Rev: July 13, 2018

Sitecore Experience Commerce Data Migration Guide

Sitecore Experience Commerce 9.0. Update-2

Instructions for migrating data from Sitecore Commerce 8.2 to Sitecore Experience Commerce 9.0

sitecore®
Own the experience™

Table of Contents

Chapter 1

Migrating Commerce Server data to Sitecore XC

This document provides instructions on how to migrate catalog and inventory data from a Sitecore Commerce 8.2 deployment to a Sitecore XC 9.0 Update-2 deployment.

This document contains the following sections:

1.1 Initial assumptions

The instructions in this document assume the following conditions for an on-premises solution:

- Sitecore XP 9.0 Update-2 is deployed.
- Sitecore XC 9.0 Update-2 is deployed, including all components (as described in the *Sitecore XC Installation Guide for On-Premise Solutions*).
- Legacy Commerce Customer Databases are accessible on SQL Server
- An ecommerce site (either your own or a custom application of the Solution Storefront) is deployed (useful for validating the migration)

Note
These instructions apply to Sitecore Commerce systems 8.2 or older. Data migration from Sitecore Commerce 8.2.1 solutions is not supported.

1.2 Export Commerce Server catalog and inventory data

You use the Commerce Server Catalog Manager to export your existing catalogs and inventory data to an XML file. Detailed information on exporting catalogs is available here.

Note
If you are exporting any Custom (Virtual) Catalogs, you must export them as Base Catalogs (that is, using the **Export as Base** setting).

For every catalog (including inventory catalogs) that you want to export, do the following:

1. Open the Commerce Server Catalog Manager (**Start** > **Sitecore Commerce 11**> **Catalog Manager**).
2. In the **Commerce Server Catalog Manager**, in the **Catalogs** pane, select the catalog you want to export.
3. In the **Task** pane, click **Export this Catalog**.
4. On the **Welcome to the Export Product Catalog Wizard** page, click **Next**.
5. On the **File Location** page, specify the following:
    - **Location**: location of the output file
    - **Export Type**: Full (to export the whole catalog)
    - **Select Language**: language in which to export the catalog
    - **Advanced**: enable
6. Click **Next**.
7. On the **Export Properties** page, specify the following:
    - **Schema Export Type**: All
    - **Export Deleted Items** (enable)
    - **Export Blank Values** (enable)
    - **Export Catalog Sets** (enable)
8. Click **Next**.
9. On the **Advanced Export Properties** page, review the export properties you specified. To change any export property values, click **Back**. To export the catalog, click **Create**.
10. On the **Completing the Export Product Catalog** wizard page, click **Finish**.

You must copy the exported XML files to your Sitecore XC 9.0 server, or to a location that the Data Migration tool can access when converting the files to JSON format.

1.3 Customize the Sitecore XC catalog properties

If your existing catalogs contain any customized attributes (that is, attributes that were not available in the default installation of your Sitecore Commerce 8.2 deployment), you must add them to your Sitecore XC 9.0 deployment before you migrate your data.

You can extend Commerce Entity properties programmatically. See this KB article for detailed instructions.

1.4 Configure the data mapping file

The Sitecore XC Data Migration tool uses a mapping file to convert Sitecore Commerce XML data to Sitecore XC 9.0 JSON entities for the Commerce Engine.

You must add definitions for any customized attributes in your Commerce Server 8.2 schema to the mapping file so that all of the catalog and inventory data is migrated properly.

You can generate the default mapping file using the Data Migration tool (Sitecore.Commerce.Migration.exe).

To generate the default data mapping file, do the following:

1. Download the Migration Tool from the Sitecore XC 9.0.2 Download page.
2. Unzip the contents of the `Sitecore.Commerce.Migration.1.0.x.zip` file.
3. Open a Powershell session, and navigate to the directory where you unzipped the Migration Tool .zip file.
4. Run the following command:

   ```
   .\Sitecore.Commerce.Migration.exe –cmo <filename>
   ```

   The tool creates a JSON file that contains all of the default data mapping definitions.

You can add your own custom attributes to existing definitions (or create new definitions as required) to ensure that all of your data is migrated. See Appendix: Catalog Mapping file for details on the default Catalog Mapping file.

1.5 Migrate the Commerce Server data

Once you have all of your Commerce Server schema defined in the catalog mapping file, you are ready to migrate your data to the Sitecore XC Commerce Engine format.

The Data Migration tool takes data from an XML file and converts the data into a format that is compatible with the Sitecore XC Commerce Engine data structure, based on the mapping definitions specified in the mapping file. The resulting JSON files are stored in .zip files.

