Sitecore Commerce Connect 7.5
Connect Installation Guide

A developer's guide to installing Sitecore Commerce Connect

Table of Contents

Introduction

This guide is for developers who are looking for information about how to install the Sitecore Commerce Connect framework and how to create product repositories.

In the following, Connect is used as abbreviation for Sitecore Comnerce Connect.

This guide also contains useful information for Sitecore developers who implement webshops and for external commerce system developers who want to use Connect for integration.

This guide describes how to:

- Install the Connect framework

- Configure a webshop

- Integrate Connect with an external commerce system and create a product repository

Installing the Connect Framework

You must install Connect as a package in Sitecore to use it for webshop development and integration with external commerce systems.

System Requirements

The following are the requirements to install the Connect framework:

- .NET Framework 4.5
- Microsoft SQL Server 2008 R2 or later
- Sitecore CXP 7.5 rev. 141003 or later. This CXP version is also known as Sitecore CXP 7.5

Note
You must run the application pool in **integrated mode**. For more information, see CXP 7.5 Installation Guide.

Installing Connect

To Install the Sitecore Connect package:

1. Use the Sitecore installation wizard to install the Sitecore Commerce Connect 7.5 zip package
2. Select **Republish** to publish everything.
3. The three engagement automation plans that comes with Connect, needs to be configured for each webshop. For more information see the next section.
4. If your external commerce system uses the product synchronization feature that comes with Connect to store product data in Sitecore, you must create a product repository. To create a product repository, see the section Integrating an External Commerce System on page 7.

Configuring a Webshop

With Connect comes three engagement automation plans:

- Abandoned Carts
- New Order Placed
- Products Back In Stock

There needs to be one instance of each plan for each web-shop defined. The following image shows the three plans for the webshop named Website
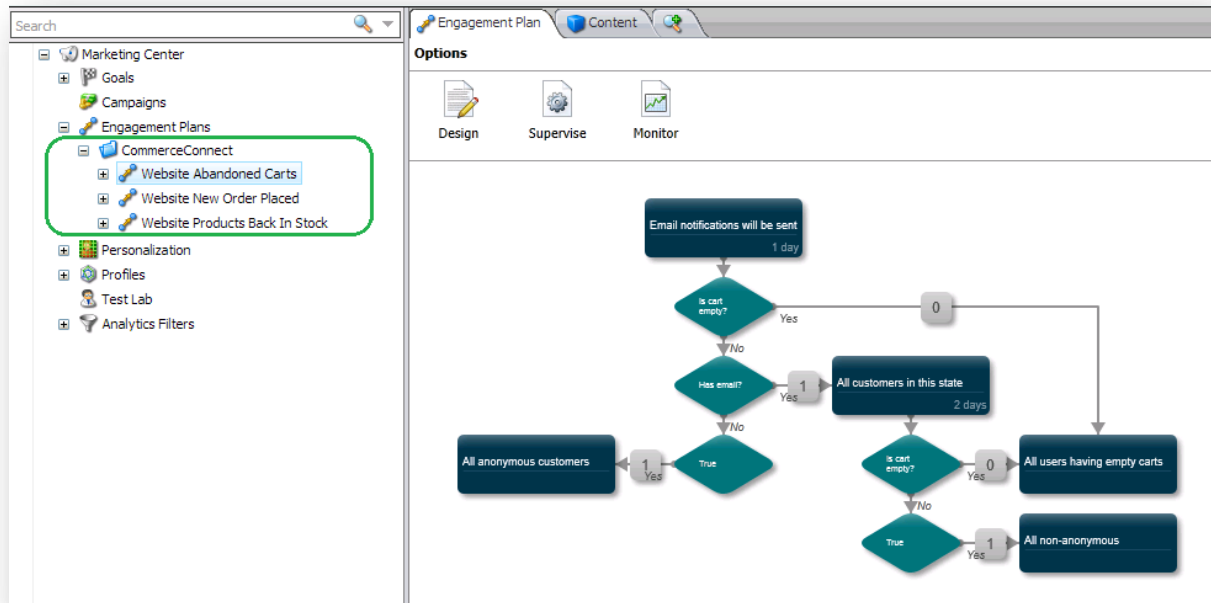
To make Connect work with multiple shops, you need to configure an engagement automation plan for each shop. The following procedure works for all three plans:

1. In the Content Editor, locate the Marketing Center

2. Create a new engagement plan under "/sitecore/system/Marketing Center/Engagement Plans" based on the preconfigured Abandoned Carts branch found under the path: "`sitecore/system/Marketing Center/Engagement Plans/CommerceConnect`".

