Advanced Category Selector

Installation:

1. Import the AdvancedCategorySelectorX_X.zip into your site.
2. **You must check** “Import Code Files” and “Import global folders.”
3. After import, you must resign your macros in order for the control to render properly. See Kentico Documentation on resigning your macros, or go to System → Macros → Signatures, check “Sign all macros” and click “Update macro signatures.”

The Advanced Category Selector should be the last Category Selector you'll ever need. It's highly configurable and capable of handling multiple different scenarios. Below I will outline the options and what they do.

- **Root Category**
  - The root of where the categories you wish to select reside. Useful if you have categories structured in the Category Tree by category type. Note you can further extend what Categories show through the “Where” condition, discussed later.
- **Category Display Mode**
  - There are 2 ways to select your categories, a searchable List, and a structured Tree (the Category Tree).
  - Tree Options
    - **Expand to Level**: You can have your Category Tree expanded to the nth level by default. Useful if you have a lot of categories in a tree and you wish to have some nodes collapsed.
    - **Only Leaf Selectable**: If checked, only the bottom level categories (leaf on the tree) can be selected. This is useful if parent Categories are more for structure and less for actual categories you wish to apply.
    - **Parent Selects Children**: If Only Leaf Selectable is true, and this is true, the parent nodes can be 'checked' but they simply act as a way to select all / deselect all their descendent leaf nodes.
- **Minimum / Maximum Categories**: Can control what number of categories can be selected, great if you wish to have only X amount or at least X categories.
- **Separator Character**: For saving to Field, you can control what character acts as the delimeter.
- **Save Mode**: The Advanced Category Selector can save the Categories in a variety of ways.
  - **Set Document Categories**: If selected, the control will not set the field value, but will assign Categories to the current document. This can only be used on Documents/Pages
  - **Set Field's Value with Category Names (Bar Separated)**: If selected, the control will set the field's value to the Categories' Field Save Type (discussed later). This will not assign the document to the categories.
    - **Allow Manual Entry (To Field Only)**
      - If checked, the user can type in the values instead of using the List/Tree selector.

    - **Set both Document Categories and Field's Value**: A combination of the previous two, will both set the Field's values to the Categories' Field Save Type, along with assign the document to the category. Note that if the Document Category changes, the field value won't change until you go to the form and hit save (the tool will prioritize the Document Categories over the Field values)
    - **To Joining Table**: Save the Object to Category Relationship into a Join Table.
        - Note: You can use Custom Tables without any customization, but in order to use Module Classes you must adjust the Form Control's code, discussed in a later section.
        - **Join Table This Object Foreign Key**: The Foreign Key that you will place in the joining table. This must be a value available to the form, such as DocumentID/Guid and NodeID/Guid for Documents.
        - **Join Table Name**: The Class name of either the Custom Table or Module Class.
        - **Join Table Left Field Name**: The Field name that stores the Object's foreign key.
        - **Join Table Right Field Name**: The Field name that stores the Category's foreign key.
        - **Join Table GUID Field Name**: If your table has a GUID, you can place the field name here so a new GUID will be assigned to it upon creating a new row.
        - **Join Table Last Modified Field Name**: If your table has a Last Modified field, you can place the field name here so the current date will be assigned to it upon creating a new row.
        - **Join Table Code Name Field Name**: If your table has a Code Name field, you can place the field name here so the tool will place a generated code name in the format of [TableClass]_[ObjectForiegnKey]_[CategoryForeignKey]
        - **Join Table SiteID Field Name**: If your table has a Site ID field, you can place the field name here so the tool will place the current Site ID in it upon creating a new row.
- **Field Save Type**: What Category Identifier you wish to use for the value of the control, or stored in the join table. Your options are Category ID, Category GUID, and Category Name.
- **Where Filter**: You can further control which categories are available by setting a Where condition that will be run against the CMS_Category table.
- **Order By**: You can determine the order the Categories Display. By default it will just use the CategoryOrder for Trees, and Category Display Name for lists.

Integrating the Advanced Category Selector with a Custom Module Class

Custom Module classes are very flexible, and offer the ability to translate one environment's IDs to another environment's IDs when properly configured. Since IDs are the fastest way to look up data, joining tables done through these can product fast queries.

