Catalog discounts, order discounts, free shipping offers and gift cards provide the option of setting conditions, which limit how the given discount or offer is applied. The conditions can be built using a pre-defined set of rules. However, you can also add new rules so that store managers can offer discounts just as they want them. You can learn more basic information about adding new discount rules in Configuring discount rules.

This page describes how to add discount rules that use a custom macro method, which developers define in the system's code. You can then create discount rules that call the custom method, and use them in the conditions of catalog discounts, order discounts, free shipping offers or gift cards.

To create a discount rule (both order and catalog rules) that calls a custom method, you need to:

1. Open your Kentico solution in Visual Studio.
2. Create a new class that inherits from the **MacroMethodContainer** class.

   Class location

   For production sites, we recommend creating a new assembly (Class library project) in your Kentico solution and including the macro container there. You need to add the appropriate references to both the assembly and the main Kentico web project. For more information, refer to Best practices for customization.

3. Register the custom macro container for specific data types or macro namespaces using the **RegisterExtension** attribute.

   For order rules

   ```
   using CMS;
   using CMS.Ecommerce;
   using CMS.MacroEngine;

   [assembly: RegisterExtension(typeof(CustomOrderRuleMacroMethods ), typeof
   (CalculatorData))]

   public class CustomOrderRuleMacroMethods : MacroMethodContainer
   {
   }
   ```

   For catalog rules

   ```
   using CMS;
   using CMS.Ecommerce;
   using CMS.MacroEngine;

   [assembly: RegisterExtension(typeof(CustomCatalogRuleMacroMethods ), typeof
   (SKUInfo))]

   public class CustomCatalogRuleMacroMethods : MacroMethodContainer
   {
   }
   ```

4. Define your custom macro methods.
5. Save the files (and build your custom class library project).
6. Sign in to the Kentico administration interface and create a discount rule using the custom macro methods.

Your store managers can now limit discounts and offers using the added discount rule. When evaluating conditions based on the rule, the system calls the created custom macro method.

Example – Adding a rule that checks product categories

The following example demonstrates how to add an order rule with a macro condition that calls a custom method. The custom discount rule checks whether products in the evaluated shopping cart or order belong to a specific page category. You can specify a minimum number of product units that must belong to the page category to be eligible for the discount.

> **Stand-alone SKUs**
>
> Even though the principle is the same, this specific example does not work for stand-alone SKUs. Page categories are available only for products that combine an SKU object and a page (the default approach in Kentico).

To create a discount rule that checks a product's category:

1. Create a macro method container class
2. Add an order discount rule that uses the custom method from the class

Creating a macro method container class

Prepare a separate project for custom classes in your Kentico solution:

1. Open your Kentico solution in Visual Studio.
2. Create a new *Class Library* project in the Kentico solution (or reuse an existing custom project).
3. Add references to the required Kentico libraries (DLLs) for the new project:
    a. Right-click the project and select **Add -> Reference**.
    b. Select the **Browse** tab of the **Reference manager** dialog, click **Browse** and navigate to the ***Lib*** folder of your Kentico web project.
    c. Add references to the following libraries:
        - **CMS.Base.dll**
        - **CMS.Core.dll**
        - **CMS.DataEngine.dll**
        - **CMS.DocumentEngine.dll**
        - **CMS.Ecommerce.dll**
        - **CMS.Helpers.dll**
        - **CMS.MacroEngine.dll**
4. Reference the custom project from the Kentico web project *(CMSApp* or *CMS)*.
5. Edit the custom project's **AssemblyInfo.cs** file (in the *Properties* folder).
6. Add the **AssemblyDiscoverable** assembly attribute:

```
using CMS;

[assembly:AssemblyDiscoverable]
```

Continue by creating the macro method container class:

1. Create a new class named **CustomEcommerceMacroMethods** under the custom project.
2. Add the following code to the class:

```
using System;
using System.Collections.Generic;
using System.Linq;

using CMS;
using CMS.DocumentEngine;
using CMS.Ecommerce;
using CMS.Helpers;
using CMS.MacroEngine;
```

```
[assembly: RegisterExtension(typeof(CustomEcommerceMacroMethods), typeof
(CalculatorData))]

public class CustomEcommerceMacroMethods : MacroMethodContainer
{
    /// <summary>
    /// Returns a true value if the shopping cart / order data contains at least
the specified number of product units in the specified category.
    /// </summary>
    [MacroMethod(typeof(bool), "Returns a true value if the shopping cart / order
data contains at least the specified number of product units in the specified
category.", 3)]
    [MacroMethodParam(0, "data", typeof(CalculatorData), "Shopping cart / order
calculation data")]
    [MacroMethodParam(1, "categoryName", typeof(string), "The name of a page
category")]
    [MacroMethodParam(2, "itemsCount", typeof(int), "The required number of
product units")]
    public static object ContainsProductsInCategory(EvaluationContext context,
params object[] parameters)
    {
        switch (parameters.Length)
        {
            case 3:
                return ContainsProductsInCategory(parameters);

            default:
                throw new NotSupportedException();
        }
    }

    private static bool ContainsProductsInCategory(object[] parameters)
    {
        // Gets the shopping cart calculation data
        CalculatorData data = (CalculatorData)parameters[0];

        // Gets the page category that is checked
        string categoryName = ValidationHelper.GetString(parameters[1], String.
Empty);

        // Gets the minimum required number of product units
        int minItemsCount = ValidationHelper.GetInteger(parameters[2], 1);

        // Gets the IDs of SKUs (products) in the shopping cart / order
        List<int> itemIds = data.Request.Items
                                // Filters out product options
                                .Where(item => !(item.SKU.IsProductOption))
                                // Ensures selection of the ID for both standard
products and variants
                                .Select<CalculationRequestItem, int>(item => item.
SKU.IsProductVariant ? item.SKU.SKUParentSKUID : item.SKU.SKUID)
                                .ToList();

        // Gets the IDs of all related product pages that belong to the specified
category
        List<TreeNode> pages = DocumentHelper.GetDocuments()
                                .OnCurrentSite()
                                .InCategories(categoryName)
                                .WhereIn("NodeSKUID", itemIds)
```

```
                                .Columns("NodeSKUID")
                                .ToList();

        // Counts the total number of product units that belong to the specified
category
        int count = 0;
        foreach (CalculationRequestItem item in data.Request.Items)
        {
            // Gets the ID of standard products or the main product's ID for
variants
            int itemId = item.SKU.IsProductVariant ? item.SKU.SKUParentSKUID :
item.SKU.SKUID;

            // Increases the count by the number of units if the product ID
matches one of the pages with the specified category
            if (pages.Exists(doc => doc.NodeSKUID == itemId))
            {
                count += (int)item.Quantity;
            }
        }

        // Returns a true value if the count is higher than or equal to the
required minimum
        return (minItemsCount <= count);
    }
}
```

3. Save all changes and **Build** the custom project.

The system now provides the custom **ContainsProductsInCategory** macro method for objects of the **CalculatorData** type (the *CalculatorData* is available within the context of order discount rules).

Adding an order rule

Follow the steps below to create an order discount macro rule that enables discounts for customers whose order contains at least a specified number of product units belonging to a selected page category.

1. Open the **Store configuration** application if you want to create an order rule for the current site. Open the **Multistore configuration** application if you run multiple sites on the same Kentico instance and you want to add the rule for all the sites.

   If you are not sure what to choose, see Choosing site or global e-commerce configuration. If you are not sure about specifics of configuring in these applications, see Configuring e-commerce settings for a specific site or globally.