3. Name the plan using the following naming convention: "`{Site name} Abandoned Carts`", where "site name" is the name of the site as defined in the <site name="…" > element in the web.config file.

   Following the naming convention will automatically bind the plan to the Connect framework. If you want to use a custom EA Plan name, add the attribute `engagementPlanName` under the site definition `engagementPlanName="My store"`. When you do this the website is connected to the "`sitecore/system/Marketing Center/Engagement Plans/My store`" plan.

   ```
   <site name="StarterKit" patch:before="site[@name='website']"
       engagementPlanName ="My store"
   …  />
   ```

   The `engagementPlanName` attribute works for the Abandoned Cart plan, but not the order and inventory plans, where the name has to be changed in code if the default is undesirable. See next section for a description of that approach.

4. Deploy the plan and publish it.

Changing the default EA plan naming scheme by code

In Connect, the default implementation searches for an EA Plan that has a name that consists of the name of the site concatenated with the name of the plan. If you use a different name for the plan, you need to update the configuration or the implementation.

The configuration approach is explained in the *previous* section and it's the easiest way to change the name. The alternative is to replace the default code. In the implementation, there is an interface called `IEaPlanProvider` with a default implementation called `EaPlanProvider`. The `EaPlanProvider` class returns the EA plan based on the naming scheme. By replacing the default EaPlanProvider and the logic, you are able to create custom naming schemes.

In the `/App_Config/Include/Sitecore.Commerce.Config` file, you can use the following setting to replace the default implementation:

```
        <eaPlanProvider type="Sitecore.Commerce.Multishop.EaPlanProvider,
Sitecore.Commerce"
         singleInstance="true" />
```

The dual purposes of the Abandoned Cart EA plan

The default configuration of Connect uses the Abandonned Cart engagement automation plan for the following two purposes:

- To track visitors and follow up when they leave the shop prematurely and abandon their shopping cart.

  If you do not want to use the Abandoned Cart plan, you can disable it by removing the following processors in the `/App_Config/Include/Sitecore.Connect.Carts.config` file:

  o The `AddVisitorToEaPlan` processor in the `SaveCart` pipeline

  o The `MoveVisitorToInitialState` processor in the `ResumeCart` pipeline

- To store a copy of the shopping cart. This is done so that the cart(s) is immediately available when the visitor is recognized by the Sitecore Experience engine. Some external systems only support part of the shopping cart functionality and data and in those cases the entire cart is stored in the EA state.

  If you do not want to save the cart to an EA state, you can disable this by removing the following processors from the pipelines found in the `/App_Config/Include/Sitecore.Connect.Carts.config` file:

  o The `SaveToEAState` processor in the `SaveCart` pipeline

  o The `FindCartInEaState` processor in the `CreateOrResumeCart` pipeline

  o The `LoadCartFromEaState` processor in the `LoadCart` pipeline

 o The `DeleteCartFromEaState` processor in the `DeleteCart` pipeline

Integrating an External Commerce System

Not all external commerce systems use product synchronization, but if you integrate Sitecore with an external commerce system that uses product synchronization, you must create a product repository. If the external commerce system does not use the product synchronization approach, skip this step.
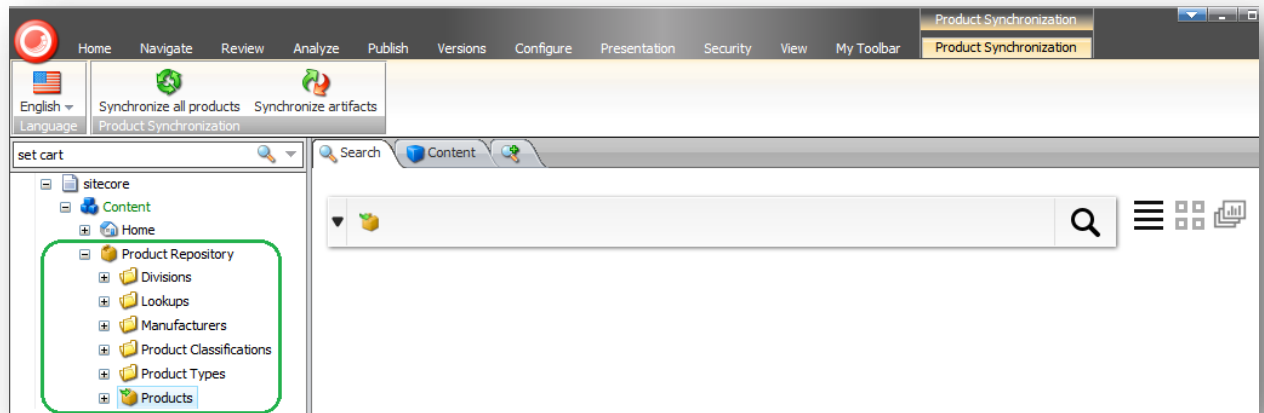
Creating a Product Repository

After you install the Connect framework package, use the following steps to create a product repository:

1. Open the Content Editor and create a product repository item that is based on the `Templates/Branches/CommerceConnect/Products/Product Repository` branch.
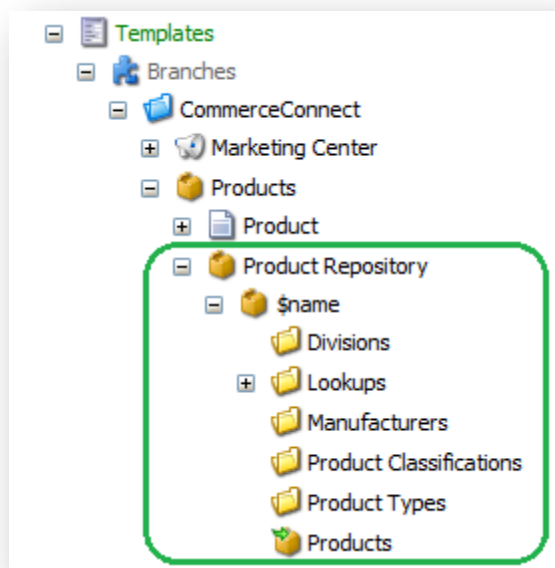
The following image shows the product repository:

Note
To avoid redundancy, create only one repository to be shared across all shops. The concept of Divisions is used to indicate which shop products belongs to.

2. Update the search Source for the Manufacturer and ProductType fields of the Product template:

   a. Browse to the item "/sitecore/content/Product Repository/Manufacturers" and copy the item ID.

   b. Browse to the template "/sitecore/templates/CommerceConnect/Products/Product" and set the Source for the "Manufacturer" to "StartSearchLocation=<itemid>" where <itemid> is the item ID you copied in the previous step.

   c. Save the template.

   d. Browse to the item "/sitecore/content/Product Repository/Product Types" and copy the item ID.

   e. Browse to the template "/sitecore/templates/CommerceConnect/Products/Product" and set the Source for the "ProductType" to "StartSearchLocation=<itemid>" where <itemid> is the item ID you copied in the previous step.

   f. Save the template.

```
<paths productRepository="/sitecore/content/Product Repository">
```

You can use this attribute to specify a custom product repository name or ID or to change the location of the product repository in Sitecore content.

3. If you change the location of the product repository from the default "/sitecore/content/Product Repository", you must also change the configuration of the following two index files (specified in bold in the following samples):