However the synchronization on Custom Module Classes is too intrinsic to be able to generate dynamically, so you must use your own ModuleInfo and ModuleInfoProvider classes

that can be generated through Kentico in order to operate.

Below are the steps to set up a Joining Table with Staging Capabilities.

Set up a Custom Module Class w/ Staging Support

1. Create a Custom Module (Modules → New)
    - For our example, I set the Module name to “My Custom Module” and Module Code Name to “MyCustomModule”
2. Go to “Classes” and Create a new Class
    - For our example, I set the Class Display Name to “My Joining Class”, Namespace to “MyNamespace” and Class to “MyJoiningClass”
3. ***Do NOT check “Is M:N,” as this renders the ID column non-auto incrementing and can throw errors when inserting.***
4. Create a field to hold the Object ID (in our case “DocumentID”), set the type to “int”, make required, and “Reference to” to what it references, in our case “Page” (“Document” in 8.0)
5. Create a field to hold the Category ID (in our case “CategoryID”), set the type to “int”, make required, and “Reference to” to what it references, in our case “Content Category”
6. Additionally, you may define a SiteID (int) with Reference to “Site”, CodeName (text), GUID, and Last Modified fields.
7. Finish creating your Class (hitting next till done).
8. On the left menu, you should now see “Code,” click it and set any System columns (Display Name, Code Name, GUID, “Last Modified”, Binary, SiteID) and adjust the Namespace if you wish. Hit “Generate Code” and lastly hit “Save Code” which will save the code to your App_Code/Modules folder.
9. Let the site reload.
10. To enable Staging, go to the Info class you generated (example App_Code/CMSModules/MyCustomModule/MyJoiningClassInfo.cs)
11. Adjust the TypeInfo portion, see documentation for 8.0/8.1/8.2, 9, and 10.
12. Remove the ObjectDependency for the SiteID, it's not needed as as long as the TypeInfo has the SiteID column set as it's SiteID definition, it handles it. It looks like this:
    1. ***new ObjectDependency("SiteID", "cms.site", ObjectDependencyEnum.Required),***
    2. Your new TYPEINFO should look like this:

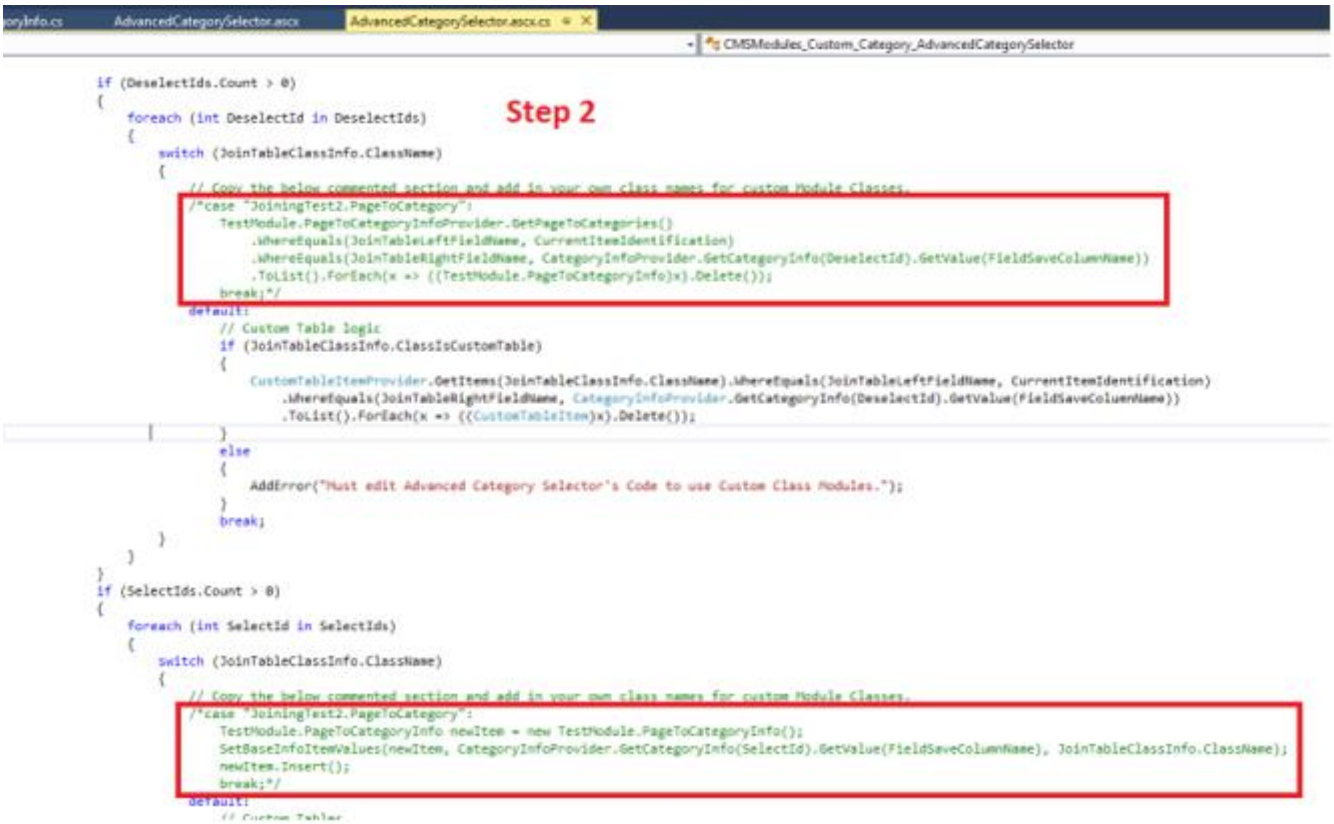
```
/// <summary>
/// Type information.
/// </summary>
public static ObjectTypeInfo TYPEINFO = new ObjectTypeInfo(typeof(MyJoiningClassInfoProvider), OBJECT_TYPE, "MyNamespace.MyJoiningClass", "MyJoiningClassID",
                                                           "MyJoiningClassLastModified", "MyJoiningClassGUID", "CodeName", null, null, "SiteID", null, null)
{
    ModuleName = "MyCustomModule",
    DependsOn = new List<ObjectDependency>()
    {
        new ObjectDependency("DocumentID", "cms.document", ObjectDependencyEnum.Required),
        new ObjectDependency("CategoryID", "cms.category", ObjectDependencyEnum.Required)
    },
    LogSynchronization = SynchronizationTypeEnum.LogSynchronization, // Enables logging of staging tasks for changes made to this object
    SynchronizationObjectTreeLocations = new List<ObjectTreeLocation>()
    {
        // Creates a new category in the 'Global objects' section of the staging object tree
        new ObjectTreeLocation(SITE, "PageToCategory")
    }
};
```