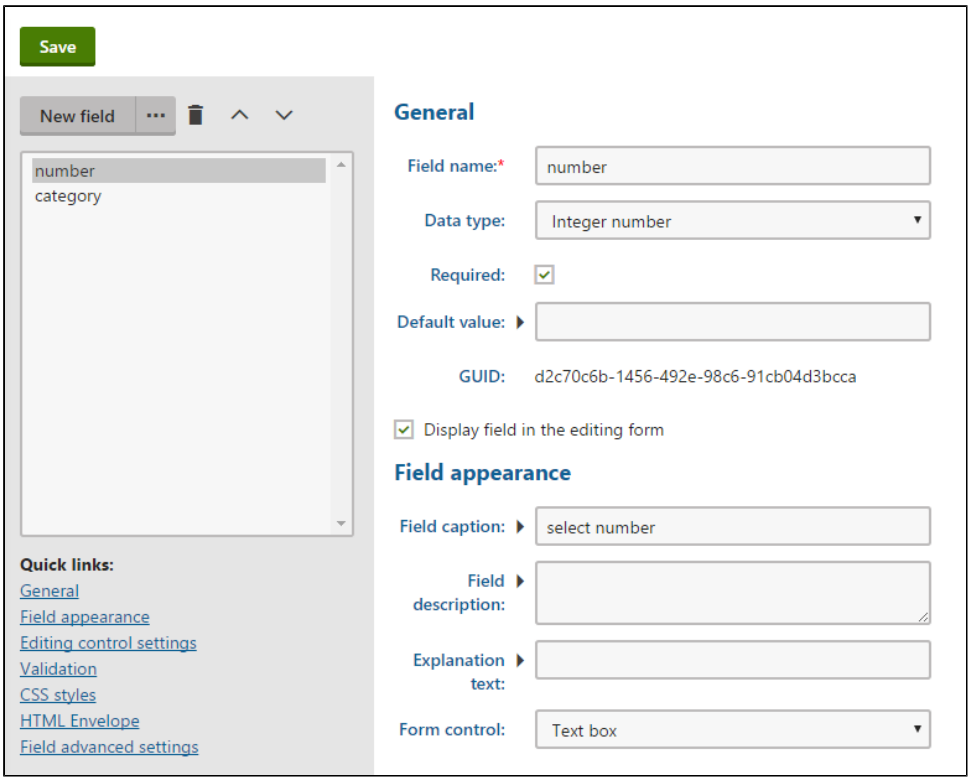
2. Switch to the **Discount rules -> Order rules** tab.
3. Click **New order rule**.
   - The system opens a new page where you can specify the order rule properties and define the rule's parameters.
4. Enter the following values for the rule's properties:
   - **Display name**: Order contains products from a category
   - **User text**: The order contains at least {number} products from the {category} category
   - **Condition**: `Data.ContainsProductsInCategory("{category}", {number})`

     Notice that the **ContainsProductsInCategory** macro method defined in the custom code is now used in the order rule's condition.

5. Click **Save**.
6. Define the rule's parameters:
    a. Switch to the **Parameters** tab.

> You specified two parameters that modify the rule's condition (*number* and *category*). The system adds them automatically. You can now adjust the parameters' settings.

    b. Select ***number*** in the left panel and set the following properties:
        - **Field name**: number
        - **Data type**: Integer number
        - **Required**: Yes (selected)
        - **Display field in the editing form**: Yes (selected)
        - **Field caption**: select number
        - **Form control**: Text box



    c. Click **Save**.
    d. Select ***category*** in the left panel and set the following properties:
        - **Field name**: category
        - **Data type**: Text
        - **Required**: Yes (selected)
        - **Display field in the editing form**: Yes (selected)
        - **Field caption**: select category
        - **Form control**: Category selector (select via the *(more items...)* option)
        - **Display personal categories**: No (cleared)
        - **Display general categories**: Yes (selected)
    e. Click **Save**.

The custom order rule with two parameters is now ready. You can create or edit order discounts, free shipping offers and gift cards with conditions based on the new rule.