

Hunk Catalog for Sitecore Commerce

Plugin for Sitecore Commerce that does the heavy lifting of importing an entity in Sitecore Commerce and allows you to write simple and maintainable custom import implementations.

**What is this?**

In most of Sitecore Commerce project you need to write custom catalog import code based on your specific requirements and catalog information in your PIM. Without a standard architecture each team does it in different way and it becomes hard to maintain such implementations in a longer time period. The idea with this plugin is to create a standard framework for import processes that can be utilized in any commerce implementation without compromising on flexibility and extensibility of Sitecore commerce architecture and also gives you type safe way to write your custom implementation.

Easy Entity Importer will give you ability to focus on your entity designs rather than import implementation. It makes you free to write lots of boilerplate code during the import process.

It supports following:

1. Creation and update of Catalog, Category and SellableItem commerce entities.
2. Creation and update of any custom commerce entities.
3. Adding, updating and removing child components on commerce entities.
4. Managing parent child relationships between catalog, category and sellable items.
5. Creation and update of item variants for sellable items.
6. Adding, updating and removing child components on item variants.
7. Managing various relationships e.g. related, cross sell on sellable items.
8. Localized content for commerce entities.
9. Localized content for commerce entities child components.
10. Localized content for item variants.
11. Localized content for item variant child components.
12. Configure entity versioning scheme.
13. Easily customizabe using Policies or custom pipeline blocks.

Note: For localization to work, make sure to set proper localization policies, please refer PlugIn.LocalizeEntities.PolicySet-1.0.0.json file.

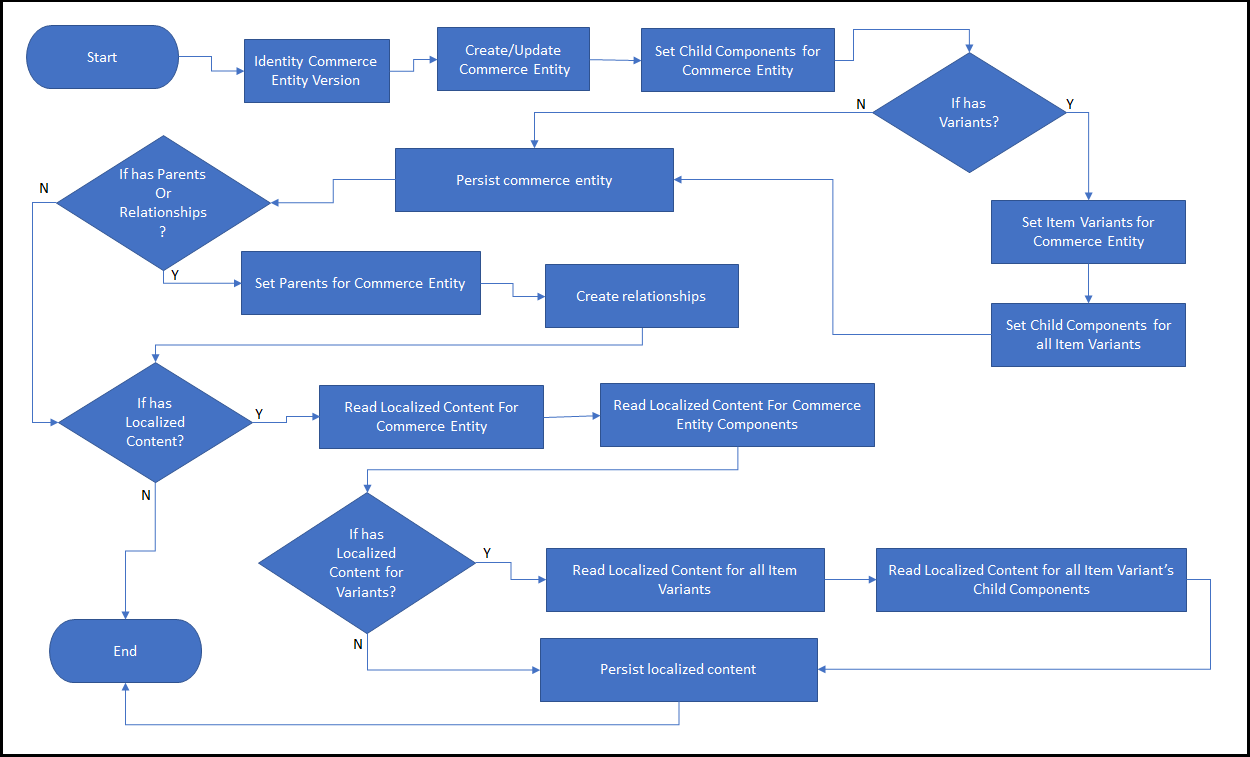
Overall it involves following:

1. Create your custom POCO classes defining your source entities.
2. Create custom Mapper classes by inheriting from standard implementation in the plugin. In mapper classes you will write simple mapping code where you will read values from your custom source entities and write to Sitecore commerce entities or Sitecore commerce components.
3. All standard mapper classes have basic implementation for most of the methods, you just need to override methods that suits your requirements.
4. Configure your mapper class in the policy values   
   or   
   create your own pipeline blocks to resolve mapper instances and replace OOTB pipeline blocks with your custom blocks.

That’s it. Once your solution is deployed you can try the import operation using Postman calls.

Entity Importer Flow chart

Below flow chart describes the overall import process for a single commerce entity.



Import operation flow:

1. Receives metadata information for the importable entity in “ImportEntity” API call.   
   Metadata information includes following:
   1. EntityType: A string value that works as an identifier for commerce entity mapper class.
   2. Components: A list of component names applicable for commerce entity. Each component name works as an identifier to identify a corresponding component mapper type.
   3. VariantComponents: A list of child component names applicable for item variant commerce component. Each component name works as an identifier to identify a corresponding mapper type.
   4. Serialized Entity: full custom source entity details in serialized form.
2. Resolves instance of entity mapper type based on the entity metadata information.
3. Deserializes source serialized entity in a type safe instance.
4. Validates that entity can be imported, if yes continues to next step otherwise aborts the process.
5. Creates or updates commerce entity.
6. Loops through all component names from metadata.
   1. For each component name resolves a component mapper instance.
   2. Maps component detail from source entity.
   3. Sets/removes commerce component on commerce entity.
7. Checks if source entity supports variants, if yes than loops through all variants in source entity.
   1. For each variant resolve item variant mapper instance.
   2. Map item variant component from source entity and source variant details.
   3. Sets/removes item variant component on commerce entity.
   4. Loops through all variant component names from metadata information.
      1. For each variant component name resolve a variant component mapper instance.
      2. Map component detail from source entity and source variant details.
      3. Sets/removes component on item variant component.
8. Checks if source entity supports localization content, if yes loops through all language specific content.
   1. For each language resolve entity localization mapper type.
   2. Reads localization values for entity.
   3. Loop through all child components on commerce entity.
      1. For each child component resolve a component localization mapper type.
      2. Reads localization values for entity components.
   4. Check if localization entities supports variants, if yes loop through for all variants
      1. For each variant resolve item variant localization mapper instance.
      2. Reads localization values.
      3. Loop through all variant child components.
         1. For each variant child component resolve child component localization mapper type.
         2. Reads localization values.
   5. Once all localization values are read, get localization entity for your commerce entity.
   6. Update localized values for your commerce entity and all your components in the localization entity and persist it.

## How to use it?

Add a dependency on Plugin.EasyImporter to the plugin that contains your catalog components:

* From the package manager console: Install-Package Plugin.EasyImporter
* Using the Nuget package manager add a dependency on Plugin.EasyImporter.

## Getting started

Let’s first add a separate plugin to your commerce solution.

This easy entity importer plugin will allow you to import any entity so let’s start from importing a catalog entity.

### Importing a catalog entity

1. Create a source catalog entity class, see example below.

|  |
| --- |
| namespace Plugin.EasyEntityImporter.CatalogImport.Sample.Entity  {  public class SourceCatalog : IEntity  {  public SourceCatalog()  {  this.Languages = new List<LanguageEntity<SourceCatalog>>();  }  [EntityId()]  public string Id { get; set; }  public string Name { get; set; }  public string DisplayName { get; set; }  public string Description { get; set; }  [Languages()]  public IList<LanguageEntity<SourceCatalog>> Languages { get; set; }  }  } |

* Make sure to inherit your class with IEntity interface.
* Make sure to have a Id attribute similar above, type of property should be string and it must be decorated with EntityId attribute.
* If you want to support localization content make sure to create a Languages property as shown in above class.