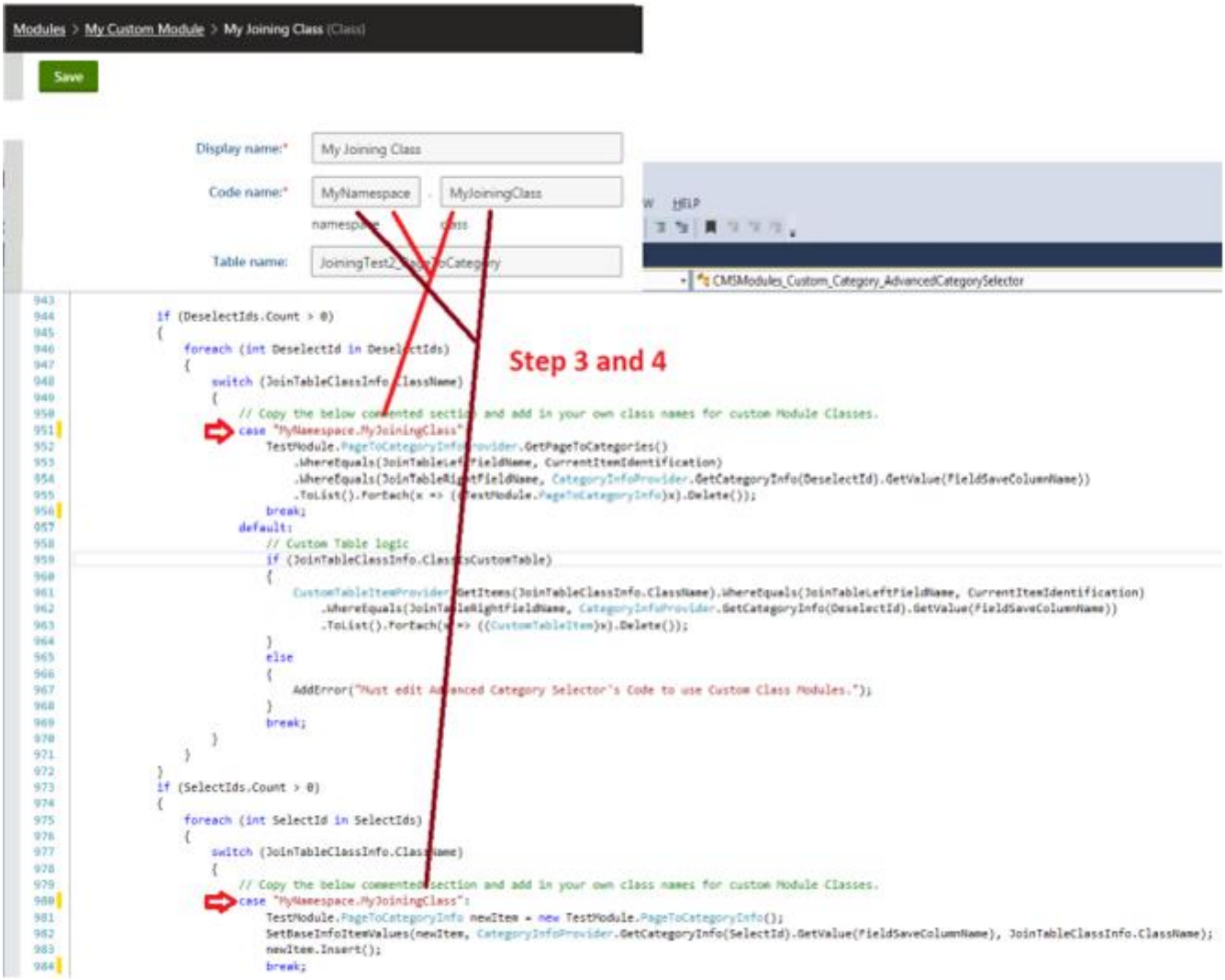
13. Let the site reload.

