Universal API Viewer

Purpose

The purpose of this universal API viewer is to allow both Transformation and Hierarchal Transformations on any and all APIs. This functions similarly to a Custom Data Source, however incorporates classes and structure to be able to define Parent-child relationship, as well as trick Kentico into thinking that the data coming back are "Page Types" so you can define your hierarchal transformations based on that set Page Type.

Features

- Convert external APIs and handle them as if they were Kentico Pages/Objects.
- Hierarchical Transformation and normal Transformation fully supported.
- Order By and Where conditions supported for the API results.
- Select Top N, Skip N, and Select Top N Level supported to allow Pagination / limiting.
    - When Selecting N items in Hierarchy Results, set the 'Select Top N Level' (0 based) to tell the system what level to limit the results by.
    - Any Parents or children of that Selected Level that are within the Skip N / Top N range will be loaded and presented for transformation.
- Works with the Webpart to API Converter tool (Additional Custom Web Part in Marketplace)

Limitations

- Could not get the Default Pagination system working with this customization.
- Requires defining custom classes to match API and logic to fill those objects.
    - Examples provided for XML and JSON parsing, and matching Kentico Page Type objects "test.Food" and "test.FoodType"

Methodology

This Universal API system works as a modified version of the "Universal Viewer with Custom Query" web part, which allows you to create a custom query of the Kentico Page structures and define in the Child-Parent relationship. Instead of querying the Kentico Tree/Document tables, it instead will take your API and build its own Data Table, including the key columns that are needed for Kentico to detect what Page Type the data row is and who's child/parent it is, and what order it's in.

This is done through a custom Class called "ApiUniversalViewerObject" which contains the fields that normally are present in the Tree/Document tables in Kentico (such as NodeID, NodeCladdID, NodeParentID, NodeLevel, NodeOrder, SiteName, Published, etc), and the logic to set them.

Configuration

Since APIs are unique in their results, this web part requires you to do some modification to the backend .cs file to prepare it for absorbing your API. Once you do this, and define in Kentico a Page Type that matches the Properties of the custom classes you create, you can then use Kentico's Transformation system to style your API results.

Step 1: Clone the UniversalAPIViewer Web Part & Set up Custom Fields

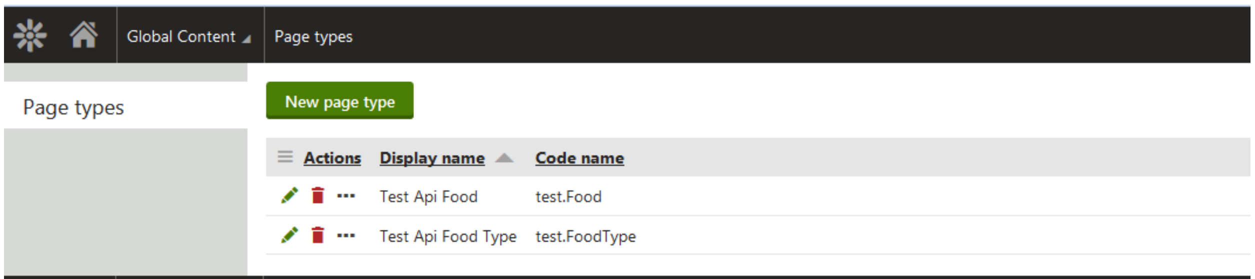
Since each API is unique, please clone the base Web Part first. You will be making back end changes, as well as you will most likely need to add custom Fields so you can properly query your API (for example, if you are referencing Google's GeoCode API, you will want to send it an address field or something similar).

Step 2: Create your custom Page Types for Transformation

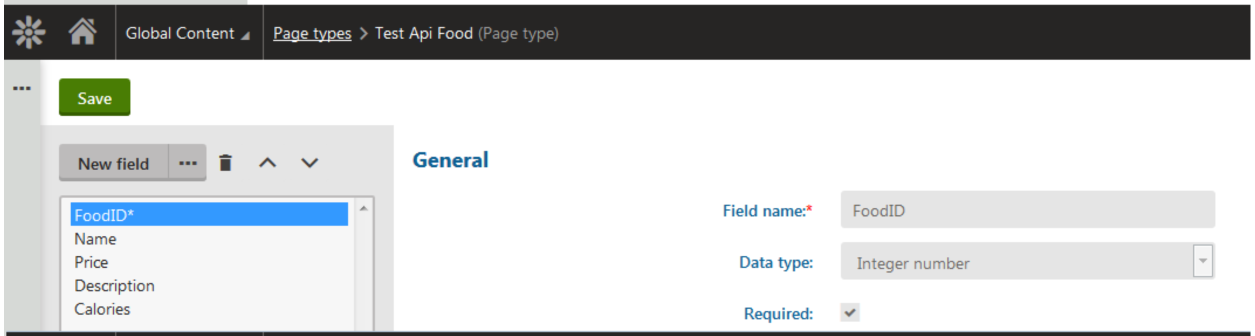
Since you will be mapping the API to Page Types in Kentico, you will need to first create the Page Type so you have a class name to pass to your custom classes.