1. Create an entity import handler class like below:

|  |
| --- |
| namespace Plugin.EasyEntityImporter.CatalogImport.Sample.EntityImportHandlers  {  public class SourceCatalogImportHandler : CatalogImportHandler<SourceCatalog>  {  public SourceCatalogImportHandler(string sourceCatalog,  CommerceCommander commerceCommander,  CommercePipelineExecutionContext context)  : base(sourceCatalog, commerceCommander, context)  {  }  protected override void Initialize()  {  this.Name = this.SourceEntity.Name;  this.DisplayName = this.SourceEntity.DisplayName;  }  public override void Map()  {  this.CommerceEntity.Name = this.SourceEntity.Name;  this.CommerceEntity.DisplayName = this.SourceEntity.DisplayName;  }  protected override void MapLocalizeValues(SourceCatalog localizedSourceEntity, Sitecore.Commerce.Plugin.Catalog.Catalog localizedTargetEntity)  {  localizedTargetEntity.DisplayName = localizedSourceEntity.DisplayName;  }  }  } |

* Make sure to inherit your class from CatalogImportHandler and provide your source catalog entity class name as type argument.
* Make sure to write your constructor like example above.
* You can access your source entity using SourceEntity property. See the use of property in Initialize and Map methods above.
* You must override Initialize method and set name and display name for your Sitecore commerce catalog, this method will be used during catalog entity creation process.
* Override Map method to update catalog commerce entity details with your source entity details during the update process.
* Override MapLocalizeValues method to read localize content from your source catalog entity.

1. Configure your custom import handler type in policy.

* Define CatalogImportPolicy.
* Configure custom entity import handler type in entity mappings. See sample below

|  |
| --- |
| {  "$type": "Plugin.EasyEntityImporter.Policy.CatalogImportPolicy, Plugin. EasyEntityImporter",  "Mappings": {  "EntityMappings": [  {  "$type": "Plugin.EasyEntityImporter.Policy.EntityMapperType, Plugin.EasyEntityImporter",  "Key": "Catalog",  "ImportHandlerTypeName": "Plugin.EasyEntityImporter.Sample.EntityImportHandlers.SourceCatalogImportHandler, Plugin.EasyEntityImporter.Sample"  }  ]  }  } |

* Make sure to setup the policy appropriately with your commerce engine environment.
* Make sure to bootstrap commerce engine.

1. (Alternate way) Create a pipeline block to resolve your entity import handler type.

* Create a new pipeline block class

|  |
| --- |
| namespace Plugin.Accelerator.CatalogImport.Sample.Pipelines.Blocks  {  [PipelineDisplayName("CustomResolveEntityImportHandler")]  public class CustomResolveEntityImportHandlerBlock : PipelineBlock<ResolveEntityImportHandlerArgument, IEntityImportHandler, CommercePipelineExecutionContext>  {  private readonly CommerceCommander \_commerceCommander;  public CustomResolveEntityImportHandlerBlock(  CommerceCommander commerceCommander)  {  this.\_commerceCommander = commerceCommander;  }  public override Task<IEntityImportHandler> Run(ResolveEntityImportHandlerArgument arg, CommercePipelineExecutionContext context)  {  var importHandler = new SourceCatalogImportHandler(arg.ImportEntityArgument.SourceEntityDetail.SerializedEntity, \_commerceCommander, context) as IEntityImportHandler;  return Task.FromResult(importHandler);  }  }  } |

* Configure the pipeline block and replace the OOTB implementation.

|  |
| --- |
| public class ConfigureSitecore : IConfigureSitecore  {  public void ConfigureServices(IServiceCollection services)  {  var assembly = Assembly.GetExecutingAssembly();  services.RegisterAllPipelineBlocks(assembly);  services.Sitecore().Pipelines(config => config  .ConfigurePipeline<IResolveEntityImportHandlerPipeline>(c =>  {  c.Replace<ResolveEntityImportHandlerBlock, CustomResolveEntityImportHandlerBlock>();  })  );  services.RegisterAllCommands(assembly);  }  } |

* Deploy commerce engine.

### Importing a category entity

1. Create a source category entity class, see example below.

|  |
| --- |
| namespace Plugin.EasyEntityImporter.CatalogImport.Sample.Entity  {  public class SourceCategory : IEntity  {  public SourceCatalog()  {  this.Parents = new List<string>();  this.Languages = new List<LanguageEntity<SourceCatalog>>();  }  [EntityId()]  public string Id { get; set; }  public string Name { get; set; }  public string DisplayName { get; set; }  public string Description { get; set; }  [Parents()]  public IList<string> Parents { get; set; }  [Languages()]  public IList<LanguageEntity<SourceCatalog>> Languages { get; set; }  }  } |

* Make sure to inherit your class with IEntity interface.
* Make sure to have an Id attribute similar above, type of property should be string and it must be decorated with EntityId attribute.
* Make sure to add a Parents property, this is needed as Commerce category entity should be assigned to a catalog and in this property you need to send catalog name. This property should return a list of strings and must be decorated with Parents attribute.
* If you want to support localized content, make sure to create a Languages property as shown in above class.

1. Create an entity import handler class like below:

|  |
| --- |
| namespace Plugin.EasyEntityImporter.CatalogImport.Sample.EntityImportHandlers  {  public class SourceCategoryImportHandler : CategoryImportHandler<SourceCategory>  {  public SourceCatalogImportHandler(string sourceCatalog,  CommerceCommander commerceCommander,  CommercePipelineExecutionContext context)  : base(sourceCatalog, commerceCommander, context)  {  }  protected override void Initialize()  {  this.Name = this.SourceEntity.Name;  this.DisplayName = this.SourceEntity.DisplayName;  }  public override void Map()  {  this.CommerceEntity.Name = this.SourceEntity.Name;  this.CommerceEntity.DisplayName = this.SourceEntity.DisplayName;  }  protected override void MapLocalizeValues(SourceCatalog localizedSourceEntity, Sitecore.Commerce.Plugin.Catalog.Catalog localizedTargetEntity)  {  localizedTargetEntity.DisplayName = localizedSourceEntity.DisplayName;  }  }  } |

* Make sure to inherit your class from CategoryImportHandler and provide your source category entity class name as type argument.
* Make sure to write your constructor like example above.
* You can access your source entity using SourceEntity property. See the use of property in Initialize and Map methods above.
* You must override Initialize method and set name and display name for your Sitecore commerce catalog, this method will be used during catalog entity creation process.
* Override Map method to update catalog commerce entity details with your source entity details during the update process.
* Override MapLocalizeValues method to read localize content from your source catalog entity.

1. Configure your custom import handler type in policy.

* Define CatalogImportPolicy.
* Configure custom entity import handler type in entity mappings. See sample below

|  |
| --- |
| {  "$type": "Plugin.EasyEntityImporter.Policy.CatalogImportPolicy, Plugin. EasyEntityImporter",  "Mappings": {  "EntityMappings": [  {  "$type": "Plugin.EasyEntityImporter.Policy.EntityMapperType, Plugin.EasyEntityImporter",  "Key": "Catalog",  "ImportHandlerTypeName": "Plugin.EasyEntityImporter.Sample.EntityImportHandlers.SourceCatalogImportHandler, Plugin.EasyEntityImporter.Sample"  }  ]  }  } |

* Make sure to setup the policy appropriately with your commerce engine environment.
* Make sure to bootstrap commerce engine.

1. (Alternate way) Create a pipeline block to resolve your entity import handler type.

* Create a new pipeline block class

|  |
| --- |
| namespace Plugin.Accelerator.CatalogImport.Sample.Pipelines.Blocks  {  [PipelineDisplayName("CustomResolveEntityImportHandler")]  public class CustomResolveEntityImportHandlerBlock : PipelineBlock<ResolveEntityImportHandlerArgument, IEntityImportHandler, CommercePipelineExecutionContext>  {  private readonly CommerceCommander \_commerceCommander;  public CustomResolveEntityImportHandlerBlock(  CommerceCommander commerceCommander)  {  this.\_commerceCommander = commerceCommander;  }  public override Task<IEntityImportHandler> Run(ResolveEntityImportHandlerArgument arg, CommercePipelineExecutionContext context)  {  var importHandler = new SourceCatalogImportHandler(arg.ImportEntityArgument.SourceEntityDetail.SerializedEntity, \_commerceCommander, context) as IEntityImportHandler;  return Task.FromResult(importHandler);  }  }  } |

* Configure the pipeline block and replace the OOTB implementation.

|  |
| --- |
| public class ConfigureSitecore : IConfigureSitecore  {  public void ConfigureServices(IServiceCollection services)  {  var assembly = Assembly.GetExecutingAssembly();  services.RegisterAllPipelineBlocks(assembly);  services.Sitecore().Pipelines(config => config  .ConfigurePipeline<IResolveEntityImportHandlerPipeline>(c =>  {  c.Replace<ResolveEntityImportHandlerBlock, CustomResolveEntityImportHandlerBlock>();  })  );  services.RegisterAllCommands(assembly);  }  } |

* Deploy commerce engine.