Sitecore Commerce Connect 7.2

Components Overview

An overview of the Sitecore Commerce Connect components

Table of Contents

Chapter 1

Introduction

Sitecore Commerce Connect is an abstract service layer and not a stand-alone solution. It is a framework and an API for e-commerce with focus on integration and at the same time, to track, act, and follow up on customer behavior. The goal is to bring the unique customer engagement features of Sitecore into e-commerce solutions, regardless of the back-end e-commerce system being used.

With the Sitecore Experience Platform and Sitecore Commerce Connect, Sitecore provides the tools to empower organizations to build the effective, meaningful relationships that win customers for life. However, applying all the great engagement features in a solution can be difficult. With Connect, the challenge is reduced by having the common e-commerce related engagement functionalities integrated into the Connect framework.

Note
In the following:

- Connect is used as an abbreviation for Sitecore Commerce Connect
- ECS is used as an acronym for the External Commerce System

1.1 Target Audience

Connect is an integration layer between a front-end webshop solution and a back-end e-commerce system. This means that there are the following target audiences:

- Sitecore partners, developers, and customers who use the Sitecore Experience Platform to build their e-commerce solutions, where it provides the perfect match. Connect provides them with built-in engagement for commerce and they can choose between a variety of third-party e-commerce vendors for the back-end system, for which Connect connectors have been created.
- Sitecore technology partners or e-commerce system vendors who want to integrate their commerce systems with Sitecore to provide developers with e-commerce functionality and engagement features. This combines the best of two worlds while maintaining the unique features of their individual e-commerce systems.

1.2 Solution Overview

Connect is designed for end-user business scenarios, including:

- Business-to-consumer (B2C) sales of tangible goods, digital goods, or online service delivery.
- Business-to-business (B2B) scenarios that include:
  - Advanced product pricing.
  - Customers having multiple users acting on their behalf.
  - Sales agents acting on behalf of multiple customers.
  - Multiple shopping carts per customer or user.

The goals for integration are:

- Making it as easy as possible to integrate with third-party commerce systems
- Supporting as many different commerce systems as possible
- Avoiding the lowest common denominator syndrome
- Maintaining the unique features of each commerce system
- Bringing the benefits of Sitecore CEP to commerce sites, such as features that track, act on, and follow up on customer behavior

With the above goals in mind, the design of the framework is based on the following principles:

- Simplicity — the features in e-commerce systems are different from each other. To avoid complicating the architecture, the most advanced scenarios, which are only present in few systems, are avoided in the base framework. To only support the most common scenarios, the default domain models provided with Connect integration layers are kept to a minimum.
- Extensibility — because the domain models are kept simple, developers who integrate Sitecore with the external commerce system must customize the domain model and the request and response parameters that are exchanged with the external commerce system.
- Each integration service layer must be able to operate independently. There must be no interdependencies. This separates concerns and allows different external commerce systems to handle different parts of the integration. However, some domain model objects are used across service layers.
- Each integration service layer must be abstract and therefore operate on neutral generic parameters that are not tied to any Sitecore-specific concepts or any external commerce systems.
- Each integration service layer must use Sitecore pipelines to host the business logic and allow for customizations and extensions.
- Connect should act as a fallback mechanism in the rare cases where the most common e-commerce scenarios for which the system has been designed, are not supported by the external e-commerce systems. In these cases, Connect must act as an intermediate storage that supports the missing scenarios and acts as a bridge between Sitecore and the external commerce system. Examples are:

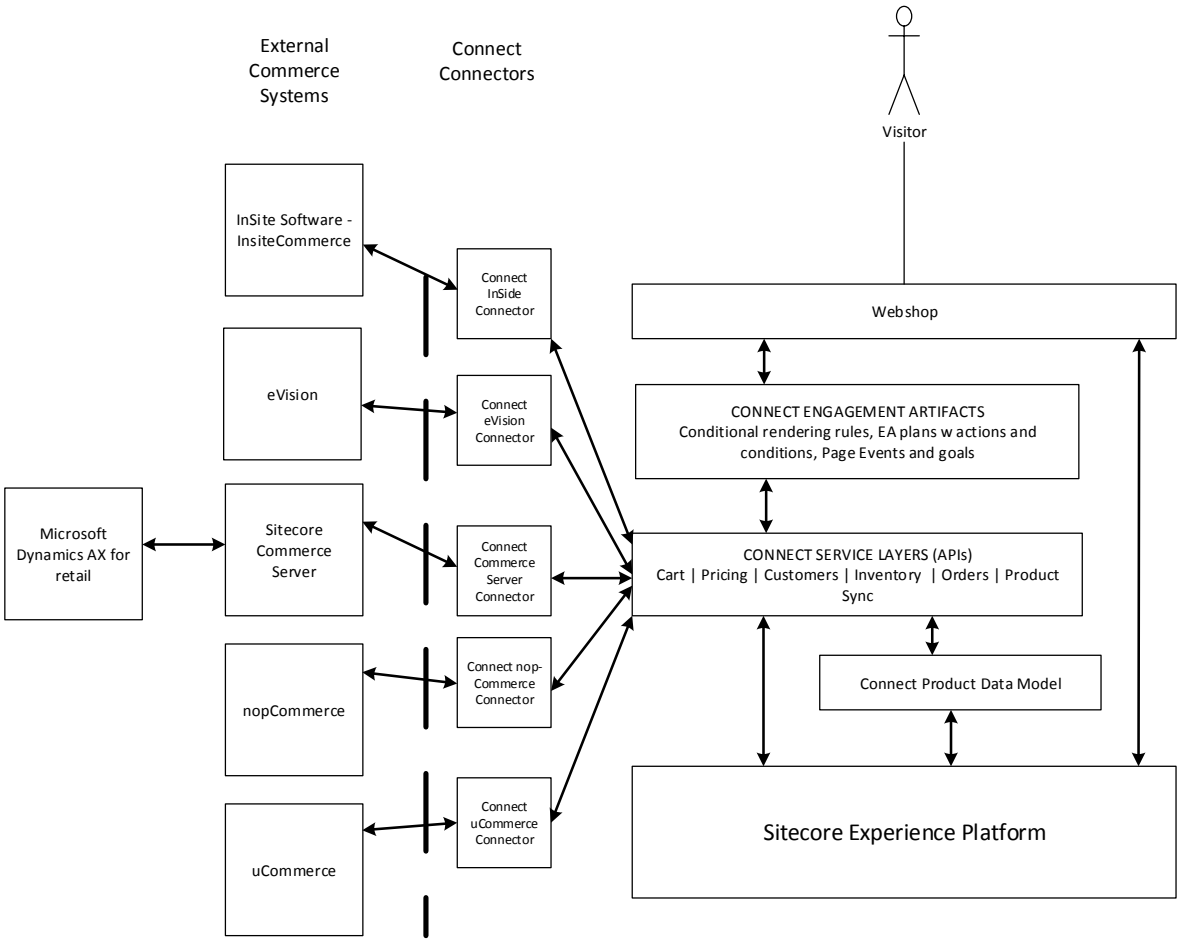
- The B2B concept of having two individual but linked entities like customers and users. Not all e-commerce systems supports that. They might support the entities, but not the many-to-many relationship between them.
- Support for multiple shopping carts per user or customer.
- Limited shopping cart information, for example, no support for nested sublines.

Chapter 2 Connect Components

Connect consists of the following components:

- The Connect Core Framework that contains the abstract Service Layers and performs engagement activities, such as tracking page events and goals, acting on conditional rendering rules, and following up on engagement automation (EA) plans, actions, and conditions.
- Connect Connectors that hook into pipelines and integrate with external commerce systems (ECS). The connectors are not a part of the core framework, but they are needed because Connect cannot run as a standalone system. Developers can use Connect connectors to integrate Sitecore with one or more commerce systems. A connector for Sitecore Commerce Server exists and more are on the way. For more information on the connector for Sitecore Commerce Server, see the Sitecore Developer Network (SDN) website.

The following diagram illustrates the Commerce Connect framework components and how they are connected:



Note
The connectors in the previous image are not all developed – they appear in the diagram for illustration purposes and are typically developed by third-party vendors.

The previous diagram illustrates the relationships between the components as follows:

- The visitor accesses the webshop
- The webshop uses the rules for personalization and EA plans to follow up.
- The webshop use the integration service layers that in turn use the connectors to integrate with and access the ECS. By using the Connect API, page events and goals are triggered implicitly and forms the basis of the personalization etc.

- The Sitecore Experience Platform runs in the background to ensure that both CMS and engagement functionalities are executed such as rendering content, tracking, acting and following up on the visitor behavior.
- The connectors ensure that proper e-commerce functionalities are available from the external commerce systems.

Note
If data and functionality are managed by different systems, developers can use one or more connectors in a one solution. For example, you can have products data exists in one system and prices are handled in another system.

2.1 The Connect Abstract Service Layers

The core framework contains a list of abstract service layers. The design of the service layers is based on common e-commerce use cases. The service layers are kept generic because they target different external commerce systems.