   o In the `Sitecore.Commerce.Products.Lucene.Index.Master.config` file:

```
<index id="commerce_products_master_index"
```

```xml
            type="Sitecore.ContentSearch.LuceneProvider.LuceneIndex,
        Sitecore.ContentSearch.LuceneProvider">
            <param desc="name">$(id)</param>
            <param desc="folder">$(id)</param>
            <!-- This initializes index property store. The ID has to be set to the index
            id -->
            <param desc="propertyStore" ref="contentSearch/databasePropertyStore"
              param1="$(id)" />
            <strategies hint="list:AddStrategy">
            <!-- NOTE: order of these is controls the execution order -->
            <strategy ref="contentSearch/indexUpdateStrategies/manual" />
            </strategies>
            <commitPolicyExecutor type="Sitecore.ContentSearch.CommitPolicyExecutor,
              Sitecore.ContentSearch">
            <policies hint="list:AddCommitPolicy">
                <policy type="Sitecore.ContentSearch.TimeIntervalCommitPolicy,
                  Sitecore.ContentSearch" />
            </policies>
            </commitPolicyExecutor>
            <locations hint="list:AddCrawler">
                <crawler type="Sitecore.ContentSearch.SitecoreItemCrawler,
                  Sitecore.ContentSearch">
                    <Database>master</Database>
                    <Root>/sitecore/content/Product Repository/Products</Root>
                </crawler>
            </locations>
        </index>
```

o   The `Sitecore.Commerce.Products.Lucene.Index.Web.config` file:

```xml
<index id="commerce_products_web_index"
type="Sitecore.ContentSearch.LuceneProvider.LuceneIndex,
Sitecore.ContentSearch.LuceneProvider">
    <param desc="name">$(id)</param>
    <param desc="folder">$(id)</param>
    <!-- This initializes index property store. The ID has to be set to the index
    id -->
    <param desc="propertyStore" ref="contentSearch/databasePropertyStore"
    param1="$(id)" />
    <strategies hint="list:AddStrategy">
    <!-- NOTE: order of these is controls the execution order -->
    <strategy ref="contentSearch/indexUpdateStrategies/onPublishEndAsync" />
    </strategies>
    <commitPolicyExecutor type="Sitecore.ContentSearch.CommitPolicyExecutor,
      Sitecore.ContentSearch">
    <policies hint="list:AddCommitPolicy">
        <policy type="Sitecore.ContentSearch.TimeIntervalCommitPolicy,
          Sitecore.ContentSearch" />
    </policies>
    </commitPolicyExecutor>
    <locations hint="list:AddCrawler">
        <crawler type="Sitecore.ContentSearch.SitecoreItemCrawler,
          Sitecore.ContentSearch">
            <Database>web</Database>
            <Root>/sitecore/content/Product Repository/Products</Root>
        </crawler>
    </locations>
</index>
```

Note

You should store the product repository under /sitecore/content, so that it does not get mistaken for the main website that typically goes under /sitecore/content/Home.

Note also, that there are two similar files for configuring indexes for SOLR, if that is th e preferred indexing engine

4. If you change the name of the Commerce Connect product index from the default "commerce_products_master_index", you must also change the configuration of the setting "ProductSynchronization.ProductIndexes" in `Sitecore.Commerce.Products.Config` file to contain the new name. The setting contains a list of index names that will be paused, resumed and rebuild during product synchronization,

```
<!--   PRODUCT INDEXES.
       The indexes used to store synchronized products.
       Can be stopped, resumed and rebuild automatically during product
synchronization.
    -->

        <setting name="ProductSynchronization.ProductIndexes"
    value="sitecore_master_index, commerce_products_master_index" />
```