Create your own Page Types that the API will use. For each Child object, you should create a separate Page type (example, if your API is a Breakfast menu with Food Items that have lists of sub Types, you should create a 'Food' Page Type and a 'Food Type' Page Type).

Define the fields for the Page Types you created. Again these field names **MUST match the Property names of the class**, as the system will convert Properties into Column Names.

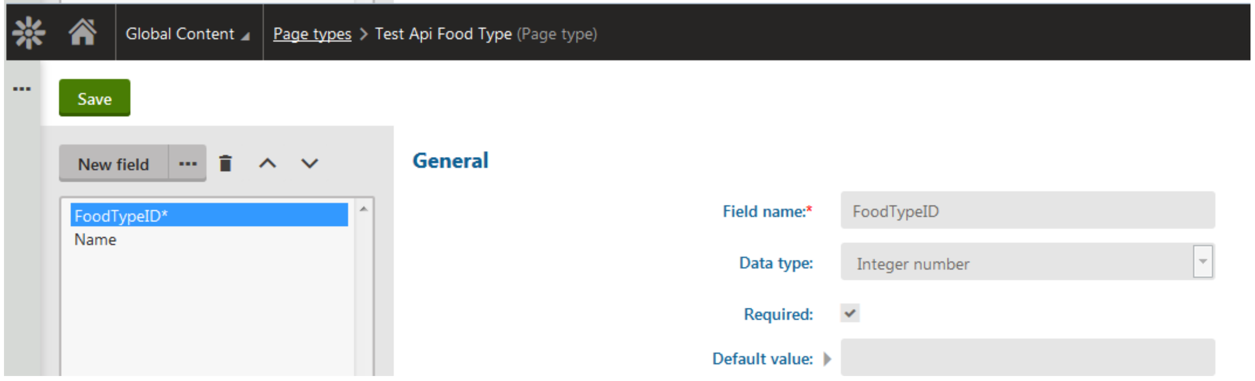


Actions	Display name	Code name
	Test Api Food	test.Food
	Test Api Food Type	test.FoodType



