Relationships Extended Documentation

In this document I will cover the various tools, user interface templates, helper methods, and other goodies that the Relationships Extended provides you. Keep in mind that I have been diligent in adding Field Descriptions to things such as the Form Controls, User Interface Properties, and code comments on methods, and to reproduce that in a text document seems pointless. Instead I'll be highlighting and categorizing the tools and giving some helpful information on them.

Quick Navigation

Helper Methods

The RelationshipsExtended module comes with a RelHelper class which provides helper methods for leveraging various features of the RelationshipsExtended.

Where Condition Generators

In some cases (such as categorization) often the bound relationships are meant as a means of tagging and filtering objects. For example, you may have Regions on a Banner and want to show only Banners in the current user's region or regions. In SQL the where condition or join can be tedious. You can leverage the following methods to generate the Where Conditions for you.

Each of these Where Condition methods features a couple options:

1. Automatic Conversion of Code Name, GUID, or IDs passed to it into ID lookups to optimize performance in queries.
2. Multiple Condition Types: (Matching Any of the items, All of the items, or none of the items passed)
3. Automatic "1=1" if no filter objects are passed to it or some misconfiguration occurs, so it will not break your where condition.
4. Each of these also comes with a Macro version through the RelHelper macro namespace.

These methods are:

- GetDocumentCategoryWhere (For Document Categories)
- GetNodeCategoryWhere (For Node Categories used with the CMS.TreeNode class that comes with this module)
- GetBindingCategoryWhere (For Object to Category bindings)
- GetBindingWhere (For any Object to Object binding)

Node-Binding Staging Task Handlers

Unlike normal Object to Object Binding classes, where you can simply set the SynchronizationSetting to TouchParent, Node to Object binding is trickier and requires a lot of manual processing if you wish to have a Node's update task contain all its related items.

But default Related Pages are already handled by Kentico, and Node Categories using the CMS.TreeNode that comes with the relationships extended also automatically are handled by my code. Any additional or custom classes need to be manually handled.

I have reduced all that complex logic into a couple lines of code thanks to some helper methods that I have created. I would refer to the Demo File's DemoInitializationModule.cs to see how each of them work and how to leverage them.

- HandleNodeBindingInsertUpdateDeleteEvent (Handles triggering the Document Update with some magic to ensure only about 1 document update is ran even if you bind a large quantity of objects)
- UpdateTaskDataWithNodeBinding (handles attaching the Bound objects to the Document Update's Task data so it can be processed)
- NewBoundObjectIDs (Used in the Staging Task Processing to convert the old object IDs into the new ones on the new environment, returning all the IDs that the node should be bound to)
- NewOrderedBoundObjectIDs (For Ordered Node Bindings, used in the Staging Task Processing to convert the old object IDs into the new ones on the new environment, returning all the IDs in order)

If you decide to not Bind the Node-Binding objects to the Node itself, there is the SetBetterBindingTaskTitle that can help convert the less-than-readable normal StagingTask into a normal one. This is used on the StagingTask.LogTask.Before hook.

CMS.TreeCategory and Node Categories

Another feature is a new class, CMS.TreeNode. This is not a Kentico created class, although I am sharing the CMS namespace as if Kentico ever did create a Tree Category, this wouldn't be needed anymore. This class operates the same as CMS.DocumentCategory, except it's attached to the Node instead of the Document, allowing you to use Categories without fear of things becoming out of sync with different localized versions of the Documents.

There is also a new UI when you edit a Page. In Properties on the page, you'll see Node Categories next to Categories, this allows you to assign any Node-category just as you would a Document category.

Advanced Category Selector and Advanced Many to Many Selector

The Advanced Category Selector form control and Advanced Many to Many selector form controls are both included with this module. The Advanced Category Selector also has a "Node Category" Save mode added which automatically integrates with the CMS.TreeCategory of the RelationshipsExtended.

Relationship Names Extended

This new User Interface (under Configuration) simply allows for the creation of AdHoc (sortable) relationship names, as by default you cannot. This is used then in conjunction with the Edit Relationship Template discussed in the next section.

User Interfaces Templates

A large part of the functionality given in the RelationshipsExtended is a handful of User interfaces which allow you to control the various binding scenarios. We'll go over them and explain some of what they do and how to use them. Remember, I won't cover every property, most are well explained when you hover over them.

Edit Relationship

This user interface allows you to manage Page Relationships (including Adhoc ones) with much greater control. The default Kentico related pages gives no control over what Page Types you can relate (you could related a Folder page for the Relationship “Banners” for example), it was clunky in its usage (you had to add pages one at a time) and it didn’t allow Adhoc relationships (orderable ones), so you couldn’t have order to the objects without making it part of a Page Type (which can get messy if multiple page types may need the same relationship, such as Banners or Quick Links).