5. In the Content Editor select the product repository item "/sitecore/content/Product Repository" and push button Synchronize All Products. As part of synchronizing, the product and master indexes are rebuilt
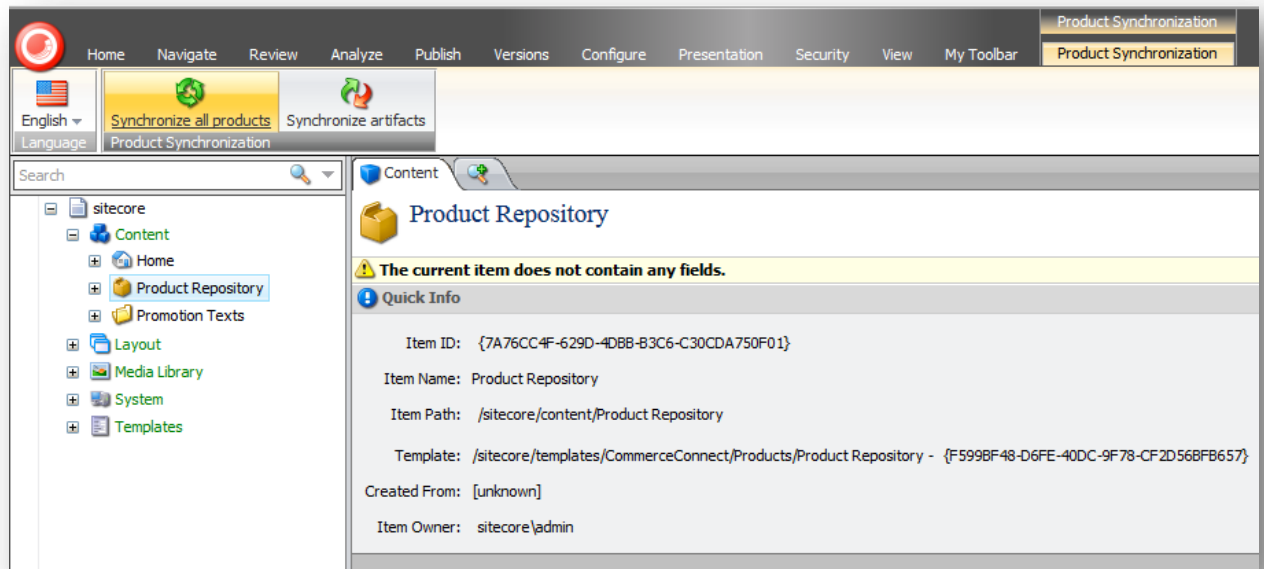
6. In the Publish group, select **Publish Site**, and then select **Republish** to publish everything.

# Installed CMS and DMS Related Items

When you install the Connect package, the following DMS items are automatically installed:

- Page events

- Conditional rendering rule conditions

- Engagement automation plans

- Engagement automation conditions and actions
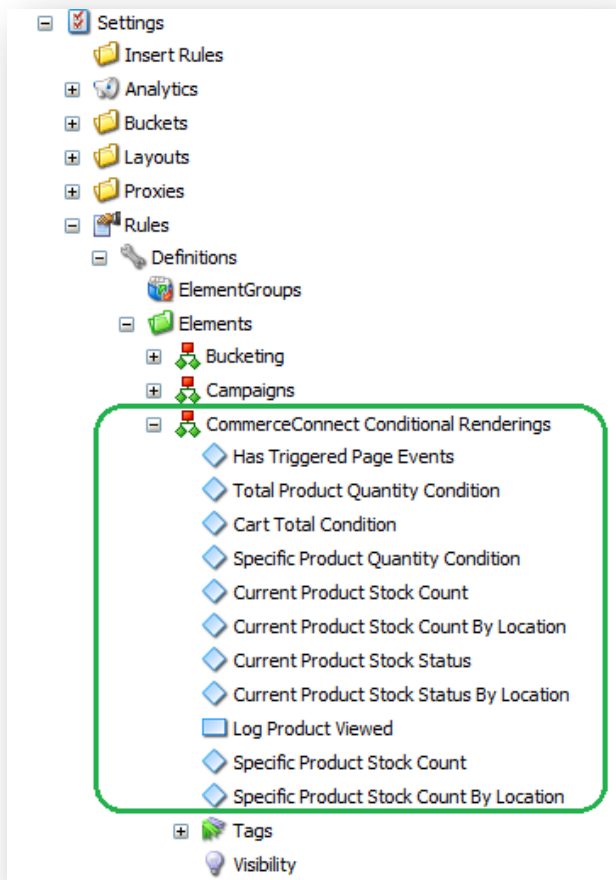
- Templates and branch templates

Page Events

The Connect package automatically installs, and then deploys the page events under the path `/sitecore/system/Settings/Analytics/Page Events/CommerceConnect`.