The Data Migration Tool is an .exe file that takes the following parameters:

Parameter	**Description**
`-catalog (-c)`	Specifies the Commerce Server catalog (in XML format) to migrate.
`-catalogout (-co)`	Specifies the output file (in .zip format) where the migrated catalog data is stored.
`-inventory (-i)`	Specifies the Commerce Server Inventory catalog (in XML format) to migrate.
`-inventoryout (-io)`	Specifies the output file (in .zip format) where the migrated inventory data is stored.
`-catalogmappings (-cm)`	Specifies the mapping file that defines how to map values from the Commerce Server catalog XML file to Commerce Entities.
`-batchsize (-b)`	Specifies the maximum number of entities included in each JSON file created during the data migration.
`-force (-f)`	Indicates that the output file should be overwritten, if it already exists.
`-verbosity (-v)`	Specifies the amount of information displayed while the command is being executed. Possible values are: Trace, Debug, Information, Wwarning, Error, Critical, and None (default is Information).
`-catalogmappingsout (-cmo)`	Specifies the file name to use when generating the default mapping definitions.

Note
You must ensure that the catalog XML files you exported from Commerce Server deployment are on your Sitecore XC 9.0 server.

1.5.1 Migrate Commerce Server catalog data

To migrate your Commerce Server catalog data, do the following:

1. Open a Powershell session, and navigate to the directory where you unzipped the Sitecore.Commerce.Migration zip file.
2. Run the following command:

   ```
   .\Sitecore.Commerce.Migration.exe -c <XML-file> -co <zip-file> -cm
   <mappings-file> -b <integer>
   ```

   The tool converts the XML data to JSON format, and stores the output in multiple JSON files (inside a zip file).

1.5.2 Migrate Commerce Server Inventory data

To migrate your Commerce Server inventory catalog data, do the following:

1. Open a Powershell session, and navigate to the directory where you unzipped the Sitecore.Commerce.Migration zip file.
2. Run the following command:

   ```
   .\Sitecore.Commerce.Migration.exe –i <XML-file> -io <zip-file> -cm
   <mappings-file> -b <integer>
   ```

   The tool converts the XML data to JSON format, and stores the output in multiple JSON files (inside a zip file).

1.6 Import the migrated catalog and inventory data

Once you have migrated all of your Commerce Server 8.2 catalog and inventory data, you can import the data into your Sitecore XC 9.0 deployment.

You import the catalog and inventory data separately, using Postman samples provided in the Commerce Engine SDK.

To import your catalog and inventory data, do the following:

1. Install Postman and launch the application.
2. Navigate to the *Sitecore Commerce SDK* folder in the Sitecore XC 9.0 release package and open the *Postman* folder.
3. Import the contents of the *Postman* folder into Postman.

   Note
   Make any necessary changes to the global variables in the environment you are using (such as ServiceHost) under the **Settings** > **Manage Environments** menu in Postman.

4. In the **Collections** pane in Postman, navigate to the *Authentication* folder.
5. Open the *Sitecore* folder and execute the *GetToken* call.

   When Postman displays an access token in the **Body** pane, authentication is successful.
6. In the **Collections** pane, navigate to the *CatalogAPISamples* folder.
7. Open the *Catalog-API* folder, and execute the *Import Catalogs* call for each .zip file that contains your migrated catalog data (you can specify the file to be imported on the **Body** tab in Postman).
8. In the **Collections** pane, navigate to the *InventoryAPISamples* folder.
9. Open the *Inventory-API* folder, and execute the *Import Inventory Sets* call for each .zip file that contains your migrated inventory data (you can specify the file to be imported on the **Body** tab in Postman).

1.7 Verify the data import

Once you have imported your migrated catalog and inventory data, you can verify the import by viewing the data in the Commerce Business Tools.

1.8 Migrate Commerce Server Profiles data

This section provides instructions on how to migrate profiles data from a Sitecore Commerce 8.2 deployment to a Sitecore XC 9.0 Update-2 deployment.

You use the `Customers.CsMigration` plugin included in the Commerce Engine SDK to migrate the profile data.

To migrate profiles data:

1. Navigate to the *Plugin.Sample.Customers.CsMigration\Policies* folder (where you extracted the Sitecore.Commerce.Engine.SDK zip file).
2. Open the `ProfilesSqlPolicy.cs` file in a text editor and update the ProfilesSqlPolicy to point to the CS Profiles database on your Sitecore Commerce 8.2 sytem.
3. Open the `ProfilePropertiesMappingPolicy.cs` file and add any necessary custom UserObject and Address properties.
4. Open Postman, and in the **Collections** pane in Postman, navigate to the *Authentication* folder.
5. Open the *Sitecore* folder and execute the *GetToken* call.