The Edit Relationship also automatically gives a “read only” view to the “Right Side” only page types. For example, if a Banner is a Right-side only on the Relationship “Banners” then if you view the Banners UI on a Banner page, it will show you all the pages that use that banner.

Noteworthy Properties

1. Left Side Macro Condition & Right Side Macro Condition
    a. To make it fully flexible, this allows you to determine if a page is considered left side or right side. This way you can say Non-banners are allowed as Left Sides in the Banners Relationship, and only Banners are allowed on the right.
    b. The most common scenario is Left Side: {% CurrentDocument.ClassName != "The.ClassName" %} and Right Side: {% CurrentDocument.ClassName == "The.ClassName" %} where “The.ClassName” is the object you are adding as a relationship (like Banners)
2. Auto Hide
    a. If this is checked, and the Vertical Listing UI element that controls this uses the RelationshipsExtended.RelationshipVerticalTabExtender extender class, then this element will automatically hide if the current page matches neither the Left nor the Right conditions.
3. Show New Page
    a. You can configure this to allow the user to easily add a new page if they don’t see one that they wish to select.
4. Selector Type
    a. There are two selection types available for you to choose from.
    b. A UniSelector so you can easily search and find items
    c. A Tree View so you can leverage the Tree structure to better organize your content
        i. You also have two Tree Selection options
        ii. Checkbox allows you to easily add many pages at once, great for many selections
        iii. Individual allows you to click and add each one individually, great for smaller selections
5. Display Name Format and Tool Tip Format
    a. You can control the Display name of the items you are selecting (default the NodeName) to make it easier for users to find and select their related pages
    b. You can also add a Tool Tip to the items so users can get a little more information as to what they are selecting (say you have Banners and two are similar in their name, you could add a hover that shows the Banner Text and Link values)
    c. Make sure to set the Additional Columns
6. UniGrid Extensions
    a. You can select custom UniGrid extensions for both the AdHoc, Read only Ad-hoc (when on the right-hand side), or Non Ordered Relationships.

Edit Categories

Those of you who have used the Advanced Category Selector already pretty much know what this is all about. The Edit Categories is the IU version of that tool. With it you can leverage Kentico categories in any way you wish, with much more control than you get with the standard Kentico category selector (which shows the entire category tree)

Noteworthy Properties

1. Root Category

    a. You can decide where the categories start, since often you organize your categories using the category tree.
2. Category Display Mode
    a. You have two options for displaying and selecting your categories
        i. Searchable List: This shows all the applicable categories in a single list with a filter on the top that filters the items as you type.
        ii. Structure Tree: This shows the tree structure (from the Root Category) so you can easily see your categories, as well as designate that selecting a parent automatically selects (or deselects) the children, and can say that only the Leaf is selectable (bottom most node)
3. Minimum / Maximum Categories
    a. Here you can designate a min and max, great if you want to enforce that you need at least X number of categories. Keep in mind that since this UI is separate from the Form tab, the only way to truly ensure they add a minimum number of categories is to add a Field without Database Representation to the Form and use a similarly-configured Advanced Category Selector form control so they must select the minimum/max categories upon creation.
4. Save Modes
    a. You have many ways to save!
        i. Set Document Categories (Default CMS_DocumentCategory)
        ii. Set Node Categories (Uses CMS_TreeCategory from the RelationshipsExtended)
        iii. To Joining Table (Use a custom Node-to-Category binding table, great if you really want to separate out your category relationships)
5. Where / Order by: You can further control what categories show and what order they show in (order only applicable I believe on list view)

Edit Bindings (Tree+Order Support)

The normal Edit bindings UI page template is great for Object to Object, non-ordered bindings. However, it lacked two features: The ability to support Node binding (it couldn't figure out that the NodeID was the ParentObjectID), and it didn't support Orderable Bindings. This of course, corrects that.

Noteworthy Properties

1. Bind on Primary Node Only
    a. There's this little thing called a Linked Node, which is a Node that really is just a placeholder, linking to the actual node. The Linked node has its own NodeID, therefore technically can have its own bound objects. Check this and it will automatically convert the Linked Page's NodeID to the NodeID of the page it's linked to.
2. WhereCondition
    a. For Nodes, it's usually (assuming the binding's NodeID reference field is "NodeID")
    NodeID = {% Convert.ToInt(QueryString.NodeID, -1) @%}
    b. For Objects, it's usually (using the example of FooBar being a Binding of Foo and Bar)
    FooID = {% Convert.ToInt(UIContext.ObjectID, -1) @%}
    or sometimes
    FooID = {% Convert.ToInt(UIContext.ParentObjectID, -1) @%}
3. Use Custom Edit Extender
    a. You can add your own Custom UI extender, as noted in the warning, you will need to leverage the original extender in creating your own, which is both in the source code of this documentation, but also in the Documentation tab of the Edit Binding (Tree+Order Support) WEBPART