FoodID*
Name
Price
Description
Calories

Field name:* FoodID
Data type: Integer number
Required: ✓



FoodTypeID*
Name

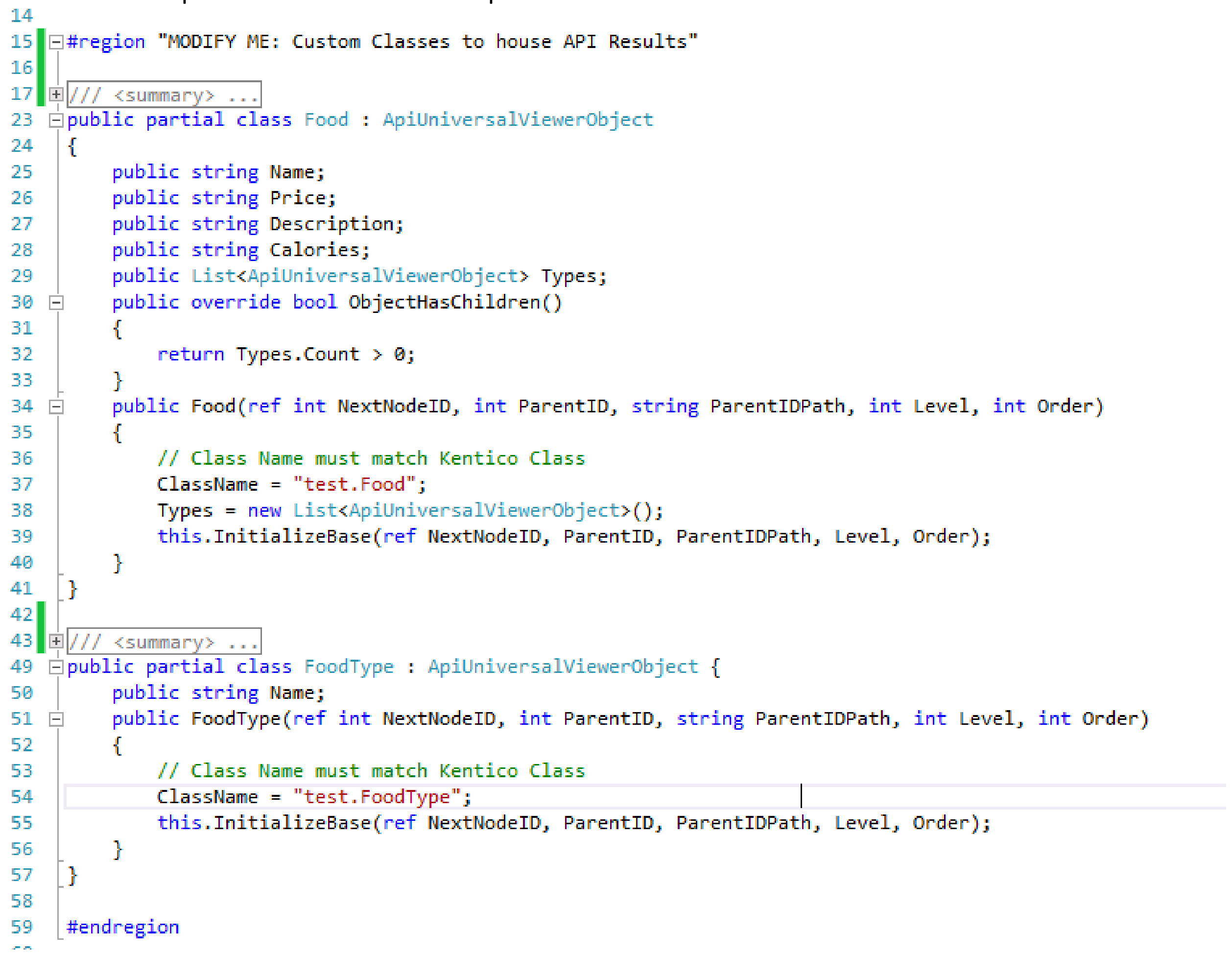
Field name:* FoodTypeID
Data type: Integer number
Required: ✓
Default value:

Step 3: Create your Custom Classes

In your copy of the ApiUniversalViewer.ascx.cs file, you need to adjust the region called "MODIFY ME: Custom Classes to house API Results." Here you will place classes that you will load the API data into. Please ensure the following are followed:

- All classes must inherit the ApiUniversalViewerObject.
- Any 'Children' or Sub-Arrays of that object must be of the type List<ApiUniversalViewerObject>.
- You must set the ClassName value to match the Page Type class name in Kentico.
- The class should at some point during initialization call the "InitializeBase" function with the Next NodeID, the ParentID, ParentIDPath, Level, and Order.
    - These fields set up the information so the system can mimic Kentico's Parent/Child structure
- You should override the ObjectHasChildren property to return true or false if any Children or Sub Arrays have values.

Here is what is provided in the Initial Sample:

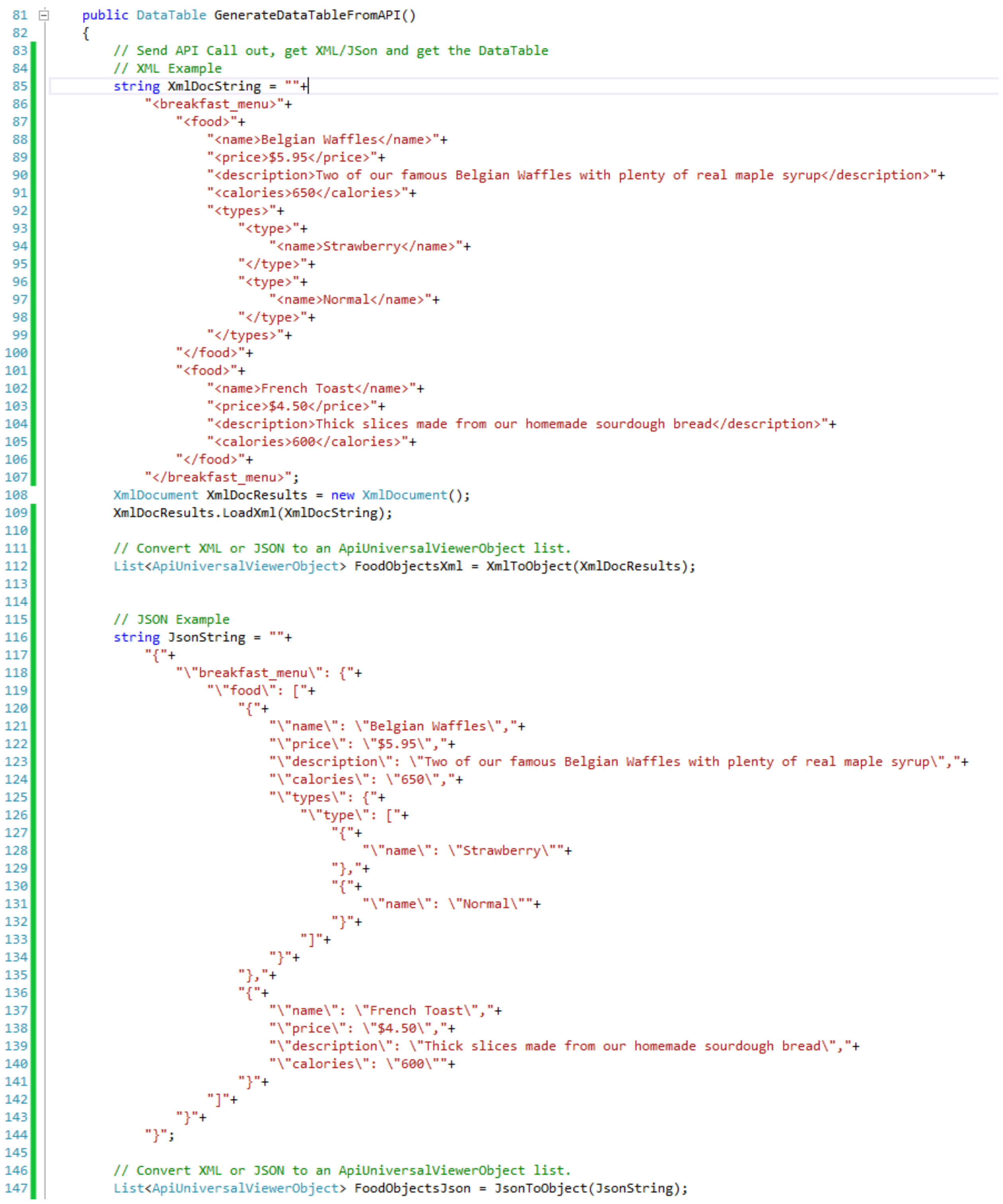


```
#region "MODIFY ME: Custom Classes to house API Results"

/// <summary> ...
public partial class Food : ApiUniversalViewerObject
{
    public string Name;
    public string Price;
    public string Description;
    public string Calories;
    public List<ApiUniversalViewerObject> Types;
    public override bool ObjectHasChildren()
    {
        return Types.Count > 0;
    }
    public Food(ref int NextNodeID, int ParentID, string ParentIDPath, int Level, int Order)
    {
        // Class Name must match Kentico Class
        ClassName = "test.Food";
        Types = new List<ApiUniversalViewerObject>();
        this.InitializeBase(ref NextNodeID, ParentID, ParentIDPath, Level, Order);
    }
}

/// <summary> ...
public partial class FoodType : ApiUniversalViewerObject {
    public string Name;
    public FoodType(ref int NextNodeID, int ParentID, string ParentIDPath, int Level, int Order)
    {
        // Class Name must match Kentico Class
        ClassName = "test.FoodType";
        this.InitializeBase(ref NextNodeID, ParentID, ParentIDPath, Level, Order);
    }
}

#endregion
```

Step 3: Modify the "GenerateDataTableFromAPI()" to Query and Parse your API Results