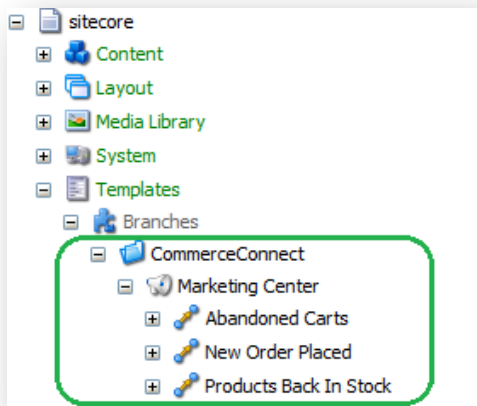The following image shows the Connect events that are installed:
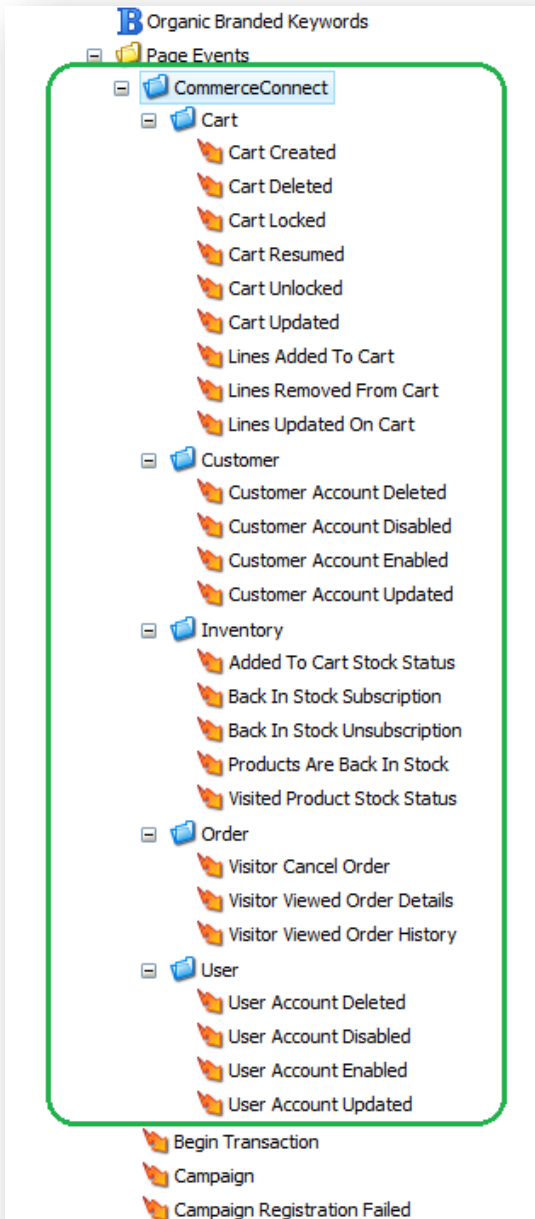
Rendering Rule Conditions

The following image shows the installed Connect rule conditions in the content tree:
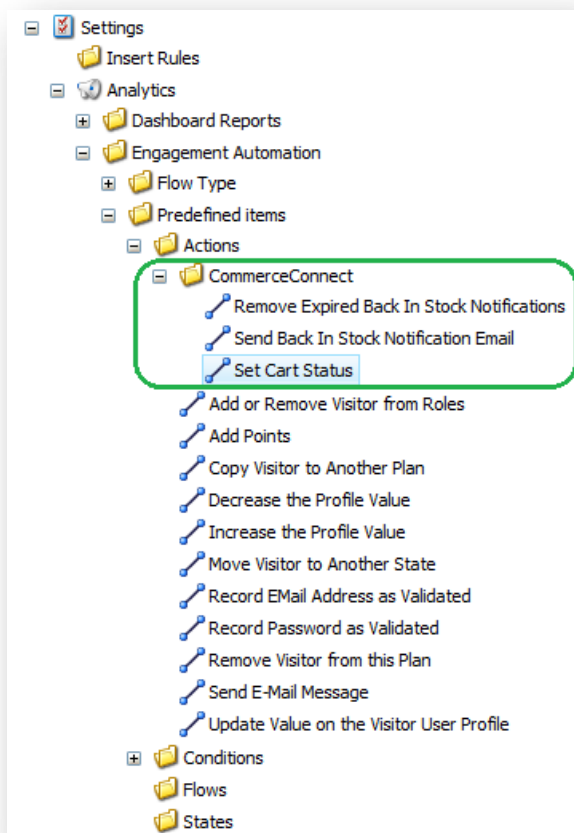
Engagement Automation