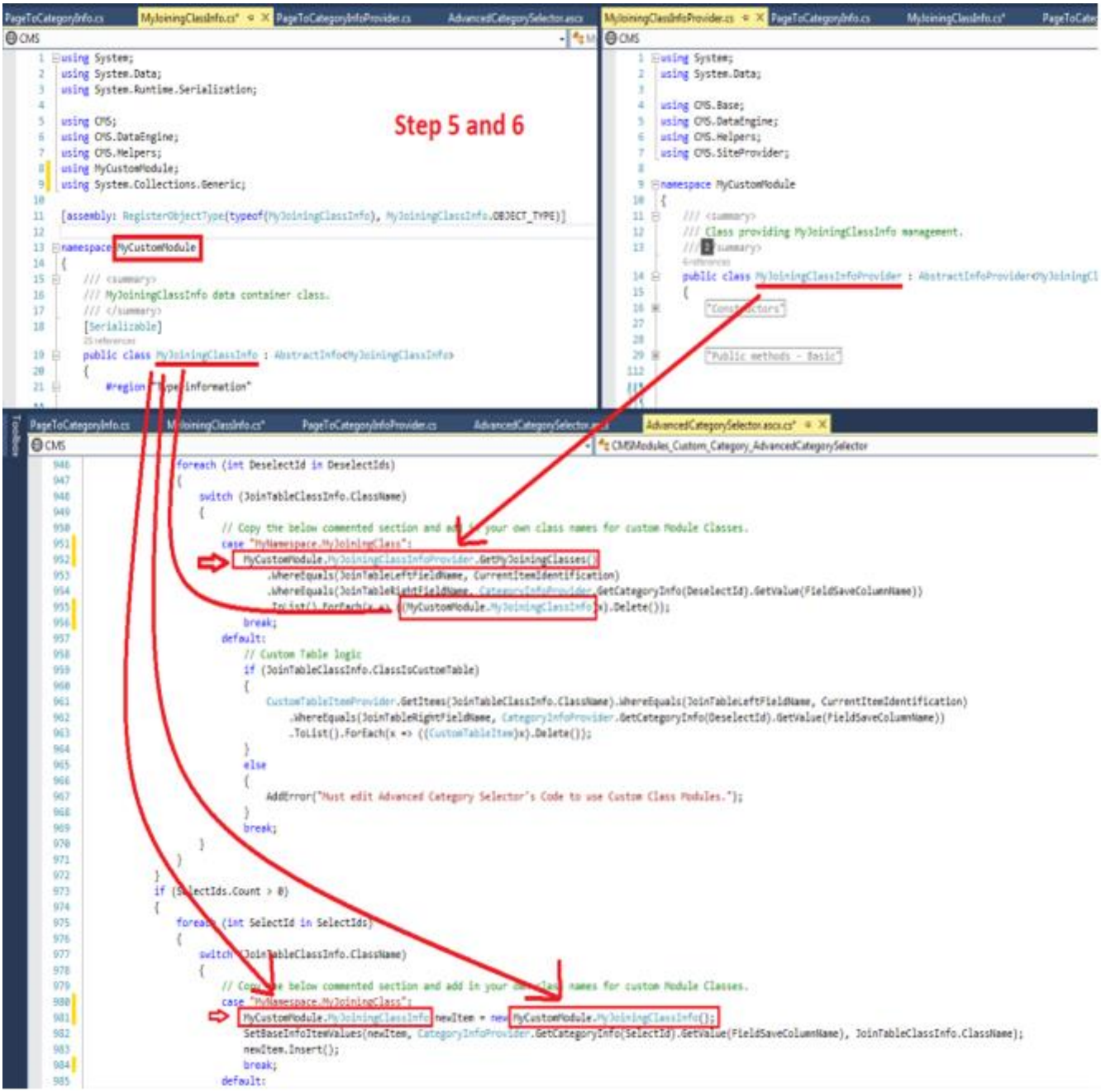
Configure the Advanced Category Selector to allow Joining on this Custom Module Class

(Screenshots below instructions to aid you)

1. Open the form control code file (~/CMSFormControls/Custom/AdvancedCategorySelector/AdvancedCategorySelector.ascx.cs)
2. Look for the commented out sections in the Form_OnAfterSaveJoinTable method (search for "// Copy the Below commented"). There are 2, one for Deleting, one for Saving
3. Uncomment or copy the case sections.
4. Replace the: case "JoiningTest2.PageToCategory" with your Custom Module Classes' Code Name.
5. Replace the TestModule.PageToCategoryInfoProvider.GetPageToCategories and TestModule.PageToCategoryInfo with the new classes that were generated in Kentico (for example MyCustomModule.MyJoiningClassInfoProvider.GetMyJoiningClasses() and MyCustomModule.MyJoiningClassInfo)
6. Replace the TestModule.PageToCategoryInfo and new TestModule.PageToCategoryInfo() with the new classes that were generated in Kentico (for example, MyCustomModule.MyJoiningClassInfo and new MyCustomModule.MyJoiningClassInfo()). in the Insert section.




Modules > My Custom Module > My Joining Class (Class)
Save
Display name:*
My Joining Class
Code name:*
MyNamespace
MyJoiningClass
Table name:
CMSModules_Custom_Category_AdvancedCategorySelector
Step 3 and 4
// Copy the below commented section and add in your own class names for custom Module Classes.
case "MyNamespace.MyJoiningClass":
// Custom Table logic
AddError("Must edit Advanced Category Selector's Code to use Custom Class Modules.");
// Copy the below commented section and add in your own class names for custom Module Classes.
case "MyNamespace.MyJoiningClass":



It may seem like a lot of work, but you are only editing about 5 lines of code. Now when you do a join table using that Custom Module Class, it will use it's InfoProvider and Info class to insert/delete, which will then trigger any Staging tasks and proper bindings.