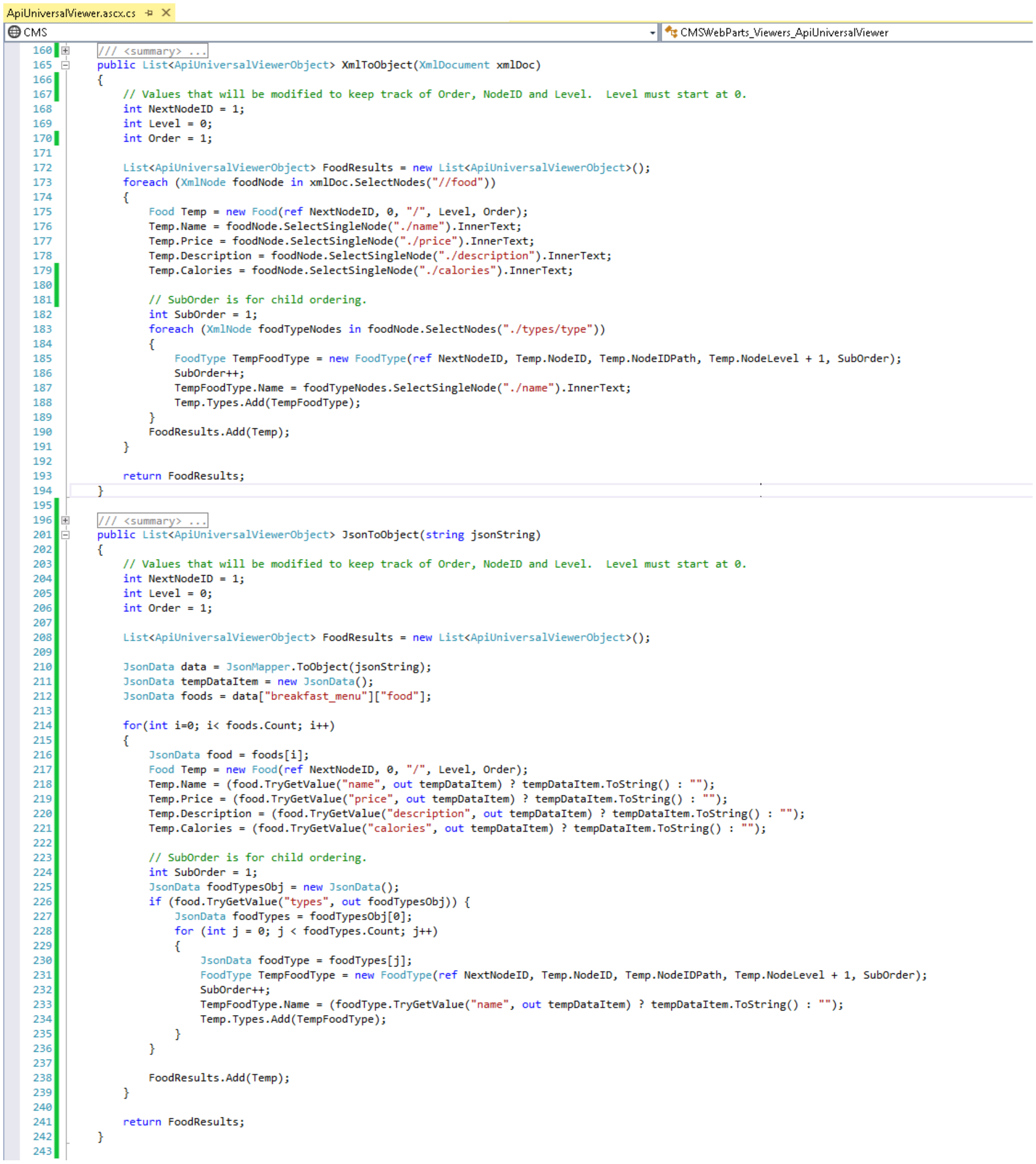
The function GenerateDataTableFromAPI() is a sample function that will Query your API and parse it into a List<UniversalApiViewerObject>, which are appended to a Data Table and returned.

The first portion is getting your data from your API. There are examples of JSON and XML values that would be returned and processed, but this you would modify to get the information you want. Use your custom fields in the web part to pass values to your API.



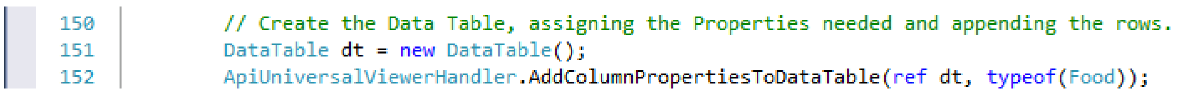
```
public DataTable GenerateDataTableFromAPI()
{
    // Send API Call out, get XML/JSon and get the DataTable
    // XML Example
    string XmlDocString = ""+
        "<breakfast_menu>"+
            "<food>"+
                "<name>Belgian Waffles</name>"+
                "<price>$5.95</price>"+
                "<description>Two of our famous Belgian Waffles with plenty of real maple syrup</description>"+
                "<calories>650</calories>"+
                "<types>"+
                    "<type>"+
                        "<name>Strawberry</name>"+
                    "</type>"+
                    "<type>"+
                        "<name>Normal</name>"+
                    "</type>"+
                "</types>"+
            "</food>"+
            "<food>"+
                "<name>French Toast</name>"+
                "<price>$4.50</price>"+
                "<description>Thick slices made from our homemade sourdough bread</description>"+
                "<calories>600</calories>"+
            "</food>"+
        "</breakfast_menu>";
    XmlDocument XmlDocResults = new XmlDocument();
    XmlDocResults.LoadXml(XmlDocString);

    // Convert XML or JSON to an ApiUniversalViewerObject list.
    List<ApiUniversalViewerObject> FoodObjectsXml = XmlToObject(XmlDocResults);


    // JSON Example
    string JsonString = ""+
        "{"+
            "\"breakfast_menu\": {"+
                "\"food\": ["+
                    "{"+
                        "\"name\": \"Belgian Waffles\","+
                        "\"price\": \"$5.95\","+
                        "\"description\": \"Two of our famous Belgian Waffles with plenty of real maple syrup\","+
                        "\"calories\": \"650\","+
                        "\"types\": {"+
                            "\"type\": ["+
                                "{"+
                                    "\"name\": \"Strawberry\""+
                                "},"+
                                "{"+
                                    "\"name\": \"Normal\""+
                                "}"+
                            "]"+
                        "}"+
                    "},"+
                    "{"+
                        "\"name\": \"French Toast\","+
                        "\"price\": \"$4.50\","+
                        "\"description\": \"Thick slices made from our homemade sourdough bread\","+
                        "\"calories\": \"600\""+
                    "}"+
                "]"+
            "}"+
        "}";

    // Convert XML or JSON to an ApiUniversalViewerObject list.
    List<ApiUniversalViewerObject> FoodObjectsJson = JsonToObject(JsonString);
```

The second portion is converting that data into your class. Example functions of “XmlToObject” and “JsonToObject” are provided to show how you would take your API results and parse them into that list of ApiUniversalViewerObjects.


ApiUniversalViewer.ascx.cs
CMS
CMSWebParts_Viewers_ApiUniversalViewer

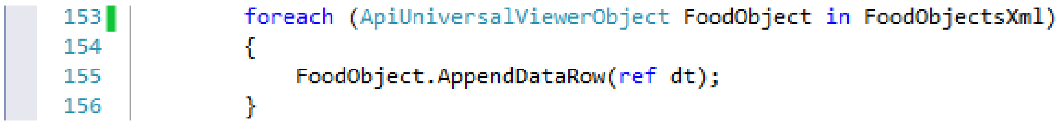
```
/// <summary> ...
public List<ApiUniversalViewerObject> XmlToObject(XmlDocument xmlDoc)
{
    // Values that will be modified to keep track of Order, NodeID and Level.  Level must start at 0.
    int NextNodeID = 1;
    int Level = 0;
    int Order = 1;

    List<ApiUniversalViewerObject> FoodResults = new List<ApiUniversalViewerObject>();
    foreach (XmlNode foodNode in xmlDoc.SelectNodes("//food"))
    {
        Food Temp = new Food(ref NextNodeID, 0, "/", Level, Order);
        Temp.Name = foodNode.SelectSingleNode("./name").InnerText;
        Temp.Price = foodNode.SelectSingleNode("./price").InnerText;
        Temp.Description = foodNode.SelectSingleNode("./description").InnerText;
        Temp.Calories = foodNode.SelectSingleNode("./calories").InnerText;

        // SubOrder is for child ordering.
        int SubOrder = 1;
        foreach (XmlNode foodTypeNodes in foodNode.SelectNodes("./types/type"))
        {
            FoodType TempFoodType = new FoodType(ref NextNodeID, Temp.NodeID, Temp.NodeIDPath, Temp.NodeLevel + 1, SubOrder);
            SubOrder++;
            TempFoodType.Name = foodTypeNodes.SelectSingleNode("./name").InnerText;
            Temp.Types.Add(TempFoodType);
        }
        FoodResults.Add(Temp);
    }

    return FoodResults;
}

/// <summary> ...
public List<ApiUniversalViewerObject> JsonToObject(string jsonString)
{
    // Values that will be modified to keep track of Order, NodeID and Level.  Level must start at 0.
    int NextNodeID = 1;
    int Level = 0;
    int Order = 1;

    List<ApiUniversalViewerObject> FoodResults = new List<ApiUniversalViewerObject>();

    JsonData data = JsonMapper.ToObject(jsonString);
    JsonData tempDataItem = new JsonData();
    JsonData foods = data["breakfast_menu"]["food"];

    for(int i=0; i< foods.Count; i++)
    {
        JsonData food = foods[i];
        Food Temp = new Food(ref NextNodeID, 0, "/", Level, Order);
        Temp.Name = (food.TryGetValue("name", out tempDataItem) ? tempDataItem.ToString() : "");
        Temp.Price = (food.TryGetValue("price", out tempDataItem) ? tempDataItem.ToString() : "");
        Temp.Description = (food.TryGetValue("description", out tempDataItem) ? tempDataItem.ToString() : "");
        Temp.Calories = (food.TryGetValue("calories", out tempDataItem) ? tempDataItem.ToString() : "");

        // SubOrder is for child ordering.
        int SubOrder = 1;
        JsonData foodTypesObj = new JsonData();
        if (food.TryGetValue("types", out foodTypesObj)) {
            JsonData foodTypes = foodTypesObj[0];
            for (int j = 0; j < foodTypes.Count; j++)
            {
                JsonData foodType = foodTypes[j];
                FoodType TempFoodType = new FoodType(ref NextNodeID, Temp.NodeID, Temp.NodeIDPath, Temp.NodeLevel + 1, SubOrder);
                SubOrder++;
                TempFoodType.Name = (foodType.TryGetValue("name", out tempDataItem) ? tempDataItem.ToString() : "");
                Temp.Types.Add(TempFoodType);
            }
        }

        FoodResults.Add(Temp);
    }

    return FoodResults;
}
```

The Third portion is defining your Data Table. Once you create a blank Data Table, call the “AddColumnPropertiesToDataTable” method, giving it the class type of the top level class. This will gather a list of all the properties of itself and any sub classes and create a data table that has those properties as columns:



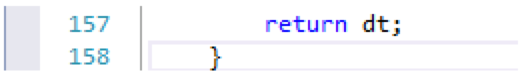
```
        // Create the Data Table, assigning the Properties needed and appending the rows.
        DataTable dt = new DataTable();
        ApiUniversalViewerHandler.AddColumnPropertiesToDataTable(ref dt, typeof(Food));
```

The Final portion is to then loop through the List of ApiUniversalViewerObjects and append the data rows to the table. The class will automatically handle all the child object rows, placing Nulls in columns that do not apply to that class.



```
        foreach (ApiUniversalViewerObject FoodObject in FoodObjectsXml)
        {
            FoodObject.AppendDataRow(ref dt);
        }
```

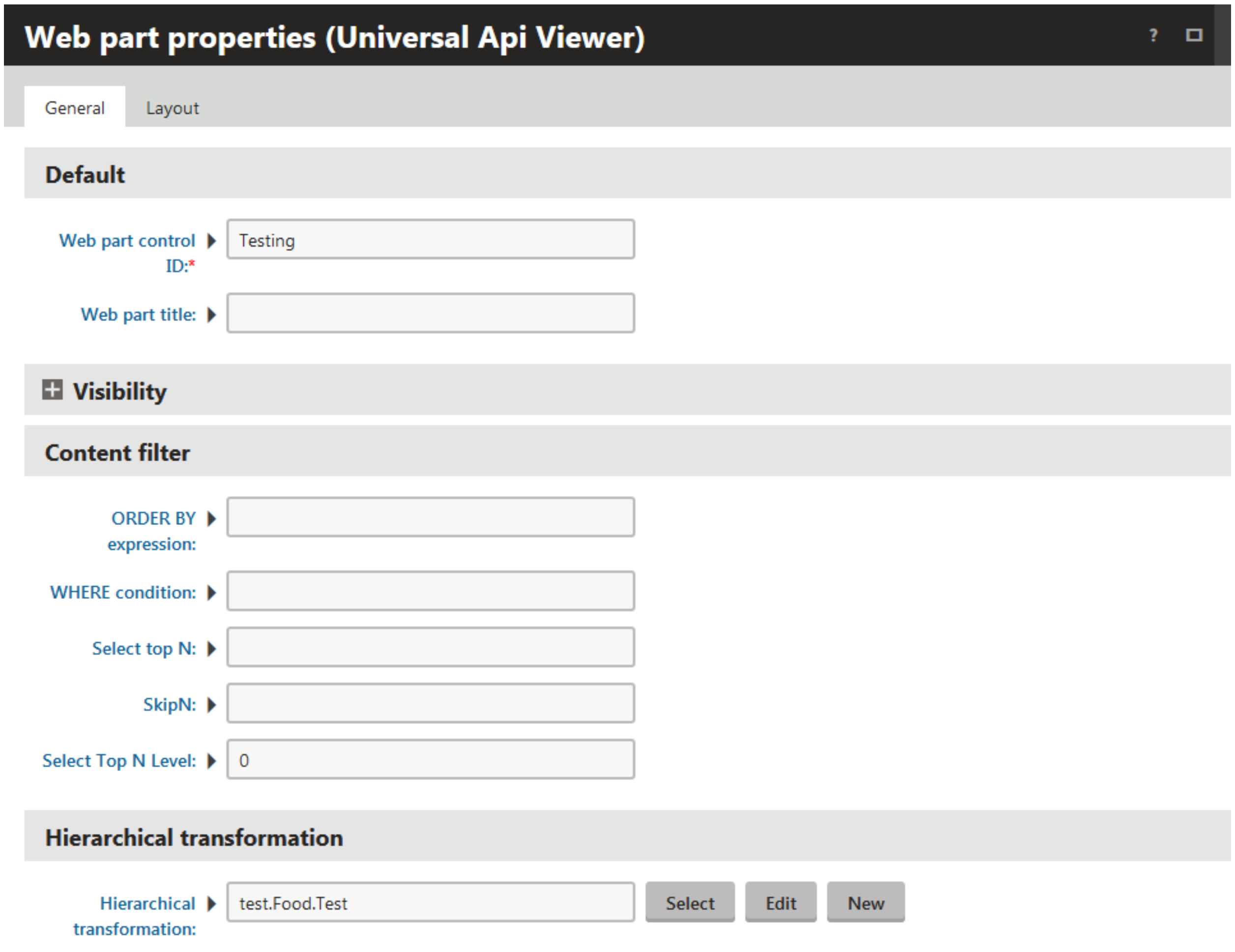
Lastly, return the data table.



```
        return dt;
    }
```

Step 4: Create your Transformations

From this point onwards, the rest behaves like normal Kentico repeaters. Simply create transformations using the Page Types that you created in step 2 to render your results. Included with this Web Part are the test.Food and test.FoodType Page Types which correspond to the sample API and renderings, along with the Hierarchy transformation “test.Food.Test”



Web part properties (Universal Api Viewer)

Default

Web part control ID:*	Testing
Web part title:	

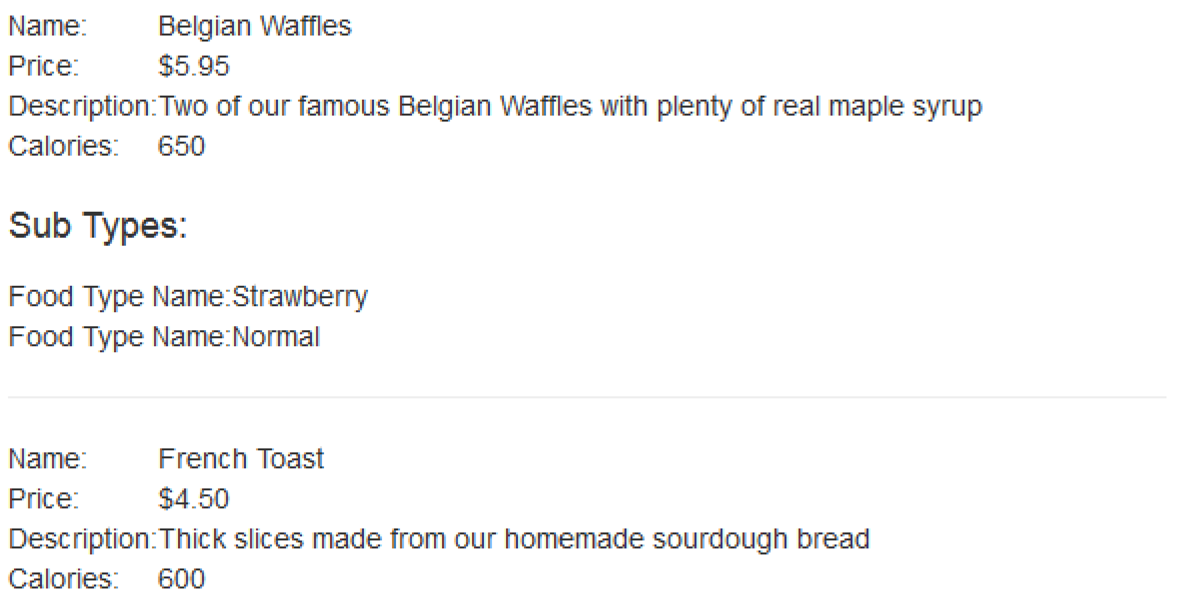
Visibility

Content filter

ORDER BY expression:	
WHERE condition:	
Select top N:	
SkipN:	
Select Top N Level:	0

Hierarchical transformation

Hierarchical transformation:	test.Food.Test



Name:	Belgian Waffles
Price:	$5.95
Description:	Two of our famous Belgian Waffles with plenty of real maple syrup
Calories:	650

Sub Types:

Food Type Name:Strawberry
Food Type Name:Normal

Name:	French Toast
Price:	$4.50
Description:	Thick slices made from our homemade sourdough bread
Calories:	600