When building an e-commerce solution using Connect, you should:

- Use the present Connect service layer API instead of accessing the external commerce system API directly. The reasons for this are that:
    - Most of the engagement is built into the service layers. If the service layer APIs are not used, the engagement part is not deployed.
    - Some external commerce systems do not support the scenarios that Connect supports. Therefore, you should use the Connect API to benefit from the fallback functionality. For more information, see the *Target Audience*

Connect is an integration layer between a front-end webshop solution and a back-end e-commerce system. This means that there are the following target audiences:

- Sitecore partners, developers, and customers who use the Sitecore Experience Platform to build their e-commerce solutions, where it provides the perfect match. Connect provides them with built-in engagement for commerce and they can choose between a variety of third-party e-commerce vendors for the back-end system, for which Connect connectors have been created.
- Sitecore technology partners or e-commerce system vendors who want to integrate their commerce systems with Sitecore to provide developers with e-commerce functionality and engagement features. This combines the best of two worlds while maintaining the unique features of their individual e-commerce systems.
    - Solution Overview section.
    - Using the standard Connect API lets you replace the external commerce systems with a minimum of effort. In this case, customizations and extensions made to accommodate unique features in the ECS cannot just be replaced if they are not already supported in the new ECS.
- Call the external commerce system API directly, only if the desired functionality does not exist or is out of the scope of Connect.

The service layers are generic and extensible and have built-in engagement features. The service layer:

- Is the carrier of essential e-commerce information between the webshop solution and the back-end e-commerce system
- Encapsulates the business logic of the external commerce system.
- Tracks and follows up on important events when interacting with the visitor. The implementation makes use of Goals, Page Events, and EA plans.

The service layers handles the events that occur in e-commerce scenarios. To encapsulate the business logic of these events, they are represented by a service method and corresponding Sitecore pipelines. Events can be divided into front-end and back-end events.

The following are examples of front-end events that are triggered by an action on the website:

- Create a customer account
- Update a wish list
- Update a shopping cart
- Place an order

Back-end events that are triggered by actions either in the third-party system or in Sitecore back-end applications.

The following are examples of back-end events:

- Update product information
- Update an order status
- Update a product price

Connect includes pipeline processors that interact with Sitecore. Processors that interact with third-party systems are included in Connect connectors and are provided as individual installation packages by Sitecore technology partners.

The Connect pipelines are open for customization, including overwriting the default Connect processors.

2.1.1 Connect Service Layer Pipeline Example

The following is an example of a pipeline that adds products to a shopping cart:

```
<Sitecore.Commerce.Cart.AddProductToShoppingCart>
  <processor type="CommerceSystem.Cart.ValidateProduct" />
  <processor type="CommerceSystem.Cart.CheckInventory" />
  <processor type="CommerceSystem.Cart.PutHoldOnProduct" />
  <processor type="Sitecore.Commerce.Cart.RecordPageEvent" />
</Sitecore.Commerce.Cart.AddProductToShoppingCart>
```

In this pipeline:

- The red processor is specific to Sitecore and provided by Connect. Developers can modify or remove it.
- The green processors are specific to the external commerce system that is integrated with Sitecore. These processors are a part of a connector to the ECS, and Sitecore with Connect does not provide them. Developers can replace or remove them.

2.1.2 Customizing the Service Layers for ECS

To ensure that the service layers fit with different external commerce systems, each service layer consists of:

- An API
  - Each service layer method takes a Request object that can be customized to satisfy the needs of the ECS.
  - Each service layer method returns a Result object that can be customized to satisfy the needs of the ECS.

- One or more pipelines for each API call
  - Each service layer method executes a pipeline that can call one or more pipelines.
- An extensible domain model
  - All domain model objects can be inherited and extended with custom properties.
  - All nested domain model objects can be inherited and extended with custom properties.
  - All service methods keep the existing defined signature, even when used with customized domain objects.

2.1.3 Implementation Details

Each service layer contains its own configuration file that defines the entities that makes up the domain model, pipelines, factories, repositories, and the service layer API implementation.

- Sitecore Factory is used to instantiate the domain model objects, hence allowing the individual entities to be customized, configured and instantiated.
- The Repository design pattern is used in processors that read and write data to Sitecore. It allows the repository to be easily replaced and customized if needed.
- The default pipeline configuration in Connect contains placeholder processors in the places where processors for external commerce systems are expected. Some service layers contains comments instead. The placeholder processors are empty and do not execute any code. An ECS connector consists of processors that interact with the ECS and a configuration file that patches the Connect configuration file, replacing the placeholder processors in the pipelines.

For more information, see the Connect Developer Guide.

2.1.4 Available Connect Service Layers

The available Connect service layers are:

- Shopping Cart
- Pricing
- Customers and Users
- Inventory
- Orders
- Product Synchronization (optional)

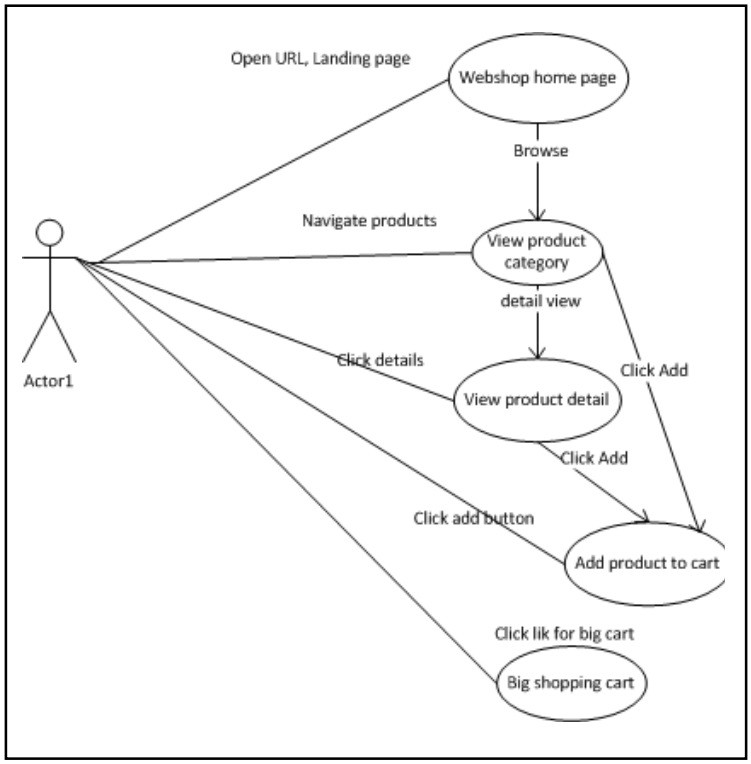
In the following sections, the Shopping Cart, Pricing and Product synchronization layers are described. For more information, see the Connect Developer Guide

2.1.5 Shopping Cart

The shopping cart service layer collects and maintains information about the shopping cart and its content. The shopping cart layer also contains a number of engagement features.

The service layer API contains the following operations:

- CRUD operations on cart
  - CreateOrResumeCart*
  - DeleteCart*
  - UpdateCart*
  - SaveCart
  - LoadCart
  - GetCarts
- CRUD operations on cart lines
  - AddCartLines*
  - UpdateCartLines*
  - RemoveCartLines*
- Locking and unlocking cart
  - LockCart*
  - UnlockCart*

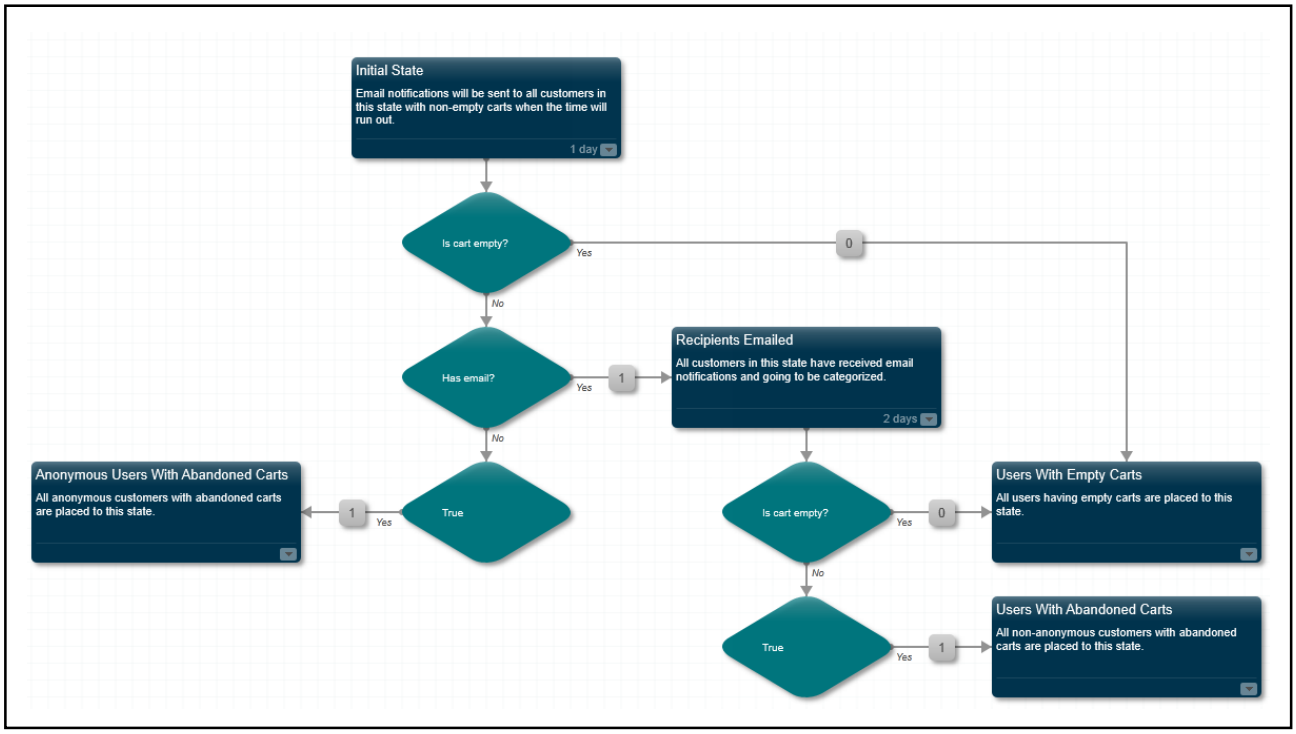


Note
All operations marked with an asterisk (*) trigger a unique page event that allows the visitor interaction to be carefully tracked, making it possible to report and act on them. Most of the events include information about the cart ID and the added products.

The complete list of installed page events is included in the Connect Installation Guide.

The Abandoned Cart Engagement Automation Plan

Connect contains a default Abandoned Cart EA plan, which lets you track, act, and follow up on shopping carts determining if they are abandoned.



When a new visitor arrives or a new cart is created, the visitor is added to the Abandoned Cart engagement automation plan.

By default, the plan works as follows:

- After a default one-day period, the system checks on whether there is a cart for the visitor and whether this cart is empty, and an e-mail is known for the visitor. If so, an e-mail is sent to the visitor reminding them that they have abandoned their cart in an attempt to get the visitor to return and place an order. If an e-mail is not known or the cart is empty, the visitor is moved to one of two end-states and nothing further happens.
- After a default two-day period, the system checks on whether there is still a cart for the visitor and whether it is empty, and the visitor is moved to one of two end-states: Abandoned or Users with Empty Carts (or not abandoned).

In Connect, you can use the Abandoned Shopping Cart EA plan as a repository for Shopping Carts. You can also store custom data with the state in the EA plans. You can use this feature, to serialize Shopping Carts and store the data in the plan. Storing carts in the plan has the following functions:

- The carts are immediately available in Sitecore when the visitor is recognized by the CEP engine.
- If the session is lost, the carts are resumed when the visitor returns, assuming that the visitor is recognized by the CEP engine or that they log in with an account.

The cart data is directly available for the conditional rendering rules, without having to query the external commerce system, which improves performance.

The EA state can be the only repository so that carts are handled solely in Connect if this is the preferred integration approach.

In case of shopping carts being purged in the external commerce systems at regular intervals by cleanup routines, the EA repository can act as backup storage.

Some of the shopping cart information from the default Connect domain model might not be covered in the external commerce system and part of the information can be persisted, retrieved, and merged when working with the cart data.

Cart-based Conditional Rendering Rule Conditions

Connect includes three rendering rule conditions that can be used to personalize the visitor experience based on the products in the cart, the cart total, and the quantity of products. The conditions can be combined in a rule in a number of different ways. For example:

- Contains product: If [shopping cart contains product A] then …
- Cart total: If [shopping cart total is > Y] then …
- Item quantity: If [shopping cart total quantity of items > N] then …
- A combination of the above: If [shopping cart contains more than N quantity of product A and cart total > Y] then …

There is a forth generic conditional rendering rule condition that allows the editor to act if a given page event P has been triggered N amount of times or more within a given timespan T.

2.1.6 Pricing

Prices are not part of the product data and can be handled by an external system that is different from the system that holds the product data. For these reasons, Connect contains a separate pricing service layer.

The pricing service layer:

- Retrieves prices of products according to a set of parameters. This includes:
  - A method that retrieves different prices at the same time for a single product, for example, price, sale price, and customer price. Each price is indicated by a price type.
  - A method that retrieves prices for a list of products to limit the amount of calls to the external system and improve performance. Only one price type is used at a time with bulk prices.
- Calculates the total and subtotal calculation for a shopping cart. This includes:
  - A method that calculates cart totals. The input argument is the cart and the output is the cart augmented with totals, subtotals, adjustments in form of charges or discounts, and, optionally, product prices or updated prices. In addition to the augmented entities, the cart content can be changed during the calculations according to the business logic. An example is the bundling of cart lines or the addition of cart lines based on campaign offers like *if you buy A and B, you will get C for free*.

To support B2C and B2B scenarios, the following input parameters are provided by default when calling the service methods:

- Product ID
- Customer ID
- Location
- Quantity
- Shop Name
- Price Type IDs, for example, list, customer, break, and sale price

When retrieving prices, the default domain model includes a price condition entity that allows the inclusion of break price information

There is no engagement imbedded in the pricing service layer. For more information about the pricing service layer and the product data model for prices, see the Connect developer guide.

2.1.7 Products

There are three obvious ways to access product data from a solution that is built with Sitecore:

- Using the product data from the content by storing and reading product data from content in the master database.
- Using a Sitecore data provider by retrieving and storing product data from an external system using a data provider.
- Indexing the external product repository by indexing product data in an external system and hydrate product data partially or in full from the index. Hydration is a new concept used in CMS 7. For more information, see the Developer's Guide to Item Buckets and Search.

Connect has built-in support for using product data from content. Connect contains its own product data model and a Product Synchronization service layer for exchanging product data with one or more external systems.

On the surface, a data provider solution may seem the most suitable because product data is typically *owned* by the external commerce system and naturally belongs there. However, there are a number of reasons to have the product data stored in Sitecore.

Product Data in Content

Augmenting and Adding Presentation Data to Products

There are typically limitations to the kind of information stored in the commerce system and the number of channels supported. Usually, the core data is not stored in a presentable way. One of the premises with Connect is that only the core product data is included in the ECS. The rest of the information needs to come from other external systems or can be added in Sitecore.

Sitecore becomes the aggregator of product information needed for customer presentation in all supported channels. Presentation and sales texts for different channels can be managed in Sitecore as the last resort before rendering.

Support for Multiple Data Sources

In some scenarios, the product data is provided by multiple sources. For example, core product data can be provided by the ECS, and the specifications along with categorization can be provided by a different provider. Two examples of classification systems are UNSPEC® and CNET DataSource™.

- The Universal Standard Products and Services Classification (UNSPEC) was created for coding products and services unambiguously according to industry-agreed naming standards for streamlining commerce among companies, particularly over the Internet. UNSPEC is an open, global electronic commerce standard that provides a logical framework for classifying goods and services.
- CNET Content solutions contains a product called DataSource. DataSource converts non-standardized product information from multiple sources into consistent content for electronic product catalogs. It is also a catalog mapping service that automatically matches external catalogs to the CNET Content Solutions DataSource product database to enrich those catalogs with DataSource specifications, product attributes, and rich content

Support for Distributed Systems

The Content Management (CM), Content Delivery (CD), and the ECS instances can be in different geographic locations and the latency might therefore be high, for example. The CM and CD might be in the cloud, whereas the ECS can be on the premises. In this scenario, it is probably not be possible nor practical to use real-time calls or an item data provider.

Once data is synchronized to Sitecore content, it is immediately and always available. Using an item data provider makes the system vulnerable in case of connectivity problems between Sitecore and the external system.

Ensure Good Performance with Semi-static Product Data

Not all external systems perform equally well. Working with products merged into Sitecore, using an item data provider, can negatively affect performance. It is desirable to have product data available in Sitecore content.

Product data does not contain prices and inventory data in Connect. Inventory data is highly volatile and prices are complex and depend on business rules that cannot easily be transferred to Sitecore. Therefore, Connect contains separate service layers for getting prices and inventory information.

Because the product repository in Connect does not contain prices or stock information, the product information becomes static. Even with a million products in the repository, only a small fraction of these will typically have to be updated, added, or removed on a regular basis.

In addition, if the solution constantly has to read products from an item data provider, this consumes much more time than if it reads them from Sitecore.

Sitecore as a Rendering Engine

Sitecore has rendering features that present information and different levels of caching mechanisms.

Scalable Default Data Model

All vendors store product data in different ways in their system. This typically represented in Sitecore.

The Connect data model is designed to avoid the most common pitfalls and provide a standard model regardless of the external system that is used.

The model is highly scalable and has the following advantages:

- A solid base for typical e-commerce scenarios
- No redundant product data
- Minimal product data that needs to be synchronized
- An easy way for e-commerce vendors to map product data
- A standard structure that allows developers to replace the external commerce system with the minimal effort and a known structure to manage products
- No typical custom model implementation problems. For more information, see the previous description
- Benefits from the CMS 7 features

Creating a connector that supports Connect product synchronization might seem like a large task. However, the amount of work of creating and supporting a Sitecore data provider and ensuring that the performance is adequate is as large.

Using a Sitecore Data Provider

There are as many reasons for storing product data in content as there are for accessing product data using a Sitecore data provider.

Because each service layer works independently of each other, you can use an item data provider for product data and still use the remainder of the service layers for integration and provide built-in engagement. If you prefer to use a data provider to inject product data into Sitecore content, you should use the Connect product data model to abstract the integration details. This provides a uniform model to build reusable UI components and tools to present and manage product data in Sitecore.

The product data model should be preserved even when using a data provider. This makes the UI framework components and the tools that manage product data in Sitecore work unaltered even when the commerce system is replaced.

The product data model exists on two levels:

- The item data structures

- The object data model

The solution scenario is to use a data provider to the external system to preserve product item structures. If this is not possible, you can still use the object domain model as an abstraction. Because of the flexibility built into the Connect product item model, it is typically more comprehensive to create a data provider that resembles that model, than the initial custom models typically created based on ECS.

Indexing the External Product Repository

A hybrid solution is to create a Sitecore product index based on product data that is either stored in the external system or in Sitecore content. You can use the object domain model in this solution to search for and return products, and you can hydrate the data model that is in CMS 7.

In Connect, the product object model can be customized to use the product ID as a reference to load the data from the ECS. By replacing the default logic, the product data can be populated from the index or ECS using a data provider or direct API calls.

Product Synchronization

Connect has its own product data model and a Product Synchronization service layer for exchanging product data with one or more external systems. The responsibility of the Product Synchronization service layer is to manage two-way synchronization of essential product data. The goal is to synchronize only the data that is needed to satisfy the most common e-commerce scenarios. In particular cases, the model can be extended. For more information on the default scenarios that are supported, see the developer guide.

The synchronization is two-way, but can be forced in only one direction if needed. All service methods take a direction parameter, which is set to ECS->Sitecore by default. A more static approach is to remove the pipeline processors, so that data is only written in one direction. For more information, see the section Two-Way Synchronization.

Synchronization of product data is done one language at a time. All service methods take a language parameter, which is set to EN by default.

As with any other service layer in Connect, there is a domain model for product synchronization that:

- During synchronization of the product data is used as a DTO to transfer information between Sitecore and the external System
- Can be used for managing product data without having to deal with the actual storage. Whether it is stored in Sitecore as items or partially in the index and partially in an ECS. For example, when querying the product index, it is possible to get instances of the product domain model in return for of a single item that only contains a fraction of the data associated with the product. The hydration functionality built into Sitecore can be used for this purpose.

The product synchronization service also includes a domain model representation in CMS content based on a number of predefined templates. The model consists of:

- A main bucketed product repository that contains composite tree structures each representing a product
- A number of associated repositories that hold data that is related to products like manufacturers, classifications, categories, product types, divisions, and resources

A typical approach to designing product repositories is to put all product data into a single item to represent a product in Sitecore. This single-item approach has the following disadvantages:

- Flat Model – resulting in redundant data and potential template overload.
- Redundant Data – the same data being duplicated into multiple product items, which share the same information. This is not best practice in a SQL database or in Sitecore. The more

duplicated data, the more space it consumes and the harder it becomes to maintain when data is changed.

- Template Overload – can lead to one of these undesirable designs:
  - One template for all products that have to include all the combinations of all fields for all the types of products in the repository. Only a small fraction of the fields are filled out for any given products.
  - One template for each type of product, with dedicated fields for the given product. Even though template inheritance seems like a good idea, it will lead to an overload of templates because of the amount of different product types.

  In Connect, the solution is to split up the product data into multiple items, with a single template for the core product data and a fixed small number of additional templates for each additional type of information.
- Difficulty of Extension – with numerous templates used for representing different types of products it becomes difficult to extend and customize the data model in Sitecore.
- Difficulty of Maintaining Redundant Data – resulting in huge costs in maintaining information when data changes.

The templates used for the item-based model remain the same, regardless of the types of products.

Inventory and Price Information

The Connect product repository does not contain prices or stock information. There are separate service layers for getting prices and inventory information.

Inventory data is not content, but application data. It is volatile information that changes rapidly and it does not belong with Sitecore content. This explains why it is not included with the product data model in Connect and that there is a separate service layer for it. Application data belongs in its own repository separate from CMS content.

Sitecore is usually used for static content. For data to be rendered to the visitors, it is published to CD servers.

You should not use Sitecore as a provider for data, such as inventory and price data, because this type of data changes frequently. Therefore, you should not:

- Store and update data directly in the web database. You should store this data in the Master database to be published to the CD servers.
- Access the Master database from the content delivery servers. For example, the CM servers can be on the premises and the CD servers in the cloud.