The following image shows the installed Connect engagement automation branch templates in the content tree:
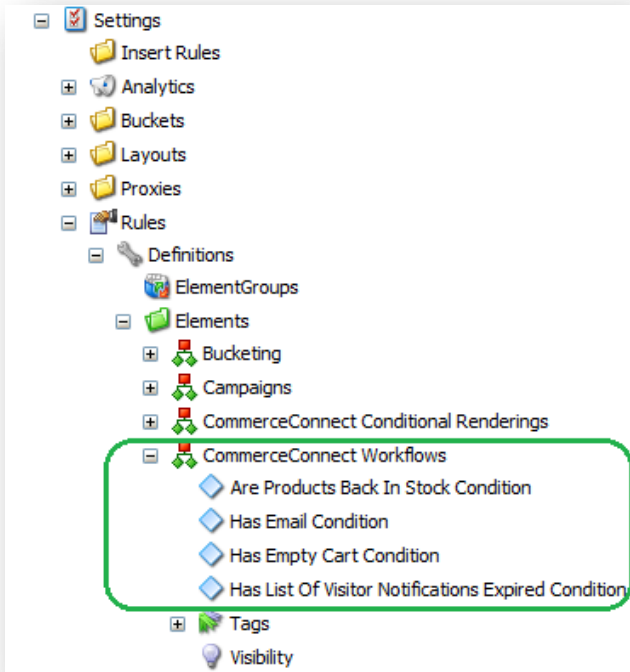
Engagement Automation Conditions and Actions

The following image shows the installed Connect engagement automation actions in the content tree:
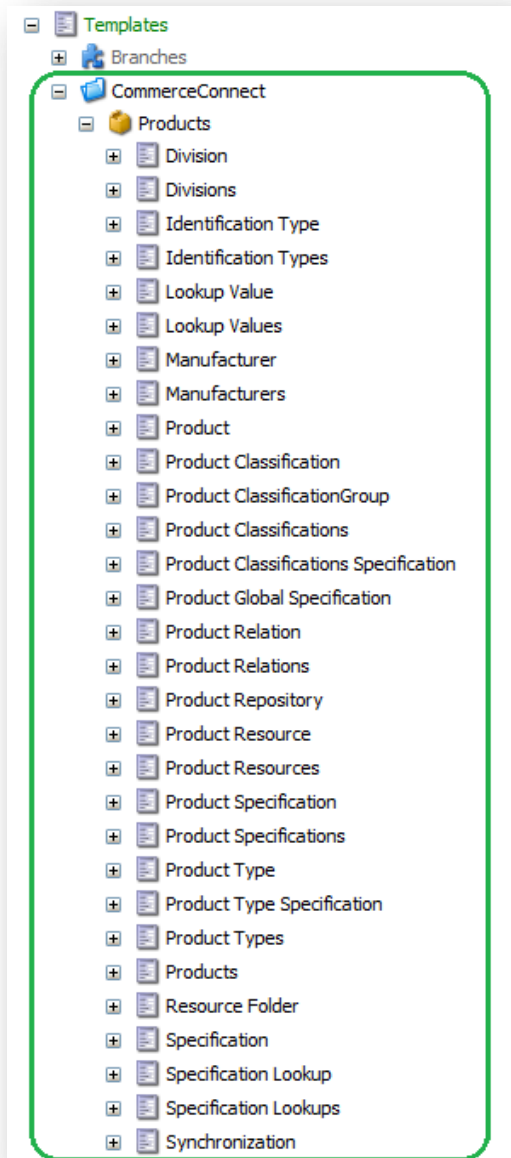
The following image shows the installed Connect engagement automation conditions in the Content Tree:

Templates

The following image shows the installed Connect templates in the content tree:

Branch Templates

The following templates are installed as branch templates.

- Abandoned Carts EA plan
- New Order Placed EA plan
- Products Back In Stock EA plan

- Product
- Product Repository

For more information about branch templates, see the Siteore *Data Definition API Cookbook*.

The following image shows the installed Connect branch templates in the Content Tree: