property. The property can be set explicitly on every call made through Connect. If the property is not set, the AuthenticationTokenProvider is instantiated and called, and the return value is assigned to the property.

The AuthenticationToken is defined as a class that can be customized for extensibility purposes.

The following is an example of how to explicitely pass in the authentication token into a service call:

```
        var request = new GetCustomerRequest(customerId);
        var token = this.EntityFactory.Create<AuthenticationToken>("AuthenticationToken");

        token.Token = "mytoken";

        request.AuthenticationToken = token;

        var result = this.CustomerServiceProvider.GetCustomer(request);
```

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

1.2 Page Events, Goals, Outcomes and Custom Analytics Data

Connect will generate page events, goals and outcomes at key points when running a storefront and the API is called. Examples are cart manipulation page events that record contact cart activities, visitor order creation goal and outcome, site language selection, searches, selected sort orders, etc. Many of the visitor activities are recorded as they navigate the web site.

The recorded activities are then used to feed the Experience Profile Commerce reports as well as the Experience Analytics Commerce dimensions, which are used to report on. While Sitecore provides a lot of important information around the visitor interactions, Connect extends it with additional events, goals and outcomes and augments the recorded data with its own custom values adding Connect state information to the visitor interactions.

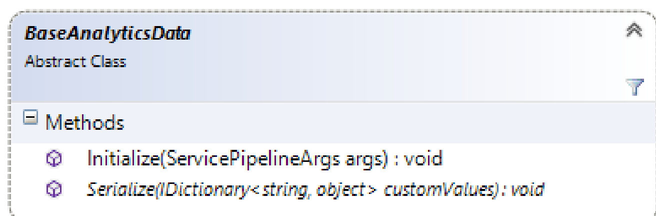
This is where the custom Analytics Data and Outcome Data classes are used. Each of these classes provide an abstraction and a mechanism to serialize data into Sitecore interactions and a mean to rehydrate the serialized values. This is currently used for reporting purposes but can be customized and used in different kinds of scenarios for searching and filtering when segmenting contacts and personalization

NOTE:
The classes provide an abstraction and a strongly typed interface to the analytics data collected in Connect and should always be used to access the data. It is *not* considered best practice to access the data directly in custom values and Sitecore may change the implementation of what data and how the data is persited and retrieved. The result of directly Accessing the data in custom values the result is undefined

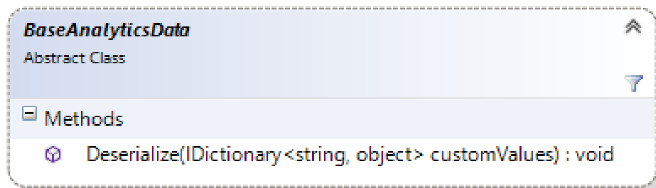
The Analytics Data and Ourcome Data classes can be found in the *Sitecore.Commerce.AnalyticsData* and *Sitecore.Commerce.OutcomeData* namespaces of the Connect framework.

1.2.1 Analytics Data Classes

The Custom Analytics Data classes expose two methods when data is being serialized into the interaction and one method to rehydrate the content.

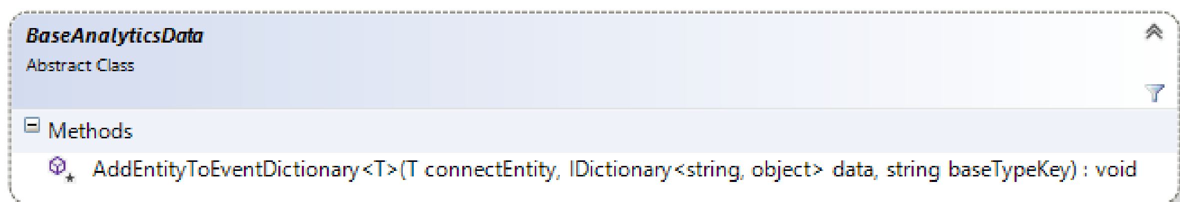


Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.



Name	Description
Initialize	In the context of a page event, call this method and pass in the arguments received during the call to the service layer. It's the responsibility of the specific AnalyticsEntity object to know how to parse the Request and Result objects to extract the data to be serialized. **NOTE:** In some implementations, an overloaded initialize has been created to accommodate for different input parameters such as an explicit order not retrieved back from the ECS .
Serialize	Call this method to serialize the content into the given dictionary which is then written into the interaction's page event or goal custom values.
Deserialize	Callthis method to rehydrate the content of the data class. Currently used by the Connect Experience Profile and Experience Analytics reporting components

When data is being serialized, the following utility methods are available:

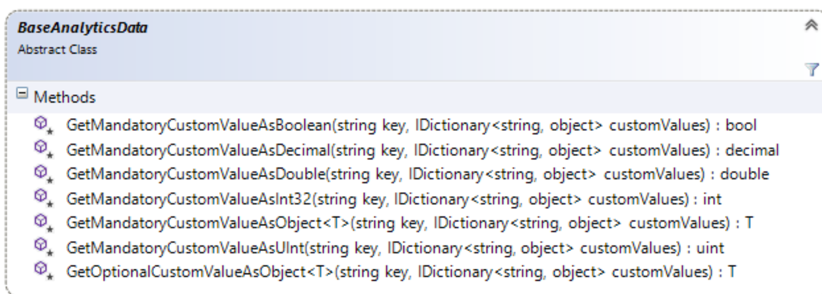


Name	Description
AddEntityToEventDictionary<T>	This templated method is used when adding Connect entities to the custom values dictionary. **NOTE** It is very important to use this method instead of adding Connect entities directly. This method does a few things. First, it saves the base entity values only, stripped out of any extension that might have been made by ECS. This because in a distributed environment, the ECS specific data types may not be available and would cause problems on the reporting and processing servers during the deserialization phase. In the case where both these servers are in your own domain and control, having the original "custom" ECS entity may be desirable. In this case, the following setting is available in the *Sitecore.Commerce.config* file:

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

	<setting name="Commerce.Analytics.EntitiesIncludedInXDB" value="Base"/> Valid values are: 1. Base 2. Custom 3. Both *Base* - saves the base entity only. *Custom* - saves the original given entity. *Both* - saves the base entity and original version. By default, *Base* is configured. For more information, see topic "Experience Profile Commerce tab - the underlying technology" on doc.sitecore.net

When data is being rehydrated, the following utility methods should be used:



The above methods allow to retrieve a value from the serialized list for any type that you require. The *GetMandatory** methods add some extra error handling code in case the mandatory parameter is not found, such as adding proper null checks and error logging.

The *Get Optional* method is the same as the GetMandatory** methods except no error logging will occur if the *key* is not present.

The following shows the Connect implementation of the order goal that is registered when a new visitor order is created :

```
public class OrderGoalAnalyticsData : BaseAnalyticsData
{
    public string ExternalId { get; set; }

    public string ShopName { get; set; }

    public Total Total { get; set; }

    public Order Order { get; set; }
```

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

```
            public bool? IsOfflineOrder { get; set; }

            public override void Initialize(Pipelines.ServicePipelineArgs args)
            {
                base.Initialize(args);

                this.Order = this.GetOrderFromArgs();

                this.ExternalId = this.Order.ExternalId;
                this.ShopName = this.Order.ShopName;
                this.Total = this.Order.Total.ToBaseType<Total>();
                this.IsOfflineOrder = this.Order.IsOfflineOrder;
            }

            public virtual void Initialize(Order order)
            {
                this.ServicePipelineArgs = null;

                this.Order = order;

                this.ExternalId = this.Order.ExternalId;
                this.ShopName = this.Order.ShopName;
                this.Total = this.Order.Total.ToBaseType<Total>();
                this.IsOfflineOrder = this.Order.IsOfflineOrder;
            }

            public override void Serialize(IDictionary<string, object> customValues)
            {
                customValues.Add(Constants.KnownPageEventDataNames.ExternalId,
this.ExternalId);
                customValues.Add(Constants.KnownPageEventDataNames.ShopName, this.ShopName);
                customValues.Add(Constants.KnownPageEventDataNames.Total, this.Total);
                customValues.Add(Constants.KnownPageEventDataNames.IsOfflineOrder,
this.IsOfflineOrder);

                this.AddEntityToEventDictionary<Order>(this.Order, customValues,
Constants.KnownPageEventDataNames.Order);
            }

            public override void Deserialize(IDictionary<string, object> customValues)
            {
                this.ExternalId =
this.GetMandatoryCustomValueAsObject<string>(Constants.KnownPageEventDataNames.ExternalId,
customValues);
                this.ShopName =
this.GetMandatoryCustomValueAsObject<string>(Constants.KnownPageEventDataNames.ShopName,
customValues);
                this.Total =
this.GetMandatoryCustomValueAsObject<Total>(Constants.KnownPageEventDataNames.Total,
customValues);
                this.Order =
this.GetOptionalCustomValueAsObject<Order>(Constants.KnownPageEventDataNames.Order,
customValues);

                var isOfflineOrder =
this.GetOptionalCustomValueAsObject<bool?>(Constants.KnownPageEventDataNames.IsOfflineOrder,
customValues);
                this.IsOfflineOrder = isOfflineOrder ?? false;

                base.Deserialize(customValues);
            }

            protected virtual Order GetOrderFromArgs()
            {
                Assert.IsTrue(this.ServicePipelineArgs.Result is SubmitVisitorOrderResult,
"args.Result is SubmitVisitorOrderResult");

                return ((SubmitVisitorOrderResult)this.ServicePipelineArgs.Result).Order;
            }
        }
```

There are a few things worth mentioning:

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

1. Two Initialize methods exist. The first takes the ServicePipelineArgs as it is being called by the trigger page event mechanism. The second one takes an Order as it is used during the registering of off line orders process.
2. When setting the Total property, the *ToBaseType<Total>()* is used. This is important as we do not want custom extensions to be saved in the page event as these data types may not be available on the reporting and processing servers (multi-server deployment).
3. The *Serialize* method makes use of the *AddEntityToEventDictionary()* template method to add the order to the page event.

The following code snipet demonstrate how you would consume the analytics data.

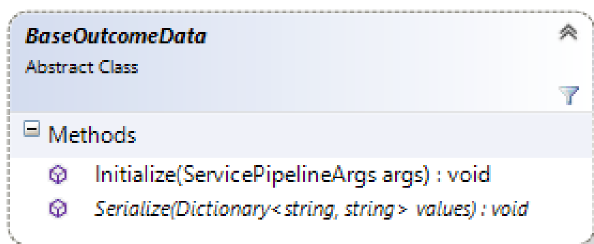
```
var customData = AnalyticsDataInitializerFactory.Create<SearchAnalyticsData>();

customData.Deserialize(pageEventData.CustomValues);
```

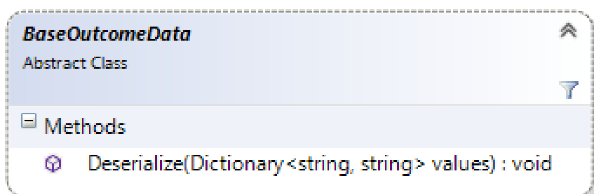
The above assumes you have the page event data from the Sitecore interaction. After calling the *Deserialize* method, the SearchAnalyticsData object properties will be populated with the information.

1.2.2 Outcome Data Classes

The Outcome Data Classes are very similar in construct than Analytics Data Classes.

The Outcome Data Classes expose two methods when data is being serialized into the interaction and one method to rehydrate the content.



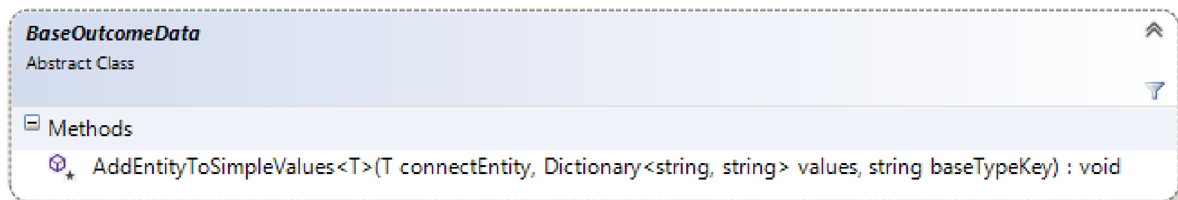


Name	Description
Initialize	In the context of triggering an oucome, call this method and pass in the arguments passed into the service request. It's the responsibility of the specific Outcome Data object to know how to parse the Request and Result objects to extract the data to be serialized.
Serialize	Call this method to serialize the content into the given dictionary which is then written into the interaction's page event or goal custom values.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Deserialize	Call this method to rehydrate the content of the data class. Currently used by the Connect Experience Profile and Experience Analytics reporting components

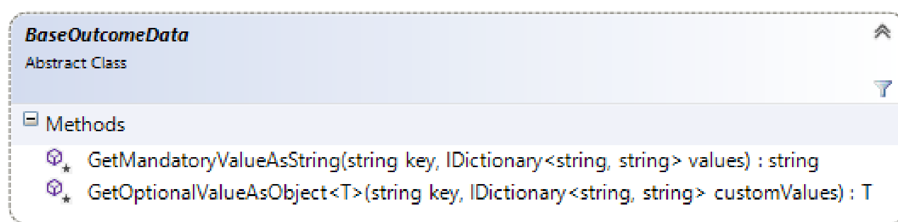
When data is being serialized, the following utility methods are available:



Name	Description
AddEntityToSimpleValues<T>	This templated method is used when adding Connect entities to the custom values dictionary. **NOTE** It is very important to use this method instead of adding Connect entities directly. This method does a few things. First, it saves the base entity values only, stripped out of any extension that might have been made by ECS. This because in a distributed environment, the ECS specific data types may not be available and would cause problems on the reporting and processing servers during the deserialization phase. In the case where both these servers are in your own domain and control, having the original "custom" ECS entity may be desirable. In this case, the following setting is available in the *Sitecore.Commerce.config* file: <setting name="Commerce.Analytics.EntitiesIncludedInXDB" value="Base"/> Valid values are: 1. Base 2. Custom 3. Both *Base* saves the base entity only. *Custom* saves the original given entity. *Both* saves the base entity and original version. By default, *Base* is configured. The outcome custom data is always a string and is handled by this method call. For more information, see topic "Experience Profile Commerce tab - the underlying technology" on doc.sitecore.net

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

When data is being rehydrated, the following utility methods should be used:



The above methods allow to retrieve a value from the serialized list for any type that you require. The *GetMandatoryValueAsString* adds some extra error handling code in case the mandatory parameter is not found, such as adding proper null checks and logging.

The *GetOptionalValueAsObject* method is the same as the *GetMandatory** methods except no error logging will occur if the *key* is not present.

The following shows the Connect implementation of the visitor order created outcome that is registered when a new visitor order is created:

```
public class SubmittedOrderOutcomeData : BaseOutcomeData
{
   public string ExternalId { get; set; }

   public string ShopName { get; set; }

   public Order Order { get; set; }

   public override void Initialize(Pipelines.ServicePipelineArgs args)
   {
       base.Initialize(args);

       this.Order = this.GetOrderFromArgs();
       if (this.Order != null)
       {
           this.ExternalId = this.Order.ExternalId;
           this.ShopName = this.Order.ShopName;
       }
   }

   public void Initialize(Order order)
   {
       this.ServicePipelineArgs = null;

       this.Order = order;
       if (this.Order != null)
       {
           this.ExternalId = this.Order.ExternalId;
           this.ShopName = this.Order.ShopName;
       }
   }

   public override void Serialize(Dictionary<string, string> values)
   {
       if (this.Order != null)
       {
```

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

```
                   values[Constants.KnownPageEventDataNames.ExternalId] = this.ExternalId;
                   values[Constants.KnownPageEventDataNames.ShopName] = this.ShopName;

                   this.AddEntityToSimpleValues<Order>(this.Order, values, 
Constants.KnownPageEventDataNames.Order);
               }
           }

           public override void Deserialize(Dictionary<string, string> values)
           {
               this.ExternalId = 
this.GetMandatoryValueAsString(Constants.KnownPageEventDataNames.ExternalId, values);
               this.ShopName = 
this.GetMandatoryValueAsString(Constants.KnownPageEventDataNames.ShopName, values);

               this.Order = 
this.GetOptionalValueAsObject<Order>(Constants.KnownPageEventDataNames.Order, values);

               base.Deserialize(values);
           }

           public override decimal GetMonetaryValue()
           {
               return this.Order.Total == null ? 0m : this.Order.Total.Amount;
           }

           public virtual bool? IsOfflineOrder()
           {
               bool? isOfflineOrder = false;

               if (this.Order != null)
               {
                 isOfflineOrder = this.Order.IsOfflineOrder;
               }

               return isOfflineOrder;
           }

           protected virtual Order GetOrderFromArgs()
           {
               if (this.ServicePipelineArgs != null && this.ServicePipelineArgs.Result is 
SubmitVisitorOrderResult)
               {
                   return ((SubmitVisitorOrderResult)this.ServicePipelineArgs.Result).Order;
               }

               return null;
           }
       }
```

There are a few things worth mentioning:

4. Two Initialize methods exist. The first takes the ServicePipelineArgs as it is being called by the trigger outcome mechanism. The second one takes an Order as it is used during the registering of off line orders process.
5. The *Serialize* method makes use of the *AddEntityToSimpleValues()* template method to add the order to the outcome.

The following code snippet demonstrate how you would consume the outcome custom data.

```
    Dictionary<string, string> customValues =
      JsonConvert.DeserializeObject<Dictionary<string, string>>
      (contactOutcome.CustomValues[Constants.KnownAnalyticsKeys.SitecoreCommerceOutcomeKey]);

    var outcomeData = OutcomeDataFactory.Create<SubmittedOrderOutcomeData>();
    outcomeData.Deserialize(customValues);
```

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

The above assumes you have the Sitecore outcome for the contact. The ContactOutcome class contains the custom value information in a serialized format. This means you need to deserialize the custom value dictionary prior to calling the SubmittedOrderOutcomeData deserialize method. In the near future, we will encapsulate this functionality in the BaseOutcomeData class.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

1.3 Federated payments

Federated payments are a method for processing commerce order payments that use a 3rd party service to capture, store, and process customer payment information. When you use a federated payment service, the merchant web site does not have any direct access to customer payment information, such as card number or card security code (CVV, CSC, and so forth).

Instead, the federated payment service receives this information directly from the customer, securely saves that information, and sends a one-time payment authorization code back to the merchant web site. This authorization code can safely be saved by the merchant web site for order fulfillment because it cannot be used for any other purchases. The advantage this offers over other payment processing methods is that sensitive customer payment information is never accessible by the merchant website, thereby reducing liability and the security attack surface.

1.3.1 Order processing

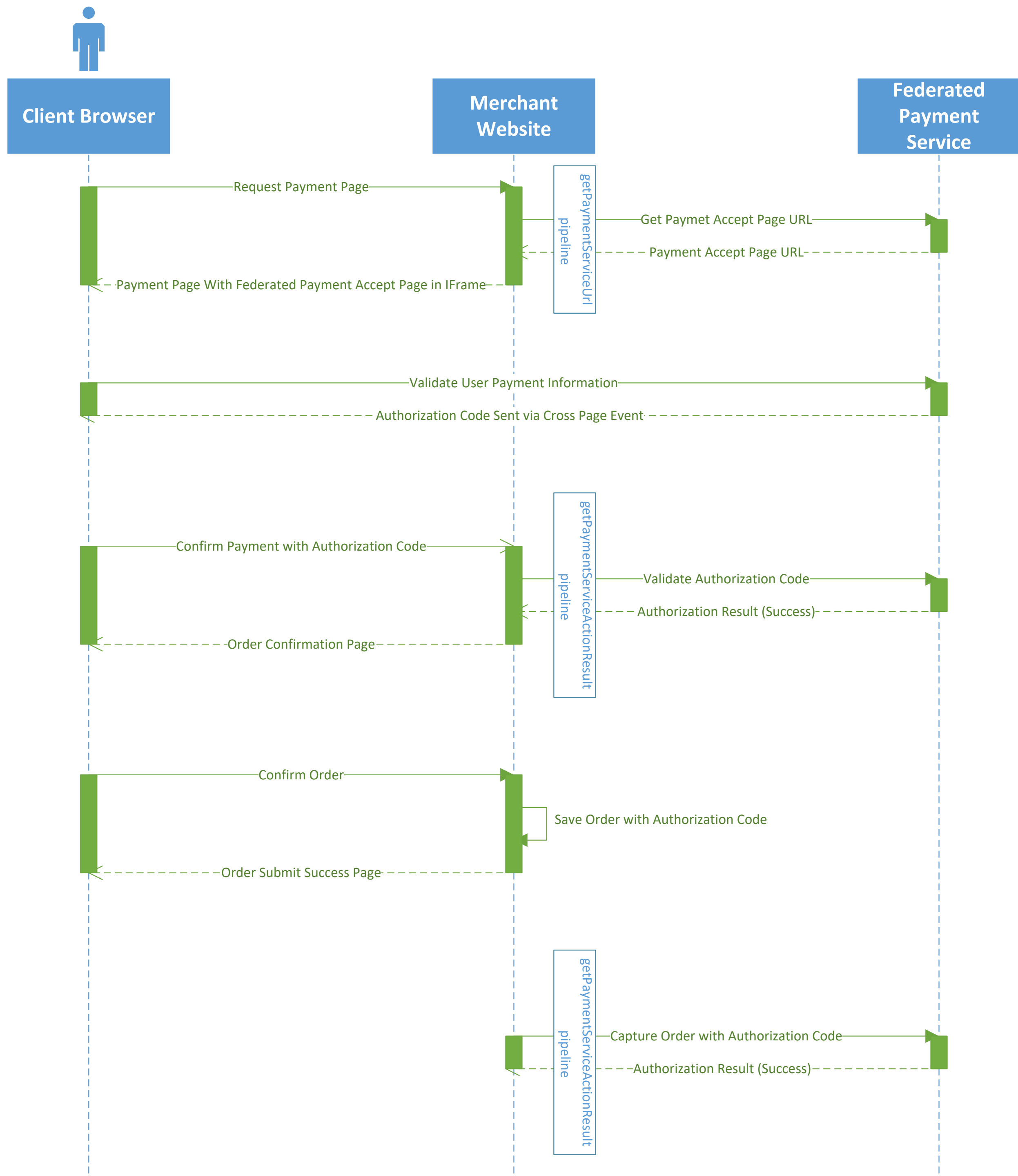
Processing an order using a federated payment service is a multi-step process that begins in the payment step of a checkout.

Commerce Connect helps facilitate implementation of a merchant web site that accepts federated payments by providing a framework for communication between the merchant web site and a federated payment service. This framework contains two methods implemented by the PaymentServiceProvider class, and their corresponding Sitecore pipelines:

- GetPaymentServiceUrl calls the commerce.payments.getPaymentServiceUrl pipeline, and is used to prepare the federated payment service for a customer interaction and retrieve the URL to the payment service payment acceptance page.
- GetPaymentServiceActionResult calls the commerce.payments.getPaymentServiceActionResult pipeline, and retrieves result of the customer interaction with the payment service and captures order payment once the order has been fulfilled.

Commerce Connect does not provide any processors that implement the actual interaction with any commercial payment providers. It is the responsibility of the merchant website or the commercial payment provider to provide these implementations. In addition to the federated service provider implementation of these pipelines, client side JavaScript code is required by the merchant website to respond to events raised by the payment service provider. The following diagram outlines the interaction between the client, the Commerce web service, and the federated payment service:

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.



In the first set of interactions with the customer client browser, the merchant website calls the GetPaymentServiceUrl method of the PaymentServiceProvider class. As part of this request, the merchant website can also provide customer information (such as payment address, supported card types, preferred currency, and so forth) so that the customer does not need to reenter it when interacting with the federated payment service. The result of this request is the URL to the payment acceptance page of the federated payment service. This URL is then used as the source of an iframe on the merchant website's payment page.

In the next step, the customer enters payment information in the iframe that hosts the federated payment service's payment acceptance page. Iframes are isolated components that restrict

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

communications between the main page and the hosted content. In this situation, use of an iframe in this step ensures that the merchant website never has access to the customer's payment information, such as card number, CVV, etc. Instead, all of this information goes directly to the federated payment service, where it can be securely stored.

If errors occur when authorizing the customer payment information, a cross page event will be raised that contains the error details (invalid CVV, insufficient funds, etc.). If the payment information is successfully validated, a cross page event will be raised that contains a one-time, one-merchant authorization code that can be used to process the payment. You can capture and interpret these events in JavaScript code in the client web browser.

Note

An agreement with a commercial payment service is typically required before this step is possible. Contact your payment provider to obtain details.

Next, the authorization code captured by the JavaScript is sent back to the merchant website. The merchant website then uses the GetPaymentServiceActionResult of the PaymentServiceProvider to verify the authorization code. This serves two purposes:

- It is an extra safeguard to ensure that malicious code on the client web browser cannot use a phony authorization code to complete a purchase.
- It allows the merchant website to retrieve a tokenized version of the customer payment information, which can be used to process future payments on the merchant website. This is essentially the same as customer asking the merchant website to save payment information so it does not need to be entered again manually. Once the authorization code is validated, the customer is then redirected to the order confirmation page of the merchant website.

Once the customer confirms the order, it is the responsibility of the merchant website to save the payment authorization code and associate it with the order placed. This authorization code will be required to capture the customer payment once the order is ready to be fulfilled. This is the last step in the order. When the order is fulfilled, the payment authorization code can then be used to finalize the customer payment with the federated payment service. Once the federated payment service confirms that the authorization code is still valid and the customer has sufficient funds, the order can be finalized.

A sample implementation of these interactions can be found in the Commerce StarterKit project on GitHub (https://github.com/Sitecore/Commerce-Connect-StarterKit/tree/release/8.2.281/master).

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Chapter 2

Using Connect to build e-commerce solutions

This chapter describes the Connect API which consists of a number of abstract service layers. Each section in this chapter describes a service layer API, associated pipelines, its relevant classes and how to use it, and a how-to section on how to configure Connect.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

2.1 Carts

2.1.1 Cart Domain Model

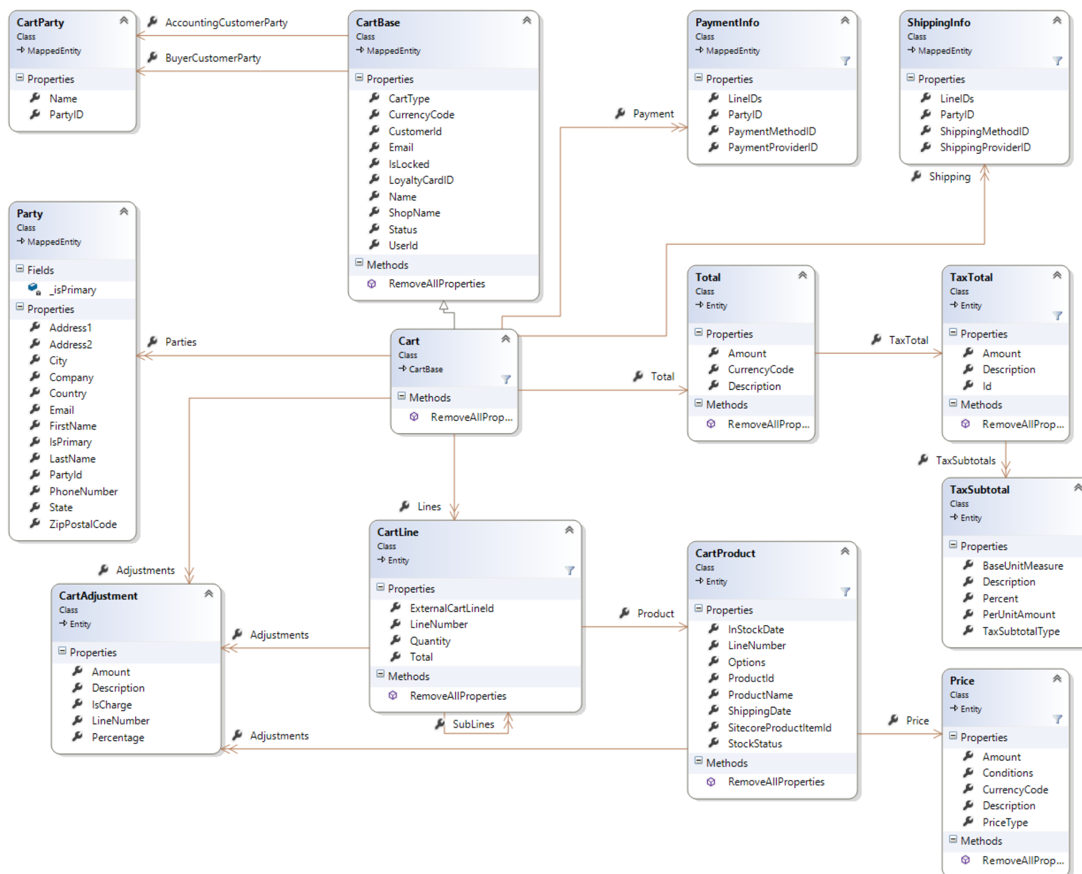
This chapter describes the domain model that represents the cart and its constituent Data Transfer Objects (DTO). The domain model is used when a Sitecore developer needs to interact with the cart and transfer cart data back and forth to the ECS. All business logic for carts is implemented in the service layer API, which must be used to manipulate carts.

Some examples are of cart domain model usage:

- To collect information from visitors
- To display information in renderings
- To use information for personalization
- To pass cart information to the external commerce system
- To return cart total from pricing service layer

Note: Because cart has strong references to Price and Total they are included in the domain model, but methods for calculating total and getting prices are part of the Pricing integration service layer.

Note: The class diagram below shows the domain model. All the Cart classes, prefixed with Cart, to the left, are described in this document, whereas the Price, Total and TaxTotal classes to the right are described in the Pricing document and is manipulated by the pricing API as well as the Party, ShippingInfo and PaymentInfo are part of the Customer, Shipping and Payment service layers respectively. For more information see the respective service layer documents for more information.



Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Note: The domain model consists of classes that make up the contracts with the external system. The contracts are defined as classes instead of interfaces to allow the model to be easily extended later if needed. This follow the best practice guidelines defined in the book Framework Design Guidelines.

Default implementation of the contracts are delivered as part of Connect. If an actual Connect provider with an external commerce system contains more functionality than provided by default, the implementation can be replaced. All instantiation of actual classes will be handled using the Factory design pattern.

IReadOnlyCollections and service API

Most collections in the domain model are specified as IReadOnlyCollections. This is done to avoid developers modifying these collections directly. All manipulation must be done through calling of the service API.

Cart Keys

A cart has two unique composite keys:

- CartId (aka. ExternalCartId), ShopName
  Some methods like LoadCart use this key to locate a cart.
- UserId, CustomerId, CartName, ShopName
  Some methods like GetCarts and CreateCart uses this key

Class: Cart Base

The Cart Base represents a summary of the main cart data and is used as return values from GetCarts and CreateOrResumeCart in case of multiple matching carts.

The Cart class is inherited from Cart Base. For more information, see class Cart below.

Name	Type	Description
Name	String	Can be used to name a cart. The property is optional to use, but can be used in solutions where a single visitor/user can have multiple carts. In order to differentiate between the carts, a string value must be provided.
UserId	String	The identifier for the user of the cart, can be: • the contact Id or Identifier provided by Sitecore • A user id provided by Sitecore • A user id provided by ECS
CustomerId	String	The identifier for the customer of the cart. The customer cannot manipulate the cart directly, but only indirectly through the user. For more information, see the Connect roles document:
ShopName	String	The shop name where the cart belongs. Is used to enable multi-shop support. Can in implementation correspond to website name if there is a one-2-one mapping between website and shop. All API methods take ShopName as parameter.
IsLocked	Boolean	Indicates if the cart is locked or not A cart is typically locked during part of the checkout process, especially after initiation of payment

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

CartType	String	Contains type values for different usage of cart. Example values are: Regular, Gift card, Wish list, Recurring-cart. In default implementation only “Regular” will be used and tested Can be extended later and customized in the actual Connect provider implementation.
CartStatus	String	A status code: InProcess, Abandoned. Can be extended later and customized in the actual Connect provider implementation. The value is updated in the Abandoned Cart EA plan, if a cart is resumed from abandoned state
CurrencyCode	String	Represents the currency code of items in the cart.

Class: Cart

The Cart is responsible for representing the collection of products a customer is planning to buy. It is used when the content of the cart needs to be displayed or when Sitecore needs to determine the content of the cart for any other purpose (such as personalization).

The Cart class inherits its base cart data from Cart Base. For more information, see class Cart Base Above.

Name	Type	Description
CartLines	IReadOnlyCollection<CartLine>	Contains the cart lines.
Adjustments	IReadOnlyCollection<Adjustment>	Collection of adjustments that describe any discrepancies between the total and the sum of the prices of the individual cart line items. Examples are coupon codes (manually triggered), and shipping surcharges (automatically triggered)
CartTotal	Total	Represents the total value for the products in the cart.

Class: Cart Line

The Cart Line represents a line in the shopping cart. It represents something that a visitor has added to his cart, along with the quantity of the item that was added to the cart. It also represents the position of the line relative to other lines in the cart (for controlling the order the lines appear when the lines are displayed).

Name	Type	Description
ExternalCartLineId	String	Unique identifier for the cart line in the commerce system. Will be empty when adding cart lines and can be set by the ECS. Can be specified when removing cart lines Can be specified when updating cart lines

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Quantity	Unsigned integer	The quantity of the specific cart product in the cart.
Product	CartProduct	The cart line product. The Cart Product object contains the reference to the actual product
LineNumber	Unsigned integer	The position of the cart line in the cart. Can be specified when removing cart lines Can be specified when updating cart lines Can be empty when adding cart lines
Adjustments	IReadOnlyCollection<Adjustment>	Collection of adjustments that describe any discrepancies between the total and the cart item price. The actual content of the adjustments are provided by the ECS.
Total	Total	Represents the total for the product represented by the cart product (sub-item).This value will be the same as the total on the cart product (sub-item) multiplied by the quantity including list of Adjustments. **Read-only**. Is reserved for Pricing Service Layer. Should not be supplied when adding, removing or updating cart lines
SubLines	IReadOnlyCollection<CartLine>	Recursive sub-lines For example, if the cart item is a bundled product, the sub-lines are the individual products that make up the bundle. Another example is services like insurance that is added to a product

Class: Cart Product

The Shopping Cart Product represents a product in a Shopping Cart Line, or a sub-item of a Shopping Cart Product in case of bundling or insurance etc.

Name	Type	Description
ProductId	String	Unique identifier for the product in the external commerce system.
SitecoreProductItemId	ID	Shortcut for referencing the product item in Sitecore directly
Options	IReadOnlyCollection<Option>	Represents visitor-specified product options (such as engraving=Adam)
LineNumber	Unsigned integer	Gets the position of the cart item in the cart.
Adjustments	IReadOnlyCollection<Adjustment>	Collection of adjustments that describe any discrepancies between the total and the price of the cart item's product.
Price	Price	Contains the product price. Can be supplied when adding or updating cart lines to cart, but can also be set by Pricing service layer during call to GetCartTotal

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Class: Cart Adjustment

A cart adjustment describes a charge or a discount given on a Cart, Cart Line or Cart Line Product.

Examples are:

- Coupon codes (manually triggered) on the Cart or the Cart Line
- Shipping surcharges on the Cart or Cart Line (automatically triggered)

The adjustment can be a fixed amount or a percentage, but not both at the same time. If both an amount and percentage is applied, then two adjustments must be given and the LineNumber specifies the order in which they are applied

Name	Type	Description
Amount	Decimal	Amount that needs to be added or subtracted to sub-total. Only Amount or Percentage (see below) can be used at any one time
Description	String	A description of the charge or discount applied. Description could include a reference to coupon code, if that's the scenario
IsCharge	Boolean	Indicates it is a charge if true, otherwise discount
Percentage	Float	A percentage that needs to be added or subtracted. Only Amount or Percentage can be used at any one time
LineNumber	Unsigned integer	If multiple adjustments are given the LineNumber specifies the order in which the charge or discount is applied. The information stored in adjustments is used for presentation, showing calculation order and not for actually calculation to the cart.

Class: Cart Option

Option represents a configurable or variable value that is specific to an item in the cart. An example of an option is engraving on a personalized product. When the "engraving" service is added to the cart, the text to engrave is also required. The Option represents this.

If an option has an additional charge, the option should be represented by a separate Cart Line Product. The option is only used to describe variables or settings on a specific Cart Line Product, not to handle adjustments or pricing changes.

Name	Type	Description
OptionId	String	An internal ID for the option, typically provided by Sitecore when the user adds an option to the cart line
Description	String	An optional description referring to the type of option. For example, Engraving, Name on t-shirt etc.
Value	String	The actual custom value

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

2.1.2 Cart Service Provider

Service providers are wrapper objects designed to make it easier to interact with Connect pipelines. The providers implement no logic other than calling Connect pipelines. All of the business logic is implemented in the pipeline processors.

For each method in the provider there is a corresponding Request and Result object used, ex. GetCarts takes a GetCartsRequest object and returns a GetCartsResult object. In some cases the response objects are re-used when returning the same data.

Customized versions of the default request and result arguments can be used by calling the overloaded generics based versions of the methods.

The Cart Service Provider contains the following methods for interacting with cart data.

GetCarts

GetCarts is used to query Cart data against the external commerce system and **doesn't** return a collection of Carts, but a collection of CartBase objects that only contains the summary of the main cart data.

Name:	**GetCarts**
Description:	Gets the carts that match the specified criteria. Calls the pipeline "GetCarts"
Usage:	Called when a list of carts is needed. Examples include: • Getting the carts for a specific visitor across all visitors • Getting one of the carts for the current visitor in a multi-cart solution • Getting the carts that have not been used within a period of time, for example, abandoned
Signature:	GetCartsResult GetCarts(GetCartsRequest request)
Input:	
	UserId – Optional - The ids of the users whose carts should be retrieved. If no value is specified, the user IDs are not considered when retrieving carts.
	CustomerId – Optional – The ids of the customers whose carts should be retrieved. If no value is specified the customer IDs are not considered when retrieving carts.
	CartName – Optional - The names of the carts that should be retrieved. If no value is specified, the cart names are not considered when retrieving carts.
	CartStatus – Optional – The status of carts that should be retrieved. Examples include "Active" and "Abandoned". If no value is specified, the cart statuses are not considered when retrieving carts. This could be used in a B2B scenario when you want to display a list of available carts to a user but only carts that are not locked.
	IsLocked – Optional – If provided it will mean the search also will filter on whether the cart is locked or not
	ShopName – Optional. Name of shop to search for carts in
Output:	

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

	IEnumerable<CartBase> – A collection of CartBase objects The lists represent the carts that match the criteria specified in the request.
	SystemMessages - Collection of messages from the external system.

Usage Example:

```
var cartServiceProvider = new CartServiceProvider();

// Create request to get the carts.
var request = new GetCartsRequest("MyStore")
{
    UserIds = new Collection<string> { "John" },
    CustomerIds = new Collection<string> { "JohnCustomerId" },
    Names = new Collection<string> { "JohnsName" },
    Statuses = new Collection<string> { "InProcess" },
    IsLocked = false
};

// Call service provider and receive the result.
var result = cartServiceProvider.GetCarts(request);
```

CreateOrResumeCart

Name:	**CreateOrResumeCart**
Description:	Initiate the creation of a shopping cart and in the process: • Tries to load persisted, potentially abandoned cart, if present • Trigger event in DMS • Enter user in Engagement Automation plan with ID of shopping cart.
Usage:	Called when a shopping cart is needed upon visitor arrival to shop.
Signature:	`CreateOrResumeCartResult CreateCart(CreateOrResumeCartRequest request)`
Input:	All four input parameters are used to search and match against existing carts for the current visitor, but only two of them are mandatory.
	UserId – Mandatory
	CustomerId – Optional
	CartName – Optional
	ShopName – Mandatory
Output:	
	Cart – A Cart object instance which represent the shopping cart. In case multiple carts already exists for the current visitor and it is undecided which one to return, then no cart is returned
	IEnumerable<CartBase> - In case multiple carts already exists for the current visitor and it is undecided which one to return, then a list of CartBase objects are returned
	SystemMessages - Collection of messages from the external system.

Usage Example:

```
var cartServiceProvider = new CartServiceProvider();

// Create the request.
var createCartRequest = new CreateOrResumeCartRequest("Autohaus", "Bred");

// Call the service provider to get the cart
var cart = cartServiceProvider.CreateOrResumeCart(createCartRequest).Cart;

// Read the id of the returned cart
var id = cart.ExternalId;
```

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

```
        // Create request to remove the cart.
        var deleteCartRequest = new DeleteCartRequest(cart);

        // Call the service provider and receive the result.
        cartServiceProvider.DeleteCart(deleteCartRequest);
```

LoadCart

Name:	**LoadCart**
Description:	Gets the cart with given Cart ID on the specified shop. Calls the pipeline "LoadCart". This method returns the full cart object with all cartlines associated.
Usage:	Called when a specific cart is needed
Signature:	`LoadCartResult LoadCart(LoadCartRequest request)`
Input:	
	CartId – required
	ShopName – required
Output:	
	Cart – A cart object instance which represent the shopping cart that matches the criteria specified in the request.
	SystemMessages - Collection of messages from the external system.

Usage Example:

```
        var cartServiceProvider = new CartServiceProvider();

        // Create the request
        var request = new LoadCartRequest("Autohaus", "Bred");

        // call the service provider to get the cart
        var cart = cartServiceProvider.LoadCart(request).Cart;
```

SaveCart

Name:	**SaveCart**
Description:	Saves the specified cart in the external system if supported as well as in Sitecore EA state. Calls the pipeline "SaveCart". Called from other service layer methods implicitly, but not called explicitly
Usage:	Called when a specific cart needs to be persisted. The method should be executed after any operation that modified the cart resulting in a change of cart. It's executed implicitly when update cart, adding, deleting or updating cart lines as well as locking and un-locking the cart.
Signature:	`SaveCartResult SaveCart(SaveCartRequest request)`
Input:	
	Cart – required
Output:	
	SystemMessages - Collection of messages from the external system.

Usage Example:

```
        var cartServiceProvider = new CartServiceProvider();

        // Create cart and lock it.
        var createCartRequest = new CreateOrResumeCartRequest("Autohaus", "Mark");
        var cart = cartServiceProvider.CreateOrResumeCart(createCartRequest).Cart;

        // add a cartline to the cart
        var cartLine1 = new CartLine
        {
```

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

```
            Quantity = 1,
            Product = new CartProduct
            {
                ProductId = "Audi Q10",
                Price = new Price(55, "USD")
            }
        };

        var cartLines = new Collection<CartLine> { cartLine1 };
        var addCartLinesRequest = new AddCartLinesRequest(cart, cartLines);
        cart = cartServiceProvider.AddCartLines(addCartLinesRequest).Cart;

        var saveCartRequest = new SaveCartRequest(cart);
        var result = cartServiceProvider.SaveCart(saveCartRequest);
```

AddCartLines

Name:	AddCartLines
Description:	Responsibility is to add lines to cart.
Usage:	Called when a list of cart lines is about to be added to the shopping cart. UI wise when the user clicks the Add-To-Cart button.
Signature:	`AddCartLinesResult AddCartLines(AddCartLinesRequest request)`
Input:	
	Cart – Required - The cart must be unmodified. Any changes made to cart instance will be disregarded. Only the cart Id and ShopName are considered for retrieving and modifying the cart.
	IEnumerable<CartLine> CartLines – Required - A collection of cart lines to add
	ForceNewLines – Optional – If true, the cart lines items will be added as new cart lines in the customer cart. Otherwise, the lines will be merged into existing cart lines if possible. *Note: This parameter is only supported if the external commerce system supports this functionality.*
Output:	
	Cart - Cart object that represent the updated cart in the external system.
	AddedCartLineExternalIds – A list of the External IDs of the lines that were added to the cart.
	SystemMessages - Collection of messages from the external system

Usage Example:

```
        var cartServiceProvider = new CartServiceProvider();

        // Prepare parameters for getting cart for visitor ID Ivan in shop Autohaus
        var createCartRequest = new CreateOrResumeCartRequest("Autohaus", "Ivan");

        // Get a cart, new or existing
        var cart = cartServiceProvider.CreateOrResumeCart(createCartRequest).Cart;

        // Create cart line with subline to add to the cart
        var cartLines = new ReadOnlyCollection<CartLine>(new Collection<CartLine>
        {
            new CartLine
            {
                Product = new CartProduct
                {
                    ProductId = "Audi",
                    Price = new Price(10000, "USD") },
             Quantity = 1,
             SubLines = new Collection<CartLine>
                    {
             new CartLine
                    {
```

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

```
                Product = new CartProduct
                {
                    ProductId = "Winter Tyres",
                    Price = new Price(100, "USD")
                },
                Quantity = 4
            },
     new CartLine
            {
                Product = new CartProduct
                {
                    ProductId = "Summer Tyres",
                    Price = new Price(80, "USD")
                },
                Quantity = 4
            }
            }
    }
}
);

// Create request with prefix and prefix lines
var request = new AddCartLinesRequest(cart, cartLines);

// Add prefix lines into prefix
var result = cartServiceProvider.AddCartLines(request);
var resultCart = result.Cart;
```

RemoveCartLines

Name:	**RemoveCartLines**
Description:	Responsibility is to remove lines from cart.
Usage:	Called when one or more cart lines are about to be removed from the shopping cart. UI wise when the user updates the cart by removing one or more lines.
Signature:	`RemoveCartLinesResult RemoveCartLines(RemoveCartLinesRequest request)`
Input:	
	Cart - Required. The cart must be unmodified. Any changes made to cart instance will be disregarded. Only the cart Id and ShopName are considered for retrieving and modifying the cart.
	IEnumerable<CartLine> CartLines – Required - A collection of cart lines to remove. ExternalCartLineId, LineNumber or object reference can be used to identify the line(s) to be removed. The default Connect based implementation removes lines by object reference. Typically
Output:	
	Cart - Cart object that represent the updated cart in the external system.
	SystemMessages - Collection of messages from the external system

Usage Example:

```
var cartServiceProvider = new CartServiceProvider();

// Create cart with "Audi Q10", "BMW X7" and "Citroen C3"
var createCartRequest = new CreateOrResumeCartRequest("Autohaus", "John");
var cart = cartServiceProvider.CreateOrResumeCart(createCartRequest).Cart;

var cartLine1 = new CartLine
{
    Quantity = 1,
    Product = new CartProduct
    {
```

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

```
                ProductId = "Audi Q10",
                Price = new Price(55, "USD")
            }
        };

        var cartLine2 = new CartLine
        {
            Quantity = 2,
            Product = new CartProduct
            {
                ProductId = "BMW X7",
                Price = new Price(10, "USD")
            }
        };

        var cartLine3 = new CartLine
        {
            Quantity = 1,
            Product = new CartProduct
            {
                ProductId = "Citroen C3",
                Price = new Price(25, "USD")
            }
        };

        var cartLines = new Collection<CartLine>
        {
            cartLine1, cartLine2, cartLine3
        };

        var addCartLinesRequest = new AddCartLinesRequest(cart, cartLines);
        cart = cartServiceProvider.AddCartLines(addCartLinesRequest).Cart;

        // Create request to remove cart line "BMW X7".
        var request = new RemoveCartLinesRequest(cart, cart.Lines.Where(l =>
l.Product.ProductId == "BMW X7").ToArray());

        // Call service provider and receive the result.
        var result = cartServiceProvider.RemoveCartLines(request);
```

UpdateCartLines

Name:	**UpdateCartLines**
Description:	Responsibility is to update lines on cart.
Usage:	Occurs when a shopping cart is about to be updated referring to lines already in the cart. UI wise it is when the user updates the cart regarding a specific product. Most typically it is when • The quantity is changed • A service is added like insurance or shipping • Promotion code is added for a given product • Product is replaced with another variant
Signature:	`UpdateCartLinesResult UpdateCartLines(UpdateCartLinesRequest request)`
Input:	
	Cart - Required. The cart must be unmodified. Any changes made to cart instance will be disregarded. Only the cart Id and ShopName are considered for retrieving and modifying the cart.
	IEnumerable<CartLine> CartLines – Required - A collection of cart lines to update on cart
Output:	
	Cart – Cart object that represent the updated cart in the external system.
	SystemMessages - Collection of messages from the external system.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Usage Example:

```
        var cartServiceProvider = new CartServiceProvider();

        // Create cart with "Audi Q10", "BMW X7" and "Citroen C3"
        var createCartRequest = new CreateOrResumeCartRequest("Autohaus", "John");
        var cart = cartServiceProvider.CreateOrResumeCart(createCartRequest).Cart;

        var cartLine1 = new CartLine
        {
            Quantity = 1,
            Product = new CartProduct
            {
                ProductId = "Audi Q10",
                Price = new Price(55, "USD")
            }
        };

        var cartLine2 = new CartLine
        {
            Quantity = 2,
            Product = new CartProduct
            {
                ProductId = "BMW X7",
                Price = new Price(10, "USD")
            }
        };

        var cartLines = new Collection<CartLine> { cartLine1, cartLine2 };

        var addCartLinesRequest = new AddCartLinesRequest(cart, cartLines);
        cart = cartServiceProvider.AddCartLines(addCartLinesRequest).Cart;

        var bmw = cart.Lines.First(i => i.Product.ProductId == "BMW X7");
        bmw.Product.Price = new Price(110000, "USD");
        bmw.Quantity = 3;

        // Create request to update cart lines.
        var updateCartLinesRequest = new UpdateCartLinesRequest(cart, new Collection<CartLine>
{ bmw });

        // Call service provider and receive the result.
        var result = cartServiceProvider.UpdateCartLines(updateCartLinesRequest);
```

DeleteCart

Name:	DeleteCart
Description:	Responsibility is to delete a cart permanently: • The cart is deleted. • Trigger event in DMS telling the cart is deleted.
Usage:	Must be called when a cart needs to be deleted. UI wise this could be • When the user has gone through the B2C scenario of paying and an order has been created and registered. • After a cart has been in abandoned state for a given time and the visitor is not expected to return.
Signature:	`DeleteCartResult DeleteCart(DeleteCartRequest request)`
Input:	
	Cart - Required. The cart must be unmodified. Any changes made to cart instance will be disregarded. Only the cart Id and ShopName are considered for retrieving and modifying the cart**.**
Output:	

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

	SystemMessages - Collection of messages from the external system.

Usage Example:

```
        var cartServiceProvider = new CartServiceProvider();

        // Create the request.
        var createCartRequest = new CreateOrResumeCartRequest("Autohaus", "Bred");

        // Call the service provider to get the cart
        var cart = cartServiceProvider.CreateOrResumeCart(createCartRequest).Cart;

        // Read the id of the returned cart
        var id = cart.ExternalId;

        // Create request to remove the cart.
        var deleteCartRequest = new DeleteCartRequest(cart);

        // Call the service provider and receive the result.
        cartServiceProvider.DeleteCart(deleteCartRequest);
```

UpdateCart

Name:	**UpdateCart**
Description:	Responsibility is to pass an updated cart to the external commerce system Trigger event in DMS telling the cart is being updated.
Usage:	The method should be executed after any operation that modifies the cart, typically when Adjustments have been added, removed or modified.
Signature:	`UpdateCartResult UpdateCart(UpdateCartRequest request)`
Input:	
	Cart - Required - The cart to be updated. The cart must be unmodified. Any changes made to cart instance will be disregarded. Only the cart Id and ShopName are considered for retrieving and modifying the cart.
	Cart Base – An instance of Cart Base containing the changes to be made to the cart Typically the only properties allowed to be modified would be: UserId, CustomerId, CartName and potentially ShopName. Whether IsLocked and CartStatus will be considered, depends on business logic in the external commerce system. Null values will not be considered, but blank values will
Output:	
	Cart – Cart object that represent the updated cart in the external system.
	SystemMessages - Collection of messages from the external system.

Usage Example:

```
        var cartServiceProvider = new CartServiceProvider();

        // Create prefix.
        var createCartRequest = new CreateOrResumeCartRequest("Autohaus", "Peter");
        var cart = cartServiceProvider.CreateOrResumeCart(createCartRequest).Cart;

        // Create the instance of the CartBase with properties that should be changed in 
existing prefix.
        var changes = new CartBase
        {
            CustomerId = "Customer Peter",
            Name = "Peter's Cart",
            ShopName = "Autohaus"
        };

        // Create request to update the prefix.
        var updateCartRequest = new UpdateCartRequest(cart, changes);
```

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

```
        // Call service provider with prepared request and receive the result.
        var result = cartServiceProvider.UpdateCart(updateCartRequest);
```

LockCart

Name:	**LockCart**
Description:	Responsibility is to set the cart in a locked state where it is ready to be committed to an order but before any optional payment transaction is performed: • Set cart to locked and save it. • Trigger event in DMS telling the cart is in locked state. When cart is in locked state it indicates two things: • It's ***not*** possible to modify the shopping cart content using the other service layer methods • It's easy to identify the carts in locked state in order to compare and confirm with payment transactions whether there are carts that have not been finalized due to some unforeseen incident in the checkout process. There is a corresponding UnlockCart method. If the cart is locked when LockCart is called, the pipeline is aborted and nothing happens.
Usage:	Is typically executed during the checkout process, just before any payment transaction is about to be executed and before turning the cart into an order. UI wise its triggered when a user in the checkout flow has selected "confirm" and in a B2C scenario is going to pay and the order is created.
Signature:	`LockCartResult LockCart(LockCartRequest request)`
Input:	
	Cart - Required
Output:	
	Cart – Cart object that represent the updated cart in the external system.
	SystemMessages - Collection of messages from the external system.

Usage Example:

```
        var cartServiceProvider = new CartServiceProvider();

        // Create a sample cart.
        var createCartRequest = new CreateOrResumeCartRequest("Autohaus", "Jho");
        var cart = cartServiceProvider.CreateOrResumeCart(createCartRequest).Cart;

        // Create request to lock this cart.
        var lockCartRequest = new LockCartRequest(cart);

        // Call service provider and receive the result.
        var result = cartServiceProvider.LockCart(lockCartRequest);
```

UnlockCart

Name:	**UpdateCartLines**
Description:	Responsibility is to set the cart in an unlocked state: • Set cart to unlocked and save it. • Trigger event in DMS telling the cart is in locked state.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

	By default a cart is in unlocked state and can be edited There is a corresponding LockCart method, which sets the state to locked. If the cart is not locked when UnlockCart is called, the pipeline is aborted and nothing happens.
Usage:	Is typically called if user returns to cart and decides to modify the content after starting the checkout process and payment transaction, but its implementation specifies how it should be handled.
Signature:	`UnlockCartResult UnlockCart(UnlockCartRequest request)`
Input:	
	Cart – Required
Output:	
	Cart – Cart object that represent the updated cart in the external system.
	SystemMessages - Collection of messages from the external system. This is how error conditions can be reported.

Usage Example:

```
var cartServiceProvider = new CartServiceProvider();

// Create cart and lock it.
var createCartRequest = new CreateOrResumeCartRequest("Autohaus", "Mark");
var cart = cartServiceProvider.CreateOrResumeCart(createCartRequest).Cart;
var lockCartRequest = new LockCartRequest(cart);
cart = cartServiceProvider.LockCart(lockCartRequest).Cart;

// Create request to unlock this cart.
var unlockCartRequest = new UnlockCartRequest(cart);

// Call service provider and receive the result.
var result = cartServiceProvider.UnlockCart(unlockCartRequest);
```

MergeCart

Name:	**MergeCart**
Description:	Responsibility is to merge two specified carts: • Both carts must have identical shop names. • Both carts must have different ExternalIDs. • Cart1 and Cart2 cart line items are merged and returned
Usage:	Is typically called when a user logs in and the system notices an anonymous cart exists.
Signature:	`CartResult MergeCart([NotNull] MergeCartRequest request)`
Input:	
	UserCart – Required **AnonymousCart - Required**
Output:	
	Cart – Cart object representing the merged user cart.

Usage Example:

```
var cartServiceProvider = new CartServiceProvider();

var userCart = new Cart
{
    ExternalId = "0",
    ShopName = "first shop",
    Lines  = new List<CartLine>
    {
        new CartLine
        {
            Quantity = 1,
```

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

```
                Product = new CartProduct
                {
                    ProductId = "Audi Q10",
                    Price = new Price(55, "USD")
                }
            }
        }
    };

    var anonymousCart = new Cart
    {
        ExternalId = "1",
        ShopName = "first shop",
        Lines  = new List<CartLine>
        {
            new CartLine
            {
                Quantity = 1,
                Product = new CartProduct
                {
                    ProductId = "BMW M5",
                    Price = new Price(75, "USD")
                }
            }
        }
    };

    var request = new MergeCartRequest(userCart, anonymousCart);

    var mergeredCart = cartServiceProvider.MergeCart(request).Cart;
    cartServiceProvider.DeleteCart(new DeleteCartRequest(anonymousCart));
```

AddParties

Name:	AddParties
Description:	Responsibility is to add parties to a cart
Usage:	Is typically called when adding party information to a cart
Signature:	`AddPartiesResult AddParties([NotNull] AddPartiesRequest request)`
Input:	
	Cart – Required **Parties -- Required**
Output:	
	Parties – The read only list of all parties associated with this cart after the add

Usage Example:

```
    var cartServiceProvider = new CartServiceProvider();

    var createCartRequest = new CreateOrResumeCartRequest("Autohaus", "John");
    var cart = cartServiceProvider.CreateOrResumeCart(createCartRequest).Cart;

    var partyList = new List<Party>
    {
        new Party
        {
            PartyId = "123", FirstName = "Joe", LastName = "Smith",
            Address1 = "123 Street", City = "Ottawa",
            State = "Ontario", Country = "Canada"
        },
        new Party
        {
            PartyId = "456", FirstName = "Jane", LastName = "Smith",
            Address1 = "234 Street", City = "Toronto",
            State = "Ontario", Country = "Canada"
        }
```

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

```
};

var addPartiesRequest = new AddPartiesRequest(cart, partyList);

var addPartiesResult = cartServiceProvider.AddParties(addPartiesRequest);
```

RemoveParties

Name:	**RemoveParties**
Description:	Responsibility is to remove parties from a cart
Usage:	Is typically called when removing party information
Signature:	`RemovePartiesResult RemoveParties([NotNull] RemovePartiesRequest request)`
Input:	
	Cart – Required **Parties – Required –** The list of parties to remove from the cart
Output:	
	Parties – The read only list of all parties associated with this cart after the remove

Usage Example:

```
var cartServiceProvider = new CartServiceProvider();

var createCartRequest = new CreateOrResumeCartRequest("Autohaus", "John");
var cart = cartServiceProvider.CreateOrResumeCart(createCartRequest).Cart;

var partyList = new List<Party>
{
    new Party
    {
        PartyId = "123", FirstName = "Joe", LastName = "Smith",
        Address1 = "123 Street", City = "Ottawa",
        State = "Ontario", Country = "Canada"
    },
    new Party
    {
        PartyId = "456", FirstName = "Jane", LastName = "Smith",
        Address1 = "234 Street", City = "Toronto",
        State = "Ontario", Country = "Canada"
    }
};

var addPartiesRequest = new AddPartiesRequest(cart, partyList);
var addPartiesResult = cartServiceProvider.AddParties(addPartiesRequest);

var removePartiesRequest = new RemovePartiesRequest(cart, new List<Party>
{
    partyList[0]
});
var removePartiesResult = cartServiceProvider.RemoveParties(removePartiesRequest);
```

UpdateParties

Name:	**UpdateParties**
Description:	Responsibility is to update a list of parties within a cart
Usage:	Is typically called when parties need to be updated
Signature:	`UpdatePartiesResult UpdateParties([NotNull] UpdatePartiesRequest request)`
Input:	
	Cart – Required **Parties – Required –** The list of parties to update in the cart

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Output:	
	Parties – The read only list of all parties associated with this cart after the update

Usage Example:

```
var cartServiceProvider = new CartServiceProvider();

var createCartRequest = new CreateOrResumeCartRequest("Autohaus", "John");
var cart = cartServiceProvider.CreateOrResumeCart(createCartRequest).Cart;

var party1 = new Party
{
    PartyId = "123",
    FirstName = "Joe",
    LastName = "Smith",
    Address1 = "123 Street",
    City = "Ottawa",
    State = "Ontario",
    Country = "Canada"
};

var party2 = new Party
{
    PartyId = "456",
    FirstName = "Jane",
    LastName = "Smith",
    Address1 = "234 Street",
    City = "Toronto",
    State = "Ontario",
    Country = "Canada"
};

var partyList = new List<Party> { party1, party2 };

var addPartiesRequest = new AddPartiesRequest(cart, partyList);
var addPartiesResult = cartServiceProvider.AddParties(addPartiesRequest);

party1.Address1 = "678 Road";
party1.City = "London";

var updatePartiesRequest = new UpdatePartiesRequest(cart, new List<Party> { party1 });
var removePartiesResult = cartServiceProvider.UpdateParties(updatePartiesRequest);
```

AddPaymentInfo

Name:	**AddPaymentInfo**
Description:	Responsibility is to add payment information to a cart
Usage:	Is typically called during a checkout flow to add the payment info for processing of an order.
Signature:	`AddPaymentInfoResult AddPaymentInfo([NotNull] AddPaymentInfoRequest request)`
Input:	
	Cart – Required **Payments – Required –** a list of payment info to be added to the cart
Output:	
	Payments – The read only list of payments associated with the cart after the add

Usage Example:

```
var cartService = new CartServiceProvider();
var cart = cartService.CreateOrResumeCart(new
    CreateOrResumeCartRequest("MyShop", "Me")).Cart;
```

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

```
var paymentList = new List<PaymentInfo>
{
    new PaymentInfo() { ExternalId = "1", PaymentMethodID = "1"},
    new PaymentInfo() { ExternalId = "2", PaymentMethodID = "2"}
};

var addPaymentRequest = new AddPaymentInfoRequest(cart, paymentList);

var addPaymentResult = cartService.AddPaymentInfo(addPaymentRequest);
```

RemovePaymentInfo

Name:	**RemovePaymentInfo**
Description:	Responsibility is to remove payment information from a cart
Usage:	Is typically called when a user wants to change their payment information
Signature:	`RemovePaymentInfoResult RemovePaymentInfo([NotNull] RemovePaymentInfoRequest request)`
Input:	**Cart – Required** **Payments – Required –** a list of payment info to be removed from the cart
Output:	**Payments** – The read only list of payments associated with the cart after the remove

Usage Example:

```
var cartService = new CartServiceProvider();
var cart = cartService.CreateOrResumeCart(new
    CreateOrResumeCartRequest("MyShop", "Me")).Cart;

var paymentList = new List<PaymentInfo>
{
    new PaymentInfo() { ExternalId = "1", PaymentMethodID = "1"},
    new PaymentInfo() { ExternalId = "2", PaymentMethodID = "2"}
};

var addPaymentRequest = new AddPaymentInfoRequest(cart, paymentList);

var addPaymentResult = cartService.AddPaymentInfo(addPaymentRequest);

var removePaymentRequest = new RemovePaymentInfoRequest(cart, new
List<PaymentInfo> {paymentList[0]});
var removeResult = cartService.RemovePaymentInfo(removePaymentRequest);
```

AddShippingInfo

Name:	
Description:	Responsibility is to add shipping information to a cart
Usage:	Is typically called during a checkout flow to add the shipping info for processing of an order.
Signature:	`AddShippingInfoResult AddShippingInfo([NotNull] AddShippingInfoRequest request)`
Input:	
	Cart – Required **ShippingInfo – Required –** a list of shipping info to add to the cart
Output:	

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

	ShippingInfo – A read only list of shipping info associated with the cart after the add

Usage Example:

```
var cartService = new CartServiceProvider();
var cart = cartService.CreateOrResumeCart(new
    CreateOrResumeCartRequest("MyShop", "Me")).Cart;

var shippingList = new List<ShippingInfo>
{
    new ShippingInfo() { ExternalId = "1", ShippingMethodID = "1"},
    new ShippingInfo() { ExternalId = "2", ShippingMethodID = "2"}
};

var addRequest = new AddShippingInfoRequest(cart, shippingList);

var addResult = cartService.AddShippingInfo(addRequest);
```

RemoveShippingInfo

Name:	
Description:	Responsibility is to remove shipping information from a cart
Usage:	Is typically called during a checkout flow to remove the shipping info for processing of an order.
Signature:	`RemoveShippingInfoResult RemoveShippingInfo([NotNull] RemoveShippingInfoRequest request)`
Input:	
	Cart – Required **ShippingInfo – Required –** a list of shipping info to remove from the cart
Output:	
	ShippingInfo – A read only list of shipping info associated with the cart after the remove

Usage Example:

```
var cartService = new CartServiceProvider();
var cart = cartService.CreateOrResumeCart(new
    CreateOrResumeCartRequest("MyShop", "Me")).Cart;

var shippingList = new List<ShippingInfo>
{
    new ShippingInfo() { ExternalId = "1", ShippingMethodID = "1"},
    new ShippingInfo() { ExternalId = "2", ShippingMethodID = "2"}
};

var addRequest = new AddShippingInfoRequest(cart, shippingList);

var addPaymentResult = cartService.AddShippingInfo(addRequest);

var removeRequest = new RemoveShippingInfoRequest(cart, new
List<ShippingInfo> {shippingList[0]});
var removeResult = cartService.RemoveShippingInfo(removeRequest);
```

2.1.3 Cart Pipelines

The integration and engagement logic used in the Cart API is implemented by pipelines that can be customized as needed. There is a pipeline for each method on the API. Some pipelines call other

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

common pipelines like SaveCart and LoadCart. In some cases the logic is split into several sub-pipelines to handle if-than-else situations like used in CreateOrResumeCart.

GetCarts

Name:	**GetCarts**
Description:	The pipeline is responsible for performing a search against all carts and return a list of CartBase instances for carts found, matching the specified search criteria. The carts might be persisted in both the ECS and/or EA state. Depending on the location of the main cart repository, the processors configured for the pipeline will differ between different Connect provider implementations.
Usage:	Called via method GetCarts on the Connect API when searching for carts.
Args:	
	Request - Contains the search criteria: UserID, CustomerID, CartName, CartStatus, IsLocked, and ShopName. Is set prior to calling the pipeline.
	Response - Contains the cart objects. Is read after the pipeline is executed.
Processors:	**GetCartsFromEAState –** **Responsibility -** To execute query against carts stored in EA states across all users which matches the input parameters UserID, CustomerID, CartName, and ShopName. In default implementation, the processor is split into two, named: • BuildQuery. • ExecuteQuery. Queries the EA repository Usage: The processor is generic for searching in EA state and returning lists of carts. *If the external commerce system supports persisting carts then the processor(s) should be replaced by the custom GetCarts processor querying against ECS, see below.* Ownership: The processor is provided with Connect Customization: If more search parameters are needed then the processor should be inherited and overwritten to search in
	GetCarts – Responsibility - To execute a query against carts stored in the external commerce system which matches the input parameters UserID, CustomerID, CartName, and ShopName. Usage: If ECS supports persistence of carts, then this processor replaces the default Connect processor(s) querying against EA state

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

	If not supported, EA state and GetCartsFromEAState processor(s) must be used instead Ownership: ECS vendor Customization: The processor is custom to the ECS. If more search parameters are needed then the processor should be extended to support that.

CreateOrResumeCart

Name:	**CreateOrResumeCart**
Description:	Initiates the creation of a shopping cart and in the process to: • Load persisted, potentially abandoned cart, if present and return that • Trigger event in DMS if new cart is created • Enter user in Engagement Automation plan with ID of new shopping cart The pipeline will call different pipelines depending on whether an existing cart is found and can be resumed or not.
Usage:	Called via method CreateOrResumeCart on the Connect API.
Args:	
	Request - Contains the essential cart parameters also used to search for existing cart. Is set prior to calling the pipeline.
	Response - Contains the cart objects or a list of CartBase instance in case multiple carts exists that matches the given parameters. Is read by the Cart Provider after the pipeline is called.
Processors:	
	FindCartInEAState – Responsibility: To locate a cart in the current contact's / visitor's EA state which matches the input parameters UserID, CustomerID, CartName, and ShopName. If a match is found then the Cart ID is set in custom pipeline argument CartID (args.CustomData["CartID"]) The processor is needed to retrieve the cart ID of the existing cart before LoadCart can be called Usage: The processor is generic for searching in EA state and returning cart ID for a cart matching the given input parameters. Ownership: The processor is provided with Connect Customization: No immediate need for overwriting the default functionality, unless further query parameters are introduced

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

RunLoadCart –

Responsibility: Is to call pipeline LoadCart and load a cart with given ID. The ID is specified in parameter CartId.

If cart was not found, cart is null or empty in pipeline arguments

Usage: Mandatory. The processor is generic for calling pipeline LoadCart that loads a cart by CartID and can be used in other pipelines.

It's assumed that the CartID to be used for loading cart is stored in customer pipeline arg "CartID" (args.CustomData["CartID"])

Pipeline RunResumeCart is assuming that the cart to be resumed is loaded before it is executed

Ownership: The processor is provided with Connect

Customization: No immediate need for overwriting the default functionality.

RunResumeCart –

Responsibility: Is to call pipeline ResumeCart in order to resume a loaded cart if possible.

It's expected that a cart is present in pipeline arguments.

If no cart is present, then the processor will not call pipeline ResumeCart

Usage: Mandatory. The processor is generic for calling pipeline ResumeCart

Ownership: The processor is provided with Connect

Customization: No immediate need for overwriting the default functionality.

RunCreateCart –

Responsibility: Is to call pipeline CreateCart in order to create a new cart.

It's expected that no cart is present in pipeline arguments.

If a cart is present, then the processor will not call pipeline CreateCart

Usage: Mandatory. The processor is generic for calling pipeline CreateCart

Ownership: The processor is provided with Connect

Customization: No immediate need for overwriting the default functionality

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

CreateCart

Name:	CreateCart
Description:	Creates and saves a new shopping cart and in the process: • Trigger event in DMS for new cart • Enter visitor / contact into Engagement Automation plan with new shopping cart
Usage:	Called from pipeline CreateOrResumeCart as one of the branches when no cart exists
Args:	
	Request - Contains the same parameters as CreateOrResumeCart pipeline. Is carried over from CreateOrResumeCart pipeline.
	Response - Contains the cart objects. Is read by the Cart Provider after the pipeline is called.
Processors:	
	CreateCart – Responsibility: Is to create a new cart, initialize values from arguments and return it. If a cart already exists in pipeline arguments, it is ignored and will be overwritten. Usage: Optional. The default processor is generic for creating a domain model cart and initialize values, but it never calls the ECS. In some integration scenarios it will be relevant to call the ECS at this point in time to create a new cart. In that case this processor can be used as a base class and extended, as it does initialize the cart domain object, or it can be replaced altogether. In other integration scenarios the ECS will not be called at this point in time. Instead it will likely be called when manipulating cart lines or only when saving the cart. It depends on the system being integrated with. Ownership: The processor is provided with Connect Customization: It can be useful to use the processor as a base class and extended it by calling the ECS after initializing the cart domain model object,
	AddVisitorToEAPlan – Responsibility: Is to add the current visitor to an EA plan, in this case the Abandoned Cart EA plan. The current implementation supports multiple shops. By using an eaPlanProvider the plan and state IDs can be retrieved. By default the eaPlanProvider tries to find an EA plan with a name prefixed with the site name. For example, the ECS could contain a site named Autohaus. According to

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

naming convention, the EA plan must be named “Autohaus Abandoned Cart”. If a different EA plan name is used, it can be configured on the site definition item.

Usage: Optional. The processor is generic for adding the visitor to an EA plan, but it expects to be in the CreateCart pipeline with the CreateCartRequest object in pipeline args.

The plan and state IDs are retrieved from the eaPlanProvider, which is specified in the processor parameter and instantiated via Sitecore Factory.

Ownership: The processor is provided with Connect

Customization: No immediate need for overwriting the default functionality

RunSaveCart –

Responsibility: Is to call pipeline SaveCart which saves the specified cart.

Usage: Optional, but should always be called when changes have been made to the cart, which is the case here. The processor is generic for calling pipeline SaveCart and can be used in other pipelines.

Ownership: The processor is provided with Connect

Customization: No immediate need for overwriting the default functionality.

TriggerPageEventForResultCart –

Responsibility: Is to trigger a specified page event and register the cart values: ExternalCartId, UserId, CartName, CartStatus

The Page Event text is localized by looking up in Sitecore dictionary.

Usage: Optional. The processor is generic for triggering a page event with the specified parameters from the cart stored in the args.Result argument.

The event to be triggered is specified in processor parameter PageEventName

The event text to be used is specified in processor parameter PageEventText

Ownership: The processor is provided with Connect

Customization: Overwrite the processor if other values from cart should be registered

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

ResumeCart

Name:	CreateOrResumeCart
Description:	Validates and resumes the cart specified in arguments and in the process: • Change the state to the initial state in abandoned cart Engagement (reboot) • Updates the cart state • Saves the cart • Trigger event in DMS
Usage:	Called via the Connect API from Sitecore.
Args:	
	Request - Contains the search criteria. Is set by the Cart Provider prior to calling the pipeline.
	Response - Contains the cart objects. Is read by the Cart Provider after the pipeline is called.
Processors:	
	CheckCanBeResumed – Responsibility: Is to check if given cart can be resumed. If visitor is not already in the abandoned cart EA plan the pipeline is aborted and if the visitor is already in the initial state, it is also aborted. Also sets three parameters in pipeline args that are used in the processors listed in parentheses: • CartSourceStateId (MoveVisitorToInitialState) • CartDestinationStateId (MoveVisitorToInitialState) • PreviousStateName (TriggerPageEventForResumedCart) Usage: Mandatory. The processor checks if the visitor is already in the plan or not. If not, there is no cart to resume. Ownership: The processor is provided with Connect Customization: No obvious customization needed
	ChangeCartStatus – Responsibility: Is to update the cart status field with the value “InProcess” taken from constant CartStatus.InProcess Usage: Optional. The processor is generic but the status value set on cart is fixed to InProcess. Ownership: The processor is provided with Connect

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Customization: The processor could be updated to take the state value from parameter instead of constant in code.

MoveVisitorToInitialState –

Responsibility: Is to move the current visitor between two states of an EA plan, in this case the Abandoned Cart EA plan.

Usage: Optional. The processor is generic for moving a visitor to initial state of EA plan, but it expects to be in the CreateCart pipeline with the CreateCartRequest object in pipeline args.

The source and destination state IDs are read from custom pipeline arguments:

- "CartSourceStateId"
- "CartDestinationStateId"

Ownership: The processor is provided with Connect

Customization: No immediate need for overwriting the default functionality

RunSaveCart –

Responsibility: Is to call pipeline SaveCart which saves the specified cart.

Usage: Mandatory. The processor is generic for calling pipeline SaveCart and can be used in other pipelines.

Ownership: The processor is provided with Connect

Customization: No immediate need for overwriting the default functionality.

TriggerPageEventForResumedCart –

Responsibility: Is to trigger a specified page event when resuming the cart and register the cart values:

- ExternalCartId, UserId, CartName, CartStatus, StateName (containing the previous state that it was moved from)

The Page Event text is localized by looking up in Sitecore dictionary.

Usage: Optional. The processor is specific for resuming a cart and for triggering a page event with the specified parameters from the cart stored in the args.Result argument.

The value for previous state is found in pipeline args "PreviousState" and is inserted in the page event text along with the ExternalCartId

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

	The event text to be used is specified in processor parameter PageEventText Ownership: The processor is provided with Connect Customization: Overwrite the processor if other values from cart should be registered or the text formatting should be different

LoadCart

Name:	**LoadCart**
Description:	Loads the cart that matches the specified criteria. For example, ID and ShopName. This pipeline is responsible for reading data for a specific cart that is managed by the commerce system. This pipeline reads the cart data from the commerce system and/or from Engagement Automation state.
Usage:	1. Called directly via the Connect API method LoadCart from Sitecore. 2. Called indirectly via the Connect API methods CreateOrResumeCart
Args:	
	Request - Contains the criteria that determine which cart should be retrieved. Is set prior to calling the pipeline.
	Response - Contains the cart object after the pipeline is called.
Processors:	
	LoadCartFromEAState – Responsibility: Load existing cart from EA state with given CartID and in the shop specified with parameter ShopName For performance reasons, the default implementation works as follows: • First carts are loaded from current visitor / contact EA state data and filtered by the given parameters. If found, it is written to pipeline result args and execution stops. • If not found, then all carts across all contacts are loaded and filtered by the given parameters. Searching across all contacts is an expensive operation. • If cart is found in EA state, it is written to pipeline Result arg. Usage: The processor is generic for loading a cart from EA state. The processor might be used alone in pipeline LoadCart, if cart persistence is not available in the ECS or if handling of carts occurs in Sitecore alone.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

It might also be left out of the pipeline if the ECS manage the cart repository alone.

Some ECS systems does not provide all the information specified in the cart domain model and a hybrid configuration might be used where the main data is read from the ECS and augmented with additional cart data stored in EA state by inserting an additional processor.

Abandoned carts might be purged from the ECS but still remain in EA state in Sitecore. In that case it might make sense to have both processors in the pipeline. Sitecore will then act as a backup storage for carts.

Ownership: The processor is provided with Connect

Customization: No immediate need for overwriting the default functionality

LoadCart –

Responsibility:

Load existing cart from ECS with given CartID and in the shop specified with parameter ShopName

Usage: The processor is specific for loading a cart from ECS.

The processor might be used alone in pipeline LoadCart if cart persistence is available in the ECS. See also scenarios in description for LoadCartFromEAState processor.

Some ECS systems does not provide all the information specified in the cart domain model and a hybrid configuration might be used where the main data is read from the ECS and augmented with additional cart data stored in EA state by inserting an additional processor.

If used in combination with LoadCartFromEAState, even though the cart might already be loaded from EA state, it is important to check with the ECS and load cart from there to ensure the latest version is used.

If cart is found in external commerce system it can overwrite or merge with the cart already stored in pipeline Response arg. It's up to the Connect provider implementation with the ECS.

Ownership: The processor is provided by the ECS

Customization: Must be built specifically for the ECS.

SaveCart

Name:	**SaveCart**
Description:	Saves the cart both to the external commerce system and in Engagement Automation state.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Usage:	Called from other service layer methods, but rarely, if never called explicitly
Args:	
	Request - Contains the criteria that determine which cart should be retrieved. Is set prior to calling the pipeline.
	Response - Contains the cart object after the pipeline is called.
Processors:	
	SaveCart – Responsibilitty: Saves the given cart to the ECS Usage: The processor is specific for saving a cart to the ECS. The processor should not be used alone in pipeline SaveCart because the feature for resuming existing carts is depending on the cart being stored in EA state as well. The processor FindCartInEAState is looking up the cart in EA state in order to get the CartId for loading the cart with LoadCart from ECS: Some ECS systems does not provide all the information specified in the cart domain model and a hybrid configuration might be used where the main data is saved to the ECS and the additional cart data is stored in EA state by. In some ECS implementations the SaveCart pipeline is the first and only place the ECS system is provided a cart from Connect. Since the ECS is the primary repository for carts, it is assumed that the unique CartID key is provided by the ECS. Ownership: The processor is provided by the ECS Customization: Must be built specifically for the ECS.
	SaveCartToEAState – Responsibility: Saves the cart to EA state Usage: Mandatory. The processor is generic for saving a cart to EA state. The processor might be used alone in pipeline LoadCart, if cart persistence is not available in the ECS or if handling of carts occurs in Sitecore alone. The processor should be used in pipeline SaveCart because the feature for resuming existing carts is depending on the cart being stored in EA state as well. The processor FindCartInEAState is looking up the cart in EA state in order to get the CartId for loading the cart with LoadCart from ECS: Some ECS systems does not provide all the information specified in the cart domain model and a hybrid configuration might be used where the main data is

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

read from the ECS and augmented with additional cart data stored in EA state by inserting an additional processor.

Abandoned carts might be purched from the ECS but still remain in EA state in Sitecore. In that case it might make sense to have both processors in the pipeline. Sitecore will then act as a backup storage for carts.

Ownership: The processor is provided with Connect

Customization: No immediate need for overwriting the default functionality

.

AddCartLines

Name:	**AddCartLines**
Description:	This pipeline is responsible for adding a new line to the shopping cart and recording a corresponding page event in DMS. This happens when a product is added to the cart.
Usage:	Called from Sitecore.
Args:	
	Request - Contains the cart to be updated, along with what lines should be added to the cart. Is set prior to calling the pipeline.
	Response - Contains the updated cart object after the pipeline is called.
Processors:	
	CheckIfLocked – Responsibility: Checks if the cart is locked and abort the pipeline if so, returning SystemMessages to signal the locked state Usage: Optional. The processor is generic for checking if cart is locked. The processor is and should be used in all pipelines that potentially modify the cart content Ownership: The processor is provided with Connect Customization: No immediate need for overwriting the default functionality
	AddLinesToCart – Responsibility: Adds the given lines to the provided cart. Calling this method will always add the given lines to the existing collection of lines in the cart, even if lines already exist on the cart containing the same product. Alternatively UpdateLinesOnCart can be called if a line already exists with a product where the quantity simply should be adjusted.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Usage: Optional. The processor is generic for adding lines to cart, but potentially should be replaced by an ECS specific implementation.

Ownership: The processor is provided with Connect

Customization: The default implementation operates on the cart domain model in Sitecore only.

In most ECS integrations it will be relevant to inherit and overwrite or replace this implementation and call the ECS, so that the changes to cart are passed on and any business logic can be applied.

RunSaveCart -

Responsibility: Is to call pipeline SaveCart which saves the specified cart.

Usage: Optional, but should always be called when changes have been made to the cart, which is the case here. The processor is generic for calling pipeline SaveCart and can be used in other pipelines.

Ownership: The processor is provided with Connect

Customization: No immediate need for overwriting the default functionality

TriggerPageEventsForCartLines –

Responsibility: Is to trigger a specified page event when adding lines to cart and register the values:

- Product ID, QTY, Price and Currency

The event to be triggered is passed in as parameter as well as the Page Event Text.

In this case the event is “Lines Added To Cart”. The Page Event text is localized by looking up in Sitecore dictionary.

Usage: Optional. The processor is generic for triggering a page event when modifying cart lines

Ownership: The processor is provided with Connect

Customization: Overwrite the processor if other values from cart should be registered or the text formatting should be different

TriggerAddToCartStockStatusPageEvent–

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Responsibility: Trigger page event AddToCartStockStatus along with the ShopName, Cart ID, Product ID , Stock Status, Pre-orderable, In-Stock Date, Shipping Date, if and only if, the stock status is NOT InStock

Usage: Mandatory

Ownership: Provided with Connect

Customization: Not needed

RemoveCartLines

Name:	RemoveCartLines
Description:	Responsibility is to remove cart lines from cart
Usage:	Called from Sitecore.
Args:	
	Request - Contains the cart to be updated along with the cart lines to be removed Is set prior to calling the pipeline.
	Response - Contains the updated cart object after the pipeline is called.
Processors:	
	CheckIfLocked – Responsibility: Checks if the cart is locked and abort the pipeline if so, returning an SystemMessages to signal the locked state Usage: Optional. The processor is generic for checking if cart is locked. The processor is and should be used in all pipelines that potentially modify the cart content Ownership: The processor is provided with Connect Customization: No immediate need for overwriting the default functionality
	RemoveLinesFromCart – Responsibility: Removes the given lines from the provided cart. The list of lines to remove must be references directly to the lines in the CartLines collection on the cart. They will be removed by reference: cart.CartLines = cart.CartLines.Except(request.CartLines).ToList();

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Usage: Optional. The processor is generic for removing lines from cart, but potentially should be replaced by an ECS specific implementation.

Ownership: The processor is provided with Connect

Customization: The default implementation operates on the cart domain model in Sitecore only.

In most ECS integrations it will be relevant to inherit and overwrite or replace this implementation and call the ECS, so that the changes to cart are passed on and any business logic can be applied.

SaveCart –

Responsibility: Is to call pipeline SaveCart which saves the specified cart.

Usage: Optional, but should always be called when changes have been made to the cart, which is the case here. The processor is generic for calling pipeline SaveCart and can be used in other pipelines.

Ownership: The processor is provided with Connect

Customization: No immediate need for overwriting the default functionality

TriggerPageEventsForCartLines –

Responsibility: Is to trigger a specified page event when adding lines to cart and register the values:

- Product ID, QTY, Price and Currency

The event to be triggered is passed in as parameter as well as the Page Event Text. In this case the event is "Lines Removed From Cart". The Page Event text is localized by looking up in Sitecore dictionary.

Usage: Optional. The processor is generic for triggering a page event when modifying cart lines

Ownership: The processor is provided with Connect

Customization: Overwrite the processor if other values from cart should be registered or the text formatting should be different

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

UpdateCartLines

Name:	**UpdateCartLines**
Description:	Responsibility is to update lines on cart
Usage:	Called from Sitecore.
Args:	
	Request - Contains the cart along with the cart lines to be updated Is set prior to calling the pipeline.
	Response - Contains the updated cart object after the pipeline is called.
Processors:	
	CheckIfLocked – Responsibility: Checks if the cart is locked and abort the pipeline if so, returning an SystemMessages to signal the locked state Usage: Optional. The processor is generic for checking if cart is locked. The processor is and should be used in all pipelines that potentially modify the cart content Ownership: The processor is provided with Connect Customization: No immediate need for overwriting the default functionality
	UpdateLinesOnCart – Responsibility: To update the given lines on the provided cart. The default implementation doesn’t process the lines on the original cart, but simple return the cart that was passed in and that already had its Cartlines updated. Hence, the list of updated lines must be references to the CartLines collection on the cart. Usage: Optional. The processor is generic for updating lines on cart, but potentially should be replaced by an ECS specific implementation. Ownership: The processor is provided with Connect Customization: The default implementation operates on the cart domain model in Sitecore only. In most ECS integrations it will be relevant to inherit and overwrite or replace this implementation and call the ECS, so that the changes to cart are passed on and any business logic can be applied.
	SaveCart –

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

	Responsibility: Is to call pipeline SaveCart which saves the specified cart. Usage: Optional, but should always be called when changes have been made to the cart, which is the case here. The processor is generic for calling pipeline SaveCart and can be used in other pipelines. Ownership: The processor is provided with Connect Customization: No immediate need for overwriting the default functionality
	TriggerPageEventsForCartLines – Responsibility: Is to trigger a specified page event when adding lines to cart and register the values: • Product ID, QTY, Price and Currency The event to be triggered is passed in as parameter as well as the Page Event Text. In this case the event is “Lines Updated On Cart”. The Page Event text is localized by looking up in Sitecore dictionary. Usage: Optional. The processor is generic for triggering a page event when modifying cart lines Ownership: The processor is provided with Connect Customization: Overwrite the processor if other values from cart should be registered or the text formatting should be different

DeleteCart

Name:	**DeleteCart**
Description:	Responsibility is to delete a cart permanently: • The cart is deleted. • Trigger event in DMS telling the cart is deleted.
Usage:	Called from Sitecore.
Args:	
	Request - Contains the cart to be deleted Is set prior to calling the pipeline.
	Response – SystemMessages
Processors:	
	CheckIfLocked –

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Responsibility: Checks if the cart is locked and abort the pipeline if so, returning an SystemMessages to signal the locked state

Usage: Optional. The processor is generic for checking if cart is locked. The processor is and should be used in all pipelines that potentially modify the cart content

Ownership: The processor is provided with Connect

Customization: No immediate need for overwriting the default functionality

DeleteCart –

Responsibility: Deletes and removes cart from storage in ECS

Usage: Mandatory

Ownership: The processor is provided by the ECS

Customization: Must be built specifically for the ECS.

DeleteCartFromEAState –

Responsibility: Locates and deletes cart from EA state

Usage: Optional. The processor is generic for deleting cart in EA state.

Ownership: The processor is provided with Connect

Customization: No immediate need for overwriting the default functionality

TriggerPageEventForRequestCart –

Responsibility: Is to trigger a specified page event when deleting a cart and register the values:

- ExternalCartId, UserId, CartName, CartStatus

The event to be triggered is passed in as parameter as well as the Page Event Text. In this case the event is “Cart Deleted”. The Page Event text is localized by looking up in Sitecore dictionary.

Usage: Optional. The processor is generic for triggering a page event for processors that takes an argument based on parameter type CartRequest

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

	Ownership: The processor is provided with Connect Customization: Overwrite the processor if other values from cart should be registered or the text formatting should be different

UpdateCart

Name:	**UpdateCart**
Description:	Responsibility is to pass an updated cart to the external commerce system Trigger event in DMS telling the cart is being updated.
Usage:	Called from Sitecore.
Args:	
	Request - Contains the cart and the data to be updated in Cart Base Is set prior to calling the pipeline.
	Response - Contains the updated cart object after the pipeline is called.
Processors:	
	CheckIfLocked – Responsibility: Checks if the cart is locked and abort the pipeline if so, returning an SystemMessages to signal the locked state Usage: Optional. The processor is generic for checking if cart is locked. The processor is and should be used in all pipelines that potentially modify the cart content Ownership: The processor is provided with Connect Customization: No immediate need for overwriting the default functionality
	UpdateCart – Responsibility: To update the cart with the updated values of the cart body (CartBase object only, not lines etc.). The default implementation will update all default properties on CartBase except “CustomerID”; “CartName”, “ShopName”. Everything including null and black values are included. Usage: Optional. The processor is generic for updating the cart body. Ownership: The processor is provided with Connect

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Customization: In case the default Connect domain model is customized, the processor should be overwritten to include the customized properties.

TriggerPageEventForRequestCartChanges –

Responsibility: Is to trigger a specified page event when deleting a cart and register the values:

- CustomerId, CartName, ShopName

The event to be triggered is passed in as parameter as well as the Page Event Text. In this case the event is “Cart Deleted”. The Page Event text is localized by looking up in Sitecore dictionary.

Usage: Optional. The processor is generic for triggering a page event for processors that takes an argument based on parameter type UpdateCartRequest

Ownership: The processor is provided with Connect

Customization: Overwrite the processor if other values from cart should be registered or the text formatting should be different

RunSaveCart –

Responsibility: Is to call pipeline SaveCart which saves the specified cart.

Usage: Optional, but should always be called when changes have been made to the cart, which is the case here. The processor is generic for calling pipeline SaveCart and can be used in other pipelines.

Ownership: The processor is provided with Connect

Customization: No immediate need for overwriting the default functionality

LockCart

Name:	**LockCart**
Description:	Responsibility is to set the cart in a locked state and prevent any modifications
Usage:	Called from Sitecore.
Args:	
	Request - Contains the cart to be locked

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

	Is set prior to calling the pipeline.
	Response - Contains the cart object after the pipeline is called.
Processors:	
	LockCart – Responsibility: Is to set the cart to locked state (IsLocked = true) Usage: Optional. The processor is generic for locking a cart. Ownership: The processor is provided with Connect Customization: The default implementation does not call the ECS. It might be relevant to overwrite or replace the implementation to call the ECS when locking.
	RunSaveCart – Responsibility: Is to call pipeline SaveCart which saves the specified cart. Usage: Optional, but should always be called when changes have been made to the cart, which is the case here. The processor is generic for calling pipeline SaveCart and can be used in other pipelines. Ownership: The processor is provided with Connect Customization: No immediate need for overwriting the default functionality
	TriggerPageEventForRequestCart – Responsibility: Is to trigger a specified page event when deleting a cart and register the values: • ExternalCartId, UserId, CartName, CartStatus The event to be triggered is passed in as parameter as well as the Page Event Text. In this case the event is “CartLocked”. The Page Event text is localized by looking up in Sitecore dictionary. Usage: Optional. The processor is generic for triggering a page event for processors that takes an argument based on parameter type CartRequest Ownership: The processor is provided with Connect Customization: Overwrite the processor if other values from cart should be registered or the text formatting should be different

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

UnlockCart

Name:	**UnlockCart**
Description:	Responsibility is to set the cart in an unlocked state
Usage:	Called from Sitecore.
Args:	
	Request - Contains the cart to unlock Is set prior to calling the pipeline.
	Response - Contains the cart object after the pipeline is called.
Processors:	
	UnlockCart – Responsibility: Is to set the cart to not-locked state (IsLocked = false) Usage: Optional. The processor is generic for unlocking a cart. Ownership: The processor is provided with Connect Customization: The default implementation does not call the ECS. It might be relevant to overwrite or replace the implementation to call the ECS when unlocking.
	RunSaveCart – Responsibility: Is to call pipeline SaveCart which saves the specified cart. Usage: Optional, but should always be called when changes have been made to the cart, which is the case here. The processor is generic for calling pipeline SaveCart and can be used in other pipelines. Ownership: The processor is provided with Connect Customization: No immediate need for overwriting the default functionality
	TriggerPageEventForRequestCart – Responsibility: Is to trigger a specified page event when deleting a cart and register the values: • ExternalCartId, UserId, CartName, CartStatus

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

	The event to be triggered is passed in as parameter as well as the Page Event Text. In this case the event is "Cart Unlocked". The Page Event text is localized by looking up in Sitecore dictionary. Usage: Optional. The processor is generic for triggering a page event for processors that takes an argument based on parameter type CartRequest Ownership: The processor is provided with Connect Customization: Overwrite the processor if other values from cart should be registered or the text formatting should be different

MergeCart

Name:	**MergeCart**
Description:	Responsibility is to merge a User cart with an Anonymous cart.
Usage:	Called from Sitecore.
Args:	
	Request - Contains the User cart and the Anonymous cart.
	Response - Contains the merged User cart.
Processors:	
	MergeCart – Responsibility: is to merge the User and Anonymous carts. Usage: Optional. The processor is generic for merging carts. Ownership: The processor is provided with Connect Customization: The default implementation does not call the ECS. It might be relevant to overwrite or replace the implementation if custom merging rules are required.
	RunSaveCart – Responsibility: Is to call pipeline SaveCart which saves the specified cart. Usage: Optional, but should always be called when changes have been made to the cart, which is the case here. The processor is generic for calling pipeline SaveCart and can be used in other pipelines.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Ownership: The processor is provided with Connect

Customization: No immediate need for overwriting the default functionality

RunDeleteCart -

Responsibility: Is to call pipeline DeleteCart which deletes the specified cart, which in this case is the cart specified in parameter Anonymous Cart

Usage: Optional, depending on whether the result of the merge is to remove one of the carts. Otherwise DeleteCart can be call explicitly

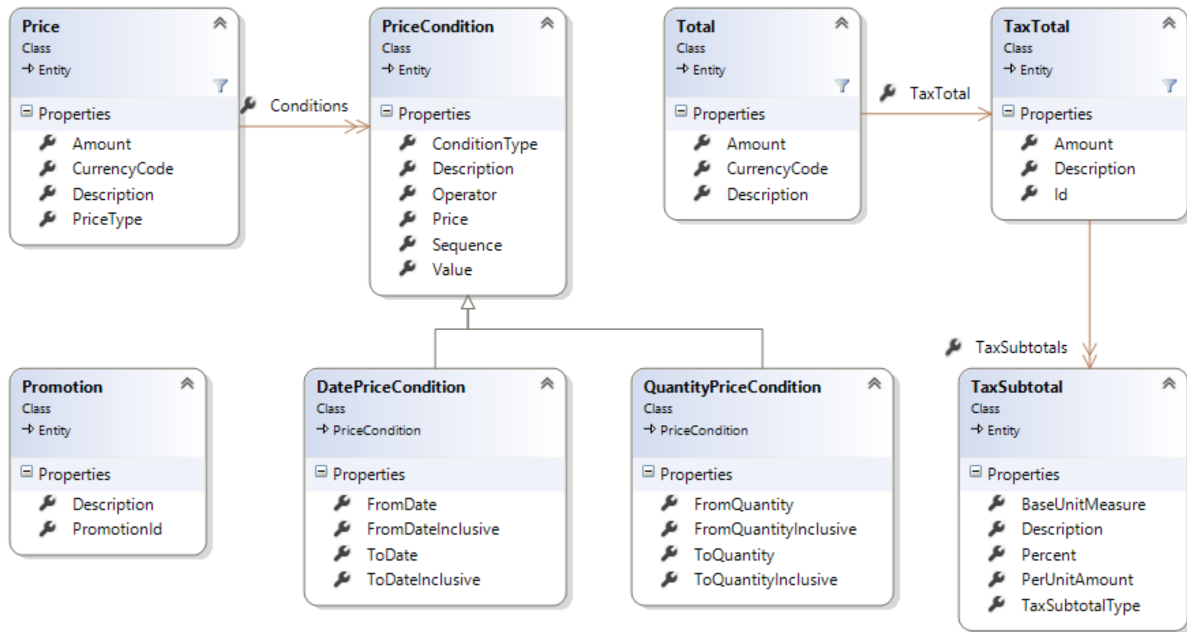
Ownership: The processor is provided with Connect

Customization: No immediate need for overwriting the default functionality

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

2.2 Pricing

2.2.1 The Pricing Domain Model



Note

The information passed in the service layer from Connect framework must be enough to meet the requirements of the external commerce system in order to do its business logic. The Connect framework doesn't perform business logic and therefore the information passed back from the external commerce system is for informational purposes.

Class: Price

Price represents the amount that a product costs. The price is used to determine what a customer has to pay for a single product, but the total cost is represented by the Total object, which takes additional information into consideration like tax, shipping etc.

Price is returned by the Pricing Service Provider as a part of the output from the GetProductPrices method. A product may have multiple prices and multiple prices might be returned from a single call. Hence a collection of Price objects is the output from the Pricing Service Provider when a single product is priced.

Name	Type	Description
PriceType	String	Examples are "List Price"(mandatory) and "Customer Price" (mandatory). Customer price means the price that the customer will have to pay taking all parameters into account. Other custom PriceTypes might be "sale price" and “break price”. There might be several prices for a single product with a given PriceType (e.g. break price), which is where the conditions are used to distinguish when the price is applicable.
Description	String	Arbitrary text description for the price

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Amount	Float	The price amount
CurrencyCode	String	Currency in which the price amount is given
Conditions	List<PriceCondition>	Used for break pricing and campaigns, where a specific price is only good when certain conditions are met (the customer has bought at least 5 products or the date is in the year 2013).

Class: PriceCondition

PriceCondition represents a condition that must be met in order for a price to apply. This interface must be extended for each commerce system with the actual ConditionTypes, operators and possible return values depends.

Note

The condition information is used for presentation and triggering engagement and not for calculation, so the value can be an arbitrary string.

Name	Type	Description
ConditionType	String	Examples are "quantity", "date", and "total"
Description	String	
Operator	String	Examples are "greater than", "equal to" and “between”
Sequence	Int	Represents the order in which the condition is evaluated. For example, one condition may only apply to "shop A" and another condition may only apply to more than 5 items. If the customer is in "shop A" and has more than 5 items, which pricing should be used? This value determines this. Conditions apply in ascending order
Value	String	**Break prices** Break prices depends on the quantity which is why the ConditionType will be Quantity. In case of ConditionType Quantity the value will typically either be: • a single integer with operator “greater than ‘>’” • a range like “5-10” with operator “between” **Campaign prices** In case of ConditionType Date the value will typically either be: • a single date with operator “greater than ‘>’” or “less than ‘<’” • a date range like “A - B” with operator “between”

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Class: DatePriceCondition

DatePriceCondition represents a date-based condition that must be met in order for a price to apply.

Name	Type	Description
FromDate	DateTime?	The date at which the price condition begins to apply. If null, the price condition is valid any time before the ToDate
FromDateInclusive	bool?	Speifies whether the FromDate is inclusive or exclusive when performing the date comparison.
ToDate	DateTime?	The date at which the price condition ends applying. If null, the price condition is valid any time after the FromDate.
ToDateInclusive	bool?	Specifies whether the ToDate is inclusive or exclusive when performing the date comparison.

Class: QuantityPriceCondition

QuantityPriceCondition represents a quantity-based condition that must be met in order for a price to apply.

Name	Type	Description
FromQuantity	decimal?	The quantity at which the price condition begins to apply. If null, the price condition is valid for any quantity less than ToQuantity
FromQuantityInclusive	bool?	Speifies whether the FromQuantity is inclusive or exclusive when performing the comparison.
ToQuantity	decimal?	The quantity at which the price condition ends applying. If null, the price condition is valid any quantity greater than FromQuantity.
ToQuantityInclusive	bool?	Speifies whether the ToQuantity is inclusive or exclusive when performing the comparison.

Class: Total

Total represents the total price a customer will have to pay for a product, cart-line or entire cart at a specific point in time including charges, discounts, coupon codes, tax and shipping etc.

Total is the output from the Pricing Service Provider when a collection of products is priced as a unit (aka bundling). It provides a total price for the entire collection.

Name	Type	Description
Description	String	Arbitrary text
Amount	Float	Representing the total amount
CurrencyCode	String	A code referring to the currency
TaxTotal	TaxTotal	A reference to TaxTotal describing the tax and how it is combined from tax sub-totals

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Class: TaxTotal

TaxTotal represents the tax that applies to something with a Total. Any object with a Total also has a TaxTotal.

Name	Type	Description
Id	String	This value is only available if an external system is used to perform the tax lookup and the external system provides an id (for audit purposes, for example).
Description	String	Arbitrary text
Amount	Float	Representing the total tax amount. Currency is assumed to be the same as for the Total
TaxSubtotals	List<TaxSubtotal>	A list of entries that affect the total tax

Class: TaxSubtotal

TaxSubtotal represents a specific tax that applies to an object, and the amount of the tax. This level of granularity is required in some countries.

Name	Type	Description
TaxSubtotalType	String	Examples are "CA state tax", "NYC city tax", "special levy 003a"
Description	String	
Percent	Float	Percentage per unit, zero if fixed value is used
PerUnitAmount	Float	Fixed value per unit, zero if percentage is used
BaseUnitMeasure	Float	The number of items in a unit for which the fixed value (PerUnitAmount) applies to. Not applicable if percentage is used.

Class: Promotion

Promotion represents a limited time discount or benefit that can be applied to a product.

Note

The promotion information is used for presentation and triggering engagement and not for calculation, so the value can be an arbitrary string.

Name	Type	Description
Description	String	A friendly description of the promotion that can be displayed to the end customer.
PromotionId	String	The unique ID of the promotion.

2.2.2 Pricing Service Methods

Service providers are wrapper objects designed to make it easier to interact with Connect pipelines. The providers implement no logic other than calling Connect pipelines. All of the business logic is implemented in the pipeline processors.

The Pricing Service Provider contains the following methods for interacting with pricing data.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

GetProductPrices

Description:	**Gets the prices for a specific product.**	
Usage:	Called when Sitecore needs the prices for a specific product.	
Signature:	`GetProductPricesResult GetProductPrices(GetProductPricesRequest request)`	
Parameters:		
	`request.ProductId` –	Required
	`Request.UserId` –	Prices typically vary depending on the actual user
	`Request.CurrencyCode` –	Required
	`Request.Location` –	Prices often depends on the location. Location can be a city or a state.
	`Request.Quantity` –	If not specified, quantity is assumed to be 1.
	`Request.DateTime` –	Needed when campaigns promote products at discount prices within a certain period of time
	`Request.ShopName` –	Multi-shop support
	`Request.PriceTypeIds` –	List of the types of prices to retrieve. If not specified, only the base/list price is returned. Examples include list, break, and sale prices. The actual PriceTypeIds depends on the specific Connect provider implementation
Returns:		
	`result.Prices` –	A collection of price objects
	`result.ExternalSystemMessages` –	Collection of messages from the external system. This is how error conditions can be reported.
Exceptions:		
		No exceptions are thrown by this method.

Usage Example:

```
var pricingServiceProvider = new PricingServiceProvider();

// Create a GetProductPricesRequest object, specify the product's ID and do not
// specify any price types. Default price type is ListPrice
var request = new GetProductPricesRequest("Audi A8L");

// Call the service provider and receive the result.
var result = pricingServiceProvider.GetProductPrices(request);

// Result prices contains the list price only.
var price = result.Prices.First().Value.Amount;

// You can use the GetProductPrices to get the prices of a specific type.
// The following sample shows an example of retrieving a price of type Customer
// opposed to the default List price type:

// Create a GetProductPricesRequest object, specify the product's ID and price type
// 'Customer'.
request = new GetProductPricesRequest("Audi A8L", "Customer");

// Call service provider and receive the result.
var result2 = pricingServiceProvider.GetProductPrices(request);

// Result prices contains the Customer price only.
var price2 = result.Prices.First().Value.Amount;
```

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

GetProductBulkPrices

Description:	**Gets the bulk prices for a specific product.**
Usage:	Called when Sitecore needs the break prices for a specific product.
Signature:	`GetProductBreakPricesResult GetProductBreakPrices(GetProductBreakPricesRequest request)`
Parameters:	
`request.ProductId` –	Required
`Request.UserId` –	Prices typically vary depending on the actual user
`Request.CurrencyCode` –	Required
`Request.Location` –	Prices often depends on the location. Location can be a city or a state.
`Request.Quantity` –	If not specified, quantity is assumed to be 1.
`Request.DateTime` –	Needed when campaigns promote products at discount prices within a certain period of time
`Request.ShopName` –	Multi-shop support
Returns:	
`result.Prices` –	A collection of price objects
`result.ExternalSystemMessages` –	Collection of messages from the external system. This is how error conditions can be reported.
Exceptions:	
	No exceptions are thrown by this method.

Usage Example:

```
var pricingServiceProvider = new PricingServiceProvider();

// Create a GetProductPricesRequest object, specify the product's ID and price type
// 'Customer'. The price type argument is optional and defaults to List.
var request = new GetProductBulkPricesRequest(
    new List<string>()
    {
        "Audi A8L",
        "Renault Grand Scenic",
        "Skoda Octavia RS"
    },
    "Customer");

// Call service provider and receive the result.
var result = pricingServiceProvider.GetProductBulkPrices(request);

// Result contains a dictionary of <key, value> pairs, where the key is the
// product ID and the value represent the corresponding Price.
var price = result.Prices["Audi A8L"].Amount;
```

GetCartTotal

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Description:	**Gets the price for a specific cart.**
Usage:	Called when Sitecore needs the price for a specific cart.
Signature:	`GetCartPriceResult GetCartPrice(GetCartPriceRequest request)`
Parameters:	
`request.Cart` –	Required
`request.UserId` –	Prices typically vary depending on the actual user
`Request.CurrencyCode` –	Required
`Request.Location` –	Prices often depends on the location. Location can be a city or a state
`Request.ShopName` –	Multi-shop support
`Request.DateTime` –	Needed when campaigns promote products at discount prices within a certain period of time
Returns:	
`result.Cart` –	An instance of a Total
`result.ExternalSystem Messages` –	Collection of messages from the external system. This is how error conditions can be reported.
Exceptions:	
	No exceptions are thrown by this method.

Usage Example:

```
var cartServiceProvider = new CartServiceProvider();
var pricingServiceProvider = new PricingServiceProvider();

// Create LoadCart request.
var cartRequest = new CreateOrResumeCartRequest("MyShop", "MyCart");

// Call CreateOrResumeCart and get the cart
var cart = cartServiceProvider.CreateOrResumeCart(cartRequest).Cart

// Create a GetCartTotalRequest object, specify the Cart and shop name
var request = new GetCartTotalRequest {Cart = cart, ShopName = "MyShop"};

// Call service provider and receive the result.
var result = pricingServiceProvider.GetCartTotal(request);

// Result contains the updated cart augmented with Total, TaxTotal,
// and TaxSubTotal instances
var cartTotal = result.Cart.Total.Amount;
```

GetSupportedCurrencies

Description:	**Gets a list of currencies supported by the ECS.**
Usage:	Called when Sitecore needs the list of supported currencies.
Signature:	`GetSupportedCurrenciesResult GetSupportedCurrencies([NotNull] GetSupportedCurrenciesRequest request)`
Parameters:	
	ShopName – Mandatory The name of the shop
Returns:	
	IReadOnlyCollection<string> – A list of all supported currency codes.
	SystemMessages - Collection of messages from the external system.
Exceptions:	
	No exceptions are thrown by this method.

Usage Example:

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

```
        var provider = (PricingServiceProvider)Factory.CreateObject("pricingServiceProvider",
true);
        var request = new GetSupportedCurrenciesRequest("StarterKit");
        var result = provider.GetSupportedCurrencies(request);
        if (result.Success)
        {
            foreach (var currency in result.Currencies)
            {
                // handle currency.
            }
        }
```

CurrencyChosen

Description:	**Raises the "Currency Chosen" page event.**
Usage:	Called when Sitecore needs to raise the "Currency Chosen" event.
Signature:	`ServiceProviderResult CurrencyChosen(CurrencyChosenRequest request)`
Parameters:	
	ShopName – Mandatory The name of the shop
	ChosenCurrency – Mandatory The chosen currency code
Returns:	
	SystemMessages - Collection of messages from the external system.
Exceptions:	
	No exceptions are thrown by this method.

Usage Example:

```
        var provider = (PricingServiceProvider)Factory.CreateObject("pricingServiceProvider",
true);
        var request = new CurrencyChosenRequest("StarterKit", "USD");
        var result = provider.CurrencyChosen(request);
        if (!result.Success)
        {
            foreach (var message in result.SystemMessages)
            {
                // handle error messages.
            }
        }
```

GetEligiblePromotionIds

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Description:	**Retrieves the IDs of promotions that can be applied to a product.**
Usage:	Called to retrieve the IDs of promotions that can be applied to a product, for example, on the product details page.
Signature:	`GetEligiblePromotionIdsResult GetEligiblePromotionIds( GetEligiblePromotionIdsRequest request)`
Parameters:	
	Shop – Mandatory The target shop.
	ProductId – Mandatory The ID of the product for which promotions should be retrieved.
	TargetTime – Optional The time context that will be used to determine which promotions are elibible.
	CustomerId – Optional The ID of the customer for which the promotions are being retrieved.
Returns:	
	PromotionIds - Collection containing the IDs of the promotions that are applicable to the product.
	SystemMessages - Collection of messages from the external system.
Exceptions:	
	No exceptions are thrown by this method.

Usage Example:

```
        var provider = (PricingServiceProvider)Factory.CreateObject("pricingServiceProvider",
true);
        var request = new GetEligiblePromotionIdsRequest()
        {
            ProductId = "Product001",
            CustomerId = "TestCustomer"
        };

        var result = provider.GetEligiblePromotionIds(request);
```

GetProductPromotionDescription

Description:	**Retrieves the description of a list of promotions for a product.**
Usage:	Called to retrieve the description of a list of promotions for display on the site.
Signature:	`GetProductPromotionDescriptionResult GetProductPromotionDescription(GetProductPromotionDescriptionRequest request)`
Parameters:	
	Shop – Mandatory The target shop.
	PromotionIds – Mandatory The IDs of the promotions to retrieve.
	CustomerId – Optional The ID of the customer for which the promotions are being retrieved.
Returns:	
	Promotions - Collection containing the requested promotions.
	SystemMessages - Collection of messages from the external system.
Exceptions:	
	No exceptions are thrown by this method.

Usage Example:

```
        var provider = (PricingServiceProvider)Factory.CreateObject("pricingServiceProvider",
true);
        var request = new GetProductPromotionDescriptionRequest()
        {
```

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

```
            PromotionIds = new List<string> { "Promotion001", "Promotion002" },
            CustomerId = "TestCustomer"
        };

        var result = provider.GetProductPromotionDescription(request);
```

2.2.3 Pricing Pipelines

GetProductPrices

Name:	**GetProductPrices**	
Description:	Gets the price object that matches the specified criteria. This pipeline is responsible for reading pricing data from a commerce system. This pipeline requests product pricing information from the commerce system and then converts the output into the proper Connect format.	
Usage:	Called by the Pricing Service Provider.	
Args Parameters:		
	`Request` –	Includes the search criteria. Is set by the Pricing Service Provider prior to calling the pipeline.
	`Response` –	Includes the price collection object. Is read by the Pricing Service Provider after the pipeline is called.
Processors:		
	`GetProductPrices` –	Retrieves the prices specified by request.ProductTypeIds.

GetProductBreakPrices

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Name:	**GetProductBreakPrices**	
Description:	Gets the break price objects with corresponding conditions that matches the specified criteria. This pipeline is responsible for reading break pricing data from a commerce system. This pipeline requests product pricing information from the commerce system and then converts the output into the proper Connect format.	
Usage:	Called by the Pricing Service Provider.	
Args Parameters:		
	`Request` -	Includes the search criteria. Is set by the Pricing Service Provider prior to calling the pipeline.
	`Response` -	Includes the price collection object. Is read by the Pricing Service Provider after the pipeline is called.
Processors:		
	`EvaluatePriceConditions` -	In a case where multiple prices exist for the product, determine which price applies. With break prices several prices needs to be returned and conditions created. It's the responsibility of this processor to build the conditions with the associated prices. For more info, see definition of condition

GetCartTotals

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Name:	GetCartTotals	
Description:	Gets the totals object that matches the specified criteria. This pipeline is responsible for reading pricing data from a commerce system. This pipeline converts the contents of a Connect cart into a format the commerce system can understand, requests the commerce system calculate the totals, and then converts the output into the proper Connect format.	
Usage:	Called by the Pricing Service Provider.	
Args Parameters:		
	`Request` –	Includes the search criteria. Is set by the Pricing Service Provider prior to calling the pipeline.
	`Response` –	Includes the totals for the cart. Is read by the Pricing Service Provider after the pipeline is called.
Processors:		
	`ApplyCartAdjustments` –	Adjustments represent charges or discounts that needs to be resolved and applied. For example, discount codes/promotions, special charges for products
	`GetTaxesForCart` –	Taxes might be calculated by a separate service
	`GetShippingChargesForCart` –	Shipping might be calculated by a separate service
	`GetPricesForCart` –	Does the final calculations based on content of cart

GetSupportedCurrencies

Name:	GetSupportedCurrencies	
Description:	Gets the list of supported currencies from the ECS.	
Usage:	Called by the Pricing Service Provider.	
Args Parameters:		
	`Request` –	Includes the current shop name. Is set by the Pricing Service Provider prior to calling the pipeline.
	`Response` –	Includes the list of available currencies. Is read by the Pricing Service Provider after the pipeline is called.
Processors:		
	`GetSupportedCurrencies` –	A placeholder processor that should be replaced with a custom processor that communicates with the ECS

CurrencyChosen

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Name:	**CurrencyChosen**	
Description:	Used to raise the “Currency Chosen” page event when the visitor changes the selected currency	
Usage:	Called by the Pricing Service Provider.	
Args Parameters:		
	`Request` –	Includes the shop name and chosen currency. Is set by the Pricing Service Provider prior to calling the pipeline.
	`Response` –	No data is returned. Is read by the Pricing Service Provider after the pipeline is called.
Processors:		
	`TriggerCurrencyChosenPageEvent` –	Raises the “Currency Chosen” event

GetEligiblePromotionIds

Name:	**commerce.prices.getEligiblePromotionIds**	
Description:	Used to retrieve a list of promotions that are eligible for a product.	
Usage:	Called by the Pricing Service Provider.	
Parameters:		
	`Request` –	A GetEligiblePromotionIdsRequest that Includes the shop name and the ID of the product for which promotions are being retrieved.
	`Response` –	A GetEligiblePromotionIdsResult that contains the IDs of the promotions that are eligible for the product.
Processors:		
	`<none>` –	There is no default implementation for this pipeline. The external commerce system must provide processors for this pipeline.

GetProductPromotionDescription

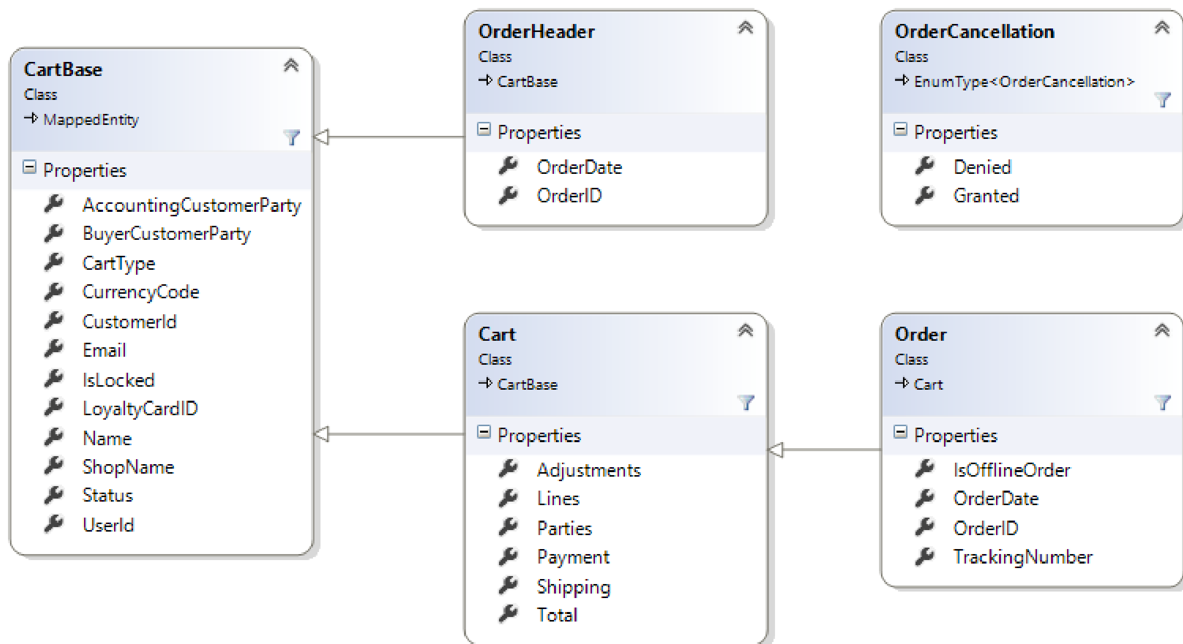
Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Name:	**commerce.prices.getProductPromotionDescription**	
Description:	Used to retrieve the description of promotions.	
Usage:	Called by the Pricing Service Provider.	
Parameters:		
	`Request` -	A GetProductPromotionDescriptionRequest that Includes the shop name and the IDs of the promotions to retrieve.
	`Response` -	A GetProductPromotionDescriptionResult that contains the descriptions of the requested promotions.
Processors:		
	`<none>` -	There is no default implementation for this pipeline. The external commerce system must provide processors for this pipeline.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

2.3 Order

2.3.1 The Order Domain Model



Class: Order

The Order class is responsible for representing an order.

Name	Type	Description
OrderId	String	Unique identifier for the order in the commerce system. This can be used to get a reference to the order using the commerce system's native API. Will typically be empty until an order has been created in external system
OrderDate	DateTime	The data the order was placed.
TrackingNumber	String	The tracking number for the order.
IsOfflineOrder	bool?	Specifies whether the order was placed offline (i.e. in an external system or physical store) and synchronized with the online order and analytics system.

Class: OrderHeader

The OrderHeader class is responsible for representing an order header.

Name	Type	Description
OrderId	String	Unique identifier for the order in the commerce system. This can be used to get a reference to the order using the commerce system's native API.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

		Will typically be empty until an order has been created in external system
OrderDate	DateTime	The date the order was placed.

Class: OrderCancellation

The OrderCancellation class is an extensible enumeration that represents the result status of a request to cancel all or part of an order.

The following table contains the list of default OrderCancellation options.

Name	Type	Description
Granted	OrderCancellation	Indicates that the order cancellation request was granted.
Denied	OrderCancellation	Indicates that the order cancellation request was denied.

2.3.2 Order Service Methods

Service providers are wrapper objects designed to make it easier to interact with Connect pipelines. The providers implement no logic other than calling Connect pipelines. All of the business logic is implemented in the pipeline processors.

The Order Service Provider contains the following methods for interacting with order data.

SubmitVisitorOrder

Name:	**SubmitVisitorOrder**
Description:	Creates an order based on the shopping cart. Calls the pipeline "SubmitVisitorOrder"
Usage:	Called from Sitecore when visitor is submitting the shopping cart to create an order.
Signature:	`SubmitVisitorOrderResult SubmitVisitorOrder(SubmitVisitorOrderRequest request)`
Input:	
	Cart – Cart. An instance of the shopping cart
Output:	
	Order – Cart. *In case of success, an order is returned and because order is inherited from Cart that will work fine but it needs to be cast as an order* *In case of failure, an instance of the cart object is returned, potentially modified and augmented with more data and error comments*
	SystemMessages - Collection of messages from the external system.

Usage Example:

```
        var cartService = new CartServiceProvider();

        // get the cart
        var cart = cartService.CreateOrResumeCart(new CreateOrResumeCartRequest("MyShop",
"Me")).Cart;

        // add parties, payment and shipping info
        cart.Parties = new List<Party>
        {
```

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

```
            new Party() { ExternalId = "1", PartyId = "{F73904C0-2A45-4A2F-A99B-F934ABDCFC99}", FirstName = "Joe", LastName = "Smith", Address1 = "123 Street", City = "Ottawa", State = "Ontario", Country = "Canada" },
            new Party() { ExternalId = "2", PartyId = "{294B7DD1-7397-4322-996C-E87E592EF621}", FirstName = "Jane", LastName = "Smith", Address1 = "234 Street", City = "Toronto", State = "Ontario", Country = "Canada" }
        };

        cart.BuyerCustomerParty = new CartParty() { ExternalId = "1", PartyID = "{F73904C0-2A45-4A2F-A99B-F934ABDCFC99}" };
        cart.AccountingCustomerParty = new CartParty() { ExternalId = "2", PartyID = "{294B7DD1-7397-4322-996C-E87E592EF621}" };

        cart.Payment = new List<PaymentInfo>
        {
            new PaymentInfo() { ExternalId = "1" },
            new PaymentInfo() { ExternalId = "2" },
        };

        cart.Shipping = new List<ShippingInfo>
        {
            new ShippingInfo() { ExternalId = "1" },
            new ShippingInfo() { ExternalId = "2" },
        };

        cartService.SaveCart(new SaveCartRequest(cart));

        var orderService = new OrderServiceProvider();

        var request = new SubmitVisitorOrderRequest(cart);
        var result = orderService.SubmitVisitorOrder(request);

        var order = result.Order;
        var orderId = order.OrderID;
```

GetAvailableCountries

Returns the list of countries supported by the ECS

Name:	**GetAvailableCountries**
Description:	Provides a list of all countries supported by the ECS
Usage:	Called from Sitecore
Signature:	`GetAvailableCountriesResult GetAvailableCountries( GetAvailableCountriesRequest request)`
Input:	
Output:	
	SystemMessages - Collection of countries from the external system.

Usage Example:

```
        var orderService = new OrderServiceProvider();

        var request = new GetAvailableCountriesRequest();

        var result = orderService.GetAvailableCountries(request);
```

GetAvailableRegions

Returns the list of regions in a country supported by the ECS

Name:	**GetAvailableRegions**
Description:	Provides a list of all the regions in a country supported by the ECS

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Usage:	Called from Sitecore
Signature:	`GetAvailableRegionsResult GetAvailableRegions( GetAvailableRegionsRequest request)`
Input:	
	CountryCode – The country to return regions for.
Output:	
	SystemMessages - Collection of countries from the external system.

Usage Example:

```
var orderService = new OrderServiceProvider();

var request = new GetAvailableRegionsRequest("Canada");

var result = orderService.GetAvailableRegions(request);
```

GetVisitorOrder

Name:	**GetVisitorOrder**
Description:	Get the order by Id placed by the visitor. Calls the pipeline " GetVisitorOrder"
Usage:	Called from Sitecore when order details are needed for specific order.
Signature:	`GetVisitorOrderResult GetVisitorOrder(GetVisitorOrderRequest request)`
Input:	
	ShopName – The name of the shop
	OrderId – The ID of the order
	CustomerID – Mandatory The ID of the customer / visitor / contact If the customer ID is not provided there is a potential security risk, that any visitor can access orders from all customers
Output:	
	Order – An instance of the order object is returned. The order object is created by the external commerce system.
	SystemMessages - Collection of messages from the external system.

Usage Example:

```
var orderService = new OrderServiceProvider();

// need a valid order id for the first param
var visitorOrderRequest = new GetVisitorOrderRequest("Order_7777", "Me", "MyShop");

var result = orderService.GetVisitorOrder(visitorOrderRequest);
```

GetVisitorOrders

Name:	**GetVisitorOrders**
Description:	Get the order summary data of orders placed by the given visitor. Calls the pipeline " GetVisitorOrders"
Usage:	Called from Sitecore when order history is needed for visitor.
Signature:	`GetVisitorOrdersResult GetVisitorOrders(GetVisitorOrdersRequest request)`
Input:	

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

	ShopName - The name of the shop
	CustomerId – Id of the buyer customer party
Output:	
	IReadOnlyCollection<OrderBase> – An instance of the order object is returned. The order object is created by the external commerce system.
	SystemMessages - Collection of messages from the external system.

Usage Example:

```
var orderService = new OrderServiceProvider();

var visitorOrdersRequest = new GetVisitorOrdersRequest("Me", "MyShop");

var result = orderService.GetVisitorOrders(visitorOrdersRequest);
```

OrderStatusChanged

Name:	**OrderStatusChanged**
Description:	Used to inform Sitecore that a change has been made to the status of an order
Usage:	Called from Sitecore to inform Sitecore about changes to an order's status.
Signature:	`OrderStatusChangedResult OrderStatusChanged(OrderStatusChangedRequest request)`
Input:	**OrderId** - The id of the order that the status has changed on
	CustomerId - The id of the customer associated to the order
	OrderStatus – The new status of the order
Output:	

Usage Example:

```
var orderService = new OrderServiceProvider();

var visitorOrdersRequest = new GetVisitorOrdersRequest("Me", "MyShop");

var result = orderService.GetVisitorOrders(visitorOrdersRequest);
```

Reorder

Name:	**Reorder**
Description:	Adds one or more items from a previously placed order into the customer's current cart. This calls the "reorder" pipeline.
Usage:	Called when a customer wishes to reorder one or more items from a previously placed order.
Signature:	`CartResult Reorder(ReorderRequest request)`
Input:	
	ShopName – The name of the shop
	OrderId – Mandatory The ID of the order
	CustomerID – Mandatory The ID of the customer / visitor / contact If the customer ID is not provided there is a potential security risk, that any visitor can access orders from all customers
	ReorderLineExternalIds – Optional

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

	The external IDs of the order lines that are being reordered. If not specified, all items from the order will be reordered.
	ForceNewLines – Optional If true, the cart lines items will be added as new cart lines in the customer cart. Otherwise, the lines will be merged into existing cart lines if possible. *Note: This parameter is only supported if the external commerce system supports this functionality.*
Output:	
	Cart – An instance of the customers new cart contents returned.
	AddedCartLineExternalIds – A list containing the IDs of the lines that were added to the customer's cart.
	SystemMessages - Collection of messages from the external system.

Usage Example:

```
var orderService = new OrderServiceProvider();

// need a valid order id for the second param
var reorderRequest = new ReorderRequest("Me", "Order_7777");

var result = orderService.Reorder(reorderRequest);
```

VisitorCancelOrder

Purpose is for a visitor to cancel an existing order if the option is present on the web shop and if business logic does not prevent it. For example, order has already been fulfilled and/or shipped.

Typically triggered when showing order details to the customer launched from the order history view and the customer chooses to cancel the order

Name:	**VisitorCancelOrder**
Description:	Is used to cancel an order placed by the visitor. The decision on whether the order is cancelled or not lies in business logic in the external commerce system. Typically an order cannot be cancelled once its shop owner has started fulfilling/processing it. If the order cannot be cancelled, it must be reflected in the returned SystemMessages
Usage:	Called from Sitecore
Signature:	`VisitorCancelOrderResult VisitorCancelOrder(VisitorCancelOrderRequest request)`
Input:	
	ShopName – Mandatory The name of the shop
	OrderId – Mandatory The ID of the order
	CustomerID – Mandatory The ID of the customer / visitor If the customer ID is not provided there is a potential security risk, that any visitor can access orders from all customers
	OrderLineExternalIds – Optional A list of the ExternalId's of the order lines to cancel. If not specified, all lines of the order will be cancelled. *Note: Connect does not support this parameter by default. It is the responsibility of the external commerce system connectors to handle this property.*
Output:	
	SystemMessages - Collection of messages from the external system.
	CancelledOrder – Contains the order that was cancelled. *Note: It is the responsibility of the external commerce system connectors to populate this result property.*

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

	CancellationStatus – An extensible enumeration value that indicates the status of the order cancellation operation (i.e. Granted, Denied, etc). *Note: It is the responsibility of the external commerce system connectors to populate this result property.*

Usage Example:

```
var orderService = new OrderServiceProvider();

// need a valid order id for the first param
var visitorCancelOrder = new VisitorCancelOrderRequest("Order_7777", "Me", "MyShop");

var result = orderService.VisitorCancelOrder(visitorCancelOrder);
```

2.3.3 Order Pipelines

SubmitVisitorOrder

Name:	**VisitorSubmitOrder**
Description:	This pipeline is responsible for creating an order. The orders are managed by the commerce system.
Usage:	Called from Sitecore.
Args:	
	Request - Contains cart with the Shop name, cart, customer ID and customer party IDs for buyer (shipping) and accounting (Invoice).
	Response - Contains the order object.
Processors:	**CreateOrder –** Creates an order in the external commerce system based on the given parameters *Note: If an error occurs during processing of the cart, the Success property of the SubmitVisitorOrderResult is set to false*
	TriggerOrderGoal– the goal “Visitor Order Created” is triggered with values ShopName, Customer ID, Order Id and total order amount. The engagement value must be set to the amount of the order total!! *Note: If the Success property of the SubmitVisitorOrderResult is false no goal is triggered*
	TriggerLoyaltyCardPurchasePageEvent– the event “Loyalty Card Purchase” is triggered with no values.
	TriggerGiftCardPurchasePageEvent– the event “Gift Card Purchase” is triggered with no values.
	TriggerOrderedProductStockStatusPageEvent– the event “Ordered Product Stock Status” is triggered with no details about each product in the cart and their statuses.
	TriggerOrderOutcome– the order placed outcome is raised along with order related data.
	AddOrderToEAPlan– Adds visitor to EA plan. For example, “New Order Placed”, which sends the order confirmation and follows-up on purchase, customer satisfaction and new offers *Note: If the Success property of the SubmitVisitorOrderResult is false no goal is triggered*

GetAvailableCountries

Name:	**GetAvailableCountries**

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Description:	Gets a list of available countries from the ECS
Usage:	Called from Sitecore.
Args:	
	Request –
	Response – A list of available countries
Processors:	
	GetAvailableCountries – Responsibility: Gets a list of countries from the ECS. Usage: The processor is mandatory Ownership: The processor is provided with the ECS connector integrating with Connect Customization: The processor must always have an implementation that works with the ECS

GetAvailableRegions

Name:	**GetAvailableRegions**
Description:	Gets a list of available regions in a country from the ECS
Usage:	Called from Sitecore.
Args:	
	Request – CountryCode
	Response – A list of available regions
Processors:	
	GetAvailableRegions – Responsibility: Gets a list of regions for a country from the ECS. Usage: The processor is mandatory Ownership: The processor is provided with the ECS connector integrating with Connect Customization: The processor must always have an implementation that works with the ECS

GetVisitorOrders

Name:	**GetVisitorOrders**
Description:	Gets a list of orders for the specified customer
Usage:	Called from Sitecore.
Args:	
	Request – ShopName and CustomerID
	Response – A list of **O**rderBase objects
Processors:	

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

	GetVisitorOrdersFromECS– Responsibility: Get the list of orders for the specified customer from the ECS. It must be possible to have the ECS and Sitecore installed in different locations, so it must be possible to access remotely. Usage: The processor is mandatory Ownership: The processor is provided with the ECS connector integrating with Connect Customization: The processor must always have an implementation that works with the ECS
	TriggerPageEvent – Responsibility: Trigger the Connect specific page event Visitor Viewed Order History along with information about the ShopName and Customer ID Usage: Mandatory. Ownership: The processor is provided with Connect Customization: Not needed, but can be overwritten if other values from the order should be registered with the page event

GetVisitorOrder

Name:	**GetVisitorOrder**
Description:	Gets the order by Id placed by the visitor. Executed from method "GetVisitorOrder"
Usage:	Called from Sitecore.
Args:	
	Request – ShopName and Order ID
	Response – An instance of an order
Processors:	
	GetVisitorOrdersFromECS– Responsibility: Get the order details for the specified order ID from the ECS. Usage: The processor is mandatory Ownership: The processor is provided with the ECS connector integrating with Connect Customization: The processor must always have an implementation that works with the ECS

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

	TriggerPageEvent – Responsibility: Trigger the Connect specific page event Visitor Viewed Order Details along with information about the ShopName, Order ID and total order amount Usage: Mandatory. Ownership: The processor is provided with Connect Customization: Overwrite the processor if other values from the order should be registered

OrderStatusChanged

Name:	**OrderStatusChanged**
Description:	Used to inform Sitecore about a change in the status of an order
Usage:	Called from Sitecore.
Args:	
	Request – Order ID, CustomerId, and OrderStatus
	Response – None
Processors:	
	TriggerOrderStatusChangedPageEvent – Responsibility: Trigger the Connect specific page event Order status changed along with information about the Order ID and new Order Status Usage: Mandatory. Ownership: The processor is provided with Connect Customization: Overwrite the processor if other values from the order should be registered

Reorder

Name:	**Reorder**
Description:	This pipeline is responsible for adding lines to a customer's cart from a previously placed order, for purposes of reordering those items.
Usage:	Called when a customer requests to reorder one or more items from a previous order.
Args:	
	Request - Contains the Shop name, order ID, and customer ID.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

	Response - Contains the resulting cart object and a list of the new cart lines added to the cart.
Processors:	**GetReorderSource –** Loads the order from which items are being reordered. This processor calls the getVisitorOrder pipeline. *Note: If an error occurs while loading the order, the Success property of the ReorderResult is set to false*
	AddReorderLinesToCart – Loads the customer's current cart and adds new lines to this cart based on the items being reordered. This processor calls the createOrResumeCart and addCartLines pipelines. *Note: If the Success property of the ReorderResult is false no items are added to the customer's cart. If the Success property is true, at least one item was successfully added to the cart.*
	AddReorderShippingInfoToCart – Adds parties and shipping information associated with the reordered items to the custommer's cart. This processor calls the addPartiesToCart and addShippingInfo pipelines.
	TriggerReorderRequestedPageEvent – Triggers the "Reorder Requested" page event. *Note: If the Success property of the ReorderResult is false no page event is triggered.*

VisitorCancelOrder

Name:	VisitorCancelOrder
Description:	Called when a visitor order is being cancelled
Usage:	Called from Sitecore.
Args:	
	Request – OrderId, CustomerId, OrderLineExternalIds and ShopName
	Response – an instance of the order
Processors:	
	VisitorCancelOrderFromECS– Responsibility: Get the order details for the specified order ID from the ECS. Usage: The processor is mandatory Ownership: The processor is provided with the ECS connector integrating with Connect Customization: The processor must always have an implementation that works with the ECS
	TriggerPageEvent – Responsibility: Trigger the Connect specific page event Visitor Cancelled Order Details along with information about the ShopName, Order ID and total order amount Usage: Mandatory. Ownership: The processor is provided with Connect

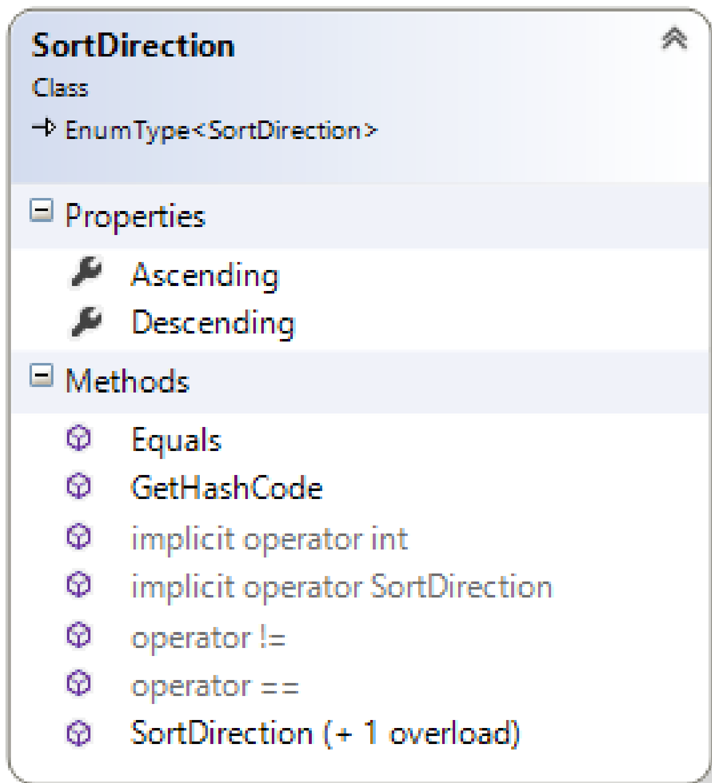
Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

	Customization: Overwrite the processor if other values from the order should be registered

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

2.4 Catalog

2.4.1 The Catalog Domain Model



Note: The domain model consists of classes that make up the contracts with the external system. The contracts are defined as classes instead of interfaces to allow the model to be easily extended later if needed. This follow the best practice guidelines defined in the book Framework Design Guidelines.

Default implementation of the contracts are delivered as part of Connect. If an actual Connect provider with an external commerce system contains more functionality that provided by default, the implementation can be replaced. All instantiation of actual classes will be handled through dependency injection.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Class: SortDirection

Class SortDirection is used as a strongly typed value to indicate the sort direction of a search. Using a strongly typed value will ease the use of the API for solution developers. The following example illustrates the use of the class as an enum-like parameter.

Name	Type	Description
Ascending	public const int Ascending = 1	Indicates that the request should be sorted ascending
Descending	public const int Descending = 2	Indicates that the request should be sorted descending

2.4.2 Catalog Service Methods

Service providers are wrapper objects designed to make it easier to interact with Connect pipelines. The providers implement no logic other than calling Connect pipelines. All of the business logic is implemented in the pipeline processors.

The Catalog Service Provider contains the following methods for interacting with catalog search data.

ProductSorting

Name:	ProductSorting
Description:	A method to let Commerce Connect know when a user has performed a page sort, so that the event can be tracked in xDB Calls the pipeline "`productSorting`".
Usage:	Called from Sitecore when user performs a sort.
Signature:	`CatalogResult ProductSorting(ProductSortingRequest request)`
Input:	
	ShopName – string. Mandatory The name of the shop
	SortKey – string. Mandatory The field that was sorted on in the search
	SortDirection – SortDirection. Mandatory The direction in which the search was sorted i.e. Asc or Desc
Output:	

Usage Example:

```
        var provider = (CatalogServiceProvider)Factory.CreateObject("catalogServiceProvider", true);
        var request = new ProductSortingRequest("StarterKit", "Price", SortDirection.Ascending);
        var result = provider.ProductSorting(request);
```

FacetApplied

Name:	FacetApplied
Description:	A method to let Commerce Connect know when a user has tried to facet some search information, so that the event can be tracked in xDB Calls the pipeline "`facetApplied`".
Usage:	Called from Sitecore when user facets on search results

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Signature:	`CatalogResult FacetApplied(FacetAppliedRequest request)`
Input:	
	ShopName – String. Mandatory The name of the shop
	Facet - String. Mandatory The facet field that the search was performed on
	IsApplied – bool. Optional Used to indicate whether or not the facet was applied
Output:	

Usage Example:

```
        var provider = (CatalogServiceProvider)Factory.CreateObject("catalogServiceProvider",
true);
        var request = new FacetAppliedRequest("StarterKit", "Manufacturer", true);
        var result = provider.FacetApplied(request);
```

VisitedCategoryPage

Name:	**VisitedCategoryPage**
Description:	A method to let Commerce Connect know when a user visits a category page, so that the event can be tracked in xDB Calls the pipeline "`visitedCategoryPage`".
Usage:	Called from Sitecore when user visits a category page
Signature:	`CatalogResult VisitedCategoryPage(VisitedCategoryPageRequest request)`
Input:	
	ShopName – String. Mandatory The name of the shop
	CategoryId - string. Mandatory The id of the category the user visited
	CategoryName – string. Manditory The name of the category the user visited
Output:	

Usage Example:

```
        var provider = (CatalogServiceProvider)Factory.CreateObject("catalogServiceProvider",
true);
        var request = new VisitedCategoryPageRequest("StarterKit", "2", "Computers");
        var result = provider.VisitedCategoryPage(request);
```

VisitedProductDetailsPage

Name:	**VisitedProductStockStatus**
Description:	A method to let Commerce Connect know when a user visits a product details page, so that the event can be tracked in xDB Calls the pipeline "`visitedProductDetailsPage`".
Usage:	Called from Sitecore when user visits a product details page
Signature:	`CatalogResult VisitedProductDetailsPage(VisitedProductDetailsPageRequest request)`
Input:	
	ShopName – string. Mandatory The name of the shop
	ProductId - string. Mandatory

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

	The id of the visited product
	ProductName – string. Mandatory The name of the visited product
	ParentCategoryId – string. Mandatory The category of the visited product
	ParentCategoryName – string. Mandatory The name of the category of the visited product
	Amount – decimal. Optional The price of the visited product
	CurrencyCode – decimal. Optional The currency the user viewed the product in
Output:	

Usage Example:

```
        var provider = (CatalogServiceProvider)Factory.CreateObject("catalogServiceProvider",
true);
        var request = new VisitedProductDetailsPageRequest("StarterKit", "23", "Diamond Pave
Earrings", "17", "Jewelry");
        var result = provider.VisitedProductDetailsPage(request);
```

SearchInitiated

Name:	**SearchInitiated**
Description:	A method to let Commerce Connect know when a user performs a keyword search, so that the event can be tracked in xDB Calls the pipeline "`searchInitiated`".
Usage:	Called from Sitecore when user visits a keyword search
Signature:	`CatalogResult SearchInitiated(SearchInitiatedRequest request)`
Input:	
	Shop Name – string. Mandatory The name of the shop for which this relates
	SearchTerm – string. Mandatory The keywords the user searched for
	NumberOfHits – int. Optional The number of results found.
Output:	

Usage Example:

```
                    var provider =
(CatalogServiceProvider)Factory.CreateObject("catalogServiceProvider", true);
                    var request = new SearchInitiatedRequest("StarterKit", "Diamond Tennis
Bracelet", 1 );
                    var result = provider.SearchInitiated(request);
```

2.4.3 Catalog Pipelines

Product Sorting

Name:	**ProductSorting**
Description:	This pipeline is responsible for triggering page event “Product Sorting”

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Usage:	Called from Sitecore.
Args:	
	Request - Contains the additional pageevent event information
	Response - None
Processors:	
	TriggerProductSortingPageEvent– Responsibility: To trigger page event “Product Sorting” to register a product sort used by the visitor Usage: Called from Sitecore and typically doesn’t call the ECS at all Ownership: The processor is provided with Connect Customization: No immediate need for overwriting the default functionality, unless more information should be registered with the page event

Facet Applied

Name:	**FacetApplied**
Description:	This pipeline is responsible for triggering page event “Facet Applied”
Usage:	Called from Sitecore.
Args:	
	Request - Contains the additional pageevent event information
	Response - None
Processors:	
	TriggerFacetAppliedPageEvent – Responsibility: To trigger page event “Facet Applied” to register the facet used by the visitor Usage: Called from Sitecore and typically doesn’t call the ECS at all Ownership: The processor is provided with Connect Customization: No immediate need for overwriting the default functionality, unless more information should be registered with the page event

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Visited Product Details Page

Name:	**VisitedProductDetailsPage**
Description:	This pipeline is responsible for triggering page event “Visited Product Details Page”
Usage:	Called from Sitecore.
Args:	
	Request - Contains the additional pageevent event information
	Response - None
Processors:	
	TriggerVisitedProductDetailsPagePageEvent – Responsibility: To trigger page event “Visited Product Details Page” to register the product details page view by the visitor Usage: Called from Sitecore and typically doesn’t call the ECS at all Ownership: The processor is provided with Connect Customization: No immediate need for overwriting the default functionality, unless more information should be registered with the page event

Visited Category Page

Name:	**VisitedCategoryPage**
Description:	This pipeline is responsible for triggering page event “Visited Category Page”
Usage:	Called from Sitecore.
Args:	
	Request - Contains the additional pageevent event information
	Response - None
Processors:	
	TriggerVisitedCategoryPagePageEvent – Responsibility: To trigger page event “Visited Category Page” to register the category page viewed by the visitor

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

	<u>Usage</u>: Called from Sitecore and typically doesn't call the ECS at all <u>Ownership</u>: The processor is provided with Connect <u>Customization</u>: No immediate need for overwriting the default functionality, unless more information should be registered with the page event

Search Initiated

Name:	**SearchInitiated**
Description:	This pipeline is responsible for triggering page event "Search"
Usage:	Called from Sitecore.
Args:	
	Request - Contains the additional pageevent event information
	Response - None
Processors:	
	TriggerSearchPageEvent – <u>Responsibility</u>: To trigger page event "Search" to register the keyword search performed by the visitor <u>Usage</u>: Called from Sitecore and typically doesn't call the ECS at all <u>Ownership</u>: The processor is provided with Connect <u>Customization</u>: No immediate need for overwriting the default functionality, unless more information should be registered with the page event

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

2.5 Globalization

2.5.1 The Globalization Domain Model

The Globalization service does not contain any custom domain models.

2.5.2 Globalization Service Methods

Service providers are wrapper objects designed to make it easier to interact with Connect pipelines. The providers implement no logic other than calling Connect pipelines. All of the business logic is implemented in the pipeline processors.

The Globalization Service Provider contains the following methods raising globalization related events.

CultureChosen

Description:	**Raises the "Culture Chosen" page event.**
Usage:	Called when Sitecore needs to raise the "Culture Chosen" event.
Signature:	`GlobalizationResult CultureChosen(CultureChosenRequest request)`
Parameters:	
	ShopName – Mandatory The name of the shop
	Culture – Mandatory The chosen culture code
Returns:	
	SystemMessages - Collection of messages from the external system.
Exceptions:	
	No exceptions are thrown by this method.

Usage Example:

```
        var provider =
(GlobalizationServiceProvider)Factory.CreateObject("globalizationServiceProvider", true);
        var request = new CultureChosenRequest("StarterKit", "en-US");
        var result = provider.CultureChosen(request);
        if (!result.Success)
        {
            // handle error.
        }
```

2.5.3 Globalization Pipelines

CultureChosen

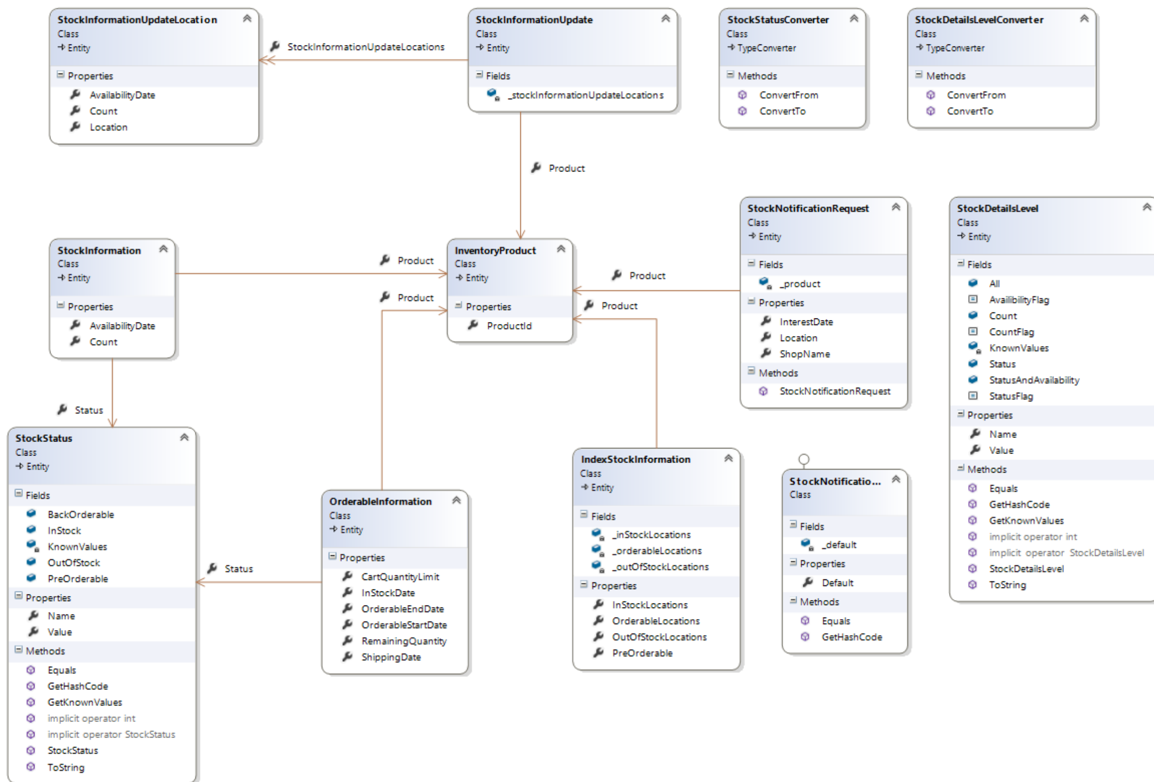
Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Name:	**CultureChosen**	
Description:	Used to raise the “Culture Chosen” page event when the visitor changes the selected currency	
Usage:	Called by the Globalization Service Provider.	
Args Parameters:		
	`Request` -	Includes the shop name and chosen culture. Is set by the Globalization Service Provider prior to calling the pipeline.
	`Response` -	No data is returned. Is read by the Globalization Service Provider after the pipeline is called.
Processors:		
	`TriggerCultureChosenPageEvent`-	Raises the “Culture Chosen” event

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

2.6 Inventory

2.6.1 The Inventory Domain Model



Note: The domain model consists of classes that make up the contracts with the external system. The contracts are defined as classes instead of interfaces to allow the model to be easily extended later if needed. This follow the best practice guidelines defined in the book Framework Design Guidelines.

Default implementation of the contracts are delivered as part of Connect. If an actual Connect provider with an external commerce system contains more functionality that provided by default, the implementation can be replaced. All instantiation of actual classes will be handled through dependency injection.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Class: StockInformation

StockInformation is used as a strongly typed composite return value for service method GetStockInformation.

Name	Type	Description
Product	InventoryProduct	Identifier for the product or product variant in the commerce system
Status	StockStatus	Default possible values are: In-Stock, Out-Of-Stock, Pre-Orderable, Back-Orderable
Count	Double	In case of products being bundled in quantities there might be fractional numbers
AvailabilityDate	DateTime	In Case the product is out-of-stock or pre-orderable, an availability date can be present

Class: OrderableInformation

OrderableInformation is used as a strongly typed composite return value for service methods GetPreOrdableInformation and GetBackOrderableInformation.

Name	Type	Description
Product	InventoryProduct	Identifier for the product or product variant in the commerce system
Status	StockStatus	Default possible values are: In-Stock, Out-Of-Stock, Pre-Orderable, Back-Orderable
InStockDate	Datetime	An ETA date for when the product is back in stock
ShippingDate	DateTime	An ETA date for when the product is shippable
CartQuantityLimit	Double	A limit for the visitor to add to his or her cart
OrderableStartDate	DateTime	A date and time for when the first orders can be placed for the given product
OrderableEndDate	DateTime	A date and time for when the last orders can be placed for the given product
RemainingQuantity	Double	In case of a pre-orderable product then there might be a remaining quantity to be placed as orders

Class: IndexStockInformation

IndexStockInformation is used as a strongly typed composite value used in pipeline StockStatusForIndexing when indexing products and including basic stock information.

The entity inherits from the base entity StockLocations. In the table, the inherited properties are marked in Italics

Name	Type	Description

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Product	InventoryProduct	Identifier for the product or product variant in the commerce system
InStockLocations	*List<string>*	*A list of locations where the product is in stock*
OutOfStockLocations	*List<string>*	*A list of locations where the product is out of stock*
OrderableLocations	*List<string>*	*A list of locations where the product can be ordered from*
PreOrderable	Boolean	Indicates if the product is pre-orderable or not

Class: StockInformationUpdate

StockInformationUpdate is used as a strongly typed composite return value from method GetBackInStockInformation to indicate the product and the locations where it will be back in stock optionally along with availability date and count.

Name	Type	Description
Product ID	String	Id of the product
StockInformationUpdateLocation	List< StockInformationUpdateLocation >	A list of locations where the product will become available along with the count and availability date as optional values

Class: StockInformationUpdateLocation

StockInformationUpdateLocation is used as a strongly typed value nested only into StockInformationUpdate returned from method GetBackInStockInformation to indicate the locations where the product will be back in stock optionally along with availability date and count.

Name	Type	Description
Location	String	Name of the location
AvailabilityDate	DateTime?	An optional date and time indicating when the product will be in stock. It can be used in comparison with the optional interest date that the visitor provided. If the interest date is before the availability date, then the notification signup becomes irrelevant and it can be acted upon
Count	Double?	An optional count of products that are in stock at the location at the specified date and time

Class: StockDetailsLevel

It is anticipated that there is a performance related difference between obtaining a simple stock status and getting the actual stock count. In order to allow for flexibility it is possible to specify the level of details that are requested.

Class StockDetailsLevel is used as a strongly typed request parameter for service method GetStockInformation to indicate the level of stock details that is requested. Using a strongly typed parameter will ease the use of the API for solution developers. The following example illustrates the use of the class as an enum-like parameter.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

```
StockInformation stockInformation = GetStockInformation(
        new StockInformationRequest {   shopName = "MyShop";
                                        products = new list<string> { "Aw123x" };
                                        detailsLevel = StockDetailsLevel.Status
                                      }).Result;
```

The following table contains the list of default StockDetailsLevel options. Below is an example of how the list of options can be extended.

Name	Type	Description
Status	public const int Status = 1	Indicates that the minimum information is to be returned, which is stock status
StatusAndAvailability	public const int StatusAndAvailability = 2	Indicates that the status and availability date information is to be returned. Availability date is relevant in case status is equal to
Count	public const int Count = 3	
All	public const int All = 4	

The class is introduced as an extensible enum. In order to extend and customize the StockDetailsLevel options:

```
public class MyECSStockDetailsLevel : StockDetailsLevel
{
      public const int MyCustomDetailLevel = 4;
      public MyECSStockDetailsLevel (int value)    : base(value)
      {    }
}
```

Class: StockStatus

Class StockStatus is used as a strongly typed value to indicate stock status. Using a strongly typed value will ease the use of the API for solution developers. The following example illustrates the use of the class as an enum-like parameter.

```
StockInformation stockInformation = GetStockInformation(
        new StockInformationRequest {   shopName = "MyShop";
                                        products = new list<string> { "Aw123x" };
                                        detailsLevel = StockDetailsLevel.Status
                                      }).Result.StockInformation[0];

If (stockInformation.Status == StockStatus.PreOrderable)
{
 // Do work
}
```

The following table contains the list of default StockStatus options. Below is an example of how the list of options can be extended.

Name	Type	Description
InStock	public const int InStock = 1	Indicates that the requested product is in stock
OutOfStock	public const int OutOfStock = 2	Indicates that the requested product is out of stock
PreOrderable	public const int PreOrderable = 3	Indicates that the requested product is not in stock yet, but is pre-orderable

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

BackOrderable	public const int BackOrderable = 4	Indicates that the requested product is out of stock, but is back-orderable

The class is introduced as an extensible enum. In order to extend and customize the StockDetailsLevel options:

```
public class MyECSStockStatus : StockStatus
{
      public const int MyCustomStatus = 4;
      public MyECSStockStatus (int value)    : base(value)
      {    }
}
```

Class: InventoryProduct

Class InventoryProduct is used as a strongly typed value to identify a product. Using a strongly typed value will ease the use of the API for solution developers.

Name	Type	Description
ProductId	string	Unique identifier for the product or product variant in the commerce system

Class: IndexStockInformation

IndexStockInformation is used as a strongly typed composite value used in pipeline GetProductStockLocations when retrieving locations for a particular product.

IndexStockInformation is also used as the base entity.

Name	Type	Description
InStockLocations	List<string>	A list of locations where the product is InStock
OutOfStockLocations	List<string>	A list of locations where the product is out of stock
OrderableLocations	List<string>	A list of locations where the product can be ordered from
PreOrderable	Bool	Is the item preorderable.

Class: StockLocation

StockLocation is used to indicate the stock location of a product.

Name	Type	Description
LocationId	String	The id of the stock location.
Name	String	The name of the stock location

2.6.2 Inventory Service Methods

Service providers are wrapper objects designed to make it easier to interact with Connect pipelines. The providers implement no logic other than calling Connect pipelines. All of the business logic is implemented in the pipeline processors.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

The Inventory Service Provider contains the following methods for interacting with inventory data.

GetStockInformation

Name:	**GetStockInformation**
Description:	Retrieves different levels of stock information from the ECS Calls the pipeline "`GetStockInformation`".
Usage:	Called from Sitecore when stock information is needed for a list of specified products
Signature:	`GetStockInformationResult GetStockInformation(GetStockInformationRequest request)`
Input:	
	ShopName – string. Mandatory The name of the shop
	Products – list<InventoryProduct>. Mandatory A list of InventoryProduct. Whether it is products or product variants is up to the implementation with the ECS
	DetailsLevel – StockDetailsLevel. Mandatory Class StockDetailsLevel is used as a strongly typed request parameter for service method GetStockInformation to indicate the level of stock details that is requested
	Location – string. Optional The specific warehouse or central storage where the stock information is Default is Central storage
	Customer ID – string. Optional The ID of the customer in case the stock information is dependent on the actual customer
Output:	
	List<StockInformation> StockInformation – A list of StockInformation objects
	SystemMessages - Collection of messages from the external system.

Usage Example:

```
var inventoryService = new InventoryServiceProvider();

var request = new GetStockInformationRequest("shopname", new List<InventoryProduct>
    {
        new InventoryProduct
        {
            ProductId = "product_1"
        },
        new InventoryProduct
        {
            ProductId = "product_2"
        },
        new InventoryProduct
        { ProductId = "product_3" }
    },
    StockDetailsLevel.StatusAndAvailability);

var result = inventoryService.GetStockInformation(request);
```

GetPreOrderableInformation

Name:	**GetPreOrderableInformation**
Description:	Retrieves the pre-orderable information

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Usage:	Called from Sitecore
Signature:	`Result [Name](Request request)`
Input:	
	ShopName – String. Mandatory The name of the shop
	Products - List< InventoryProduct **>. Mandatory** The list of InventoryProduct
	Visitor ID – string. Optional **The ID of the visitor / contact or customer**
Output:	
	OrderableInformation – List<OrderableInformation> A list of strongly typed objects each with the information for a specific product
	SystemMessages - Collection of messages from the external system.

Usage Example:

```
var inventoryService = new InventoryServiceProvider();

var request = new GetPreOrderableInformationRequest("shopname",
    new List<InventoryProduct>
    {
        new InventoryProduct
        {
            ProductId = "product_1"
        },
        new InventoryProduct
        {
            ProductId = "product_2"
        },
        new InventoryProduct
        { ProductId = "product_3" }
    });

var result = inventoryService.GetPreOrderableInformation(request);
```

GetBackOrderableInformation

Name:	**[Name]**
Description:	Gets the back-orderable information
Usage:	Called from Sitecore
Signature:	`Result [Name](Request request)`
Input:	
	ShopName – String. Mandatory The name of the shop
	Products - List< InventoryProduct **>. Mandatory** The list of InventoryProduct
	Visitor ID – string. Optional **The ID of the visitor / contact or customer**
Output:	
	OrderableInformation – List<OrderableInformation> A list of strongly typed objects each with the information for a specific product
	SystemMessages - Collection of messages from the external system.

Usage Example:

```
var inventoryService = new InventoryServiceProvider();

var request = new GetBackOrderableInformationRequest("shopname",
    new List<InventoryProduct>
    {
```

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

```
            new InventoryProduct
            {
                ProductId = "product_1"
            },
            new InventoryProduct
            {
                ProductId = "product_2"
            },
            new InventoryProduct
            { ProductId = "product_3" }
        });

    var result = inventoryService.GetBackOrderableInformation(request);
```

VisitedProductStockStatus

Name:	**VisitedProductStockStatus**
Description:	Should be called in the event of the customer pays a visit to a product details page which shows
Usage:	Called from Sitecore
Signature:	`VisitedProductStockStatusResult VisitedProductStockStatus (VisitedProductStockStatusRequest request)`
Input:	
	ShopName – string. Mandatory The name of the shop
	StockInformation - StockInformation. Mandatory The stock information previously retrieved by calling GetStockInformation
	Location – string. Optional The specific warehouse or central storage where the stock information is Default is Central storage
Output:	
	SystemMessages - Collection of messages from the external system.

Usage Example:

```
    var inventoryService = new InventoryServiceProvider();

    var stockInfo = new StockInformation
    {
        Product = new InventoryProduct
        {
            ProductId = "product_1"
        },
        Status = StockStatus.BackOrderable
    };

    var request = new VisitedProductStockStatusRequest("shopname", stockInfo);

    var result = inventoryService.VisitedProductStockStatus(request);
```

ProductsAreBackInStock

Name:	**ProductsAreBackInStock**
Description:	A method that is exposed so that it can be called remotely to signal when products are back in stock. Executes the corresponding pipeline ProductsAreBackInStock
Usage:	Called from ECS

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Signature:	`ProductsAreBackInStockResult ProductsAreBackInStock(ProductsAreBackInStockRequest request)`
Input:	
	Shop Name – string. Mandatory The name of the shop for which this relates
	Products – list< InventoryProduct**>. Mandatory** A list of InventoryProduct that signals which products have updated stock information
Output:	
	SystemMessages - Collection of messages from the external system.

Usage Example:

```
var inventoryService = new InventoryServiceProvider();

var request = new ProductsAreBackInStockRequest("shopname",
    new List<InventoryProduct>
    {
        new InventoryProduct
        {
            ProductId = "product_1"
        },
        new InventoryProduct
        {
            ProductId = "product_2"
        },
        new InventoryProduct
        { ProductId = "product_3" }
    });

var result = inventoryService.ProductsAreBackInStock(request);
```

VisitorSignUpForStockNotification

Name:	**VisitorSignUpForStockNotification**
Description:	Is used to add visitor to EA plan so they can be notified when the product gets back in stock
Usage:	Called from Sitecore
Signature:	`VisitorSignUpForStockNotificationResult VisitorSignUpForStockNotification(VisitorSignUpForStockNotificationRequest request)`
Input:	
	ShopName – string. Mandatory
	Visitor ID – string. Mandatory The ID of the current visitor / contact
	Email – string. Mandatory E-mail address to send the notification to
	Product – InventoryProduct**. Mandatory**
	Location – string. Optional
	InterestDate – DateTime. Optional A date and time that signals the deadline for which to notify the visitor Default is 6 months from now
Output:	
	SystemMessages - Collection of messages from the external system.

Usage Example:

```
var inventoryService = new InventoryServiceProvider();
```

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

```
var request = new VisitorSignUpForStockNotificationRequest("shopname",
    "visitorId",
    "email",
    new InventoryProduct { ProductId = "product_1" });

var result = inventoryService.VisitorSignUpForStockNotification(request);
```

RemoveVisitorFromStockNotification

Name:	RemoveVisitorFromStockNotification
Description:	Typically called from Sitecore, when the visitor has decided to unsubscribe from the stock notification and hence the EA plan
Usage:	Called from Sitecore
Signature:	`RemoveVisitorFromStockNotificationResult RemoveVisitorFromStockNotification(RemoveVisitorFromStockNotificationRequest request)`
Input:	
	ShopName – string. Mandatory
	Visitor ID – string. Mandatory The ID of the current visitor / contact
	Product ID – string. Mandatory
Output:	
	SystemMessages - Collection of messages from the external system.

Usage Example:

```
var inventoryService = new InventoryServiceProvider();

var request = new RemoveVisitorFromStockNotificationRequest("shopname",
    "visitorId",
    new InventoryProduct { ProductId = "product_1" });

var result = inventoryService.RemoveVisitorFromStockNotification(request);
```

GetBackInStockInformation

Name:	GetBackInStockInformation
Description:	The method is used to get the updated stock information from the ECS The method is normally called because of method ProductsAreBackInStock has been called remotely and in that case the pipeline is implicitly called The difference between GetStockInformation and GetBackInStockInformation is that the first queries for stock information in a specified location and for a given customer, where the latter gets the stock information for all locations ignoring customer context
Usage:	Called from Sitecore
Signature:	`GetBackInStockInformationResult GetBackInStockInformation (GetBackInStockInformationRequest request)`
Input:	
	Shop Name – string. Mandatory The name of the shop for which this relates
	Products – list< InventoryProduct **>. Optional** A list of InventoryProduct to get updated stock information from. If the list is empty, for example no list is provided, it is up to the ECS to return stock information updates for the products that have been updated.
Output:	

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

	StockInformationUpdates – list<StockInformationUpdate>. Mandatory A list of StockInformationUpdate object that each signals the product and a list of locations for which the product is back in stock, when (availability date) and the count, where the latter two are optional values.
	SystemMessages - Collection of messages from the external system.

Usage Example:

```
        var inventoryService = new InventoryServiceProvider();

        var request = new GetBackInStockInformationRequest("shopname")
        {
            Products = new List<InventoryProduct>
            {
                new InventoryProduct
                {
                    ProductId = "product_1"
                },
                new InventoryProduct
                {
                    ProductId = "product_2"
                },
                new InventoryProduct
                {
                    ProductId = "product_3"
                }
            }
        };

        var result = inventoryService.GetBackInStockInformation(request);
```

GetStockLocations

Name:	**GetStockLocations**
Description:	Retrieves all of the stock locations from the ECS
Usage:	Called from Sitecore when a list of stock locations in the ECS is required
Signature:	`GetStockLocationsResult GetStockLocations(GetStockLocationsRequest request)`
Input:	
	ShopName – string. The name of the shop
	CustomerId – string. Optional The id of the customer who needs the list of stock locations
Output:	
	List<StockLocation> StockLocations – A list of StockLocation objects
	SystemMessages - Collection of messages from the external system.

Usage Example:

```
        var provider =
(InventoryServiceProvider)Factory.CreateObject("inventoryServiceProvider", true);
        var request = new GetStockLocationsRequest("StarterKit");
        var result = provider.GetStockLocations(request);
        if (result.Success)
        {
            foreach (var stockLocation in result.StockLocations)
            {
                // handle stock location
            }
        }
```

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

GetProductStockLocations

Name:	**GetProductStockLocations**
Description:	Retrieves all of the stock locations from the ECS for a particular product
Usage:	Called from Sitecore when a list of stock locations for a product in the ECS is required
Signature:	`GetProductStockLocationsResult GetProductStockLocations( GetProductStockLocationsRequest request)`
Input:	
	ShopName – string. The name of the shop
	CustomerId – string. Optional The id of the customer who needs the list of stock locations
	ProductId – string. The id of the product to look up.
Output:	
	List<StockInformation> StockInformation – A list of StockInformation objects for each location
	SystemMessages - Collection of messages from the external system.

Usage Example:

```
        var provider =
(InventoryServiceProvider)Factory.CreateObject("inventoryServiceProvider", true);
        var request = new GetProductStockLocationsRequest("StarterKit", "6");
        var result = provider.GetProductStockLocations(request);
        if (result.Success)
        {
            foreach (var stockInfo in result.StockInformation)
            {
                // handle location stock information.
            }
        }
```

2.6.3 Inventory Pipelines

GetStockInformation

Name:	**GetStockInformation**
Description:	This pipeline is responsible for retrieving stock information for one or more products specified
Usage:	Called from Sitecore.
Args:	
	Request - Contains the list of InventoryProduct, details level, shop name, location and customer ID. Is set prior to calling the pipeline.
	Response - Contains the user object. Is read after the pipeline is called.
Processors:	
	GetStockInformation – Responsibility: To retrieve stock information for the list of specified product IDs

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Usage: Calls the ECS to get the stock information

Ownership: The processor is provided by the ECS

Customization: Must be created as part of the connector integrating with the ECS

StockStatusForIndexing

Developer story:

- As a developer I have a StockStatusForIndexing pipeline that runs when the crawler is indexing products and that returns the information to be indexed.

Acceptance criteria

- Check pipeline StockStatusForIndexing has no service layer method associated
- Check pipeline is separate from the pipeline associated with GetProductStockInformation
- Check the output retrieves in-stock or our-of-stock information associated with each location
- Check the output includes which location the product is sold

Notes

- User story is 405718

- As a developer I get stock status information included in the product index so when I query the index I can include criteria regarding stock status in connection with location

Acceptance criteria

- Check that we have a InStock index field that lists all the locations where the product is in stock
- Check that we have a OutOfStock index field that lists all the locations where the product is out of stock
- Check that we have a Location index field that contains the locations where the product is orderable from
- Check that we have a Pre-Orderable index field (Boolean) that indicates whether the product is pre-orderable or not

Notes
The index contains only stock status information per product and not per variant

The table below shows an example of the product index content for a T-shirt product that comes in different variants and with the In-Stock and Out-of-Stock columns

Product ID (not variant)	Size	Color	In-Stock	Out-Of-Stock	Location	Pre-orderable

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Aw123-04	S, M, L, XL	R, B, G, O	Central Store, Store1, Store2	Store3	Central Store 1, Store 2, Store 3	Yes

Name:	**StockStatusForIndexing**
Description:	Called during indexing to populate the index with stock information
Usage:	Called from Sitecore.
Args:	
	Request – A list of Product IDs is provided from the Sitecore indexing
	Response – A list of IndexStockInformation objects is returned for processing and inclusion into the index
Processors:	
	StockStatusForIndexing – Responsibility: To call the ECS and retrieve stock information used to populate the product index Usage: Called when crawling product repository and indexing the products. Ownership: Custom processor provided with the connector to the ECS Customization:

GetPreOrderableInformation

Name:	**GetPreOrderableInformation**
Description:	Contacts the ECS to get the pre-orderable information
Usage:	Called from Sitecore
Args:	
	Request – ShopName, list of InventoryProduct, Visitor ID, Location
	Response – A list of OrderableInformation
Processors:	
	GetPreOrderableInformation – Responsibility: Contacts the ECS to get the pre-orderable information Usage: Called from Sitecore to retrieve information to be used for rendering to the visitor as well as restricting placing orders Ownership: Provided with the ECS connector

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

	Customization: Must be customized

GetBackOrderableInformation

Name:	**GetBackOrderableInformation**
Description:	Contacts the ECS to get the back-orderable information
Usage:	Called from Sitecore
Args:	
	Request – ShopName, list of InventoryProduct, Visitor ID, Location
	Response – A list of OrderableInformation
Processors:	
	GetBackOrderableInformation – Responsibility: Contacts the ECS to get the back-orderable information Usage: Called from Sitecore to retrieve information to be used for rendering to the visitor as well as restricting placing orders Ownership: Provided with the ECS connector Customization: Must be customized

ProductsAreBackInStock

Name:	**ProductsAreBackInStock**
Description:	Triggers the page event Products Back In Stock, so that the visitors which have signed up, can be notified
Usage:	Called from Sitecore.
Args:	
	Request – Shop name and a list of InventoryProduct for which the product is back in stock
	Response – None
Processors:	
	TriggerPageEvent–

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

	Responsibility: Trigger page event Products Back In Stock along with the shop name and a list of product IDs. Firing the event will trigger the EA plan to re-evaluate the visitors and determine whether they should be notified Usage: Called from the ECS to signal when products have come back in stock Ownership: Provided with Connect Customization: No immediate need

GetBackInStockInformation

Name:	**GetBackInStockInformation**
Description:	The method and pipeline is used to get the updated stock information from the ECS The method is normally called because of method ProductsAreBackInStock has been called remotely and in that case the pipeline is implicitly called The difference between GetStockInformation and GetBackInStockInformation is that the first queries for stock information in a specified location and for a given customer, where the latter gets the stock information for all locations ignoring customer context
Usage:	Called from Sitecore.
Args:	
	Request – Shop name and optionally a list of InventoryProduct.
	Response – A list of StockInformationUpdate objects
Processors:	
	GetStockInformationUpdates– Responsibility: To retrieve a list of StockInformation Updates objects from the ECS, each describing the product and a list of locations for which the product is back in stock, when (availability date) and the count, where the latter two are optional values The parameters can optionally include a list of product IDs specifying the products for which a stock update is requested. If the list is empty then it is up to the ECS to keep track of which products that new stock updates. It’s needed in case the ECS is not able to notify Connect of stock update changes. In That case Connect should be able to query for any updates in order for the EA plan to work Usage: Called from the Sitecore to get the information needed to follow-up in the EA plan

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

	Ownership: Provided with Connect Customization: No immediate need

VisitorSignUpForStockNotification

Name:	**VisitorSignUpForStockNotification**
Description:	Called from Sitecore when a visitor wants to be notified when a product gets back in stock
Usage:	Called from Sitecore.
Args:	
	Request – ShopName, Visitor ID, InventoryProduct **and interest Date**
	Response – None
Processors:	
	VisitorSignUpForStockNotification – Responsibility: - Check visitors, who are **not** already in the EA plan Back In Stock Notification, are added - Check Product ID and interest date are stored in the EA state - Check visitors who **are** already in the plan stays in the same state, but has an additional product ID and interest date added to the list Usage: Ownership: Provided with Connect Customization: No immediate need
	TriggerPageEvent Responsibility: - Check that page event Back In Stock Subscription is triggered which includes the product ID, e-mail address and interest date Usage: Ownership: Provided with Connect Customization: No immediate need

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

RemoveVisitorFromStockNotification

Name:	**RemoveVisitorFromStockNotification**
Description:	Typically called from Sitecore, when the visitor has decided to unsubscribe from the stock notification and hence the EA plan Removes the visitor from the EA plan and triggers a page event
Usage:	Executed from Sitecore, when the method with the same name is called
Args:	
	Request – ShopName, VisitorID and ProductID
	Response – None
Processors:	
	RemoveVisitorFromStockNotification – Responsibility: - Check visitor ID and product ID are provided - Check that the product ID and interest date is removed from the visitors list and saved to EA state - Check that if the visitors list of product IDs is empty, then the visitor is removed from the plan all together Usage: Ownership: Provided with Connect Customization: No immediate need for customization
	TriggerPageEvent – Responsibility: Check that page event Back In Stock Unsubscription is triggered which includes the product ID, e-mail address and interest date. Usage: Ownership: Provided with Connect Customization: No immediate need for customization

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

OrderedProductStockStatus

Name:	**OrderedProductStockStatus**
Description:	Pipeline is called as a part of the SubmitOrder pipeline
Usage:	Called implicitly from pipeline AddLinesToCart from the Cart service layer to trigger a page event whenever a product which is our of stock is added to the cart
Args:	
	Request – ShopName and Cart
	Response - None, except for external system messages
Processors:	
	TriggerPageEvent– Responsibility: For each order line, **t**rigger **a** page event ProductsOutOfStockOrdered along with the ShopName, Order ID, Product ID, Stock Status, if and only if, the stock status is NOT InStock Usage: Mandatory Ownership: Provided with Connect Customization: Not needed

GetStockLocations

Name:	**GetStockLocations**
Description:	
Usage:	Called from Sitecore.
Args:	
	Request – ShopName, and CustomerId
	Response – List of StockLocations
Processors:	
	GetStockLocations– Responsibility: A placeholder pipeline that should be replaced with a processor that performs the same request against an ECS. Usage: Mandatory Ownership: Provided with Connect

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Customization: Not needed

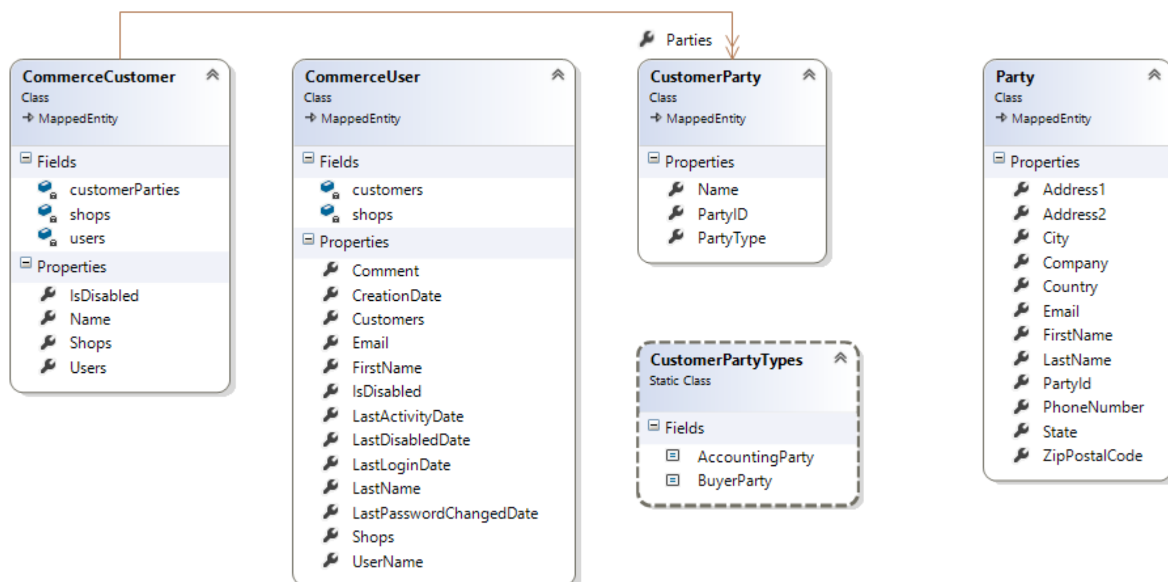
GetProductStockLocations

Name:	GetProductStockLocations
Description:	
Usage:	Called from Sitecore.
Args:	
	Request – ShopName, CustomerId, and ProductId
	Response – List of StockInformation
Processors:	
	GetProductStockLocations– Responsibility: A placeholder pipeline that should be replaced with a processor that performs the same request against an ECS. Usage: Mandatory Ownership: Provided with Connect Customization: Not needed

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

2.7 Customer

2.7.1 The Customer Domain Model



Class: CommerceUser

The CommerceUser class is responsible for representing a user account. A user resembles a visitor of a webshop (website) who has identified him- or herself explicitly by creating a login account by which the person can be (re-)authorized.

The CommerceUser entity can be extended to include custom information particular to the external commerce system, but the default implementation will work if users are stored in Sitecore only for authentication purposes.

The following assertions relates to a user:

- A user represents a person who can log in to a website. For example, meaning that an account exists for that user in the system.
- A visitor going through the checkout without registering a user account. For example, anonymous checkout, will be created as a customer, but without a user account
- Customer and User relationship:
    - A user can represent multiple different customers acting as an agent on the customers behalf
    - A customer, such as a company, can have multiple users representing the company. For example, employees of different departments
    - Based on the two previous statements it means that there is a many to many relationship between the two.
- Multiple users can manage the same shopping cart on behalf of the same customer or have individual shopping carts

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Usage scenarios:

- When a new account is explicitly created on the site
- When an existing user account is used during checkout
- When the returning user authenticates him- or herself to get the customer specific product prices and discounts
- When the user and/or customer profile is edited by the user
- When the shop owner sends out welcome mail to new users
- When the shop owner wants to follow-up on new users that hasn't returned to the shop for some time (B2C)

Name	Type	Description
ExternalID	String	Unique identifier for the user in the commerce system. This can be used to get a reference to the user using the commerce system's native API. Will be empty until account has been created in external system
ID	String	Unique ID
Email	String	The users e-mail
FirstName	String	
LastName	String	
Shops	List<String>	The list of shops the user has access to.
Disabled	Boolean	Indicates if the user account is disabled or not
Comment	String	Free text comment
CreationDate	DateTime	Gets the date and time when the account was added.
LastActivityDate	DateTime	Gets or sets the date and time when the user was last authenticated or accessed the shop.
LastLoginDate	DateTime	Gets or sets the date and time when the user was last authenticated.
LastDisabledDate	DateTime	Gets the most recent date and time when the user account was disabled.
LastPasswordChangedDate	DateTime	Gets the date and time when the user's password was last updated.
Customers	List<string>	The list of customer IDs of the customers that the user is associated with

Class: CommerceCustomer

The concept of a customer is determined by the integrated commerce system and the e-shop solution. In B2C solutions, the customer typically represents a person whereas in B2B scenarios a customer typically represents a company.

The CommerceCustomer entity will always be extended to include custom information particular to the external commerce system.

Definition

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

- A customer represents a person in a B2C shops and a company in a B2B shops.
- A customer account **cannot** be used to log in to the webshop. In order to log in, a user account is needed. A Customer account is different from a user account and the two can carry different information.
- Not all commerce systems support the concept of both a user and a customer. Example: SES only has users whereas InSite Commerce has both customers and users.
    - When the external system does not support customers, the information might be stored together in the User profile, but the Relation will typically not be available.
- In B2C scenarios a customer and a user is typically the same
- In B2B scenarios a customer typically has 1-many users associated:
    - A customer is typically the one paying the invoices for the orders placed in the system by its users
    - Users are the ones managing shopping carts on behalf of the customer and go through the checkout process, placing the order.
    - A customer can have multiple shopping carts associated and they can be managed by one or many different users

Usage scenarios:

- When a new customer account is created implicitly during anonymous checkout
- When the customer profile is edited by the shop owner
- When the customer places an order and gets an order confirmation mail
- When the shop owner sends out welcome mail to new customers in B2B solutions
- When the shop owner wants to make a campaign or promote special offers in B2B solutions
- When the shop owner wants to follow-up on new customers that hasn't placed orders in the shop for some time

Name	Type	Description
ExternalID	String	Unique identifier for the user in the commerce system. This can be used to get a reference to the user using the commerce system's native API. If ASP.NET is used, then the ExternalUserId will equal UserName Will be empty until account has been created in external system
Name	String	The name of the customer
ID	String	Unique ID
IsDisabled	Boolean	Indicates if the customer account is disabled or not
Shops	List<String>	The list of shops the customer has access to
CustomerParties	List<CustomerParty>	The list of parties consisting of contact and address information
CommerceUsers	List<string>	The list of user IDs of the users that the customer is associated with

Class: CustomerParty

CustomerParty is used to represent the type and 0-to-many relationship between the customer and a list of parties

Name	Type	Description
ExternalID	String	Unique identifier for the party
PartyId	String	ID of the party. Reserved for future use in Sitecore

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Name	String	An optional string for that describes the relationship
Type	CustomerPartyTypes	Represent the type of relationship

Class: CustomerPartyTypes

Class CustomerPartyTypes is used to indicate the type of relationship between the customer and party.

The class is introduced as an extensible enum. In order to extend and customize the CustomerPartyTypes options:

```
public class MyECSCustomerPartyTypes : CustomerPartyTypes
{
      public const int MyCustomPartyType = 3;
      public MyECSCustomerPartyTypes(int value)   : base(value)
      {   }
}
```

Name	Type	Description
BuyerParty	public const int BuyerParty = 1	BuyerParty represents the buyer and are typically used as the party where products are shipped to
AccountingParty	public const int AccountingParty = 2	AccountingParty represents the buyer and are typically used as the party where the invoice is sent to

Class: Party

The CustomerParty entity represents address contact information and a list of CustomerParty objects is included in the Customer class

Note
The CustomerParty object is used in both the customer and order service layers.

Name	Type	Description
ExternalID	String	Unique identifier for the party
PartyID	String	ID of the party
FirstName	String	First name
LastName	String	Last name
Email	String	E-mail address
Company	String	Company name
Address1	String	Streetname
Address2	String	Region, District, County etc
ZipPostalCode	String	Zip/Postal code
City	String	City name
State	String	State
Country	String	Country name
PhoneNumber	String	Phone number

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

2.7.2 Customer Service Methods

Service providers are wrapper objects designed to make it easier to interact with Connect pipelines. The providers implement no logic other than calling pipelines. All of the business logic is implemented in the pipeline processors.

For each method there is a corresponding Request and Result object getting returned, ex. CreateUser takes a CreateUserRequest object and returns a CreateUserResult object.

The Customer Service Provider contains the following methods for interacting with customer and user data.

CreateUser

Name:	**CreateUser**
Description:	Creates a user account by which the user can re-authenticate him- or herself upon return. By default the account is disabled until it has be confirmed by visitor to be a valid request and ActivateUserAccount has been called Calls the pipeline "`CreateUser`".
Usage:	Called from Sitecore when a visitor is registering for an account. It could be during the checkout process or through plain signup.
Signature:	`CreateUserResult CreateUser(CreateUserRequest request)`
Input:	
	Username – string. Mandatory The user name for the new user.
	Email – string. Mandatory The e-mail address for the new user.
	Password – string. Mandatory The password for the new user.
	Shops– Mandatory An instance of the CommerceUser object is parsed in. Mandatory fields: Username, Email, Shops.
Output:	
	User – An instance of the user object is returned. The user object is updated by the external commerce system by supplying the External ID value.
	SystemMessages - Collection of messages from the external system.

Usage Example:

```
        var customerService = new CustomerServiceProvider();

        var request = new CreateUserRequest("JohnSmith", "password", "john@abczyx.net",
"webstore");

        var result = customerService.CreateUser(request);
```

UpdateUser

Name:	**UpdateUser**
Description:	Updates an existing user account. Calls the pipeline "UpdateUser"
Usage:	Called from Sitecore when visitor wants to update the information stored on the account.
Signature:	`UpdateUserResult UpdateUser(UpdateUserRequest request)`
Input:	
	CommerceUser – An instance of the modified CommerceUser object is passed in

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Output:	
	User – An instance of the user object is returned.
	SystemMessages - Collection of messages from the external system.

Usage Example:

```
        var customerService = new CustomerServiceProvider();

        // create a user
        var request = new CreateUserRequest("JohnSmith", "password", "john@abczyx.net", 
"webstore");

        var user = customerService.CreateUser(request).CommerceUser;
        user.FirstName = "John";

        // update the user
        var updateRequest = new UpdateUserRequest(user);
        var result = customerService.UpdateUser(updateRequest);
```

DeleteUser

Name:	**DeleteUser**
Description:	Deletes a user account. Calls the pipeline "`DeleteUser`".
Usage:	Called from Sitecore when the shop owner wants to delete an account. It's a solution business decision whether the account is actually deleted or simply disabled
Signature:	`DeleteUserResult DeleteUser(DeleteUserRequest request)`
Input:	
	CommerceUser – An instance of the CommerceUser object is parsed in
Output:	
	SystemMessages - Collection of messages from the external system.

Usage Example:

```
        var customerService = new CustomerServiceProvider();

        // create a user
        var request = new CreateUserRequest("JohnSmith", "password", "john@abczyx.net", 
"webstore");
        var user = customerService.CreateUser(request).CommerceUser;

        // delete the user
        var deleteRequest = new DeleteUserRequest(user);
        var result = customerService.DeleteUser(deleteRequest);
```

DisableUser

Name:	**DisableUser**
Description:	Disables a user account. Calls the pipeline "DisableU`ser`"
Usage:	Called from Sitecore when the user account should be disabled
Signature:	`DisableUserResult DisableUser(DisableUserRequest request)`
Input:	
	CommerceUser – Mandatory An instance of the user object is parsed in
	Comment – Optional

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

	An optional string that can explain why the user account was disabled. Will be put in the Page Event as explanation
Output:	
	CommerceUser – The disabled CommerceUser entity
	SystemMessages - Collection of messages from the external system.

Usage Example:

```
        var customerService = new CustomerServiceProvider();

        // create a user
        var request = new CreateUserRequest("JohnSmith", "password", "john@abczyx.net",
"webstore");
        var user = customerService.CreateUser(request).CommerceUser;

        // disable the user
        var disableRequest = new DisableUserRequest(user);
        var disableResult = customerService.DisableUser(disableRequest);

        // enable the user
        var enableRequest = new EnableUserRequest(user);
        var enableResult = customerService.EnableUser(enableRequest);
```

EnableUser

Name:	**EnableUser**
Description:	Enables a user account. Calls the pipeline "EnableUser" A user account can be disabled for different reasons and triggered by shop owner, visitor or by system (EA plan). When a user account is disabled it must be possible to enable it again, which is the purpose of this method. The method seems similar to ActivateUserAccount in the way that both enable the account. The difference lies in the usage scenario and possible actions associated. EnableUser is a generic method whereas UserAccountActivation is used in a specific scenario when a new account is created and must be validated and then activated. The two trigger different page events, where the event User Account Activated triggers the EA plan User Account Registration to proceed.
Usage:	Called from Sitecore when the user account should be enabled
Signature:	`EnableUserResult EnableUser(EnableUserRequest request)`
Input:	
	CommerceUser – Mandatory An instance of the user object is parsed in
	Comment – Optional An optional string that can explain why the user account was enabled. Will be put in the Page Event as explanation
Output:	
	CommerceUser – The enabled CommerceUser entity
	SystemMessages - Collection of messages from the external system.

Usage Example:

```
        var customerService = new CustomerServiceProvider();

        // create a user
        var request = new CreateUserRequest("JohnSmith", "password", "john@abczyx.net",
"webstore");
        var user = customerService.CreateUser(request).CommerceUser;
```

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

```
        // disable the user
        var disableRequest = new DisableUserRequest(user);
        var disableResult = customerService.DisableUser(disableRequest);

        // enable the user
        var enableRequest = new EnableUserRequest(user);
        var enableResult = customerService.EnableUser(enableRequest);
```

GetUser

Name:	**GetUser**
Description:	Returns a single user account. Calls the pipeline "GetUser".
Usage:	Called from Sitecore when searching for one or more accounts
Signature:	`GetUserResult GetUser(GetUserRequest request)`
Input:	
	ShopName – Mandatory
	UserName – Mandatory The ID of the user to retrieve
Output:	
	User – A single instance of a User
	SystemMessages - Collection of messages from the external system.

Usage Example:

```
        var customerService = new CustomerServiceProvider();

        // create a user
        var request = new CreateUserRequest("JohnSmith", "password", "john@abczyx.net",
"webstore");
        var user = customerService.CreateUser(request).CommerceUser;

        var getRequest = new GetUserRequest("JohnSmith");
        var result = customerService.GetUser(getRequest);
```

GetUsers

Name:	**GetUsers**
Description:	Queries and returns user accounts. Calls the pipeline "GetUsers". Different input parameters can be provided and they will be combined using logical *and*
Usage:	Called from Sitecore when searching for one or more accounts
Signature:	`GetUsersResult GetUsers(GetUsersRequest request)`
Input:	
	ExternalIDs – Optional. Can be a single or a list of IDs. When provided, it takes precedence
	SitecoreIDs – Optional Can be a single or a list of IDs
	UserName – optional
	Email – optional
	ExternalCustomerIDs – optional Can be a single or a list of IDs. Used when looking for users associated with a given customer
	Disabled - optional
	ShopName – optional
Output:	
	List<User>

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

	SystemMessages - Collection of messages from the external system.

Usage Example:

```
        var customerService = new CustomerServiceProvider();

        // create some users
        var request = new CreateUserRequest("JohnSmith", "password", "john@abczyx.net", 
"webstore");
        var result = customerService.CreateUser(request);
        request = new CreateUserRequest("JaneSmith", "passWord", "jane@jane.net", "webstore");
        result = customerService.CreateUser(request);
        request = new CreateUserRequest("Rob", "abcdefghij", "r@r.com", "webstore");
        result = customerService.CreateUser(request);

        var getRequest = new GetUsersRequest(new UserSearchCriteria { ShopName = "webstore" 
});
        var getResult = customerService.GetUsers(getRequest);
        Sitecore.Diagnostics.Assert.AreEqual(3, getResult.CommerceUsers.Count, "should have a 
count of 3");

        getRequest = new GetUsersRequest(new UserSearchCriteria { UserName = "JohnSmith" });
        getResult = customerService.GetUsers(getRequest);
        Sitecore.Diagnostics.Assert.AreEqual(1, getResult.CommerceUsers.Count, "should have a 
count of 1");
```

CreateCustomer

Name:	**CreateCustomer**
Description:	Creates a customer. Calls the pipeline "`CreateCustomer`"
Usage:	Typically called when a visitor is going through the checkout process
Signature:	`CreateCustomerResult CreateCustomer(CreateCustomerRequest request)`
Input:	
	CommerceCustomer – Mandatory An instance of the CommerceCustomer object is parsed in Mandatory field values are: Name and Shops
Output:	
	CommerceCustomer – An instance of the customer object is returned. The customer object is updated by the external commerce system by supplying the External ID.
	SystemMessages - Collection of messages from the external system.

Usage Example:

```
        var customerService = new CustomerServiceProvider();

        customerService.CreateCustomer(new CreateCustomerRequest(
            new CommerceCustomer
            {
                Name = "Jeff",
                Shops = new[] { "autohaus" },
                IsDisabled = true,
                ExternalId = "Jeff",
                Users = new[] { "Jeff" }
            }));

        customerService.CreateCustomer(new CreateCustomerRequest(
            new CommerceCustomer
            {
                Name = "Bob",
                Shops = new[] { "webstore" },
                IsDisabled = false,
                ExternalId = "Jeff",
                Users = new[] { "Bob" }
            }));
```

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

UpdateCustomer

Name:	**UpdateCustomer**
Description:	Updates an existing customer account. Calls the pipeline "UpdateCustomer"
Usage:	Called from Sitecore when visitor or shop owner wants to update the information stored on the customer account.
Signature:	`UpdateCustomerResult UpdateCustomer(UpdateCustomerRequest request)`
Input:	
	CommerceCustomer – An instance of the modified CommerceCustomer object is parsed in
Output:	
	CommerceCustomer – An instance of the customer object is returned.
	SystemMessages - Collection of messages from the external system.

Usage Example:

```
var customerService = new CustomerServiceProvider();

customerService.CreateCustomer(new CreateCustomerRequest(
    new CommerceCustomer
    {
        Name = "Jeff",
        Shops = new[] { "autohaus" },
        IsDisabled = true,
        ExternalId = "Jeff",
        Users = new[] { "Jeff" }
    }));

var customer = customerService.CreateCustomer(new CreateCustomerRequest(
    new CommerceCustomer
    {
        Name = "Bob",
        Shops = new[] { "webstore" },
        IsDisabled = false,
        ExternalId = "Jeff",
        Users = new[] { "Bob" }
    })).CommerceCustomer;

customer.IsDisabled = true;
customer.Name = "Bobby";

var request = new UpdateCustomerRequest(customer);
var result = customerService.UpdateCustomer(request);
```

DisableCustomer

Name:	**DisableCustomer**
Description:	Disables a customer account. Calls the pipeline "DisableCustomer"
Usage:	Called from Sitecore when the customer account should be disabled
Signature:	`DisableCustomerResult DisableCustomer(DisableCustomerRequest request)`
Input:	
	CommerceCustomer – Mandatory An instance of the customer object is parsed in
	Comment – Optional An optional string that can explain why the user account was disabled. Will be put in the Page Event as explanation

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Output:	
	CommerceCustomer – The disabled customer
	SystemMessages - Collection of messages from the external system.

Usage Example:

```
var customerService = new CustomerServiceProvider();

var customer = customerService.CreateCustomer(new CreateCustomerRequest(
    new CommerceCustomer
    {
        Name = "Bob",
        Shops = new[] { "webstore" },
        IsDisabled = false,
        ExternalId = "Jeff",
        Users = new[] { "Bob" }
    })).CommerceCustomer;

var disableRequest = new DisableCustomerRequest(customer);
var disableResult = customerService.DisableCustomer(disableRequest);

var enableRequest = new EnableCustomerRequest(customer);
var enableResult = customerService.EnableCustomer(enableRequest);
```

EnableCustomer

Name:	**EnableCustomer**
Description:	Enables a customer account. Calls the pipeline "EnableCustomer"
Usage:	Called from Sitecore when the customer account should be enabled
Signature:	`EnableCustomerResult EnableCustomer(EnableCustomerRequest request)`
Input:	
	CommerceCustomer – Mandatory An instance of the customer object is parsed in
	Comment – Optional An optional string that can explain why the user account was enabled. Will be put in the Page Event as explanation
Output:	
	SystemMessages - Collection of messages from the external system.

Usage Example:

```
var customerService = new CustomerServiceProvider();

var customer = customerService.CreateCustomer(new CreateCustomerRequest(
    new CommerceCustomer
    {
        Name = "Bob",
        Shops = new[] { "webstore" },
        IsDisabled = false,
        ExternalId = "Jeff",
        Users = new[] { "Bob" }
    })).CommerceCustomer;

var disableRequest = new DisableCustomerRequest(customer);
var disableResult = customerService.DisableCustomer(disableRequest);

var enableRequest = new EnableCustomerRequest(customer);
var enableResult = customerService.EnableCustomer(enableRequest);
```

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

DeleteCustomer

Name:	DeleteCustomer
Description:	Deletes a customer account. Calls the pipeline "DeleteUser".
Usage:	Called when an account should be deleted It's a solution business decision whether the account is actually deleted or simply disabled
Signature:	DeleteCustomerResult DeleteCustomer(DeleteCustomerRequest request)
Input:	
	CommerceCustomer – An instance of the CommerceCustomer object is parsed in
Output:	
	SystemMessages - Collection of messages from the external system.

Usage Example:

```
var customerService = new CustomerServiceProvider();

var customer = customerService.CreateCustomer(new CreateCustomerRequest(
    new CommerceCustomer
    {
        Name = "Bob",
        Shops = new[] { "webstore" },
        IsDisabled = false,
        ExternalId = "Jeff",
        Users = new[] { "Bob" }
    })).CommerceCustomer;

var deleteRequest = new DeleteCustomerRequest(customer);
var deleteResult = customerService.DeleteCustomer(deleteRequest);
```

GetCustomer

Name:	GetCustomer
Description:	Returns a single customer instance. Calls the pipeline "GetCustomer".
Usage:	Called from Sitecore when searching for an account
Signature:	GetCustomerResult GetCustomer(GetCustomerRequest request)
Input:	
	ShopName – Mandatory
	ExternalID – Mandatory The unique ID of the customer in the given shop.
Output:	
	CommerceCustomer – An instance of the CommerceCustomer if it exists
	SystemMessages - Collection of messages from the external system.

Usage Example:

```
var customerService = new CustomerServiceProvider();

customerService.CreateCustomer(new CreateCustomerRequest(
    new CommerceCustomer
    {
        Name = "Bob",
        Shops = new[] { "webstore" },
        IsDisabled = false,
        ExternalId = "1234567890",
        Users = new[] { "Bob" }
    }));
```

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

```
        var getRequest = new GetCustomerRequest("1234567890");
        var result = customerService.GetCustomer(getRequest);
```

GetCustomers

Name:	**GetCustomers**
Description:	Queries and returns customer entities. Calls the pipeline "GetCustomers". Different input parameters can be provided and they will be combined using logical *and*
Usage:	Called from Sitecore when searching for an account
Signature:	`GetCustomersResult GetCustomers(GetCustomersRequest request)`
Input:	
	ExternalIDs – optional Can be a single or a list of IDs. When provided, it takes precedence
	SitecoreIDs – optional Can be a single or a list of IDs.
	Name – optional
	ExternalUserIDs – Optional. Can be a single or a list of IDs. Used when looking for customers associated with a given user
	Disabled – optional
	ShopName – optional
Output:	
	List<CommerceCustomer>
	SystemMessages - Collection of messages from the external system.

Usage Example:

```
        var customerService = new CustomerServiceProvider();

        customerService.CreateCustomer(new CreateCustomerRequest(
            new CommerceCustomer
            {
                Name = "Jeff",
                Shops = new[] { "autohaus" },
                IsDisabled = true,
                ExternalId = "Jeff",
                Users = new[] { "Jeff" }
            }));
        customerService.CreateCustomer(new CreateCustomerRequest(
            new CommerceCustomer
            {
                Name = "Bob",
                Shops = new[] { "webstore" },
                IsDisabled = false,
                ExternalId = "Jeff",
                Users = new[] { "Bob" }
            }));
        customerService.CreateCustomer(new CreateCustomerRequest(
            new CommerceCustomer
            {
                Name = "Michael",
                Shops = new[] { "autohaus" },
                IsDisabled = false,
                ExternalId = "Michael",
                Users = new[] { "Michael" }
            }));
        customerService.CreateCustomer(new CreateCustomerRequest(
            new CommerceCustomer
            {
                Name = "Kerry",
                Shops = new[] { "webstore" },
                IsDisabled = true,
                ExternalId = "Michael",
```

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

```
                Users = new[] { "Michael" }
            }));

        var result = customerService.GetCustomers(new GetCustomersRequest(new
CustomerSearchCriteria { IsDisabled = true }));
        Sitecore.Diagnostics.Assert.AreEqual(2, result.CommerceCustomers.Count, "Should have 2
disabled");

        result = customerService.GetCustomers(new GetCustomersRequest(new
CustomerSearchCriteria { ExternalIDs = new[] { "Jeff" }, Name = "Bob" }));
        Sitecore.Diagnostics.Assert.AreEqual(2, result.CommerceCustomers.Count, "Should have 1
match");
```

AddCustomers

Name:	**AddCustomers**
Description:	Add a customer to a user. Calls the pipeline "`AddCustomers`"
Usage:	Called to associated a visitor to a user
Signature:	`AddCustomersResult AddCustomers(AddCustomersRequest request)`
Input:	
	List<string> CustomerIds – the customer ids to add to the user
Output:	
	IReadOnlyCollection<string> CustomerIds – the list of customer ids associated with the user
	SystemMessages - Collection of messages from the external system.

Usage Example:

```
        var customerService = new CustomerServiceProvider();

        // create a customer
        customerService.CreateCustomer(new CreateCustomerRequest(
            new CommerceCustomer
            {
                Name = "Jeff",
                Shops = new[] { "autohaus" },
                ExternalId = "1234567890"
            }));

        // create a user
        var request = new CreateUserRequest("JohnSmith", "password", "john@abczyx.net",
"webstore");
        var user = customerService.CreateUser(request).CommerceUser;

        // add the customer to the user
        var addRequest = new AddCustomersRequest(user, new List<string> { "1234567890" });
        var addResult = customerService.AddCustomers(addRequest);
```

AddUsers

Name:	**AddUsers**
Description:	Add a user to a customer. Calls the pipeline "`AddUsers`"
Usage:	Called to associated a user to a visitor
Signature:	`AddUsersResult AddUsers(AddUsersRequest request)`
Input:	
	List<string> UserIds – the user ids to add to the customer
Output:	
	IReadOnlyCollection<string> UserIds – the list of user ids associated with the customer

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

	SystemMessages - Collection of messages from the external system.

Usage Example:

```
        var customerService = new CustomerServiceProvider();

        // create a user
        var request = new CreateUserRequest("JohnSmith", "password", "john@abczyx.net",
"webstore");
        var user = customerService.CreateUser(request).CommerceUser;

        // create a customer
        var customer = customerService.CreateCustomer(new CreateCustomerRequest(
            new CommerceCustomer
            {
                Name = "Jeff",
                Shops = new[] { "autohaus" },
                ExternalId = "1234567890"
            })).CommerceCustomer;

        // add the user to the customer
        var addRequest = new AddUsersRequest(customer, new List<string> { user.ExternalId });
        var addResult = customerService.AddUsers(addRequest);
```

RemoveCustomers

Name:	**RemoveCustomers**
Description:	Removes customers from a user. Calls the pipeline "`RemoveCustomers`"
Usage:	Called to remove an associated visitor to a user
Signature:	`RemoveCustomersResult RemoveCustomers(RemoveCustomersRequest request)`
Input:	
	List<string> CustomerIds – the customer ids to remove from the user
Output:	
	IReadOnlyCollection<string> CustomerIds – the list of customer ids associated with the user
	SystemMessages - Collection of messages from the external system.

Usage Example:

```
        var customerService = new CustomerServiceProvider();

        // create 2 customers
        customerService.CreateCustomer(new CreateCustomerRequest(
            new CommerceCustomer
            {
                Name = "Jeff",
                Shops = new[] { "webstore" },
                ExternalId = "1234567890"
            }));
        customerService.CreateCustomer(new CreateCustomerRequest(
            new CommerceCustomer
            {
                Name = "Bob",
                Shops = new[] { "webstore" },
                ExternalId = "9876543210"
            }));

        // create a user
        var request = new CreateUserRequest("JohnSmith", "password", "john@abczyx.net",
"webstore");
        var user = customerService.CreateUser(request).CommerceUser;

        // add the customers to the user
        var addRequest = new AddCustomersRequest(user, new List<string> { "1234567890",
"9876543210" });
        var addResult = customerService.AddCustomers(addRequest);
```

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

```
        // remove a customer
        var removeRequest = new RemoveCustomersRequest(user, new List<string> { "1234567890"
});
        var removeResult = customerService.RemoveCustomers(removeRequest);
```

RemoveUsers

Name:	RemoveUsers
Description:	Removes users from a customer. Calls the pipeline "`RemoveUsers`"
Usage:	Called to remove an associated user to a visitor
Signature:	`RemoveUsersResult RemoveUsers(RemoveUsersRequest request)`
Input:	
	List<string> UserIds – the user ids to remove from the customer
Output:	
	IReadOnlyCollection<string> UserIds – the list of user ids associated with the customer
	SystemMessages - Collection of messages from the external system.

Usage Example:

```
        var customerService = new CustomerServiceProvider();

        // create 2 users
        var request = new CreateUserRequest("JohnSmith", "password", "john@abczyx.net",
"webstore");
        var user = customerService.CreateUser(request).CommerceUser;
        request = new CreateUserRequest("JaneSmith", "passWord", "jane@jane.net", "webstore");
        var user2 = customerService.CreateUser(request).CommerceUser;

        // create a customer
        var customer = customerService.CreateCustomer(new CreateCustomerRequest(
            new CommerceCustomer
            {
                Name = "Jeff",
                Shops = new[] { "autohaus" },
                ExternalId = "1234567890"
            })).CommerceCustomer;

        // add the users to the customer
        var addRequest = new AddUsersRequest(customer, new List<string> { user.ExternalId,
user2.ExternalId });
        var addResult = customerService.AddUsers(addRequest);

        // remove a user
        var removeRequest = new RemoveUsersRequest(customer, new List<string> {
user.ExternalId });
        var removeResult = customerService.RemoveUsers(removeRequest);
```

AddCustomerParties

Name:	AddCustomerParties
Description:	This method is responsible for adding one or more given customer parties to the specified customer
Usage:	Called from Sitecore when adding parties to a customer account, typically during checkout or editing the customer account
Signature:	`AddCustomerPartiesResult AddCustomerParties (AddCustomerPartiesRequest request)`
Input:	
	Customer – CommerceCustomer. Mandatory

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

	An instance of the customer
	CustomerParties – List<CustomerParty>**.** Mandatory A list of customer parties to associate with the customer
Output:	
	Customer - Customer. Mandatory
	SystemMessages - Collection of messages from the external system.

Usage Example:

```
        var customerService = new CustomerServiceProvider();

        // create a customer
        var customer = customerService.CreateCustomer(new CreateCustomerRequest(
            new CommerceCustomer
            {
                Name = "Jeff",
                Shops = new[] { "autohaus" },
                ExternalId = "1234567890"
            })).CommerceCustomer;

        // create the add request
        var request = new AddCustomerPartiesRequest(customer,
            new List<CustomerParty>
            {
                new CustomerParty {ExternalId = "1234567890", PartyID = "1", Name =
"HomeAddess", PartyType = 2}
            });

        var result = customerService.AddCustomerParties(request);
```

RemoveCustomerParties

Name:	**RemoveCustomerParties**
Description:	This method is responsible for removing one or more given customer parties from the specified customer
Usage:	Called from Sitecore when removing parties to a customer account, typically when editing the customer account
Signature:	`RemoveCustomerPartiesResult RemoveCustomerParties (RemoveCustomerPartiesRequest request)`
Input:	
	Customer – CommerceCustomer. Mandatory An instance of the customer
	Parties – List<Party>**.** Mandatory A list of customer parities to un-associate with the customer
Output:	
	Customer - Customer. Mandatory
	SystemMessages - Collection of messages from the external system.

Usage Example:

```
        var customerService = new CustomerServiceProvider();

        // create a customer
        var customer = customerService.CreateCustomer(new CreateCustomerRequest(
            new CommerceCustomer
            {
                Name = "Jeff",
                Shops = new[] { "autohaus" },
                ExternalId = "1234567890"
            })).CommerceCustomer;

        var parties = new List<CustomerParty>
        {
            new CustomerParty {ExternalId = "1234567890", PartyID = "1", Name = "HomeAddress",
PartyType = 2}
```

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

```
        };

        // create the add request
        var request = new AddCustomerPartiesRequest(customer, parties);
        var result = customerService.AddCustomerParties(request);

        // remove the parties
        var removeRequest = new RemoveCustomerPartiesRequest(customer, parties);
        var removeResult = customerService.RemoveCustomerParties(removeRequest);
```

UpdateCustomerParties

Name:	**UpdateCustomerParties**
Description:	This method is responsible for updating one or more given customer parties on the specified customer
Usage:	Called from Sitecore when updating parties on a customer account, typically when editing the customer account
Signature:	`UpdateCustomerPartiesResult UpdateCustomerParties (UpdateCustomerPartiesRequest request)`
Input:	
	Customer – CommerceCustomer. Mandatory An instance of the customer
	Parties – List<Party>**.** Mandatory A list of customer parities to update on the customer
Output:	
	Customer - Customer. Mandatory
	SystemMessages - Collection of messages from the external system.

Usage Example:

```
        var customerService = new CustomerServiceProvider();

        // create a customer
        var customer = customerService.CreateCustomer(new CreateCustomerRequest(
            new CommerceCustomer
            {
                Name = "Jeff",
                Shops = new[] { "autohaus" },
                ExternalId = "1234567890"
            })).CommerceCustomer;

        var parties = new List<CustomerParty>
        {
            new CustomerParty {ExternalId = "1234567890", PartyID = "1", Name = "HomeAddess",
PartyType = 2}
        };

        // create the add request
        var request = new AddCustomerPartiesRequest(customer, parties);
        var result = customerService.AddCustomerParties(request);

        parties.Add(new CustomerParty { ExternalId = "9876543210", PartyID = "2", Name =
"BillingAddess", PartyType = 1 });

        // update the parties
        var updateRequest = new UpdateCustomerPartiesRequest(customer, parties);
        var updateResult = customerService.UpdateCustomerParties(updateRequest);
```

AddParties

Name:	**AddParties**

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Description:	This method is responsible for adding one or more given customer parties to the specified customer
Usage:	Called from Sitecore when adding parties to a customer account, typically during checkout or editing the customer account
Signature:	`AddPartiesResult AddParties(AddPartiesRequest request)`
Input:	
	Customer – CommerceCustomer. Mandatory An instance of the customer
	Parties – List<Party>**.** Mandatory A list of customer parties to add
Output:	
	Customer - Customer. Mandatory
	SystemMessages - Collection of messages from the external system.

Usage Example:

```
        var customerService = new CustomerServiceProvider();

        // create a customer
        var customer = customerService.CreateCustomer(new CreateCustomerRequest(
            new CommerceCustomer
            {
                Name = "Jeff",
                Shops = new[] { "autohaus" },
                ExternalId = "1234567890"
            })).CommerceCustomer;

        var parties = new List<Party>
        {
            new Party {ExternalId = "1234567890", Address1 = "123 My Street", City = "My
City", Country = "My Country", PartyId = "1"}
        };

        // add the party
        var addRequest = new AddPartiesRequest(customer, parties);
        var addResult = customerService.AddParties(addRequest);
```

GetParties

Name:	**GetParties**
Description:	This method is responsible for getting all the parties
Usage:	Called from Sitecore when adding parties to a customer account, typically during checkout or editing the customer account
Signature:	`AddPartiesResult AddParties(AddPartiesRequest request)`
Input:	
	Customer – CommerceCustomer. Mandatory An instance of the customer
Output:	
	Parties – List<Party> A list of parties
	SystemMessages - Collection of messages from the external system.

Usage Example:

```
        var customerService = new CustomerServiceProvider();

        // create a customer
        var customer = customerService.CreateCustomer(new CreateCustomerRequest(
            new CommerceCustomer
            {
                Name = "Jeff",
                Shops = new[] { "autohaus" },
```

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

```
                ExternalId = "1234567890"
            })).CommerceCustomer;

        var parties = new List<Party>
        {
            new Party {ExternalId = "1234567890", Address1 = "123 My Street", City = "My
City", Country = "My Country", PartyId = "1"}
        };

        // add the party
        var addRequest = new AddPartiesRequest(customer, parties);
        var addResult = customerService.AddParties(addRequest);

        // get the party
        var getRequest = new GetPartiesRequest(customer);
        var getResult = customerService.GetParties(getRequest);
```

RemoveParties

Name:	RemoveParties
Description:	This method is responsible for removing one or more given customer parties from the specified customer
Usage:	Called from Sitecore when removing parties to a customer account, typically when editing the customer account
Signature:	`RemovePartiesResult RemoveParties(RemovePartiesRequest request)`
Input:	
	Customer – CommerceCustomer. Mandatory An instance of the customer
	Parties – List<Party>**.** Mandatory A list of customer parties to remove
Output:	
	Customer - Customer. Mandatory
	SystemMessages - Collection of messages from the external system.

Usage Example:

```
        var customerService = new CustomerServiceProvider();

        // create a customer
        var customer = customerService.CreateCustomer(new CreateCustomerRequest(
            new CommerceCustomer
            {
                Name = "Jeff",
                Shops = new[] { "autohaus" },
                ExternalId = "1234567890"
            })).CommerceCustomer;

        var parties = new List<Party>
        {
            new Party {ExternalId = "1234567890", Address1 = "123 My Street", City = "My
City", Country = "My Country", PartyId = "1"}
        };

        // add the party
        var addRequest = new AddPartiesRequest(customer, parties);
        var addResult = customerService.AddParties(addRequest);

        // remove the party
        var removeRequest = new RemovePartiesRequest(customer, parties);
        var getResult = customerService.RemoveParties(removeRequest);
```

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

UpdateParties

Name:	**UpdateParties**
Description:	This method is responsible for updating one or more given customer parties on the specified customer
Usage:	Called from Sitecore when updating parties on a customer account, typically when editing the customer account
Signature:	`UpdatePartiesResult UpdateParties(UpdatePartiesRequest request)`
Input:	
	Customer – CommerceCustomer. Mandatory An instance of the customer
	Parties – List<Party>**.** Mandatory A list of parties to update
Output:	
	Customer - Customer. Mandatory
	SystemMessages - Collection of messages from the external system.

Usage Example:

```
        var customerService = new CustomerServiceProvider();

        // create a customer
        var customer = customerService.CreateCustomer(new CreateCustomerRequest(
            new CommerceCustomer
            {
                Name = "Jeff",
                Shops = new[] { "autohaus" },
                ExternalId = "1234567890"
            })).CommerceCustomer;

        var parties = new List<Party>
        {
            new Party {ExternalId = "1234567890", Address1 = "123 My Street", City = "My
City", Country = "My Country", PartyId = "1"}
        };

        // add the party
        var addRequest = new AddPartiesRequest(customer, parties);
        var addResult = customerService.AddParties(addRequest);

        parties.Add(new Party { ExternalId = "9876543210", Address1 = "123 My Road", City =
"My Town", Country = "Your Country", PartyId = "2" });

        // update the parties
        var updateRequest = new UpdatePartiesRequest(customer, parties);
        var getResult = customerService.UpdateParties(updateRequest);
```

UpdatePassword

Name:	**UpdatePassword**
Description:	Change the user password. Calls the pipeline "UpdatePassword"
Usage:	Called from Sitecore when password needs to be changed.
Signature:	`UpdatePasswordResult UpdatePassword(UpdatePasswordRequest request)`
Input:	
	UserID – Mandatory The ID of the user to change the password for
	OldPassword – **Mandatory**. The old password
	NewPassword – Mandatory The new password

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Output:	
	SystemMessages - Collection of messages from the external system.

Usage Example:

```
        var customerService = new CustomerServiceProvider();

        // create a customer
        var request = new CreateUserRequest("JohnSmith", "password", "john@abczyx.net",
"webstore");
        customerService.CreateUser(request);

        // update the password
        var updateRequest = new UpdatePasswordRequest("JohnSmith", "password", "nEwPaSsWoRd");
        var result = customerService.UpdatePassword(updateRequest);
```

2.7.3 Customer Pipelines

CreateUser

Depending on the actual integration with ECS and the solution then the user can be disabled by default and will get activated when ActivateUserAccount is called

Name:	**CreateUser**
Description:	This pipeline is responsible for creating a user account
Usage:	Called from Sitecore.
Args:	
	Request - Contains the user entity and a password. Is set prior to calling the pipeline.
	Response - Contains the user object. Is read after the pipeline is called.
Processors:	
	CreateUserInECS – Optional Creates a user in the external commerce system and updates the field ExternalID on the user entity. **Note:** If External ID is parsed in and the user already exists, nothing should happen **Note:** This processor is optional but recommended if CommerceUsers are supported in the ECS.
	CreateUserInCMS – (Mandatory) Creates the user but more importantly stores the relation to CommerceCustomer in Sitecore, using the membership and profile providers. If user already exists, then the pipeline should be aborted to avoid triggering the goal.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

	Note: This processor is mandatory to store the 1-to-many relationship between the CommerceUser and CommerceCustomer. If that is handled in the ECS, the processor is not mandatory
	TriggerDMSGoal – The goal “User Account Created” is triggered with values User name and ShopName.
	AddVisitorToEAPlan – Adds visitor / contact to EA plan. For example, “User Account Registered”, which sends an account activation mail and later a welcome mail when the account has been activated

UpdateUser

Name:	**UpdateUser**
Description:	This pipeline is responsible for updating an existing user account.
Usage:	Called from Sitecore.
Args:	
	Request - Contains the user entity. Is set prior to calling the pipeline.
	Response - Contains the user entity. Is read after the pipeline is called.
Processors:	
	UpdateUserInECS – Optional Updates an existing user account in the external commerce system. **Note:** This processor is optional but recommended if CommerceUsers are supported in the ECS.
	UpdateUserInCMS – Mandatory Updates the user account in the Sitecore. Since the user is updated externally first, then the user data can be changed there and the final version is stored in CMS. **Note:** This processor is mandatory to store the 1-to-many relationship between the CommerceUser and CommerceCustomer. If that is handled in the ECS, the processor is not mandatory
	TriggerDMSEvent – the page event “User Account Updated” is triggered with values User name and ShopName.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

DeleteUser

Name:	**DeleteUser**
Description:	This pipeline is responsible for deleting an existing user account.
Usage:	Called from Sitecore.
Args:	
	Request - Contains the user entity. Is set prior to calling the pipeline.
	Response -
Processors:	
	DeleteUserInECS – Optional Deletes an existing user in the external commerce system. **Note:** If user account doesn't exists it can be noted in returned collection of External System Messages, but the pipeline must continue **Note:** This processor is optional but recommended if CommerceUsers are supported in the ECS.
	DeleteUserInCMS – Mandatory Deletes the user in the Sitecore. **Note:** If user account doesn't exist, the pipeline is aborted **Note:** This processor is mandatory to store the 1-to-many relationship between the CommerceUser and CommerceCustomer. If that is handled in the ECS, the processor is not mandatory
	TriggerDMSEvent – the page event "User Account Deleted" is triggered with values User name and ShopName.

DisableUser

Name:	**DisableUser**
Description:	This pipeline is responsible for disabling an existing user account.
Usage:	Called from Sitecore.
Args:	
	Request - Contains the user entity and a comment. Is set prior to calling the pipeline.
	Response - Is read after the pipeline is called.
Processors:	
	DisableUserInECS – Optional Disables an existing user in the external commerce system.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

	DisableUserInCMS – Mandatory Disables the user in Sitecore. **Note:** Since the user is attempted disabled externally first, then it is possible that the pipeline is aborted due to business rules and the users are still synchronized **Note:** This processor is mandatory to store the 1-to-many relationship between the CommerceUser and CommerceCustomer. If that is handled in the ECS, the processor is not mandatory
	TriggerDMSEvent – the page event "User Account Disabled" is triggered with values User name, ShopName and the provided comment.

EnableUser

Name:	**EnableUser**
Description:	This pipeline is responsible for enabling an existing user account.
Usage:	Called from Sitecore.
Args:	
	Request - Contains the user entity and a comment. Is set prior to calling the pipeline.
	Response - Is read after the pipeline is called.
Processors:	
	EnableUserInECS – Optional Enables an existing user in the external commerce system. **Note:** Either this processor or the next should be in the pipeline, normally not both
	EnableUserInCMS – Mandatory Enables the user in Sitecore. **Note:** Since the user is attempted enabled externally first, then it is possible that the pipeline is aborted due to business rules and the users are still synchronized **Note:** This processor is mandatory to store the 1-to-many relationship between the CommerceUser and CommerceCustomer. If that is handled in the ECS, the processor is not mandatory
	TriggerDMSEvent – the goal "User Account Enabled" is triggered with values User name, ShopName and the provided comment.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

GetUsers

Name:	**GetUsers**
Description:	Queries and returns user accounts. Different input parameters can be provided and they will be combined using logical *and*
Usage:	Called from Sitecore when a visitor is registering for an account. It could be during the checkout process or through plain signup.
Args:	
	Request - Contains search parameters. Is set prior to calling the pipeline.
	Response – Returns a list of user entities. Is read after the pipeline is called.
Processors:	
	GetUsersFromECS – Optional Queries against users in the external commerce system. **Note:** Either this processor or the next should be in the pipeline, normally not both **Note:** This processor is optional but recommended if CommerceUsers are supported in the ECS.
	GetUsersFromCMS – Mandatory Queries against users in CMS. **Note:** Either this processor or the next should be in the pipeline, normally not both

GetUser

Name:	**GetUser**
Description:	Returns a single user account.
Usage:	Called from Sitecore when a visitor a specific user account is needed
Args:	
	Request - Contains ShopName and User ID. Is set prior to calling the pipeline.
	Response – Returns a single user entity. Is read after the pipeline is called.
Processors:	
	GetUserFromECS – Optional Gets the specified user from the external commerce system.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

	Note: Either this processor or the next should be in the pipeline, normally not both **Note:** This processor is optional but recommended if CommerceUsers are supported in the ECS.
	GetUserFromCMS – Mandatory Gets the specified user from CMS. **Note:** Either this processor or the next should be in the pipeline, normally not both

CreateCustomer

Name:	**CreateCustomer**
Description:	This pipeline is responsible for creating a customer account. The customers are managed by the commerce system.
Usage:	Called from Sitecore
Args:	
	Request - Contains the customer entity. Is set prior to calling the pipeline.
	Response - Contains the customer entity after the pipeline is called.
Processors:	
	CreateCustomerInECS – Optional Depends on whether Customers are supported in the ECS Creates customer in external commerce system. It involves: o Create unique customer account in external system o Create mapping between Customer and any given user accounts **Note:** If External ID is parsed in and the customer already exists, nothing should happen
	CreateCustomerInSitecore – (Mandatory) Creates customer in Sitecore **Note:** This processor is mandatory to store the 1-to-many relationship between the CommerceUser and CommerceCustomers. If that is handled in the ECS, the processor is not mandatory
	AddCustomerToEAplan– Adds the customer to EA plan Customers storing a reference to the CommerceCustomer in EA state Since the customer potentially is created externally first, then the external ID of the customer is given and can be stored in Sitecore too, making the connection between the two.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

	Note: If External ID is parsed in and the customer already exists, then the pipeline should be aborted to avoid triggering the goal.
	TriggerDMSGoal – the goal "Customer Account Created" is triggered with values customer name and ShopName

GetCustomers

Name:	**GetCustomers**
Description:	Queries and returns customer accounts. Different input parameters can be provided and they will be combined using logical *and* Whether the customers are retrieved from ECS or CMS depends on the support of customers in ECS
Usage:	Called from Sitecore
Args:	
	Request - Contains search parameters.. Is set prior to calling the pipeline.
	Response - Returns a list of customer entities after the pipeline is called.
Processors:	
	GetCustomersFromECS – optional Queries against customers in the external commerce system. If the required search functionality is not supported in the ECS, then it can potentially be handled by searching for customers in CMS. **Note:** Either this processor or the next should be in the pipeline, normally not both
	GetCustomersFromCMS – optional Queries against Customers in CMS. If the required search functionality is not supported in CMS, then it can potentially be handled by searching for users in the ECS **Note:** Either this processor or the previous should be in the pipeline, normally not both

GetCustomer

Name:	**GetCustomer**
Description:	Returns the single customer entity with the specified ID Whether the customer is retrieved from ECS or CMS depends on the support of customers in ECS
Usage:	Called from Sitecore
Args:	
	Request - Contains shop name and Customer ID. Is set prior to calling the pipeline.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

	Response - Returns a single customer entity after the pipeline is called.
Processors:	
	GetCustomerFromECS – optional Gets the specified customer from the external commerce system. **Note:** Either this processor or the next should be in the pipeline, normally not both
	GetCustomersFromCMS – Mandatory Gets the specified customer from CMS. **Note:** Either this processor or the previous should be in the pipeline, normally not both

UpdateCustomer

Name:	**UpdateCustomer**
Description:	This pipeline is responsible for updating an existing customer account. The customers are managed by the commerce system.
Usage:	Called from Sitecore
Args:	
	Request - Contains the customer entity. Is set prior to calling the pipeline.
	Response - Contains the customer object. Is read after the pipeline is called.
Processors:	
	UpdateCustomerInECS – Mandatory Updates customer in external commerce system.
	SaveCustomertoEAState – mandatory Updates the customer stored in EA state based on updated CommerceCustomer returned from previous processor
	TriggerDMSEvent – the goal “Customer Account Updated” is triggered with values customer name and ShopName

DeleteCustomer

Name:	**DeleteCustomer**
Description:	This pipeline is responsible for deleting an existing customer account. The customers are managed by the commerce system.
Usage:	Called from Sitecore
Args:	
	Request - Contains the customer entity. Is set prior to calling the pipeline.
	Response -
Processors:	

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

	DeleteCustomerInECS – Mandatory Deletes customer in external system. **Note:** If customer account doesn't exists it can be noted in returned collection of External System Messages, but the pipeline must continue
	RemoveCustomerFromEAState – Mandatory Removes the customer stored in EA state
	TriggerDMSEvent – the goal "Customer Account Deleted" is triggered with values customer name and ShopName

DisableCustomer

Name:	**DisableCustomer**
Description:	This pipeline is responsible for disabling an existing customer account.
Usage:	Called from Sitecore.
Args:	
	Request - Contains the customer entity and a comment. Is set prior to calling the pipeline.
	Response – The disabled customer entity**.** Is read after the pipeline is called.
Processors:	
	DisableCustomerInECS – Disables an existing customer in the external commerce system.
	SaveCustomertoEAState – mandatory Updates the customer stored in EA state based on updated CommerceCustomer returned from previous processor
	TriggerDMSEvent – the page event "Customer Account Disabled" is triggered with values Name, ShopName and the provided comment.

EnableCustomer

Name:	**EnableCustomer**
Description:	This pipeline is responsible for enabling an existing customer account.
Usage:	Called from Sitecore.
Args:	
	Request - Contains the customer entity and a comment. Is set prior to calling the pipeline.
	Response – The enabled customer entity**.** Is read after the pipeline is called.
Processors:	
	EnableCustomerInECS – Enables an existing customer in the external commerce system.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

	SaveCustomertoEAState – mandatory Updates the customer stored in EA state based on updated CommerceCustomer returned from previous processor
	TriggerDMSEvent – the pageevent "Customer Account Enabled" is triggered with values Name, ShopName and the provided comment.

AddCustomerParties

Name:	**AddCustomerParties**
Description:	This pipeline is responsible for adding customer parties to the specified customer
Usage:	Called from Sitecore.
Args:	
	Request - Contains a customer and a list of CustomerParty instances. Is set prior to calling the pipeline.
	Response – A new instance of the customer Is read after the pipeline is called.
Processors:	
	AddCustomerParties – Responsibility: Is to add the provided parties to the customer account and persist them Usage: Mandatory. Ownership: The processor is provided with Connect and stores the parties with the customer account using the Sitecore membership provider Customization: No immediate need to customize. There should be a separate processor for storing the parties in the ECS. The processor should either replace this processor or be added in addition to this processor

RemoveCustomerParties

Name:	**RemoveCustomerParties**
Description:	This pipeline is responsible for removing customer parties from the specified customer
Usage:	Called from Sitecore.
Args:	

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

	Request - Contains a customer and a list of CustomerParty instances. Is set prior to calling the pipeline.
	Response – A new instance of the customer Is read after the pipeline is called.
Processors:	
	RemoveCustomerParties – Responsibility: Is to remove the provided parties from the customer account Usage: Mandatory. Ownership: The processor is provided with Connect and removes the parties which will no longer be persisted anywhere in Sitecore Customization: No immediate need to customize. There should be a separate processor for storing the parties in the ECS. The processor should either replace this processor or be added in addition to this processor

UpdateCustomerParties

Name:	**UpdateCustomerParties**
Description:	This pipeline is responsible for updating customer parties on the specified customer
Usage:	Called from Sitecore.
Args:	
	Request - Contains a customer and a list of CustomerParty instances. Is set prior to calling the pipeline.
	Response – A new instance of the customer Is read after the pipeline is called.
Processors:	
	UpdateCustomerParties – Responsibility: Is to update the provided parties on the customer account and persist them Usage: Mandatory. Ownership: The processor is provided with Connect and stores the updated parties with the customer account using the Sitecore membership provider Customization: No immediate need to customize.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

	There should be a separate processor for storing the parties in the ECS. The processor should either replace this processor or be added in addition to this processor

GetParties

Name:	**GetParties**
Description:	This pipeline is responsible for getting the parties
Usage:	Called from Sitecore.
Args:	
	Request - Contains a customer. Is set prior to calling the pipeline.
	Response – A list of parties. Is read after the pipeline is called.
Processors:	
	GetParties – Responsibility: Is to return the complete list of parties stored with the customer Usage: Mandatory. Ownership: The processor is provided with Connect and stores the parties with the customer account using the Sitecore membership provider Customization: No immediate need to customize. There should be a separate processor for storing the parties in the ECS. The processor should either replace this processor or be added in addition to this processor

AddParties

Name:	**AddParties**
Description:	This pipeline is responsible for adding parties to the specified customer
Usage:	Called from Sitecore.
Args:	
	Request - Contains a customer and a list of CustomerParty instances. Is set prior to calling the pipeline.
	Response – A new instance of the customer Is read after the pipeline is called.
Processors:	
	AddParties –

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

	Responsibility: Is to add the provided parties to the customer account and persist them Usage: Mandatory. Ownership: The processor is provided with Connect and stores the parties with the customer account using the Sitecore membership provider Customization: No immediate need to customize. There should be a separate processor for storing the parties in the ECS. The processor should either replace this processor or be added in addition to this processor

RemoveParties

Name:	**RemoveParties**
Description:	This pipeline is responsible for removing parties stored with the specified customer
Usage:	Called from Sitecore.
Args:	
	Request - Contains a customer and a list of party instances. Is set prior to calling the pipeline.
	Response – A new instance of the customer Is read after the pipeline is called.
Processors:	
	RemoveParties – Responsibility: Is to remove the provided parties from the customer account Usage: Mandatory. Ownership: The processor is provided with Connect and removes the parties which will no longer be persisted anywhere in Sitecore Customization: No immediate need to customize. There should be a separate processor for storing the parties in the ECS. The processor should either replace this processor or be added in addition to this processor

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

UpdateParties

Name:	**UpdateParties**
Description:	This pipeline is responsible for updating customer parties on the specified customer
Usage:	Called from Sitecore.
Args:	
	Request - Contains a customer and a list of CustomerParty instances. Is set prior to calling the pipeline.
	Response – A new instance of the customer Is read after the pipeline is called.
Processors:	
	UpdateParties – Responsibility: Is to update the provided parties on the customer account and persist them Usage: Mandatory. Ownership: The processor is provided with Connect and stores the updated parties with the customer account using the Sitecore membership provider Customization: No immediate need to customize. There should be a separate processor for storing the parties in the ECS. The processor should either replace this processor or be added in addition to this processor

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

2.8 Product Sync

2.8.1 The Product Sync Domain Model

This chapter describes the domain model that represents the product objects where the following three assertions holds true:

- The domain model primarily serves as DTO objects for transferring information between the external commerce system and Sitecore.
- The domain model contains the minimum required information as contracts and will typically be extended, when integrated with a specific commerce system.
- The domain model is used when a Sitecore developer needs to synchronize product data with the external commerce system or product data is pushed into Sitecore from the external commerce system

The class diagram below shows the domain model.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Note: The domain model consists of abstract classes that make up the contracts with the external system. The contracts are defined as abstract classes instead of interfaces to allow the model to be easily extended later if needed. This follow the best practice guidelines defined in the book Framework Design Guidelines.

Default implementation of the contracts are delivered as part of Connect. If an actual Connect provider with an external commerce system contains more functionality that provided by default, the implementation can be replaced. All instantiation of actual classes will be handled through dependency injection.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Class: Product

The product class is responsible for representing a product or any variant of it, hence a variant is a product in this model.

Name	Type	Description
ExternalId	string	Unique identifier for the product in the commerce system. This can be used to get a reference to the product using the commerce system's native API.
SitecoreItemId	string	Returns the Sitecore ID
Name	string	Name of the product
ShortDescription	string	The short description of the product.
FullDescription	string	The full description of the product.
ProductType	ProductType	A reference to the product type
Manufacturers	Manufacturer	Reference to the manufacturers
ClassificationGroups	IReadOnlyCollection<ProductClassificationGroup>	Reference to the associated classifications and categories
Specifications	ProductSpecifications	Collection of specifications set directly on the product
VariantSpecifications	ProductVariantSpecifications	List of references to specifications that tells the variants apart and which potentially can be selectable to the visitor
Resources	IReadOnlyCollection<ProductResource>	Reference to the associated resources
Divisions	IReadOnlyCollection<Division>	Reference to the associated divisions
RelationTypes	IReadOnlyCollection<ProductRelationType>	Reference to the related products
Created	DateTime	Date of creation
Updated	DateTime	Date of last update

Class: ProductSpecifications

Name	Type	Description

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

ExternalId	string	Unique identifier used to identify the item in an external system.
SitecoreItemId	string	Represents the Sitecore Id.
Specifications	IReadOnlyCollection<ProductSpecification>	
Created	DateTime	
Updated	DateTime	

Class: ProductSpecification

Name	Type	Description
ExternalId	string	Unique identifier used to identify the item in an external system.
SitecoreItemId	string	Represents the Sitecore Id.
Group	String	
Key	String	
Value	String	
Created	DateTime	
Updated	DateTime	

Class: ProductClassification

Name	Type	Description
ExternalId	string	Unique identifier used to identify the item in an external system.
ExternalParentId	string	
SitecoreItemId	string	Represents the Sitecore Id.
Name	string	
Description	string	
Specifications	ProductSpecifications	
Created	DateTime	
Updated		

Class: ProductType

Name	Type	Description
ExternalId	string	Unique identifier used to identify the item in an external system.
SitecoreItemId	string	Represents the Sitecore Id.
ProductTypeId	String	
Description	String	
Specifications	ProductSpecifications	
Created	DateTime	
Updated	DateTime	

Class: ProductManufacturer

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Name	Type	Description
ExternalId	string	Unique identifier used to identify the item in an external system.
SitecoreItemId	string	Represents the Sitecore Id.
Name	String	
Description	String	
WebSiteUrl	String	
ProductTypes	IReadOnlyCollection<ProductType>	
Created	DateTime	
Updated	DateTime	

Class: ProductClassificationGroup

Name	Type	Description
ExternalId	string	Unique identifier used to identify the item in an external system.
SitecoreItemId	string	Represents the Sitecore Id.
Name	String	
Description	String	
Classifications	IReadOnlyCollection<ProductClassification>	
Created	DateTime	
Updated	Name	

Class: ProductVariantSpecificaions

Name	Type	Description
ExternalId	string	Unique identifier used to identify the item in an external system.
SitecoreItemId	string	Represents the Sitecore Id.
Specifications	IReadOnlyCollection<ProductSpecification>	
Created	DateTime	
Updated	DateTime	

Class: ProductResource

Name	Type	Description
ExternalId	string	Unique identifier used to identify the item in an external system.
SitecoreItemId	string	Represents the Sitecore Id.
Created	DateTime	
Updated	DateTime	

Class: Division

Name	Type	Description

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

ExternalId	string	Unique identifier used to identify the item in an external system.
SitecoreItemId	string	Represents the Sitecore Id.
Name	String	
SubDivisions	IReadOnlyCollection<Division>	
Created	DateTime	
Updated	DateTime	

Class: ProductRelation

Name	Type	Description
ExternalId	string	Unique identifier used to identify the item in an external system.
SitecoreItemId	string	Represents the Sitecore Id.
Product	Product	
ReferredProduct	Product	
Created	DateTime	
Updated	DateTime	

Class: ProductRelationType

Name	Type	Description
ExternalId	string	Unique identifier used to identify the item in an external system.
SitecoreItemId	string	Represents the Sitecore Id.
Name	String	
Relations	IReadOnlyCollection<ProductRelation>	
Created	DateTime	
Updated	DateTime	

2.8.2 Product Sync Service Methods

Service providers are wrapper objects designed to make it easier to interact with Connect pipelines. The providers implement no logic other than calling Connect pipelines. All of the business logic is implemented in the pipeline processors.

The Product Sync Service Provider contains the following methods for interacting with product sync data.

SynchronizeProducts

SyncronizeProducts is used to synchronize a collection of products between the external commerce system and Sitecore.

The synchronization can go both ways, so changes made to product data in CMS content are pushed to the external commerce system.

A log must be kept of events registered during synchronization. At minimum it should contain a list of products successfully updated. It would be better

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Upon return the result contains the list of messages generated during synchronization, which would be the Ids of the products that failed during synchronization

Name:	**SynchronizeProducts**
Description:	SynchronizeProducts calls the pipeline " SynchronizeProducts" to synchronize changes to all updated products and related repositories
Usage:	Called from Sitecore or the ECS when the product manager wants to update products both from and to the ECS
Signature:	`SynchronizeProductsResult SynchronizeProducts(SynchronizeProductsRequest request)`
Input:	
	Language – string, optional. The language for the product data being synchronized. Default is English ("en" or "US-EN")
	Direction – optional**.** An enum type indicating whether synchronization goes from ECS -> SC, SC -> ECS or both ways. The default is ECS -> SC. See section **Error! Reference source not found.** for more.
Output:	
	SystemMessages - Collection of messages from the external system.

SynchronizeProductList

Name:	**SynchronizeProductList**
Description:	SynchronizeProductList calls the pipeline " SynchronizeProductList"
Usage:	Called from Sitecore or the ECS when the product manager wants to update a list of products both from and to the ECS
Signature:	`SynchronizeProductListResult SynchronizeProductList(SynchronizeProductListRequest request)`
Input:	
	List<ExternalProductIds> - List of strings, mandatory The list of external product ids to synchronize
	Language – string, optional. The language for the product data being synchronized. Default is English ("en" or "US-EN")
	Direction – enum, optional An enum type indicating whether synchronization goes from ECS -> SC, SC -> ECS or both ways. The default is ECS -> SC. See section **Error! Reference s ource not found.** for more.
Output:	
	SystemMessages - Collection of messages from the external system.

SynchronizeProduct

SyncProduct is used to synchronize a single product between the external commerce system and Sitecore. The product to synchronize is given by ID.

The synchronization can go both ways, so changes made to product data in CMS content are pushed to the external commerce system as well.

Name:	**SynchronizeProduct**
Description:	SynchronizeProduct calls the pipeline " SynchronizeProduct"

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Usage:	Called from Sitecore or the ECS when the product manager wants to update a single product both from and to the ECS
Signature:	`SynchronizeProductResult SynchronizeProduct(SynchronizeProductRequest request)`
Input:	
	ProductId – string, mandatory The external product id to be synchronized
	Language – string, optional. The language for the product data being synchronized. Default is English ("en" or "US-EN")
	Direction – enum, optional An enum type indicating whether synchronization goes from ECS -> SC, SC -> ECS or both ways. The default is ECS -> SC. See section **Error! Reference source not found.** for more.
Output:	
	SystemMessages - Collection of messages from the external system.

SynchronizeArtifacts

SynchronizeArtifacts is responsible for synchronizing all the related repositories: Manufacturers, Types, Classifications, Divisions, Resources, and Specifications before the individual products are synchronized the references to repositories are updated.

Name:	**SynchronizeArtifacts**
Description:	SynchronizeArtifacts calls the pipeline " SynchronizeArtifacts" to synchronize all the related repositories: Manufacturers, Types, Classifications, Divisions, Resources, Specifications
Usage:	Called from Sitecore or the ECS when the product manager wants to update the product related repositories
Signature:	`SynchronizeArtifactsResult SynchronizeArtifacts (SynchronizeArtifactsRequest request)`
Input:	
	Language – string, optional. The language for the product data being synchronized. Default is English ("en" or "US-EN")
Output:	
	SystemMessages - Collection of messages from the external system.

2.8.3 Product Sync Pipelines

SynchronizeProducts

SyncronizeProducts is used to synchronize products between the external commerce system and Sitecore.

Name:	**SynchronizeProducts**
Description:	This pipeline is responsible for obtaining the lists of product Ids to be synchronized and iterate over them
Usage:	Called from Sitecore or the external commerce system
Args:	

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

	Request – Is empty by default. Is set prior to calling the pipeline.
	Response - Contains the list of messages generated during synchronization, which would be the Ids of the products that failed during synchronization Is read after the pipeline is called.
Processors:	
	RunSynchronizeArtifacts – Calls individual pipeline to synchronize all the related repositories: Manufacturers, Types, Classifications, Divisions, Resources, Specifications
	RunGetSCProductList – Obtain the list of product ids to synchronize from Sitecore. This processor can be left out if product data is only pushed from the external system
	RunGetECSProductList - Obtain the list of product ids to synchronize from ECS
	EvaluateProductListUnionToSynchronize – Creates the union of product Ids to be synchronized based on the two lists obtained from ECS and SC
	RunSynchronizeProductList – Calls individual pipeline with the evaluated list of product IDs to synchronize

SynchronizeProductList

SyncronizeProductlist is used to synchronize a given list of products between the external commerce system and Sitecore.

Name:	**SynchronizeProductList**
Description:	This pipeline is responsible for iterating over the given list of product Ids and synchronize
Usage:	Called from Sitecore or the external commerce system
Args:	
	Request – List of product Ids to synchronize. Is set prior to calling the pipeline.
	Response - Contains the list of messages generated during synchronization, which would be the Ids of the products that failed during synchronization Is read after the pipeline is called.
Processors:	
	SynchronizeProductList - Iterates over the given list of product Ids and runs pipeline SynchronizeProduct for each product

GetExternalCommerceSystemProductList

Name:	**GetExternalCommerceSystemProductList**
Description:	This pipeline is responsible for obtaining the list of product Ids to be synchronized from the external commerce system
Usage:	Called internally from SynchronizeProducts but can also be called explicitly from both ECS or SC
Args:	
	Request – No default data. Is set prior to calling the pipeline.
	Response - Contains the list of product Ids to be synchronized and SystemMessages. Is read after the pipeline is called.
Processors:	
	GetExternalCommerceSystemProductList – Get list of IDs from ECS

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

GetSitecoreProductList

Name:	**GetSitecoreProductList**
Description:	This pipeline is responsible for obtaining the list of product Ids to be synchronized from Sitecore
Usage:	Called internally from SynchronizeProducts but can also be called explicitly from both ECS or SC
Args:	
	Request – No default data. Is set prior to calling the pipeline.
	Response - Contains the list of product Ids to be synchronized and SystemMessages. Is read after the pipeline is called.
Processors:	
	GetSitecoreProductList – Get list of IDs from Sitecore

SynchronizeArtifacts

Name:	**SynchronizeArtifacts**
Description:	This pipeline is responsible for synchronizing all the related repositories: Manufacturers, Types, Classifications, Divisions, Resources, Specifications
Usage:	Called from Sitecore or the external commerce system
Args:	
	Request – Is empty by default. Is set prior to calling the pipeline.
	Response - Contains the list of messages generated during synchronization
Processors:	
	RunSynchronizeManufacturers – Calls individual pipeline to synchronize manufacturers respository
	RunSynchronizeTypes - Calls individual pipeline to synchronize types repository
	RunSynchronizeClassifications - Calls individual pipeline to synchronize classifications respository
	RunSynchronizeDivisions - Calls individual pipeline to synchronize divisions respository
	RunSynchronizeTypes - Calls individual pipeline to synchronize type repository
	RunSynchronizeResources - Calls individual pipeline to synchronize resources respository
	RunSynchronizeSpecifications - Calls individual pipeline to synchronize global, Category and type specifications

SynchronizeManufacturers

Name:	**SynchronizeManufacturers**
Description:	This pipeline is responsible for synchronizing all manufacturers in the separate Manufactures repository
Usage:	Called from pipeline SynchronizeArtifacts as initialization of separate repositories before synchronizing products and their references to these repositories.
Args:	

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

	Request – Is empty by default. Is set prior to calling the pipeline.
	Response - Contains the list of messages generated during synchronization
Processors:	
	ReadManufacturersFromSC – Optional Reads the manufacturers to synchronize from SC. This processor can be skipped if changes only are pushed from ECS to SC.
	ReadManufacturersFromECS – Mandatory Reads the manufacturers to synchronize from ECS
	ResolveManufacturersChanges – Optional Resolves differences between ECS and SC**.** This processor can be skipped if changes only are pushed from ECS to SC.
	SaveManufacturersToECS – Optional Saves synchronized manufacturers to ECS. This processor can be skipped if changes only are pushed from ECS to SC.
	SaveManufacturersToSC – Mandatory Saves synchronized manufacturers to SC.

SynchronizeClassifications

Name:	**SynchronizeClassifications**
Description:	This pipeline is responsible for synchronizing all classifications in the separate Classifications repository Since multiple different classification schemes are supported, this pipeline is responsible for synchronizing all schemes
Usage:	Called from pipeline SynchronizeArtifacts as initialization of separate repositories before synchronizing products and their references to these repositories.
Args:	
	Request – Is empty by default. Is set prior to calling the pipeline.
	Response - Contains the list of messages generated during synchronization
Processors:	
	ReadClassificationsFromSC – Optional Reads the classifications to synchronize from SC. This processor can be skipped if changes only are pushed from ECS to SC.
	ReadClassificationsFromECS – Mandatory Reads the classifications to synchronize from ECS
	ResolveClassificationsChanges – Optional Resolves differences between ECS and SC**.** This processor can be skipped if changes only are pushed from ECS to SC.
	SaveClassificationsToECS – Optional Saves synchronized classifications to ECS. This processor can be skipped if changes only are pushed from ECS to SC.
	SaveClassificationsToSC – Mandatory Saves synchronized classifications to SC.

SynchronizeTypes

Name:	**SynchronizeTypes**
Description:	This pipeline is responsible for synchronizing all Types in the separate Product Types repository

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Usage:	Called from pipeline SynchronizeArtifacts as initialization of separate repositories before synchronizing products and their references to these repositories.
Args:	
	Request – Is empty by default. Is set prior to calling the pipeline.
	Response - Contains the list of messages generated during synchronization
Processors:	
	ReadTypesFromSC – Optional Reads the types to synchronize from SC. This processor can be skipped if changes only are pushed from ECS to SC.
	ReadTypesFromECS – Mandatory Reads the types to synchronize from ECS
	ResolveTypesChanges – Optional Resolves differences between ECS and SC. This processor can be skipped if changes only are pushed from ECS to SC.
	SaveTypesToECS – Optional Saves synchronized types to ECS. This processor can be skipped if changes only are pushed from ECS to SC.
	SaveTypesToSC – Mandatory Saves synchronized types to SC.

SynchronizeDivisions

Name:	**SynchronizeDivisions**
Description:	This pipeline is responsible for synchronizing all divisions in the separate Divisions repository
Usage:	Called from pipeline SynchronizeArtifacts as initialization of separate repositories before synchronizing products and their references to these repositories.
Args:	
	Request – Is empty by default. Is set prior to calling the pipeline.
	Response - Contains the list of messages generated during synchronization
Processors:	
	ReadDivisionsFromSC – Optional Reads the divisions to synchronize from SC. This processor can be skipped if changes only are pushed from ECS to SC.
	ReadDivisionsFromECS – Mandatory Reads the divisions to synchronize from ECS
	ResolveDivisionsChanges – Optional Resolves differences between ECS and SC. This processor can be skipped if changes only are pushed from ECS to SC.
	SaveDivisionsToECS – Optional Saves synchronized divisions to ECS. This processor can be skipped if changes only are pushed from ECS to SC.
	SaveDivisionsToSC – Mandatory Saves synchronized divisions to SC.

SynchronizeResources

Name:	**SynchronizeResources**

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Description:	This pipeline is responsible for synchronizing all resources in Sitecore Media Library
Usage:	Called from pipeline SynchronizeArtifacts as initialization of separate repositories before synchronizing products and their references to these repositories. NB In case resources are kept only in the ECS, then this pipeline can be skipped or configured empty
Args:	
	Request – Is empty by default. Is set prior to calling the pipeline.
	Response - Contains the list of messages generated during synchronization
Processors:	
	ReadResourcesFromSC – Optional Reads the resources to synchronize from SC. This processor can be skipped if changes only are pushed from ECS to SC.
	ReadResourcesFromECS – Mandatory Reads the resources to synchronize from ECS
	ResolveResourcesChanges – Optional Resolves differences between ECS and SC**.** This processor can be skipped if changes only are pushed from ECS to SC.
	SaveResourcesToECS – Optional Saves synchronized resources to ECS. This processor can be skipped if changes only are pushed from ECS to SC.
	SaveResourcesToSC – Mandatory Saves synchronized resources to SC.

SynchronizeSpecifications

Name:	**SynchronizeSpecifications**
Description:	This pipeline is responsible for synchronizing specifications on type, category and globally by running separate pipelines for each
Usage:	Called from pipeline SynchronizeArtifacts as initialization of separate repositories before synchronizing products and their references to these repositories
Args:	
	Request - Is set prior to calling the pipeline.
	Response - Is read after the pipeline is called.
Processors:	
	RunSynchronizeGlobalSpecifications – Runs a separate pipeline to synchronize global specifications
	RunSynchronizeTypeSpecifications - Runs a separate pipeline to synchronize type specifications
	RunSynchronizeClassificationSpecifications - Runs a separate pipeline to synchronize category specifications

SynchronizeGlobalSpecifications

Name:	**SynchronizeGlobalSpecifications**
Description:	This pipeline is responsible for synchronizing global specifications

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

	The specifications and the tables for fixed key-value pairs are stored under "/sitecore/content/Product Artifacts/Lookups/Global Product Specification Lookups"
Usage:	Called internally from pipeline SynchronizeSpecifications
Args:	
	Request - Is set prior to calling the pipeline.
	Response - Is read after the pipeline is called.
Processors:	
	ReadGlobalSpecificationsFromSC – Optional Reads the product specifications data from SC. This processor can be skipped if changes only are pushed from ECS to SC.
	ReadGlobalSpecificationsFromECS – Mandatory Reads the product specifications data from ECS
	ResolveGlobalSpecificationsChanges – Optional Resolves differences between ECS and SC. This processor can be skipped if changes only are pushed from ECS to SC.
	SaveGlobalSpecificationsToECS – Optional Saves synchronized product specifications data to ECS. This processor can be skipped if changes only are pushed from ECS to SC.
	SaveGlobalSpecificationsToSC – Mandatory Saves synchronized product specifications data to SC.

SynchronizeTypeSpecifications

Name:	**SynchronizeTypeSpecifications**
Description:	This pipeline is responsible for synchronizing type specifications The specifications and the tables for fixed key-value pairs are stored under "/sitecore/content/Product Artifacts/Product Types" Note: For types also specification options and default values are synchronized as part of this pipeline
Usage:	Called internally from pipeline SynchronizeSpecifications
Args:	
	Request - Is set prior to calling the pipeline.
	Response - Is read after the pipeline is called.
Processors:	
	ReadTypeSpecificationsFromSC – Optional Reads the product specifications data from SC. This processor can be skipped if changes only are pushed from ECS to SC.
	ReadTypeSpecificationsFromECS – Mandatory Reads the product specifications data from ECS
	ResolveTypeSpecificationsChanges – Optional Resolves differences between ECS and SC. This processor can be skipped if changes only are pushed from ECS to SC.
	SaveTypeSpecificationsToECS – Optional Saves synchronized product specifications data to ECS. This processor can be skipped if changes only are pushed from ECS to SC.
	SaveTypeSpecificationsToSC – Mandatory Saves synchronized product specifications data to SC.

SynchronizeClassificationSpecifications

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Name:	**SynchronizeClassificationSpecifications**
Description:	This pipeline is responsible for synchronizing Category specifications The specifications and the tables for fixed key-value pairs are stored under "/sitecore/content/Product Artifacts/Classifications" Note: Since multiple different classification schemes are supported, this pipeline is responsible for synchronizing specifications for all schemes
Usage:	Called internally from pipeline SynchronizeSpecifications
Args:	
	Request - Is set prior to calling the pipeline.
	Response - Is read after the pipeline is called.
Processors:	
	ReadClassificationSpecificationsFromSC – Optional Reads the classifications specifications data from SC. This processor can be skipped if changes only are pushed from ECS to SC.
	ReadClassificationSpecificationsFromECS – Mandatory Reads the classifications specifications data from ECS
	ResolveClassificationSpecificationsChanges – Optional Resolves differences between ECS and SC**.** This processor can be skipped if changes only are pushed from ECS to SC.
	SaveClassificationSpecificationsToECS – Optional Saves synchronized classifications specifications data to ECS. This processor can be skipped if changes only are pushed from ECS to SC.
	SaveClassificationSpecificationsToSC – Mandatory Saves synchronized classifications specifications data to SC.

After running the pipeline the categories will have a folder called Specifications containing all the specifications for the category, including tables with fixed set key-value pairs for reference from products.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

SynchronizeProduct

SynchronizeProduct is used to synchronize a single product between the external commerce system and Sitecore. The product to synchronize is given by Id.

Name:	**SynchronizeProduct**
Description:	This pipeline is responsible for synchronizing a single product by calling a number of individual pipelines. Each pipeline will update the references between the product and the related separate repositories except pipeline SynchronizeProductItem, which operates on the main product data on the product item itself.
Usage:	Called directly from service method SynchronizeProduct and from SynchronizeProductList indirectly
Args:	
	Request - Contains the external product Id. Is set prior to calling the pipeline.
	Response - Contains the SystemMessages. Is read after the pipeline is called.
Processors:	
	RunSyncronizeProductManufacturers – Synchronizes relations to manufacturers
	RunSyncronizeProductType – Synchronizes relation to product type
	RunSyncronizeProductClassifications – Synchronizes relations to classifications
	RunSyncronizeProductResources – Synchronizes resources and relations to resources
	RunSyncronizeProductRelations – Synchronizes relations to other products thorugh cross-sell, variants etc
	RunSyncronizeProductDivisions – Synchronizes relations to divisions
	RunSyncronizeProductItem – Synchronizes main product data on the product item itself
	RunSynchronizeProductSpecifications – Calls individual pipeline to synchronize product specifications

SynchronizeProductManufacturers

SynchronizeProductManufacturers is used to synchronize references between a single product and separate repository Manufacturers between the external commerce system and Sitecore. The product to synchronize is given by external product Id.

Name:	**SynchronizeProductManufacturers**
Description:	This pipeline is responsible for synchronizing and updating the relation between a given product and manufacturers. It's assumed that manufacturers are already synchronized and present in CMS The references to manufacturers are stored directly on the main product item
Usage:	Called internally from pipeline SynchronizeProduct
Args:	
	Request - Contains the external product Id. Is set prior to calling the pipeline.
	Response - Contains the Manufacturer. Is read after the pipeline is called.
Processors:	
	ReadProductManafacturersFromSC – Optional Reads the product manufacturers reference data from SC. This processor can be skipped if changes only are pushed from ECS to SC.
	ReadProductManafacturersFromECS – Mandatory Reads the product manufacturers reference data from ECS
	ResolveProductManafacturersChanges – Optional

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

	Resolves differences between ECS and SC. This processor can be skipped if changes only are pushed from ECS to SC.
	SaveProductManafacturersToECS – Optional Saves synchronized product manufacturers reference data to ECS. This processor can be skipped if changes only are pushed from ECS to SC.
	SaveProductManafacturersToSC – Mandatory Saves synchronized product manufacturers reference data to SC.

SynchronizeProductType

Name:	**SynchronizeProductType**
Description:	This pipeline is responsible for synchronizing and updating the references between a given product and its product type. It's assumed that types are already synchronized and present in CMS The references to product type is stored directly on the main product item
Usage:	Called internally from pipeline SynchronizeProduct
Args:	
	Request - Contains the external product Id. Is set prior to calling the pipeline.
	Response - Contains the Manufacturer. Is read after the pipeline is called.
Processors:	
	ReadProductTypeFromSC – Optional Reads the product type reference data from SC. This processor can be skipped if changes only are pushed from ECS to SC.
	ReadProductTypeFromECS – Mandatory Reads the product type reference data from ECS
	ResolveProductTypeChanges – Optional Resolves differences between ECS and SC. This processor can be skipped if changes only are pushed from ECS to SC.
	SaveProductTypeToECS – Optional Saves synchronized product type reference data to ECS. This processor can be skipped if changes only are pushed from ECS to SC.
	SaveProductTypeToSC – Mandatory Saves synchronized product type reference data to SC.

SyncronizeProductClassifications

Name:	**SynchronizeProductClassifications**
Description:	This pipeline is responsible for synchronizing and updating the references between a given product and associated classifications and categories within. It's assumed that classifications are already synchronized and present in CMS The references to categories are stored directly on the main product item
Usage:	Called internally from pipeline SynchronizeProduct
Args:	
	Request - Contains the external product Id. Is set prior to calling the pipeline.
	Response - Contains the Manufacturer. Is read after the pipeline is called.
Processors:	
	ReadProductClassificationsFromSC – Optional Reads the product classification reference data from SC. This processor can be skipped if changes only are pushed from ECS to SC.
	ReadProductClassificationsFromECS – Mandatory Reads the product classification reference data from ECS
	ResolveProductClassificationsChanges – Optional

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

	Resolves differences between ECS and SC. This processor can be skipped if changes only are pushed from ECS to SC.
	SaveProductClassificationsToECS – Optional Saves synchronized product classification reference data to ECS. This processor can be skipped if changes only are pushed from ECS to SC.
	SaveProductClassificationsToSC – Mandatory Saves synchronized product classification reference data to SC.

SynchronizeProductEntity

Name:	**SynchronizeProductEntity**
Description:	This pipeline is responsible for synchronizing and updating the main product entity (data(for the product with the given external product ID.
Usage:	Called internally from pipeline SynchronizeProduct
Args:	
	Request - Contains the external product Id. Is set prior to calling the pipeline.
	Response - Contains the Manufacturer. Is read after the pipeline is called.
Processors:	
	ReadProductFromSC – Optional Reads the product data from SC. This processor can be skipped if changes only are pushed from ECS to SC.
	ReadProductFromECS – Mandatory Reads the product data from ECS
	ResolveProductChanges – Optional Resolves differences between ECS and SC. This processor can be skipped if changes only are pushed from ECS to SC.
	SaveProductToECS – Optional Saves synchronized product data to ECS. This processor can be skipped if changes only are pushed from ECS to SC.
	SaveProductToSC – Mandatory Saves synchronized product data to SC.

SynchronizeProductDivisions

Name:	**SynchronizeProductDivisions**
Description:	This pipeline is responsible for synchronizing and updating the references between a given product and associated divisions. It's assumed that divisions are already synchronized and present in CMS The references to divisions are stored directly on the main product item
Usage:	Called internally from pipeline SynchronizeProduct
Args:	
	Request - Contains the external product Id. Is set prior to calling the pipeline.
	Response - Contains the Manufacturer. Is read after the pipeline is called.
Processors:	
	ReadProductDivisionsFromSC – Optional Reads the product divisions reference data from SC. This processor can be skipped if changes only are pushed from ECS to SC.
	ReadProductDivisionsFromECS – Mandatory Reads the product divisions reference data from ECS
	ResolveProductDivisionsChanges – Optional Resolves differences between ECS and SC. This processor can be skipped if changes only are pushed from ECS to SC.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

	SaveProductDivisionsToECS – Optional Saves synchronized product divisions reference data to ECS. This processor can be skipped if changes only are pushed from ECS to SC.
	SaveProductDivisionsToSC – Mandatory Saves synchronized product divisions reference data to SC.

SynchronizeProductResources

Name:	**SynchronizeProductDivisions**
Description:	This pipeline is responsible for synchronizing and updating the references between a given product and associated resources. It's assumed that resources are already synchronized and present in CMS or that the references are external using an URI to point to the location The references to resources are stored under the main product item on the path "[Product Item]/Resources/[Resource]"
Usage:	Called internally from pipeline SynchronizeProduct
Args:	
	Request - Contains the external product Id. Is set prior to calling the pipeline.
	Response - Contains the Manufacturer. Is read after the pipeline is called.
Processors:	
	ReadProductResourcesFromSC – Optional Reads the product resources reference data from SC. This processor can be skipped if changes only are pushed from ECS to SC.
	ReadProductResourcesFromECS – Mandatory Reads the product resources reference data from ECS
	ResolveProductResourcesChanges – Optional Resolves differences between ECS and SC**.** This processor can be skipped if changes only are pushed from ECS to SC.
	SaveProductResourcesToECS – Optional Saves synchronized product resources reference data to ECS. This processor can be skipped if changes only are pushed from ECS to SC.
	SaveProductResourcesToSC – Mandatory Saves synchronized product resources reference data to SC.

SynchronizeProductRelations

Name:	**SynchronizeProductRelations**
Description:	This pipeline is responsible for synchronizing product relations for a single product
Usage:	Called internally from pipeline SynchronizeProduct
Args:	
	Request - Contains the external product Id . Is set prior to calling the pipeline.
	Response - Contains the Product relations. Is read after the pipeline is called.
Processors:	
	ReadProductRelationsFromSC - Reads the product relations data from SC. This processor can be skipped if changes only are pushed from ECS to SC.
	ReadProductRelationsFromECS – Reads the product relations data from ECS
	ResolveProductRelationsChanges – Resolves differences between ECS and SC**.** This processor can be skipped if changes only are pushed from ECS to SC.
	SaveProductRelationsToECS – Saves synchronized product relations data to ECS. This processor can be skipped if changes only are pushed from ECS to SC.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

	SaveProductRelationsToSC - Saves synchronized product relations data to SC.

SynchronizeProductSpecifications

Name:	**SynchronizeProductSpecifications**
Description:	This pipeline is responsible for synchronizing product specifications for a single product. It's assumed that specification tables of fixed key-value pairs (lookups) are already synchronized and present in CMS when this pipeline is run The references to specifications are stored under the main product item on the path "[Product Item]/Specifications/[Specification]" Specifications that reference lookup tables can point to specification tables located under global, classification or type.
Usage:	Called internally from pipeline SynchronizeProduct
Args:	
	Request - Contains the external product Id . Is set prior to calling the pipeline.
	Response - Contains the Product relations. Is read after the pipeline is called.
Processors:	
	ReadProductSpecificationsFromSC - Optional Reads the product specification data from SC. This processor can be skipped if changes only are pushed from ECS to SC.
	ReadProductSpecificationsFromECS – Mandatory Reads the product specification data from ECS
	ResolveProductSpecificationChanges – Optional Resolves differences between ECS and SC**.** This processor can be skipped if changes only are pushed from ECS to SC.
	SaveProductSpecificationsToECS – Optional Saves synchronized product specification data to ECS. This processor can be skipped if changes only are pushed from ECS to SC.
	SaveProductSpecificationsToSC – Mandatory Saves synchronized product specification data to SC.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

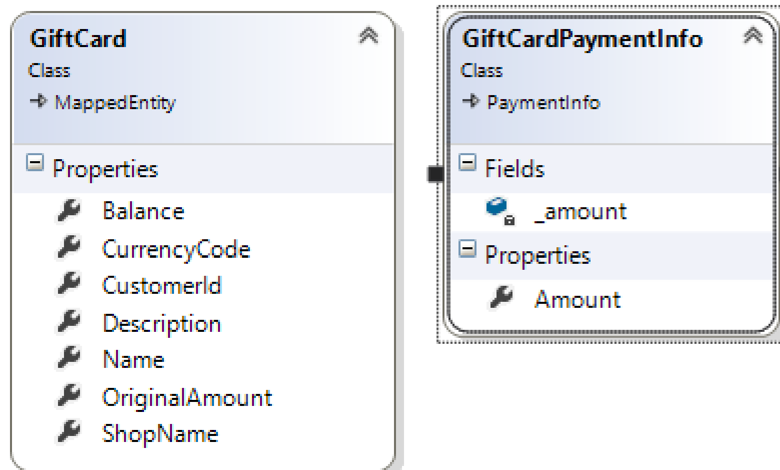
2.9 Gift Cards

2.9.1 The Gift Cards Domain Model

This chapter describes the domain model that represents the gift card objects where the following three assertions holds true:

- The domain model primarily serves as DTO objects for transferring information between the external commerce system and Sitecore.
- The domain model contains the minimum required information as contracts and will typically be extended, when integrated with a specific commerce system.
- The domain model is used when a Sitecore developer needs to synchronize product data with the external commerce system or product data is pushed into Sitecore from the external commerce system

The class diagram below shows the domain model.



Note: The domain model consists of abstract classes that make up the contracts with the external system. The contracts are defined as abstract classes instead of interfaces to allow the model to be easily extended later if needed. This follow the best practice guidelines defined in the book Framework Design Guidelines.

Default implementation of the contracts are delivered as part of Connect. If an actual Connect provider with an external commerce system contains more functionality that provided by default, the implementation can be replaced. All instantiation of actual classes will be handled through dependency injection.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Class: GiftCard

The GiftCard class is responsible for representing a gift card entity in an external system.

Name	Type	Description
ExternalId	string	Unique identifier for the product in the commerce system. This can be used to get a reference to the product using the commerce system's native API.
Name	string	The name, if any, of the gift card
CustomerId	string	The identifier of the owner of the card
ShopName	string	The shop the card is associated to if any
CurrencyCode	string	The currency of the card value
Balance	decimal	The remaining value of the card
OriginalAmount	decimal	The initial value of the card
Description	string	Reference to the manufacturers

Class: GiftCardPaymentInfo

The GiftCardPaymentInfo class is an implementation of the PaymentInfo class that allows a payment to be made with a gift card.

Name	Type	Description
Amount	decimal	The amount being charged against the gift card.
CardNumber	string	The ID of the gift card being used for payment.

2.9.2 Gift Cards Service Provider

Service providers are wrapper objects designed to make it easier to interact with Connect pipelines. The providers implement no logic other than calling Connect pipelines. All of the business logic is implemented in the pipeline processors.

For each method in the provider there is a corresponding Request and Result object used, ex. GetCarts takes a GetCartsRequest object and returns a GetCartsResult object. In some cases the response objects are re-used when returning the same data.

Customized versions of the default request and result arguments can be used by calling the overloaded generics based versions of the methods.

The Gift Card Provider contains the following methods for interacting with gift card data.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

GetGiftCard

GetGiftCard is used to query the external commerce system to get details for a specific gift card.

Name:	GetGiftCard
Description:	Gets the gift card that matches the specified criteria. Calls the pipeline " commerce.giftCards.getGiftCard"
Usage:	Called when full details of a gift card is needed. Examples include: • Checking gift card balance
Signature:	GetGiftCardResult GetGiftCard([NotNull]GetGiftCardRequest request)
Input:	
	GiftCardId - The ids of the gift card to request
	ShopName – Name of shop to search for the gift card in
Output:	
	GiftCard – A single gift card entity representing the requested card
	SystemMessages - Collection of messages from the external system.

Usage Example:

```
var provider = new GiftCardServiceProvider();
var request = new GetGiftCardRequest("1", "StarterKit");

var result = provider.GetGiftCard(request);
var card = result.GiftCard;
```

2.9.3 Gift Cards Pipelines

The integration and engagement logic used in the Gift Card API is implemented by pipelines that can be customized as needed. There is a pipeline for each method on the API.

GetGiftCard

Name:	GetGiftCard
Description:	The location for a third party ECS to add a processor and make a request for a gift card
Usage:	Called via method GetGiftCard on the Connect API when requesting a specific gift card.
Args:	
	Request - Contains the search criteria: GiftCardId and ShopName. Is set prior to calling the pipeline.
	Response - Contains the giftcard object. Is read after the pipeline is executed.
Processors:	This pipeline has no default processors, an ECS processor is required here.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

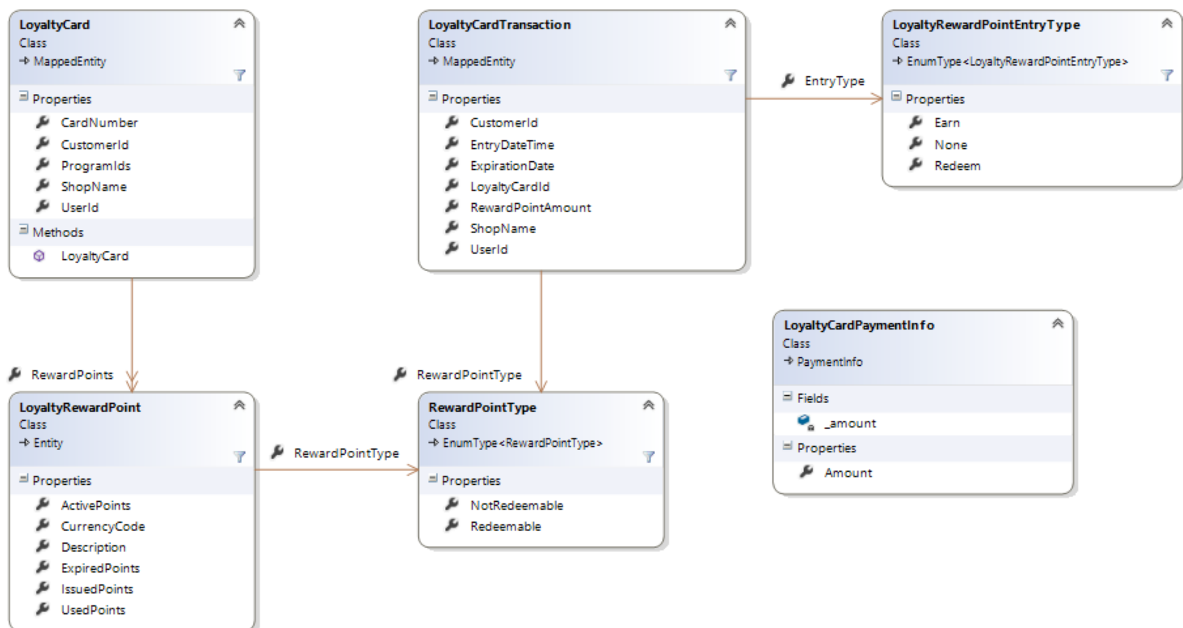
2.10 Loyalty Programs & Cards

2.10.1 The Loyalty Programs & Cards Domain Model

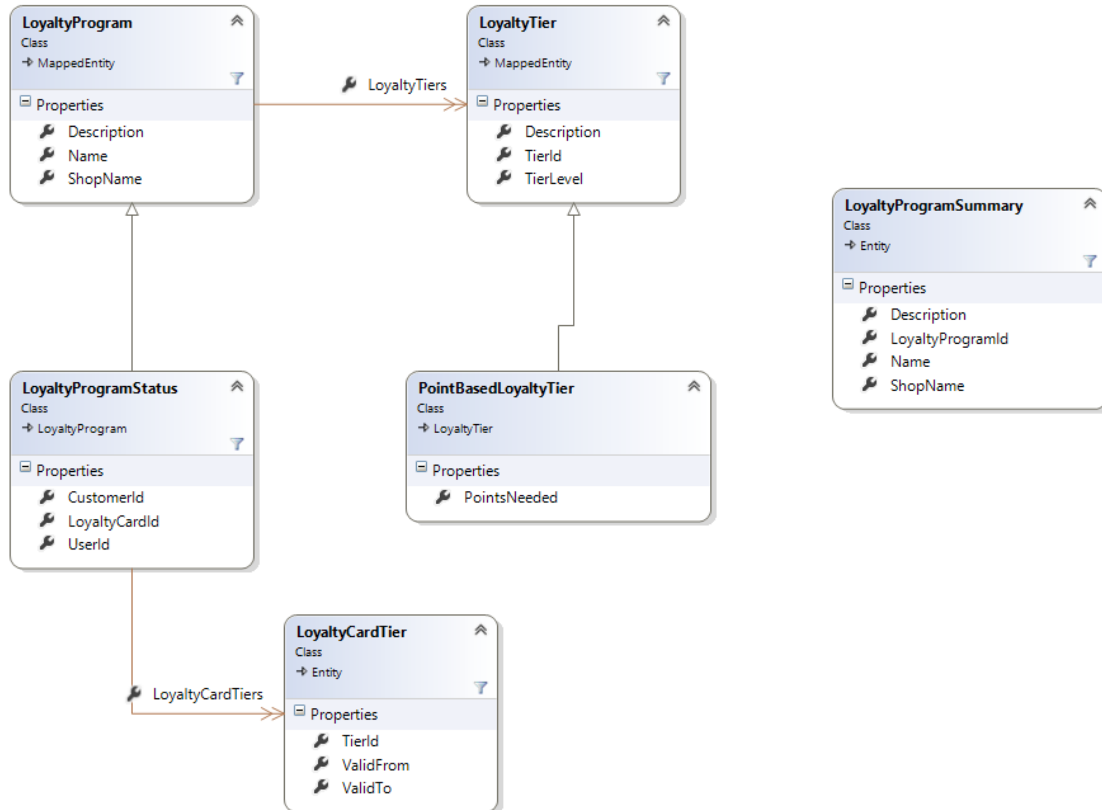
This chapter describes the domain model that represents the loyalty program and card objects where the following three assertions holds true:

- The domain model primarily serves as DTO objects for transferring information between the external commerce system and Sitecore.
- The domain model contains the minimum required information as contracts and will typically be extended, when integrated with a specific commerce system.
- The domain model is used when a Sitecore developer needs to synchronize product data with the external commerce system or product data is pushed into Sitecore from the external commerce system

The class diagram below shows the domain model.



Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.



Note: The domain model consists of abstract classes that make up the contracts with the external system. The contracts are defined as abstract classes instead of interfaces to allow the model to be easily extended later if needed. This follow the best practice guidelines defined in the book Framework Design Guidelines.

Default implementation of the contracts are delivered as part of Connect. If an actual Connect provider with an external commerce system contains more functionality that provided by default, the implementation can be replaced. All instantiation of actual classes will be handled through dependency injection.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Class: LoyaltyCard

The LoyaltyCard object is used to represent a loyalty card owned by a customer

Name	Type	Description
ExternalId	string	Unique identifier for the loyalty card in the external commerce system
CustomerId	string	The customerid associated to the card
UserId	string	The user id associated to the card
ShopName	string	The shop name the card is associated to
CardNumber	string	The number of the card
ProgramIds	ReadOnlyCollection<string>	The ids of any programs the card is associated wuth
RewardPoints	ReadOnlyCollection<LoyaltyRewardPoint>	Any reward points om the card.

Class: LoyaltyCardPaymentInfo

The LoyaltyCardPaymentInfo extends the payment info class and makes it possible for a customer to pay for an order using their loyalty card

Name	Type	Description
Amount	decimal	The amount of points to charge to the card

Class: LoyaltyCardTier

The LoyaltyCardTier object is used to represent a tier that a loyalty card migh be involved in e.g. bronze, silver, gold.

Name	Type	Description
Tier	string	The tier the card is associated with
ValidFrom	dateTime	When the tier becomes valid
ValidTo	dateTime	When the card is no longer elidgible to be part of the tier

Class: LoyaltyCardTransaction

The LoyaltyCardTransaction object is used to represent a record of the addition or removal of points on to a loyalty card

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Name	Type	Description
LoyaltyCardId	string	Unique identifier for the loyalty card in the external commerce system
CustomerId	string	The customerid associated to the card
UserId	string	The user id associated to the card
ShopName	string	The shop name the card is associated to
RewardPointType	RewardPointType	The type of point transaction that occured
RewardPointAmount	decimal	The amount of points that were awarded
EntryType	LoyaltyRewardPointEntryType	The type of entry for this transaction
ExpirationDate	Datetime	The date the reward expires
EntryDateTime	DateTime	The date the transaction occurred.

Class: LoyaltyProgram

The LoyaltyProgram object is used to represent a a program that a loyalty card is involved in

Name	Type	Description
ExternalId	string	Unique identifier for the loyalty program in the external commerce system
Name	string	The name of the program
Description	string	The name of the description
ShopName	string	The name of the shop that the program is associated with
LoyaltyTiers	ReadOnlyCollection<LoyaltyTier>	The tiers that exist within the program

Class: LoyaltyProgramStatus

The LoyaltyProgramStatus object extends from the LoyaltyProgram object to represent the card tiers a user is part of.

Name	Type	Description
CustomerId	string	The id of the customer that owns the card.
UserId	string	The id of the user that owns the card.
LoyaltyCardId	string	The id of the loyalty card

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

LoyaltyCardTiers	ReadOnlyCollection<LoyaltyCardTier>	The tiers available to this loyalty card.

Class: LoyaltyProgramSummary

The LoyaltyProgramSummary object is used to represent a summary of the loyalty program.

Name	Type	Description
Name	string	The name of the program
Description	string	A description of the program
ShopName	string	The name of the shop that the program is associated with
LoyaltyProgramId	string	The id of the loyalty program for this summary

Class: LoyaltyRewardPoint

The LoyaltyRewardPoint object is used to represent point activity on a card

Name	Type	Description
Description	string	A description of the reward points
RewardPointType	RewardPointType	The type of points rewarded
CurrencyCode	String	The currency of the points rewarded
IssuedPoints	Decimal	The amount of point issued
UsedPoints	Decimal	The amount of points used
ExpiredPoints	Decimal	The amount of expired points
ActivePoints	Decimal	The amount of active points

Class: LoyaltyRewardPointEntryType

The LoyaltyRewardPointEntryType object is an enum of different reward point entry types

Name	Type	Description
None		Unassigned reward point entry type
Earn		An earned reward point
Redeem		A redeemed reward point

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Class: LoyaltyTier

The LoyaltyTier object is used to represent a loyalty tier within a program

Name	Type	Description
Description	string	A description of the tier
TierId	String	The id of the tier
TierLevel	Decimal	The level of the tier

Class: PointBasedLoyaltyTier

The PointBasedLoyaltyTier object is an extension of LoyaltyTier and is used to points based loyalty tier within a program

Name	Type	Description
PointsNeeded	decimal	The amount of points required to meet the tier

Class: RewardPointType

The RewardPointType object is an enum of different reward point entry types

Name	Type	Description
NotRedeemable		The points are not redeemable
Redeemable		The points are redeemable

2.10.2 Loyalty Programs & Cards Service Provider

Service providers are wrapper objects designed to make it easier to interact with Connect pipelines. The providers implement no logic other than calling Connect pipelines. All of the business logic is implemented in the pipeline processors.

For each method in the provider there is a corresponding Request and Result object used, ex. GetCarts takes a GetCartsRequest object and returns a GetCartsResult object. In some cases the response objects are re-used when returning the same data.

Customized versions of the default request and result arguments can be used by calling the overloaded generics based versions of the methods.

The Cart Service Provider contains the following methods for interacting with cart data.

GetLoyaltyPrograms

GetLoyaltyPrograms is used to query for a lists of loyalty programs against the external commerce system.

Name:	**GetLoyaltyPrograms**
Description:	Gets the loyalty programs that match the specified criteria. Calls the pipeline "GetLoyaltyPrograms"

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Usage:	Called when a list of loyalty programs is needed. Examples include: • Getting the programs for a shop • Getting the programs for a user at a particular shop
Signature:	GetLoyaltyProgramsResult GetLoyaltyPrograms([NotNull]GetLoyaltyProgramsRequest request)
Input:	
	UserId – Optional - The ids of the users whose associated programs you need returned
	ShopName – Name of shop to search for carts in
Output:	
	ReadOnlyCollection<LoyaltyProgramSummary> – A collection of LoyaltyProgramSummary objects The lists represent the programs that match the criteria specified in the request.
	SystemMessages - Collection of messages from the external system.

Usage Example:

```
var loyaltyProgramService = new LoyaltyProgramServiceProvider();

var request = new GetLoyaltyProgramsRequest("myShop", "John");
var result = loyaltyProgramService.GetLoyaltyPrograms(request);
```

GetLoyaltyProgram

GetLoyaltyProgram is used to query for a specific loyal program against the external commerce system.

Name:	**GetLoyaltyProgram**
Description:	Gets the loyalty program that match the specified criteria. Calls the pipeline "GetLoyaltyProgram"
Usage:	Called when a specific loyalty program is needed. Examples include: • Getting the program for a specific shop and program id • Getting the program for a specific shop, program id, and user
Signature:	GetLoyaltyProgramResult GetLoyaltyProgram([NotNull]GetLoyaltyProgramRequest request)
Input:	
	UserId - Optional - The ids of the users whose carts should be retrieved. If no value is specified, the user IDs are not considered when retrieving carts. **ProgramId –** The id of the program you need returned
	ShopName – Optional. Name of shop to search for carts in

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Output:	
	LoyaltyProgram – The requested loyalty program The object represents the program that matches the criteria specified in the request.
	SystemMessages - Collection of messages from the external system.

Usage Example:

```
var loyaltyProgramService = new LoyaltyProgramServiceProvider();

var request = new GetLoyaltyProgramRequest("myShop", "John");
var result = loyaltyProgramService.GetLoyaltyProgram(request);
```

GetLoyaltyProgramStatus

GetCarts is used to query Cart data against the external commerce system and **doesn't** return a collection of Carts, but a collection of CartBase objects that only contains the summary of the main cart data.

Name:	**GetLoyaltyProgramStatus**
Description:	Gets the status of a loyalty card against all or a specific program. Calls the pipeline "GetLoyaltyProgramStatus"
Usage:	Called when loyalty card program status is needed. Examples include: • Getting the status of a loyalty card against all programs • Getting the status of a loyalty card against a specific programs
Signature:	GetLoyaltyProgramStatusResult GetLoyaltyProgramStatus([NotNull]GetLoyaltyProgramStatusRequest request)
Input:	
	LoyaltyCard – The loyalty card to check for status against
	ProgramIds – Optional – An optional list of program ids to check for status against
Output:	
	ReadOnlyCollection<LoyaltyProgramStatus> – A collection of statuses against specific programs
	SystemMessages - Collection of messages from the external system.

Usage Example:

```
var loyaltyProgramService = new LoyaltyProgramServiceProvider();

var loyaltyCard = new LoyaltyCard
{
    CardNumber = "1234567890",
    CustomerId = "John",
    ProgramIds = new ReadOnlyCollection<string>(new List<string> {"ProgamId1"})
};
```

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

```
        var request = new GetLoyaltyProgramStatusRequest(loyaltyCard);
        var result = loyaltyProgramService.GetLoyaltyProgramStatus(request);
```

JoinLoyaltyProgram

JoinLoyaltyProgram is used to add a user to a loyalty program in the external commerce system.

Name:	**JoinLoyaltyProgram**
Description:	Joins a user to the loyalty program in the specified criteria. Calls the pipeline "JoinLoyaltyProgram"
Usage:	Called when a user needs to join a loyalty program. Examples include: • Join a user to all loyalty programs in a store • Join a user to a specific loyalty program in a store
Signature:	JoinLoyaltyProgramResult JoinLoyaltyProgram([NotNull]JoinLoyaltyProgramRequest request)
Input:	
	UserId - The id of the user who needs to join the program
	ShopName – The shop the user is registering for the program in
	ProgramId – Optional. The program to add the user to
Output:	
	LoyaltyCard – The loyalty card created in the program
	SystemMessages - Collection of messages from the external system.

Usage Example:

```
        var loyaltyProgramService = new LoyaltyProgramServiceProvider();

        var request = new JoinLoyaltyProgramRequest("Rob", "webShop");
        var result = loyaltyProgramService.JoinLoyaltyProgram(request);
        var loyaltyCard = result.LoyaltyCard;
```

GetLoyaltyCards

GetLoyaltyCards is used to return a collection of the loyalty cards.

Name:	**GetLoyaltyCards**
Description:	Gets the loyalty cards that match the specified criteria. Calls the pipeline "GetLoyaltyCards"
Usage:	Called when a list (or one) of loyalty card is needed. Examples include: • Get all of the cards for a user at a store • Get a specific card for a user at a store

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Signature:	GetLoyaltyCardsResult GetLoyaltyCards([NotNull]GetLoyaltyCardsRequest request)
Input:	
	UserId- The id of the user whose card(s) should be retrieved.
	ShopName - Name of shop to search for cards in
	CardId – Optional - The id of a card if you are looking for specific one
Output:	
	ReadOnlyCollection<LoyaltyCard> – The cards found that match the specified criteria
	SystemMessages - Collection of messages from the external system.

Usage Example:

```
var loyaltyProgramService = new LoyaltyProgramServiceProvider();

var request = new GetLoyaltyCardsRequest("Rob", "WebShop");
var result = loyaltyProgramService.GetLoyaltyCards(request);
```

GetLoyaltyCardTransactions

GetLoyaltyCardTransactions returns a list of transaction for a specific card.

Name:	**GetLoyaltyCardTransactions**
Description:	Gets the list of transactions that match the specified criteria. Calls the pipeline "GetLoyaltyCardTransactions"
Usage:	Called when a list of transactions for a card is needed.
Signature:	GetLoyaltyCardTransactionsResult GetLoyaltyCardTransactions([NotNull]GetLoyaltyCardTransactionsRequest request)
Input:	
	LoyaltyCard – The loyalty card to retrieve transactions for
Output:	
	ReadOnlyCollection<LoyaltyCardTransaction> – A collection of transactions for the card
	SystemMessages - Collection of messages from the external system.

Usage Example:

```
var loyaltyProgramService = new LoyaltyProgramServiceProvider();

var loyaltyCard = new LoyaltyCard
{
    CardNumber = "1234567890",
    CustomerId = "Mike",
    ProgramIds = new ReadOnlyCollection<string>(new List<string> {"ProgamId1"})
};
```

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

```
var request = new GetLoyaltyCardTransactionsRequest(loyaltyCard);
var result = loyaltyProgramService.GetLoyaltyCardTransactions(request);
```

2.10.3 Loyalty Programs & Cards Pipelines

The integration and engagement logic used in the Loyalty Program and Card API is implemented by pipelines that can be customized as needed. There is a pipeline for each method on the API.

GetLoyaltyPrograms

Name:	**GetLoyaltyPrograms**
Description:	Gets a collection of loyalty programs for a shop or user in a shop
Usage:	Called via method GetLoyaltyPrograms on the Connect API when searching for loyalty programs.
Args:	
	Request - Contains the search criteria: ShopName, UserId. Is set prior to calling the pipeline.
	Response - Contains the collection of LoyaltyProgramSummary objects. Is read after the pipeline is executed.
Processors:	No default processors.

GetLoyaltyProgram

Name:	**GetLoyaltyProgram**
Description:	Gets a specific loyalty program for a shop or user in a shop.
Usage:	Called via method GetLoyaltyProgram on the Connect API when searching for loyalty programs.
Args:	
	Request - Contains the search criteria: UserID, ShopName, and ProgramID. Is set prior to calling the pipeline.
	Response - Contains the LoyaltyProgram object. Is read after the pipeline is executed.
Processors:	No default processors.

GetLoyaltyProgramStatus

Name:	**GetLoyaltyProgramStatus**
Description:	Gets the statuses for a loyalty card or specific programs that card is associated to.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Usage:	Called via method GetLoyaltyProgramStatus on the Connect API when searching for program statuses.
Args:	
	Request - Contains the search criteria: LoyaltyCard object and a list of programIds. Is set prior to calling the pipeline.
	Response – A collection of LoyaltyProgramStatus objects. Is read after the pipeline is executed.
Processors:	No default processors.

JoinLoyaltyProgram

Name:	**JoinLoyaltyProgram**
Description:	Joins a user to all or a specific program in a store
Usage:	Called via method JoinLoyaltyProgram on the Connect API when when trying to join a user to a program(s) .
Args:	
	Request - Contains the search criteria: userId, shopName, and programId. Is set prior to calling the pipeline.
	Response – A LoyaltyCard object. Is read after the pipeline is executed.
Processors:	
	TriggerLoyaltyProgramJoinedGoal – Triggers the “Loyalty Program Joined” goal **AddVisitorToEaPlan** – Adds the user to a EAP for joining a loyalty program

GetLoyaltyCards

Name:	**GetLoyaltyCards**
Description:	Gets the loyalty cards for the user in a store or a specific card.
Usage:	Called via method GetLoyaltyCards on the Connect API when trying to retrieve a card.
Args:	
	Request - Contains the search criteria: UserID, ShopName, and CardId. Is set prior to calling the pipeline.
	Response - Contains the collection of LoyaltyCard objects. Is read after the pipeline is executed
Processors:	No default processors.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

GetLoyaltyCardTransactions

Name:	**GetLoyaltyCardTransactions**
Description:	Gets all of the transactions for a specific card.
Usage:	Called via method GetLoyaltyCardTransactions on the Connect API when retrieving for statuses.
Args:	
	Request - Contains the search criteria: LoyaltyCard. Is set prior to calling the pipeline.
	Response - Contains the collection of LoyaltyCardTransaction objects. Is read after the pipeline is executed.
Processors:	No default processors.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

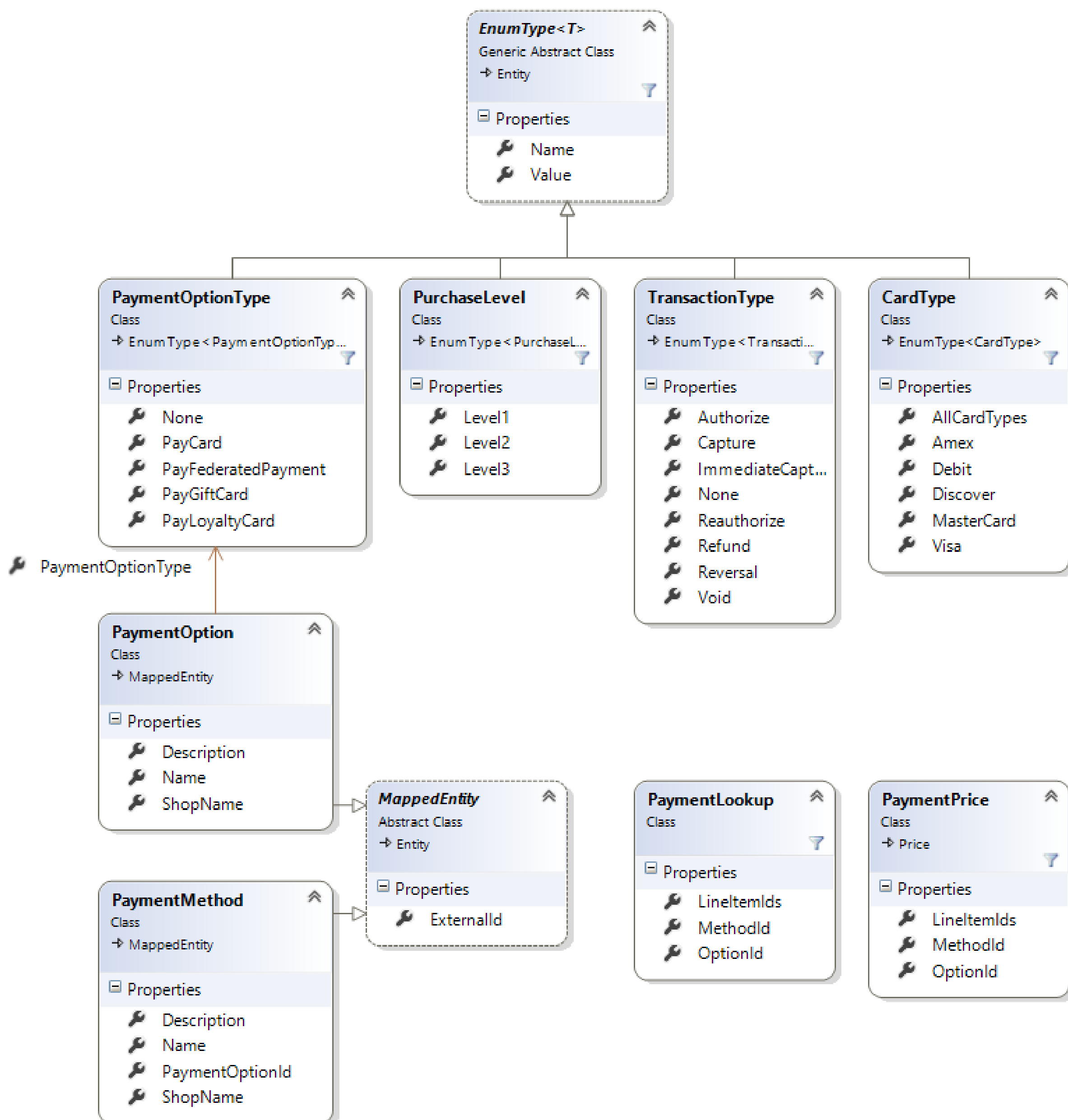
2.11 Payments

2.11.1 Payments Domain Model

This chapter describes the domain model that represents the payment information objects where the following three assertions holds true:

- The domain model primarily serves as DTO objects for transferring information between the external commerce system and Sitecore.
- The domain model contains the minimum required information as contracts and will typically be extended, when integrated with a specific commerce system.
- The domain model is used when a Sitecore developer needs to synchronize product data with the external commerce system or product data is pushed into Sitecore from the external commerce system

The class diagram below shows the domain model.



Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Note: The domain model consists of abstract classes that make up the contracts with the external system. The contracts are defined as abstract classes instead of interfaces to allow the model to be easily extended later if needed. This follow the best practice guidelines defined in the book Framework Design Guidelines.

Default implementation of the contracts are delivered as part of Connect. If an actual Connect provider with an external commerce system contains more functionality that provided by default, the implementation can be replaced. All instantiation of actual classes will be handled through dependency injection.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Class: PaymentMethod

The PaymentMethod class is responsible for representing a type of payment option e.g. Visa would be of type PayCard option.

Name	Type	Description
ExternalId	string	Unique identifier for the payment method in the commerce system.
Description	string	A description of the payment method.
Name	string	The name of the payment method.
PaymentOptionId	string	The payment option associated with this payment method.
ShopName	string	The shop name if any associated with this payment method.

Class: PaymentOption

The PaymentOption class is responsible for representing different payment option types e.g.Visa.

Name	Type	Description
ExternalId	string	Unique identifier for the payment option in the commerce system.
Description	string	A description of the payment option
Name	string	Name of the payment option
ShopName	string	The shop name, if any, associated to the payment option

Class: PaymentOptionType

The PaymentOptionType is a virtual enum class is responsible for representing known payment option types e.g. gift card, credit card, etc.

Name	Type	Description
None		An unassigned payment option
PayCard		Credit cards, debit cards, etc
PayGiftCard		Represents gift cards
PayLoyaltyCard		Represents loyalty cards

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Class: PaymentLookup

The PaymentLookup is used to hold the information required to lookup payment prices for a collection of line items.

Name	Type	Description
OptionId	String	An unassigned payment option
MethodId	string	Credit cards, debit cards, etc
LineItemIds	ReadOnlyCollection<string>	The line items to check the pricing for

Class: PaymentPrice

The PaymentPrice inherits from the Price object and is used to hold the pricing information for a payment option, method, and list of line items..

Name	Type	Description
OptionId	string	An unassigned payment option
MethodId	string	Credit cards, debit cards, etc
LineItemIds	ReadOnlyCollection<string>	The list of line items that the pricing applies to.

Class: CardType

The CardType is a virtual enum class is responsible for representing the type of payment cards that can be accepted when processing a payment.

Name	Type	Description
Visa		The Visa card type.
MasterCard		The Master Card card type.
Discover		The Discover card type.
Debit		The debit card type.
AllCardTypes		Represents all supported card types.

Class: PurchaseLevel

The PurchaseLevel is a virtual enum class that represents the level of card processing that will be performed during a payment service transaction.

Name	Type	Description
Level1		Represents level 1 card processing.
Level2		Represents level 2 card processing.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Level3		Represents level 3 card processing.

Class: TransactionType

The TransactionType is a virtual enum class that represents the type of transaction being performed with a payment service.

Name	Type	Description
None		No transaction is performed.
Authorize		Authorize a payment.
Reauthorize		Re-authorize a payment.
Capture		Capture a payment.
ImmediateCapture		Perform a capture with no delay.
Void		Void a payment.
Reversal		Reverse a payment.
Refund		Refund a payment.

2.11.2 Payments Service Provider

Service providers are wrapper objects designed to make it easier to interact with Connect pipelines. The providers implement no logic other than calling Connect pipelines. All of the business logic is implemented in the pipeline processors.

For each method in the provider there is a corresponding Request and Result object used, ex. GetCarts takes a GetCartsRequest object and returns a GetCartsResult object. In some cases the response objects are re-used when returning the same data.

Customized versions of the default request and result arguments can be used by calling the overloaded generics based versions of the methods.

The Payments Service Provider contains the following methods for interacting with payment data.

GetPaymentOptions

GetPaymentOptions is used to query payment option data against the external commerce system and returns a collection of payment options.

Name:	**GetPaymentOptions**
Description:	Gets the carts that match the specified criteria. Calls the pipeline "GetPaymentOptions"
Usage:	Called when a list of payment options is needed. Examples include: • Getting the list if payment options for a shop
Signature:	GetPaymentOptionsResult GetPaymentOptions([NotNull]GetPaymentOptionsRequest request)
Input:	

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

	ShopName – Optional. Name of shop to retrieve payment options for
Output:	
	IEnumerable<PaymentOption> – A collection of payment options for the requested store.
	SystemMessages - Collection of messages from the external system.

Usage Example:

```
var paymentService = new PaymentServiceProvider();

var request = new GetPaymentOptionsRequest("WebShop");
var result = paymentService.GetPaymentOptions(request);
```

GetPaymentMethods

GetPaymentMethods is used to query payment data against the external commerce system to retrieve all of the payment methods for a particular payment option.

Name:	**GetPaymentMethods**
Description:	Gets the payment methods that match the specified criteria. Calls the pipeline "GetPaymentMethods"
Usage:	Called when a list of payment methods is needed. Examples include: • Getting the payment methods for a particular payment option type
Signature:	GetPaymentMethodsResult GetPaymentMethods([NotNull]GetPaymentMethodsRequest request)
Input:	
	PaymentOption – The payment option to retrieve payment methods for. **ShopName - optional** – The name of the
Output:	
	IEnumerable<PaymentMethod> – A collection of payment methods for the requested option.
	SystemMessages - Collection of messages from the external system.

Usage Example:

```
var paymentService = new PaymentServiceProvider();
var paymentOption = new PaymentOption
{
    Name = "Visa",
    PaymentOptionType = new PaymentOptionType(1, "CreditCard"),
    ShopName = "webShop"
};

var request = new GetPaymentMethodsRequest(paymentOption);
var result = paymentService.GetPaymentMethods(request);
```

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

GetPricesForPayments

GetPricesForPayments is used to get a list of possible payment prices for all items in a cart.

Name:	GetPricesForPayments
Description:	Gets the payment costs that match the specified criteria. Calls the pipeline "GetPricesForPayments"
Usage:	Called when a list of payment costs is needed.
Signature:	GetPricesForPaymentsResult GetPricesForPayments(GetPricesForPaymentsRequest request)
Input:	
	ShopName - Manditory – The name of the current shop.
	PaymentLookup – Manditory – The combination of payment method and options to lookup
	Cart – Manditory – The cart containing the line items to look up
Output:	
	ReadOnlyCollection<PaymentPrice> – A collection of payment prices mapped to items in the cart
	SystemMessages - Collection of messages from the external system.

Usage Example:

```
        var paymentLookupList = new List<PaymentLookup>
        {
            new PaymentLookup
            {
                OptionId = "{B83E4AA3-5F98-4FEE-95C0-BDE9CE572E7C}",
                LineItemIds = new List<string> {"1", "2"}.AsReadOnly()
            },
            new PaymentLookup
            {
                OptionId = "{B83E4AA3-5F98-4FEE-95C0-BDE9CE572E7C}",
                MethodId = "{8BC8E1DA-4FA8-4D20-AB53-53C104CCC2CC}"
            }
        };

        var cartProvider = (CartServiceProvider)Factory.CreateObject("cartServiceProvider",
true);
        var cartRequest = new LoadCartRequest("StarterKit", "cartid");
        var cartResult = cartProvider.LoadCart(cartRequest);
        var cart = cartResult.Cart;

        var provider = (PaymentServiceProvider)Factory.CreateObject("paymentServiceProvider",
true);
        var request = new GetPricesForPaymentsRequest("StarterKit", paymentLookupList, cart);
        var result = provider.GetPricesForPayments(request);
        if (result.Success && result.PaymentPrices != null)
        {
            foreach (var paymentPrice in result.PaymentPrices)
            {
                // handle payment price.
            }
        }
```

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

GetPaymentServiceUrl

GetPaymentServiceUrl is used to initialize a transaction with a payment service and retrieve the URL of payment acceptance page of the payment service.

Name:	**GetPaymentServiceUrl**
Description:	Retrieves the URL to the payment acceptance page of a federated payment service.
Usage:	Called when a website is ready to retrieve payment information from a customer. Examples include: • When rendering the payment acceptance page during a checkout.
Signature:	GetPaymentServiceUrlResult GetPaymentServiceUrl([NotNull] GetPaymentServiceUrlRequest request)
Input:	
	HostPageOrigin – Required. Specifies root URL of the merchant website page that will host the payment acceptance page.
	AllowPartialAuthorization – Optional. Specifies whether partial authorizations are allowed in the transaction.
	AllowVoiceAuthorization – Optional. Specifies whether voice authorization is allowed in the transaction.
	CardType – Optional. Specifies the type of payment cards that can be accepted in the transaction.
	City – Optional. Specifies the city of the billing or shipping address.
	ColumnNumber – Optional. Specifies the number of columns the payment acceptance page will use when displaying input fields.
	Country – Optional. Specifies the country of the billing or shipping address.
	CurrencyCode – Optional. Specifies the the currency that will be used in the transaction. The format of the currency code is specific to the payment service provider being used.
	DisabledTextBackgroundColor - Optional. Specifies the background color of the disabled text on the payment acceptance page.
	FontFamily – Optional. Specifies the font family of the text on the payment acceptance page.
	FontSize – Optional. Specifies the size of the text on the payment acceptance page.
	IndustryType – Optional. Specifies the type of industry for the transaction.
	LabelColor – Optional. Specifies the color of label text on the payment acceptance page.
	Locale – Optional. Specifies the region / locale / language of the payment acceptance page.
	PageBackgroundColor – Optional. Specifies the background color of the payment acceptance page.
	PageWidth – Optional. Specifies the width of the payment acceptance page.
	PostalCode – Optional. Specifies the postal code of the billing or shipping address.
	PurchaseLevel – Optional. Specifies the purchase level of the transaction being performed.
	ShowSameAsShippingAddress – Optional. Specifies whether or not to display a checkbox that indicates the billing and shipping addresses are the same.
	State – Optional. Specifies the state of the billing or shipping address.
	StreetAddress – Optional. Specifies the street address of the billing or shipping address.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

	SupportCardSwipe – Optional. Specifies whether a card swipe is supported to complete the transaction.
	SupportCardTokenization – Optional. Specifies whether the card information input during the transaction will be tokenized.
	TextBackgroundColor – Optional. Specifies the background of text on the payment acceptance page.
	TextColor – Optional. Specifies the color of text on the payment acceptance page.
	TransactionType – Optional. Specifies the type of transaction being performed.
Output:	
	Url – The URL of the payment acceptance page that will perform a transaction with the provided options.
	SystemMessages - Collection of messages from the external system.

Usage Example:

```
        var request = new GetPaymentServiceUrlRequest()
        {
            AllowPartialAuthorization = false,
            AllowVoiceAuthorization = false,
            CardType = CardType.AllCardTypes,
            ColumnNumber = 1,
            CurrencyCode = "USD",
            DisabledTextBackgroundColor = "#E4E4E4",
            FontFamily = "\"Helvetica Neue\", Helvetica, Arial, sans-serif",
            FontSize = "14px",
            IndustryType = "Ecommerce",
            LabelColor = "black",
            Locale = Context.Culture.Name,
            PageBackgroundColor = "white",
            PageWidth = "429px",
            PurchaseLevel = PurchaseLevel.Level1,
            ShowSameAsShippingAddress = false,
            SupportCardSwipe = false,
            SupportCardTokenization = true,
            TextBackgroundColor = "white",
            TextColor = "black",
            TransactionType = TransactionType.Authorize,
        };

        if (HttpContext.Current != null && HttpContext.Current.Request != null)
        {
            request.HostPageOrigin = string.Format(
                System.Globalization.CultureInfo.InvariantCulture,
                "{0}://{1}",
                HttpContext.Current.Request.Url.Scheme,
                HttpContext.Current.Request.Url.Authority);
        }

        var provider = (PaymentServiceProvider)Factory.CreateObject("paymentServiceProvider",
true);
        var result = provider.GetPaymentServiceUrl(request);
```

GetPaymentServiceActionResult

GetPaymentServiceActionResult is used retrieve the result of a transaction or action against a payment service.

Name:	**GetPaymentServiceActionResult**

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Description:	Retrieves the result of a federated payment service transaction / action.
Usage:	Called when a website is ready to retrieve the result of a transaction. Examples include: • Retrieving the result of a customer's transaction with a federated payment service.
Signature:	GetPaymentServiceActionResultResult GetPaymentServiceActionResult([NotNull] GetPaymentServiceActionResultRequest request)
Input:	
	Locale – Required. Specifies the locale of the merchant website.
	PaymentAcceptResultAccessCode – Required. Specifies the result access code from the federated payment service. This code is typically sent to the merchant website through a cross page JavaScript event.
Output:	
	CardToken – A card token that repsents the payment information that the customer provided to the payment service.
	AuthorizationResult – A value that identifies the result of the transaction with the payment service. The format of this string is specific to the payment service being used.
	SystemMessages - Collection of messages from the external system.

Usage Example:

```
        var request = new GetPaymentServiceActionResultRequest()
        {
            Locale = Context.Culture.Name,
            PaymentAcceptResultAccessCode = "accessCode"
        };

        var provider = (PaymentServiceProvider) Factory.CreateObject("paymentServiceProvider",
true);
        var result = provider.GetPaymentServiceActionResult(request);
```

2.11.3 Payments Pipelines

The integration and engagement logic used in the Payments API is implemented by pipelines that can be customized as needed. There is a pipeline for each method on the API.

GetPaymentOptions

Name:	**GetPaymentOptions**
Description:	The pipeline is responsible for performing a search against all payment options in the external commerce system.
Usage:	Called via method GetPaymentOptions on the Connect API when searching for payment options.
Args:	
	Request - Contains the search criteria: ShopName. Is set prior to calling the pipeline.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

	Response - Contains a collection of paymentOption objects. Is read after the pipeline is executed.
Processors:	**No default processors**

GetPaymentMethods

Name:	**GetPaymentMethods**
Description:	The pipeline is responsible for performing a search against all payment methods in the external commerce system.
Usage:	Called via method GetPaymentMethods on the Connect API when searching for payment methods.
Args:	
	Request - Contains the search criteria: PaymentOption. Is set prior to calling the pipeline.
	Response - Contains a collection of PaymentMethod objects. Is read after the pipeline is executed.
Processors:	**No default processors**

GetPricesForPayments

Name:	**GetPricesForPayments**
Description:	The pipeline is responsible for getting the payment pricing for a cart and a select group of payment and methods and options from the external commerce system.
Usage:	Called via method GetPricesForPayments on the Connect API when searching for payment methods.
Args:	
	Request - Contains the search criteria: ShopName, PaymentLookups, and Cart. Is set prior to calling the pipeline.
	Response - Contains a list of PaymentPrice objects. Is read after the pipeline is executed.
Processors:	**No default processors**

GetPaymentServiceUrl

Name:	**GetPaymentServiceUrl**

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Description:	The pipeline is responsible for retrieveing the URL for the card acceptance page of a federated payment service.
Usage:	Called via method GetPaymentServiceUrl on the Connect API when performing checkout.
Args:	
	Request – Contains the merchant and customer information the payment service requires.
	Response - Contains the URL for the card acceptance page.
Processors:	**No default processors**

GetPaymentServiceActionResult

Name:	**GetPaymentServiceActionResult**
Description:	The pipeline is responsible for retrieveing the result of a transaction with a payment service.
Usage:	Called via method GetPaymentServiceActionResult on the Connect API when performing checkout.
Args:	
	Request – Contains the result access code that uniquely identifies the transaction.
	Response - Contains the result of the transaction as well as the card token that identifies the customer payment information.
Processors:	**No default processors**

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

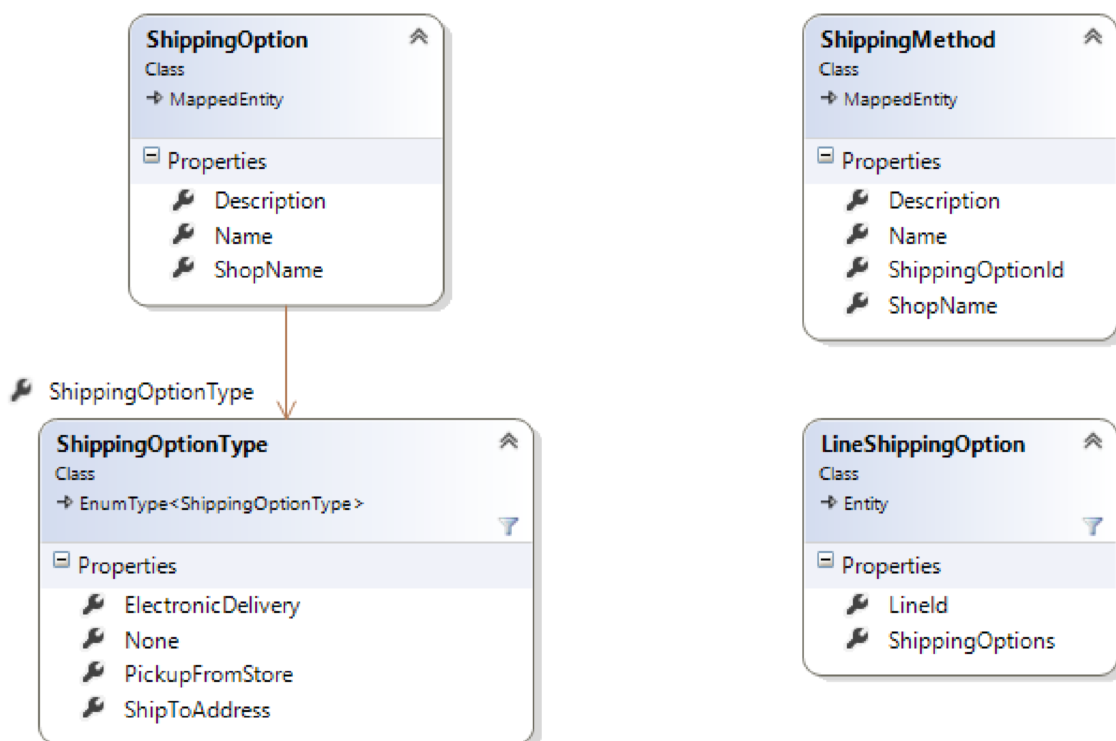
2.12 Shipping

2.12.1 Shipping Domain Model

This chapter describes the domain model that represents shipping objects where the following three assertions holds true:

- The domain model primarily serves as DTO objects for transferring information between the external commerce system and Sitecore.
- The domain model contains the minimum required information as contracts and will typically be extended, when integrated with a specific commerce system.
- The domain model is used when a Sitecore developer needs retrieve shipping data with the external commerce system.

The class diagram below shows the domain model.



Note: The domain model consists of abstract classes that make up the contracts with the external system. The contracts are defined as abstract classes instead of interfaces to allow the model to be easily extended later if needed. This follow the best practice guidelines defined in the book Framework Design Guidelines.

Default implementation of the contracts are delivered as part of Connect. If an actual Connect provider with an external commerce system contains more functionality that provided by default, the implementation can be replaced. All instantiation of actual classes will be handled through dependency injection.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Class: ShippingMethod

The ShippingMethod class is responsible for representing a type of shipping option e.g. FedEx would be of type ShipToAddress option.

Name	Type	Description
ExternalId	string	Unique identifier for the shipment method in the commerce system.
Description	string	A description of the shipment method.
Name	string	The name of the shipment method.
ShippingOptionId	string	The shipment option associated with this shipment method.
ShopName	string	The shop name if any associated with this shipment method.

Class: ShippingMethodPerItem

Used to return the valid shipping options on a per cart line item basis.

Name	Type	Description
LineId	string	The line item identifier.
ShippingMethods	IReadOnlyCollection<ShippingMethod>	The list of allowable shipping methods for the line item.

Class: ShippingOption

The ShippingOption class is responsible for representing different shipping option types e.g.ShipToAddress.

Name	Type	Description
ExternalId	string	Unique identifier for the shipping option in the commerce system.
Description	string	A description of the shipping option
Name	string	Name of the shipping option
ShopName	string	The shop name, if any, associated to the shipping option

Class: ShippingOptionType

The ShippingOptionType is a virtual enum class is responsible for representing known shipping option types e.g. pick up from store, electronic delivery, etc.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Name	Type	Description
None		An unassigned shipping option
ElectronicDelivery		Downloadable content
PickupFromStore		Buy on the site and pick from from a specific store
ShipToAddress		Ship to a physical address

Class: LineShippingOption

The LineShippingOption is used to hold shipping information for a particular line item.

Name	Type	Description
LineId	String	The id of the line item this is associated to
ShippingOptions	ReadOnlyCollection<ShippingOption>	The shipping options associated to this line item

Class: ShippingLookup

The ShippingLookup is used to hold the information required to lookup shipping prices for a collection of line items.

Name	Type	Description
OptionId	String	The id of the desired shipping option
MethodId	string	The id of the desired shipping method
LineItemIds	ReadOnlyCollection<string>	The collection of line item ids to look up the shipping costs for

Class: ShippingPrice

The ShippingPrice inherits from the Price object and is used to hold the pricing information for a shipping option, method, and list of line items.

Name	Type	Description
OptionId	String	The id of the shipping option
MethodId	string	The id of the shipping method
LineItemIds	ReadOnlyCollection<string>	The collection of line item ids that the price is for

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

2.12.2 Shipping Service Provider

Service providers are wrapper objects designed to make it easier to interact with Connect pipelines. The providers implement no logic other than calling Connect pipelines. All of the business logic is implemented in the pipeline processors.

For each method in the provider there is a corresponding Request and Result object used, ex. GetCarts takes a GetCartsRequest object and returns a GetCartsResult object. In some cases the response objects are re-used when returning the same data.

Customized versions of the default request and result arguments can be used by calling the overloaded generics based versions of the methods.

The Shipping Service Provider contains the following methods for interacting with payment data.

GetShippingOptions

GetShippingOptions is used to query shipping option data against the external commerce system and returns a collection of shipping options.

Name:	**GetShippingOptions**
Description:	Gets the shipping options for a cart. Calls the pipeline "GetShippingOptions"
Usage:	Called when a list of shipping options is needed. Examples include: • Getting all available shipping options • Getting the shipping options for the contents of a user's cart
Signature:	GetShippingOptionsResult GetShippingOptions(GetShippingOptionsRequest request)
Input:	
	Cart – Optional. Figure out the shipping options available for a specific cart
Output:	
	ReadOnlyCollection<ShippingOptions> – A collection of shipping options for the requested scenario. ReadOnlyCollection<LineShippingOption> - The shipping options available for different line items in the cart.
	SystemMessages - Collection of messages from the external system.

Usage Example:

```
var shippingService = new ShippingServiceProvider();

var request = new GetShippingOptionsRequest();
var result = shippingService.GetShippingOptions(request);
```

GetShippingMethods

GetShippingMethods is used to query shipping data against the external commerce system to retrieve all of the shipping methods for a particular payment option.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Name:	GetShippingtMethods
Description:	Gets the shipping methods that match the specified criteria. Calls the pipeline "GetShippingMethods"
Usage:	Called when a list of shipping methods is needed. Examples include: • Getting the shipping methods for a particular shipping option • Getting the shipping methods for a particular shipping option and party
Signature:	GetShippingMethodsResult GetShippingMethods(GetShippingMethodsRequest request)
Input:	
	ShippingOption – The shipping option to retrieve shipping methods for. **Party – Optional** – Limit the options based on a party
Output:	
	ReadOnlyCollection<ShippingMethod> – A collection of shipping methods for the requested option.
	SystemMessages - Collection of messages from the external system.

Usage Example:

```
var shippingService = new ShippingServiceProvider();

var shippingOption = new ShippingOption
{
    Description = "Super Fast Shipping",
    Name = "SuperFast",
    ShippingOptionType = new ShippingOptionType(1, "Courier"),
    ShopName = "webShop"
};

var request = new GetShippingMethodsRequest(shippingOption);
var result = shippingService.GetShippingMethods(request);
```

GetShippingMethod

GetShippingMethod is used to get the full details for a shipping method from the ECS.

Name:	GetShippingtMethod
Description:	Gets the full details for a shipping method that match the specified criteria. Calls the pipeline "GetShippingMethod"
Usage:	Called when the full details of a shipping method is needed.
Signature:	GetShippingMethodResult GetShippingMethod(GetShippingMethodRequest request)
Input:	
	ShopName – Optional – The name of the current shop. **ExternalId – Manditory** – The id of the shipping method in the ECS
Output:	

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

	ShippingMethod – The requested shipping method. **ShippingMethodPerItem -**
	SystemMessages - Collection of messages from the external system.

Usage Example:

```
        var provider =
(ShippingServiceProvider)Factory.CreateObject("shippingServiceProvider", true);
        var request = new GetShippingMethodRequest("Next Day");
        var result = provider.GetShippingMethod(request);
```

GetPricesForShipments

GetPricesForShipments is used to get a list of possible shipping prices for all items in a cart.

Name:	**GetPricesForShipments**
Description:	Gets the shipping costs that match the specified criteria. Calls the pipeline "GetPricesForShipments"
Usage:	Called when a list of shipping costs is needed.
Signature:	GetPricesForShipmentsResult GetPricesForShipments(GetPricesForShipmentsRequest request)
Input:	
	ShopName - Manditory – The name of the current shop. **ShippingLookup – Manditory** – The combination of shipping method and options to lookup
	Cart – Manditory – The cart containing the line items to look up
Output:	
	ReadOnlyCollection<ShippingPrice> – A collection of shipping prices mapped to items in the cart
	SystemMessages - Collection of messages from the external system.

Usage Example:

```
        var shippingLookupList = new List<ShippingLookup>
        {
            new ShippingLookup
            {
                LineItemIds = new List<string> { "1", "2" }.AsReadOnly()
            }
        };

        var cartProvider = (CartServiceProvider)Factory.CreateObject("cartServiceProvider",
true);
        var cartRequest = new LoadCartRequest("StarterKit", "cartid");
        var cartResult = cartProvider.LoadCart(cartRequest);
        var cart = cartResult.Cart;

        var provider =
(ShippingServiceProvider)Factory.CreateObject("shippingServiceProvider", true);
        var request = new GetPricesForShipmentsRequest("StarterKit", shippingLookupList,
cart);
        var result = provider.GetPricesForShipments(request);
        if (result.Success && result.ShippingPrices != null)
```

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

```
        {
            foreach (var shippingPrice in result.ShippingPrices)
            {
                // handle shipping prices
            }
        }
```

2.12.3 Shipping Pipelines

GetShippingOptions

Name:	GetShippingOptions
Description:	The pipeline is responsible for performing a search against all shipping options in the external commerce system.
Usage:	Called via method GetShippingOptions on the Connect API when searching for shipping options.
Args:	
	Request - Contains the search criteria: Cart. Is set prior to calling the pipeline.
	Response - Contains a collection of ShippingOption and LineShippingOption objects. Is read after the pipeline is executed.
Processors:	**No default processors**

GetShippingMethods

Name:	GetShippingMethods
Description:	The pipeline is responsible for performing a search against all shipping methods in the external commerce system.
Usage:	Called via method GetShippingMethods on the Connect API when searching for shipping methods.
Args:	
	Request - Contains the search criteria: ShippingOption, Party. Is set prior to calling the pipeline.
	Response - Contains a collection of ShippingMethod objects. Is read after the pipeline is executed.
Processors:	**No default processors**

GetShippingMethod

Name:	GetShippingMethod
Description:	The pipeline is responsible for performing a query against the ECS to retrieve full details for a shipping method in the external commerce system.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Usage:	Called via method GetShippingMethod on the Connect API when requesting details for a shipping method.
Args:	
	Request - Contains the search criteria: ShopName, and ExternalId (Shipping method id)
	Response - A ShippingMethod object. Is read after the pipeline is executed.
Processors:	**No default processors**

GetPricesForShipments

Name:	**GetPricesForShipments**
Description:	The pipeline is responsible for getting the shipment pricing for a cart and a select group of shipping and methods and options from the external commerce system.
Usage:	Called via method GetPricesForShipments on the Connect API when searching for shipping methods.
Args:	
	Request - Contains the search criteria: ShopName, ShippingLookups, and Cart. Is set prior to calling the pipeline.
	Response - Contains a list of ShippingPrice objects. Is read after the pipeline is executed.
Processors:	**No default processors**

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

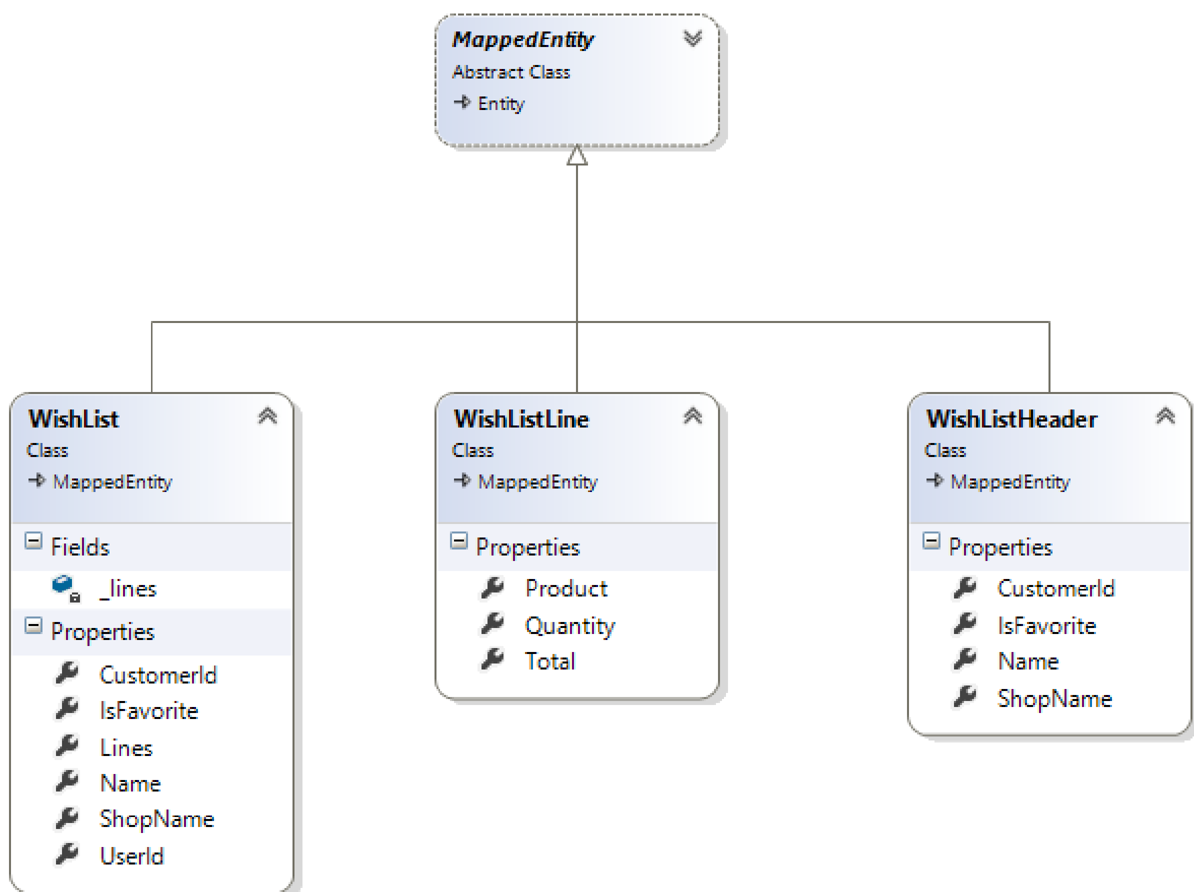
2.13 Wish Lists

2.13.1 Wish Lists Domain Model

This chapter describes the domain model that represents the product objects where the following three assertions holds true:

- The domain model primarily serves as DTO objects for transferring information between the external commerce system and Sitecore.
- The domain model contains the minimum required information as contracts and will typically be extended, when integrated with a specific commerce system.
- The domain model is used when a Sitecore developer needs to synchronize product data with the external commerce system or product data is pushed into Sitecore from the external commerce system

The class diagram below shows the domain model.



Note: The domain model consists of abstract classes that make up the contracts with the external system. The contracts are defined as abstract classes instead of interfaces to allow the model to be easily extended later if needed. This follow the best practice guidelines defined in the book Framework Design Guidelines.

Default implementation of the contracts are delivered as part of Connect. If an actual Connect provider with an external commerce system contains more functionality that provided by default, the

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

implementation can be replaced. All instantiation of actual classes will be handled through dependency injection.

Class: WishList

The WishList class is responsible for representing a wish list or any variant of it.

Name	Type	Description
ExternalId	string	Unique identifier for the wish list in the commerce system.
CustomerId	string	The id of the customer in the external commerce system
IsFavorite	bool	Is this the customer's preferred wish list
Lines	ReadOnlyCollection<WishListLine>	The line items within the wish list
Name	string	The friendly name of the wish list
ShopName	String	The shop the wishlist is associated to
UserId	string	The Sitecore user id

Class: WishListLine

The WishListLine class is responsible for representing a wish list line or any variant of it.

Name	Type	Description
ExternalId	string	Unique identifier for the product in the commerce system. This can be used to get a reference to the product using the commerce system's native API.
Quantity	unint	The quantity of product on the line item
Product	CartProduct	The product in the line item
Total	Total	The total cost of the line item

Class: WishListHeader

The WishListHeader class is responsible for representing a wish list header or any variant of it. The wish list header acts as a summary of the wish list.

Name	Type	Description
ExternalId	string	Unique identifier for the wish list in the commerce system.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

ShopName	string	The shop name associated to the wish list
Name	string	The friendly name of the wish list
CustomerId	string	The id of the customer in the external commerce system
IsFavorite	bool	Is this the user's default wish list

2.13.2 Wish Lists Service Provider

Service providers are wrapper objects designed to make it easier to interact with Connect pipelines. The providers implement no logic other than calling Connect pipelines. All of the business logic is implemented in the pipeline processors.

For each method in the provider there is a corresponding Request and Result object used, ex. GetCarts takes a GetCartsRequest object and returns a GetCartsResult object. In some cases the response objects are re-used when returning the same data.

Customized versions of the default request and result arguments can be used by calling the overloaded generics based versions of the methods.

The Wish List Service Provider contains the following methods for interacting with cart data.

CreateWishList

CreateWishList is used to query create a wish list in the external commerce system and returns the newly created WishList object.

Name:	**CreateWishList**
Description:	Creates a wish list that matches the specified criteria. Calls the pipeline "CreateWishList"
Usage:	Called when creating a new wish list. Examples include: • Creating a wish list for a user with a specific name in a specific store
Signature:	CreateWishListResult CreateWishList([NotNull]CreateWishListRequest request)
Input:	
	UserId – The id of the user to create the cart for
	WishListName – The friendly name to assign to the wish list.
	ShopName – The name of the shop the wish list is being created in
Output:	
	WishList – The newly created wish list
	SystemMessages - Collection of messages from the external system.

Usage Example:

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

```
        var wishListService = new WishListServiceProvider();

        var request = new CreateWishListRequest("Bob", "Bobs wish list", "webShop");
        var result = wishListService.CreateWishList(request);
```

DeleteWishList

DeleteWishList is used to query delete a wish list in the external commerce system and returns the deleted wish list object.

Name:	**DeleteWishList**
Description:	Deletes a wish list that matches the specified criteria. Calls the pipeline "DeleteWishList"
Usage:	Called when a wish list needs to be deleted. Examples include: • Delete a specific wish list
Signature:	DeleteWishListResult DeleteWishList([NotNull]DeleteWishListRequest request)
Input:	
	WishList - The wish list to delete from the ecommerce system
Output:	
	WishList – The deleted wish list
	SystemMessages - Collection of messages from the external system.

Usage Example:

```
        var wishListService = new WishListServiceProvider();

        var createRequest = new CreateWishListRequest("Bob", "Bob's wish list", "webShop");
        var wishList = wishListService.CreateWishList(createRequest).WishList;
        var deleteRequest = new DeleteWishListRequest(wishList);
        var result = wishListService.DeleteWishList(deleteRequest);
```

EmailWishLists

EmailWishLists is used to raise a page event when a user tries to email a wish list.

Name:	**EmailWishLists**
Description:	Emails some wish lists based on the specified criteria. Calls the pipeline "emailWishLists"
Usage:	Called wish lists need to be emailed.
Signature:	EmailWishListsResult EmailWishLists([NotNull] EmailWishListsRequest request)
Input:	
	IEnumerable<WishList> - The wish lists to email
Output:	

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

	IEnumerable<WishList> – The wish lists that were emailed.
	SystemMessages - Collection of messages from the external system.

Usage Example:

```
var wishListService = new WishListServiceProvider();

var line = new WishListLine
{
    Product = new Entities.Carts.CartProduct
    {
        ProductId = "Product1",
        Price = new Entities.Prices.Price((decimal)22.99, "USD")
    },
    Quantity = 1
};
var wishList = new WishList
{
    CustomerId = "Bob",
    Name = "Bob's wish list",
    ShopName = "webShop",
    Lines = new List<WishListLine>() {line}.AsReadOnly()
};

var wishLists = new Collection<WishList> { wishList };

var request = new EmailWishListsRequest(wishLists);
var result = wishListService.EmailWishLists(request);
```

GetWishList

GetWishList is used to query wish list data against the external commerce system and returns a single wish list.

Name:	**GetWishList**
Description:	Gets the wish list that matches the specified criteria. Calls the pipeline "GetWishList"
Usage:	Called when a wish list is needed. Examples include: • Getting the wish list for a user based on the wish list id and shop name
Signature:	GetWishListResult GetWishList([NotNull]GetWishListRequest request)
Input:	
	UserId – The id of the user who owns the wish list
	WishListId – The id of the wish list being search for.
	ShopName – The name of the store containing the wish list
Output:	
	WishList – The wish list found based on the search citeria
	SystemMessages - Collection of messages from the external system.

Usage Example:

```
var wishListService = new WishListServiceProvider();
```

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

```
var createRequest = new CreateWishListRequest("Bob", "Bob's wish list", "webShop");
var wishList = wishListService.CreateWishList(createRequest).WishList;

var request = new GetWishListRequest("Bob", wishList.ExternalId, "webShop");
var result = wishListService.GetWishList(request);
Assert.AreEqual(wishList.Name, result.WishList.Name, "Names should be the same");
```

GetWishLists

GetWishLists is used to query wish list data against the external commerce system and returns a collection wish list.

Name:	**GetWishLists**
Description:	Gets the wish lists that match the specified criteria. Calls the pipeline "GetWishLists"
Usage:	Called when all of the wish lists for a user are needed.
Signature:	GetWishListsResult GetWishLists([NotNull]GetWishListsRequest request)
Input:	
	UserId – The id of the user to search for
	ShopName – The store containing the wish lists
Output:	
	ReadOnlyCollection<WishListHeader>– WishListHeader objects for all of the wish lists found
	SystemMessages - Collection of messages from the external system.

Usage Example:

```
var wishListService = new WishListServiceProvider();
var request = new GetWishListsRequest("Bob", "webShop");
var result = wishListService.GetWishLists(request);
```

UpdateWishList

UpdateWishList is used to update wish list data in the external commerce system returns the updated wish list object.

Name:	**UpdateWishList**
Description:	Updates the specified wish list. Calls the pipeline "UpdateWishList"
Usage:	Called when a wish list needs to be updated.
Signature:	GetCartsResult GetCarts(GetCartsRequest request)
Input:	
	WishList – The wish list to update
Output:	
	WishList – The updated wish list
	SystemMessages - Collection of messages from the external system.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Usage Example:

```
var wishListService = new WishListServiceProvider();
var createRequest = new CreateWishListRequest("Bob", "Bob's wish list", "webShop");
var wishList = wishListService.CreateWishList(createRequest).WishList;

wishList.IsFavorite = true;

var request = new UpdateWishListRequest(wishList);
var result = wishListService.UpdateWishList(request);
```

AddLinesToWishList

AddLinesToWishList is used to add line items to a wish list in the external commerce system and returns the updated wish list and added line items.

Name:	**AddLinesToWishList**
Description:	Adds the specified line items to the wish list. Calls the pipeline "AddLinesToWishList"
Usage:	Called when line items need to be added to wish list.
Signature:	AddLinesToWishListResult AddLinesToWishList([NotNull]AddLinesToWishListRequest request)
Input:	
	WishList - The wish list to add line items to.
	IEnumerable<WishListLine> – The line items to add to the wish list
Output:	
	WishList – The updated wish list object
	ReadOnlyCollection<WishListLine> - The line items added to the wish list
	SystemMessages - Collection of messages from the external system.

Usage Example:

```
var wishListService = new WishListServiceProvider();
var createRequest = new CreateWishListRequest("Bob", "Bob's wish list", "webShop");
var wishList = wishListService.CreateWishList(createRequest).WishList;

var line = new WishListLine
{
    Product = new Entities.Carts.CartProduct
    {
        ProductId = "Product1",
        Price = new Entities.Prices.Price((decimal)22.99, "USD")
    },
    Quantity = 1
};
var lines = new List<WishListLine> {line}.AsReadOnly();

var request = new AddLinesToWishListRequest(wishList, lines);
var result = wishListService.AddLinesToWishList(request);
```

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

UpdateWishListLines

UpdateWishListLines is used to update line items in a wish list in an external commerce system and returns the updated wish list and line items added to it.

Name:	**UpdateWishListLines**
Description:	Updates specific line items on a wish list. Calls the pipeline "UpdateWishListLines"
Usage:	Called when some line items on a wish list need to be updated.
Signature:	UpdateWishListLinesResult UpdateWishListLines([NotNull]UpdateWishListLinesRequest request)
Input:	
	WishList – The wish list that contains the line items that need to be updated.
	IEnumerable<WishListLine> – The wish list items to update
Output:	
	WishList – The updated wish list
	ReadOnlyCollection<WishListLine> – The wish list items that were updated
	SystemMessages - Collection of messages from the external system.

Usage Example:

```
var wishListService = new WishListServiceProvider();
var createRequest = new CreateWishListRequest("Bob", "Bob's wish list", "webShop");
var wishList = wishListService.CreateWishList(createRequest).WishList;

var line = new WishListLine
{
    Product = new Entities.Carts.CartProduct
    {
        ProductId = "Product1",
        Price = new Entities.Prices.Price((decimal)22.99, "USD")
    },
    Quantity = 1
};
var lines = new List<WishListLine> { line };

var addLinesRequest = new AddLinesToWishListRequest(wishList, lines);
wishList = wishListService.AddLinesToWishList(addLinesRequest).WishList;

line = new WishListLine
{
    Product = new Entities.Carts.CartProduct
    {
        ProductId = "Product2",
        Price = new Entities.Prices.Price((decimal)33.77, "USD")
    },
    Quantity = 2
};
lines = wishList.Lines.ToList();
lines.Add(line);

var request = new UpdateWishListLinesRequest(wishList, lines);
var result = wishListService.UpdateWishListLines(request);
```

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

RemoveWishListLines

RemoveWishListLines is used to remove some specific line items from a wish list, and return the updated wish list.

Name:	**RemoveWishListLines**
Description:	Removes lines items from a wish list that match the specified criteria. Calls the pipeline "RemoveWishListLines"
Usage:	Called when a line items need to be removed from a wish list.
Signature:	RemoveWishListLinesResult RemoveWishListLines([NotNull]RemoveWishListLinesRequest request)
Input:	
	WishList – The wish list requiring the items to be removed
	IEnumerable<string> – The ids of the line items to remove
Output:	
	WishList – The wish list with the line items removed
	ReadOnlyCollection<WishListLine> – The line items that were removed from the wish list
	SystemMessages - Collection of messages from the external system.

Usage Example:

```
        var wishListService = new WishListServiceProvider();
        var createRequest = new CreateWishListRequest("Bob", "Bob's wish list", "webShop");
        var wishList = wishListService.CreateWishList(createRequest).WishList;

        var line = new WishListLine
        {
            Product = new Entities.Carts.CartProduct
            {
                ProductId = "Product1",
                Price = new Entities.Prices.Price((decimal)22.99, "USD")
            },
            Quantity = 1
        };
        var line2 = new WishListLine
        {
            Product = new Entities.Carts.CartProduct
            {
                ProductId = "Product2",
                Price = new Entities.Prices.Price((decimal)33.77, "USD")
            },
            Quantity = 2
        };
        var lines = new List<WishListLine> { line, line2 };

        var addLinesRequest = new AddLinesToWishListRequest(wishList, lines);
        wishList = wishListService.AddLinesToWishList(addLinesRequest).WishList;

        var request = new RemoveWishListLinesRequest(wishList, new List<string> {
wishList.Lines[0].ExternalId });
        var result = wishListService.RemoveWishListLines(request);
```

PrintWishList

PrintWishList is used to raise a page event when a user tries to print a wish list.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Name:	PrintWishList
Description:	Gets the wish list that matches the specified criteria. Calls the pipeline "PrintWishList"
Usage:	Called when a copy of the wish list is needed for printing
Signature:	PrintWishListResult PrintWishList([NotNull] PrintWishListRequest request)
Input:	
	WishList – The wish list required for printing.
Output:	
	WishList – The wish list for printing
	SystemMessages - Collection of messages from the external system.

Usage Example:

```
        var wishListService = new WishListServiceProvider();
        var line = new WishListLine
        {
            Product = new Entities.Carts.CartProduct
            {
                ProductId = "Product1",
                Price = new Entities.Prices.Price((decimal)22.99, "USD")
            },
            Quantity = 1
        };
        var wishList = new WishList
        {
            CustomerId = "Bob",
            Name = "Bob's wish list",
            ShopName = "webShop",
            Lines = new List<WishListLine>() { line }.AsReadOnly()
        };

        var request = new PrintWishListRequest(wishList);
        var result = wishListService.PrintWishList(request);
```

2.13.3 Wish Lists Pipelines

The integration and engagement logic used in the Cart API is implemented by pipelines that can be customized as needed. There is a pipeline for each method on the API. Some pipelines call other common pipelines like SaveCart and LoadCart. In some cases the logic is split into several sub-pipelines to handle if-than-else situations like used in CreateOrResumeCart.

CreateWishList

Name:	CreateWishList
Description:	The pipeline is responsible for creating a wish list and rasing an associated page event.
Usage:	Called via method CreateWishList on the Connect API when creating wish lists.
Args:	
	Request - Contains the search criteria: UserID, WishListName and ShopName. Is set prior to calling the pipeline.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

	Response - Contains the created wish list object. Is read after the pipeline is executed.
Processors:	**TriggerWishListCreatedPageEvent–** **Responsibility** – Rasises a page event called “WishList Created”
	AddVisitorToEaPlan– **Responsibility** – Adds the user to a wish list releated EA plan

DeleteWishList

Name:	**DeleteWishList**
Description:	The pipeline is responsible for deleteing a wish list and raising an associated page event.
Usage:	Called via method DeleteWishList on the Connect API when delete a wish ist
Args:	
	Request - Contains the search criteria: WishList. Is set prior to calling the pipeline.
	Response - Contains the WishList object. Is read after the pipeline is executed.
Processors:	**TriggerWishListDeletedPageEvent–** **Responsibility** – Raises a page event called “WishLists Deleted”

EmailWishLists

Name:	**EmailWishLists**
Description:	Emails a specified wish list and raises and associated page event
Usage:	Called via method EmailWishLists on the Connect API when emailing wish lists.
Args:	
	Request - Contains the search criteria: IEnumerable<WishList>. Is set prior to calling the pipeline.
	Response - Contains an IEnumerable<WishList>. Is read after the pipeline is executed.
Processors:	**TriggerWishListsEmailedPageEvent –** **Responsibility** – Raises a page event called “WishList Emailed”

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

GetWishList

Name:	GetWishList
Description:	The pipeline is responsible for performing a search against all wish lists and returning a WishList instance that matches the specified search criteria.
Usage:	Called via method GetWishLists on the Connect API when searching for a wish list.
Args:	
	Request - Contains the search criteria: UserId, WishlistId, and ShopName. Is set prior to calling the pipeline.
	Response - Contains the matching WishList object. Is read after the pipeline is executed.
Processors:	**No default processors**

GetWishLists

Name:	GetWishLists
Description:	The pipeline is responsible for performing a search against all wish lists and returning a list of WishListHeader instances for carts found, matching the specified search criteria.
Usage:	Called via method GetWishLists on the Connect API when searching for wish lists.
Args:	
	Request - Contains the search criteria: UserID and ShopName. Is set prior to calling the pipeline.
	Response - Contains the WishListHeader objects. Is read after the pipeline is executed.
Processors:	**No default processors**

UpdateWishList

Name:	UpdateWishList
Description:	The pipeline is responsible for updating select line items on a wish list
Usage:	Called via method UpdateWishList on the Connect API when updating a wish list.
Args:	
	Request - Contains the search criteria: WishList. Is set prior to calling the pipeline.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

	Response - Contains the WishList object. Is read after the pipeline is executed.
Processors:	**No default processors**

AddLinesToWishList

Name:	**AddLinesToWishList**
Description:	The pipeline is responsible for add new line items to a cart and raising an associated page event.
Usage:	Called via method AddLinesToWishList on the Connect API when adding new lines to wish list.
Args:	
	Request - Contains the criteria: WishList and a collection of WishListItems. Is set prior to calling the pipeline.
	Response - Contains the updated WishList and the initial WishListItem objects. Is read after the pipeline is executed.
Processors:	**TriggerLinesAddedToWishListPageEvent–** **Responsibility –** Raises a page event called “Lines Added To WishList”

UpdateWishListLines

Name:	**UpdateWishListLines**
Description:	The pipeline is responsible for updating specific line items already added to the wish list.
Usage:	Called via method UpdateWishListLines on the Connect API when updating line items on a wish list.
Args:	
	Request - Contains the search criteria: WishList and a collection of WishListItems. Is set prior to calling the pipeline.
	Response - Contains the updated wish list and the requested wish list items to update. Is read after the pipeline is executed.
Processors:	**TriggerLinesUpdatedOnWishListPageEvent –** **Responsibility** – Raises a page event called “Lines Updated On WishList”

RemoveWishListLines

Name:	**RemoveWishListLines**

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Description:	The pipeline is responsible for removing selected line items from a wish list.
Usage:	Called via method RemoveWishListLines on the Connect API when searching for carts.
Args:	
	Request - Contains the search criteria: WishList and a list of line item ids to remove. Is set prior to calling the pipeline.
	Response - Contains the updated wish list and the line items object that were removed. Is read after the pipeline is executed.
Processors:	**TriggerLinesRemovedFromWishListPageEvent –** **Responsibility** – Raises a page event called “Lines Removed From WishList”

PrintWishList

Name:	**PrintWishList**
Description:	The pipeline is responsible for retrieving a wish list object for printing
Usage:	Called via method PrintWishList on the Connect API when searching for a wish list to print.
Args:	
	Request - Contains the search criteria: WishList object. Is set prior to calling the pipeline.
	Response - Contains the WishList object to print. Is read after the pipeline is executed.
Processors:	**TriggerWishListPrintedPageEvent –** **Responsibility** – Raises a page event called “Lines Removed From WishList”

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

2.14 Connect Configuration

You can use the Sitecore.Commerce.config file and the individual service layer configuration files to register entities, repositories, pipeline processors and service providers. In the following sections the Cart service layer is used as example.

2.14.1 Factories and entities

The Factory Method Pattern is an object-oriented creational design pattern that implements the concept of factories. You can create objects without basing it on a specific class. The core of this pattern is to define an interface for creating an object, but let the classes that implement the interface decide which class to instantiate. The Factory method lets a class defer instantiation to subclasses.

To configure the entity factory to use, set the type. By default the Sitecore Factory is used implicitly through Connect:

```
        <!-- ENTITY FACTORY Creates an entity by entity name. Allows to substitute default
entity
        with extended one. -->
        <entityFactory type="Sitecore.Commerce.Entities.EntityFactory, Sitecore.Commerce"
        singleInstance="true" />
```

To configure custom objects in the Sitecore.Commerce.Carts.config file:

```
        <!--  Connect ENTITIES Contains all the Connect entities. The configuration can be
used to
        substitute the default entity implementation with extended one. -->
        <commerce.Entities>
            <CartBase type="Sitecore.Commerce.Entities.Carts.CartBase, Sitecore.Commerce" />
            <Cart type="Sitecore.Commerce.Entities.Carts.Cart, Sitecore.Commerce" />
        </commerce.Entities>
```

In an actual Connect provider implementation, the custom objects are known and do not necessarily have to be created through use of Factory.

You can use references to the factory as a parameter in some processors, for example, CreateCart:

```
        <processor type="Sitecore.Commerce.Pipelines.Carts.CreateCart.CreateCart,
Sitecore.Commerce">
            <param ref="entityFactory" />
        </processor>
```

You must use the Factory.Create method to get an instance of the needed type. For example, in the following code snippet, we need a cart and calls the factory to create and return a cart. The cart domain model can be completely modified and customized so that you can replace the default cart type with your own implementation:

```
        public override void Process(ServicePipelineArgs args)
        {
          var result = (CartResult)args.Result;
          var cart = this.entityFactory.Create<Cart>("Cart");
          var request = (CreateOrResumeCartRequest)args.Request;
          cart.UserId = request.UserId;
          cart.ShopName = request.ShopName;
          cart.CartName = request.CartName;
          cart.CustomerId = request.CustomerId;
          cart.CartStatus = CartStatus.InProcess;
          result.Cart = cart;
        }
```

2.14.2 Pipelines for Methods

In the Sitecore.Commerce.Carts.config file, you can set your processors into pipelines that to inject business logic.

The following table contains some examples of pipelines in the configuration file of Connect:

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

Pipeline	Description
`getCarts`	This pipeline searches for all carts that match some specific criteria. The carts are managed by the commerce system. This pipeline reads cart data from the commerce system and converts that data into Connect format.
`createOrResumeCart`	This pipeline: • Initiates the creation of a shopping cart. • Loads persisted, abandoned cart, if present. • Calls `resumeCart` pipeline to resume loaded cart. • Calls `createCart` pipeline to create cart if no cart was found in the previous steps.
`createCart`	This pipeline: • Is internally used by the `createOrResumeCart` pipeline if the existing cart was not found and should be created. • Creates a cart with the minimal number of required fields. • Moves a visitor / contact to the initial state in the engagement plan. • Saves a cart to the storage and triggers the `CartCreate` event.
`resumeCart`	This pipeline: • Is internally used by the `createOrResumeCart` pipeline if a cart was loaded and should be resumed. • Sets the initial state to the loaded cart, moves a visitor / contact to the initial state in the engagement plan. • Saves a cart to the storage and triggers the `CartResume` event.
`loadCart`	This pipeline: • Gets a cart object that matches a specified criteria. • Reads data for a cart that is managed by the commerce system. • Reads the cart data from the commerce system and converts that data into the Connect format.
`saveCart`	This pipeline saves the cart object to an external system and in Sitecore EA state.
`addCartLines`	This pipeline adds a new line to the shopping cart and records a corresponding page event in DMS. This happens when a product is added to the cart.
`removeCartLines`	This pipeline removes cart lines from cart.
`updateCartLines`	This pipeline updates lines on cart.
`deleteCart`	This pipeline: • Deletes a cart permanently: • When the cart is deleted, it triggers the event in DMS to indicate that the cart is deleted.
`updateCart`	This pipeline: • Passes an updated cart to the external commerce system. • Triggers an event in DMS to indicate that the cart is being updated.
`lockCart`	This pipeline sets the cart to a locked state and prevents any modifications.
`unlockCart`	This pipeline sets the cart to an unlocked state.
`getCartTotal`	This pipeline: • Gets the totals object that matches the specified criteria. • Is responsible for reading pricing data from a commerce system. • Converts the contents of a Connect cart into a format the commerce system can work with.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.

	• Sends a request to the commerce system to calculate the totals, and converts the output into the proper Connect format.
getProductPrices	This pipeline: • Gets the price object that matches the specified criteria. • Reads the pricing data from a commerce system. • Requests the product pricing information from the commerce system and converts the output into the proper Connect format.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2017 Sitecore. All rights reserved.