   When Postman displays an access token in the **Body** pane, authentication is successful.
6. In the **Collections** pane, navigate to the *SitecoreCommerce_DevOps* folder.
7. Open the *1 Environment Bootstrap* folder, and execute the *Bootstrap Sitecore Commerce* call.
8. In the **Collections** pane, navigate to the *CustomersAPISamples* folder.
9. Open the *API* folder, and execute the *MigrateCS Customers* call.

Note
The Profiles Migration tool migrates customer data, but passwords associated with each customer are not migrated. You must develop your own solution to allow customers to reset their passwords on the new site.

Chapter 2 Appendix: Catalog Mapping file

The Sitecore XC Data Migration tool uses a mapping file to convert Commerce Server XML data to Sitecore XC 9.0 JSON data.

.NET types

The Catalog Mapping file defines the .NET types for the catalog and sellable item entities used in the Commerce Engine.

```
$type : Sitecore.Commerce.Migration.CatalogMappings, Sitecore.Commerce.Migration

CategoryType : Sitecore.Commerce.Plugin.Catalog.Category,
               Sitecore.Commerce.Plugin.Catalog, Version=2.2.0.0, Culture=neutral,
               PublicKeyToken=null

SellableItemType : Sitecore.Commerce.Plugin.Catalog.SellableItem,
                   Sitecore.Commerce.Plugin.Catalog, Version=2.2.0.0, Culture=neutral,
                   PublicKeyToken=null
```

If you are using a custom implementation for your category or sellable item entities, you should extend the default Sitecore.Commerce.Plugin.Catalog.Category (see the Developer's Guide).

If you have your own category or sellable item implementation, you must substitute the location of the category or sellable item class in the Catalog Mapping file. For example:

```
CategoryType : MyCompany.MyCategory, MyCompany.Plugin.MyCateogry, Version=9.9.0.0,
               Culture=neutral, PublicKeyToken=null

SellableItemType : MyCompany.MySellableItem, MyCompany.Plugin.MyCatalog,
                   Version=9.9.0.0, Culture=neutral, PublicKeyToken=null
```

Mappings

The Catalog Mapping file also includes the mapping definitions for converting Sitecore Commerce XML attributes to Commerce Engine JSON entities.

Each mapping definition instructs the the migration tool to take a specific Commerce Server element or attribute and convert the data to a target Commerce Engine entity. The target entity is the Commerce Engine implementation that represents the Commerce Server element that is being converted.

For example:

- a Commerce Server Category element is converted to a Commerce Engine Category entity (defined by CategoryType)
- a Commerce Server Product element is converted to a Commerce Engine SellableItem entity (defined by SellableItemType)
- a Commerce Server ProductVariant element is converted to a Commerce Engine ItemVarationComponent entity

Each mapping definition includes the following information:

- type of mapping (for example, attribute to property)
- XML element path and attritube name in the source file
- property name in the destination JSON file
- data type for the new property name

For example, the following mapping definition converts a Commerce Server XML attribute to a Sitecore XC 9.0 JSON entity's property. The mapping converts the "lastmodified" attribute of a Category element to a JSON entity property called "DateUpdated" (of data type "utcdateoffset").

```
$type : Sitecore.Commerce.Migration.AttributePropertyMapping, Sitecore.Commerce.Migration
PropertyName : DateUpdated
ElementPath : Category
AttributeName : lastmodified
ConversionType : utcdateoffset
```

The following table summarizes the default mapping definitions provided with the Data Migration tool.

Mapping Definition	**Description**
AttributeIgnoreMapping	Ignores the specified Commerce Server attribute so that the data is not converted to a Commerce Engine entity. **Example**: The Commerce Server "PrimaryParentCategory" attribute of a Category element is not mapped because the Commerce Engine data structure does not implement the concept of primary parent categories.
AttributePropertyMapping	Converts a Commerce Server attribute to the property of a Commerce Engine entity. **Example**: The Commerce Server "Brand" attribute of a Product element is converted to the "Brand" property of a Commerce Engine sellable item entity.
ParentCategoryElementMapping	Converts a Commerce Server "ParentCategory" element to the Commerce Engine relationships that define the catalog hierarchy (CatalogToCategory, CatalogToSellableItem, CategoryToCategory, CategoryToSellableItem). **Example**: The Commerce Server ParentCategory for a Product element is converted to a parent-child hierarchy relationship for the equivalent Commerce Engine sellable item entity.
RelationshipElementMapping	Converts the Commerce Server representation of relationships into Commerce Engine relationships. **Example**: A Commerce Server relationship named "CrossSell" is converted to a Commerce Engine relationship. If the Commerce Engine does not include a realationship definition for the Sitecore Commerce relationship, the tool creates a new relationship.

LocalizedElementMapping	Converts the Commerce Server representation of a localized property to the Commerce Engine representation of a localized property. In Commerce Engine, localized properties are represented by a separate LocalizationEntity that is linked to a SellableItem or Category. This mapping creates the LocalizationEntity and links to the target Commerce Engine entity. If necessary, the tool sets the default value for the Commerce Engine entity property based on the default language defined in the Commerce Server catalog (for example, <Catalog name="Adventure Works Catalog" DefaultLanguage="en-US">). **Example**: All localized "DisplayName" elements of Commerce Server Product elements are mapped to the localized Commerce Engine "SellableItem.DisplayName" property.
AttributeComponentMapping	Converts a Commerce Server attribute to a Commerce Engine entity component. **Example:** The "Variant_Images" attribute of a Commerce Server ProductVariant element is mapped to Images property of a Commerce Engine component.
AttributeComponentChildComponentsMapping	Converts a Commerce Server attribute to the property of a Commerce Engine entity component's child component. **Example**: The Commerce Server "Definition" attribute of a Product element is mapped to the "ItemDefinition" property of a Commerce Engine SellableItem CatalogsComponent/CatalogComponent component.
AttributePolicyMapping	Converts a Commerce Server attribute to a Commerce Engine entity policy. **Example**: The Commerce Server "listprice" attribute of the Product element is mapped to the "ListPricingPolicy" of a Commerce Engine Sellable Item.
VariationElementMapping	This mapping is unique because it contains child mappings that are only applied to the ItemVariationComponent of migrated SellableItem entities. Since Commerce Server Product Variants can be extended, the child mappings provide a way to convert those Commerce Server variant customizations to Commerce Engine entities.

AttributeIgnoreMapping	Ignores the specified Commerce Server attribute so that the data is not converted to a Commerce Engine entity. **Example**: The Commerce Server "PrimaryParentCategory" attribute of a Category element is not mapped because the Commerce Engine data structure does not implement the concept of primary parent categories.

If you want to implement your own mapping definition, you must create a new entry in the Mappings collection with a `$type` that points to your mapping implementation.

Conversion types

Most mappings include a ConversionType property that describes how to convert a value from a string in the Commerce Server XML file to the value expected by a Commerce Engine entity property or constructor parameter.

The following conversion types are defined for the Data Migration tool:

- **none**: indicates that the Commerce Server value is not used by the mapping. For example, in the AttributeIgnoreMapping mapping, the mapping does not use any value conversion.
- **string**: maps the Commerce Server value exactly as represented in the Commerce Server XML (since the target Commerce Engine property is a string).
- **utcdateoffset**: converts the Commerce Server value to a DateTimeOffset and converts the value from local time to UTC time (in Commerce Engine, all dates are represented in UTC time).
- **dateoffset**: same as utcdateoffset but does not convert the Commerce Server value to UTC time.
- **guidlist**: converts the Commerce Server value (represented as a "|"-delimited list of GUID values) to a List<string>.
- **taglist**: converts a Commerce Server string into a list of Commerce Engine Tag objects (only used in Product and ProductVariation mappings).
- **money**: converts the Commerce Server value to a Commerce Engine Money object.
- **moneylist**: converts the Commerce Server valuie to a list of Commerce Engine Money objects (List<Money>).
- **boolean**: converts the Commerce Server value into a .NET bool value (handles values "0", "1", "true", "false" or any other value recognized by bool.Parse()).
- **integer**: converts the Commerce Server value into a .NET System.Int32 value.

Chapter 3 Appendix: Large catalog imports

Catalog and Inventory Import file sizes are limited to a maximum of 2GB by the Commerce Engine Service. The maximum size for Catalog and Inventory Import can further be limited for security reasons by modifying the `requestLimits maxAllowedContentLength` value.

To modify configuration for large catalog imports:

1. Navigate to the Commerce Authoring service folder and open the `web.config` file.
2. Change the value of the `maxAllowedContentLength` parameter to your desired maximum in bytes:

```
<configuration>
  <system.webServer>
    <security>
      <requestFiltering>
        <requestLimits maxAllowedContentLength="2147483648" />
      </requestFiltering>
    </security>
  </system.webServer>
</configuration>
```