Two-way Synchronization

The service layer is designed for two-way synchronization. In most setups, the product data is owned by the ECS and only supports one-way synchronization; however the framework has been created to support two-way synchronization.

The implementation of Connect uses a pattern similar to the Bridge Design Pattern. The product domain model serves as the data abstraction that hides its implementation in the ECS as well as in Sitecore. Each entity is read from both the ECS and Sitecore. A comparison of the values between identical instances is executed, and the result is written back to both the ECS and Sitecore. This means that each pipeline has the same pattern of processors for each entity, where the entity can be product, manufacturer, division, or classification.

The two-way synchronization takes place in the following order:

1. The entity is read from the ECS.
2. The entity is read entity from the CMS.

3. The entities are compared and the differences are resolved.
4. The results are written to the ECS.
5. The results are written to the CMS.

The service layer methods take a direction parameter with the following possible values:

- ECS->CMS: one-way synchronization, inbound. This is the default value.
- CMS->ECS: one-way synchronization, outbound
- CMS<->ECS: two-way synchronization, both directions

Depending on the value of the Direction parameter, some of the previously listed processors skip execution. For example, if the Direction parameter is set to inbound (ECS->CMS), there is no need to read the entity from CMS or write it back to the ECS.

2.1.8 Performance Considerations

In the implementation of the product synchronization service layer, a number of initiatives have been deployed to make it as fast as possible:

- Buckets are used to store product and manufacturer data. Global specification buckets store, index and let you easily search huge amounts of data, such as product data.
- When a new item is created in a bucket, it is automatically placed in the root folder. If you want to move it into a different location, you need to synchronize the bucket. You can use one of the following methods:
  - You can synchronize a product by a single operation, by calling the `BucketManager.MoveItemIntoBucket(entityItem, root)` method.
  - You can use bulk synchronization by calling the `SynchronizeProducts` or `SynchronizeProductList` method. The bucket synchronization is delayed until all new product items have been processed. To further reduce the time spent, a temporary bucket is used for new product items. The temporary bucket is synchronized after all the products have been processed and then the bucket content is moved to the main bucket. This eliminates the time spent accessing all existing items in the bucket, which could be significant.
  - You can use multi-threaded synchronization. By default, a single thread is spawned from the processors that synchronize products, manufacturers, types, resources, divisions, and specifications. The threads are spawned for each repository being synchronized. The number of threads can be configured in the `Sitecore.Commerce.Products.config` file.
- Connect disables the triggering of item events and indexing during synchronization to avoid wasting resources on firing events or starting indexing before synchronization is done. Indexing is turned on after synchronization has finished. The following context and disablers are instantiated during synchronization:
  - `Sitecore.SecurityModel.SecurityDisabler()`
  - `Sitecore.Data.Proxies.EventDisabler()`
  - `Sitecore.Data.Proxies.ProxyDisabler()`
  - `Sitecore.Data.DatabaseCacheDisabler()`
  - `Sitecore.Data.BulkUpdateContext()`
- Reading product data once and then processing it in multiple pipelines reduces the number of calls between Sitecore and the external systems. All product entities in Connect are synchronized using the Connect pipeline, which reads the data from the external system in the individual pipelines. In this case, synchronizing a single product can amount to many

calls between the systems, and each call takes time and resources. The design does not prevent product data being read once initially and then passed on to the individual subpipelines for processing, which reduces the number of calls between the systems.

- Resources can be located externally. Resources in Sitecore are stored as media items in the media library. Media items are binary blobs and can be large and time-consuming to import into Sitecore. Therefore, resources can either be imported into the Sitecore media library or simply referred to externally. If resources are imported, they are stored in a bucketed Products folder under the media library. If not imported, they can be referred to by a URI stored on the resource reference item.
- By design, the remote product repository is always regarded as the main repository, which by default owns the products. This makes the ID of the products and artifacts in the external system the primary key. In Sitecore, the ID of the corresponding items for products and artifacts is generated by Connect instead of relying on the default Sitecore implementation that automatically generates a new GUID. By using a hash algorithm, you can generate a direct mapping between the IDs coming from the external system and the item IDs in Sitecore. This has the following benefits:
  - There is no need for mapping tables, which take up space.
  - Getting the ID of the item IDs is very fast.
  - There is no need for searching for items in Sitecore – you can use the external ID instead.

  The default implementation is based on the MD5 hash algorithm and has the following format:

```
Item.ID = MD5.ComputeHash(Prefix + ExternalID);
```