The Kentico E-commerce Solution allows you to customize how the system applies discounts to products and orders. For example, you can:

- Define and apply custom discounts (for example discounts provided by an external system or application)
- Adjust or override the functionality of the default discount types

> **Note**: If you only need to set up custom conditions under which discounts are applied, you may be able to achieve your goal by creating discount macro rules.

To customize discounts:

1. Open your Kentico project in Visual Studio.
2. Create new classes that implement one or more of the discount customization interfaces (described below).

   > **Class location**
   >
   > For production sites, we recommend creating a new assembly (Class library project) in your Kentico solution and including the classes there. Then, add the appropriate references to both the assembly and the main Kentico web project. For more information, see Best practices for customization.

3. Implement all methods required by the given interfaces.
4. Register your implementations of the interfaces using the **RegisterImplementation** assembly attribute.

When you reload the website, the system uses the custom implementations that you registered instead of the default discount functionality.

Discount customization interfaces

> All mentioned interfaces can be found in the **CMS.Ecommerce** namespace.

- Product-level discounts (catalog and volume)
- Product coupons
- Buy X Get Y discounts
- Order discounts
- Shipping discounts
- Gift cards

Product-level discounts (catalog and volume)

The system uses the following interfaces to load discounts that apply to individual units of products (includes Catalog and Volume discounts by default):

- **ICatalogDiscountSource** – implementations must contain the *GetDiscounts* method, which returns an *IEnumerable* collection of *DiscountInfo* objects (representing applied catalog discounts) for a specified product (*SKUInfo*). Additional data related to the discount calculation context is available in the method's *PriceParameters* parameter.

- **IVolumeDiscountSource** – implementations must contain the *GetDiscount* method, which returns a *VolumeDiscountInfo* object for a specified product (*SKUInfo*) and unit quantity. Return *null* to apply no volume discount.

- **IProductDiscountSource** – encapsulates the overall retrieval of product-level discounts. By default uses the *ICatalogDiscountSource* and *IVolumeDiscountSource* implementations to load catalog and volume discounts. Implementations of *IProductDiscountSource* must contain the *GetDiscounts* method, which returns an *IEnumerable* collection of *DiscountCollection* objects (representing applied discounts) for a specified product (*SKUInfo*). Additional data related to the discount calculation context is available in the method's *PriceParameters* parameter.

The **PriceParameters** parameter available in the methods of *ICatalogDiscountSource* and *IProductDiscountSource* provides the following properties:

- **Currency** – the currency (*CurrencyInfo*) in which the discount is calculated (for discounts with fixed values).
- **Quantity** – the number of product units.
- **SiteID** – an identifier of the site on which the discount is calculated.
- **User** – the user object (*UserInfo*) for which the discount is calculated. Is *null* for anonymous customers who are not registered as website users.
- **Customer** – the customer object (*CustomerInfo*) for which the discount is calculated. Is *null* when calculating prices for anonymous visitors before they enter customer details in the checkout process.
- **CalculationDate** – a *DateTime* value storing the time for which the discount is calculated (can be used to check the validity of discounts). For most calculations, the value is equal to the current time. When calculating discounts for existing orders, the value is equal to the date and time when the order was created.

See the following customization examples:

- Example - Adding a custom product discount
- Example - Using custom volume discounts

The E-commerce API uses the following additional interfaces to connect the discount sources with the overall product price calculation logic (see Customizing product prices to learn more):

- **IDiscountApplicator** – general service for applying multiple discounts (also used for order discounts and free shipping offers). Implementations must contain the *ApplyDiscounts* method, which performs the following functions:
  - Processes parameters that specify a base price (*decimal*) and an *IEnumerable* collection of *DiscountCollection* objects (representing applied discounts).
  - Returns a summary of the applied discounts as a *ValuesSummary* object containing discount name and value pairs.
- **IProductDiscountService** – builds the overall summary of applied product-level discounts. Implementations must contain the *GetProductDiscounts* method, which returns a *ValuesSummary* object for a specified product (*SKUInfo*), base price (*decimal*), and price calculation parameters (*PriceParameters*). The default *IProductDiscountService* implementation gets the collection of *DiscountCollection* objects using the registered *IProductDiscountSource*, and then creates the *ValuesSummary* using the *IDiscountApplicator* implementation.

Customization of these interfaces is not required or recommended for typical discount customization scenarios.

> Back to the discount interface list

Product coupons

The system uses the **IProductCouponSource** interface to load discounts that require a coupon code and apply to shopping cart lines (each line represents one or more units of a product). See Working with product coupons to learn about the default functionality.

The **IProductCouponSource** interface is only intended for customization scenarios that utilize the default Kentico product coupon objects (i.e. objects represented by the *MultiBuyDiscountInfo* class in the API and managed in the *Product coupons* application of the administration interface).

If you need to integrate product coupons from an external source or service, we recommend adding a custom step (*IShoppingCartCalculator* implementation) into the overall shopping cart calculation process. See Customizing the shopping cart calculation for more information.

Implementations of *IProductCouponSource* must contain the **GetDiscounts** method. The method provides a **DiscountsParameters** parameter with the following properties:

- **CouponCodes** – a collection of coupon codes that the customer added to the order.

- **Currency** – the currency (*CurrencyInfo*) in which the discount is calculated.
- **DueDate** – a *DateTime* value used to evaluate the validity of discounts. For most calculations, the value is equal to the current time. When calculating discounts for existing orders, the value is equal to the date and time when the order was created.
- **SiteID** – an identifier of the site on which the discount is calculated.
- **User** – the user object (*UserInfo*) for which the discount is calculated. Is *null* for anonymous customers who are not registered as website users.

The *GetDiscounts* method must return an *IEnumerable* collection of **IMultiBuyDiscount** objects representing the applied discounts (we recommended using instances of the default **MultiBuyDiscount** class).

> The E-commerce API additionally uses the **IMultiBuyDiscountsApplicator** and **IProductCouponService** interfaces to connect the product coupon source with the overall shopping cart and order calculation logic. However, customization of these interfaces is not required for typical scenarios. For such advanced cases, we instead recommend adding a custom step (*IShoppingCartCalculator* implementation) into the overall shopping cart calculation process (see Customizing the shopping cart calculation for more information).

> Back to the discount interface list

Buy X Get Y discounts

The system uses the **IMultiBuyDiscountSource** interface to load discounts that work based on combinations of products and apply to shopping cart lines (each line represents one or more units of a product). See Working with Buy X Get Y discounts to learn about the default functionality.

Implementations of *IMultiBuyDiscountSource* must contain the **GetDiscounts** method, which provides a **DiscountsParameters** parameter (see the Product coupons section to learn about the available properties). The *GetDiscounts* method must return an *IEnumerable* collection of **IMultiBuyDiscount** objects representing the applied discounts (we recommended using instances of the default **MultiBuyDiscount** class).

If you need to modify how the system evaluates whether the content of a shopping cart is eligible for a Buy X Get Y discount, create a custom implementation of the **IMultiBuyDiscountsEvaluator** interface. We recommend using the following general approach:

1. Add a custom class that inherits from the **MultiBuyDiscountsEvaluator** class (the default *IMultiBuyDiscountSource* implementation).
2. Override the virtual methods that you wish to change.
3. Register your custom class using the **RegisterImplementation** assembly attribute.

> The E-commerce API additionally uses the **IMultiBuyDiscountsApplicator** and **IMultiBuyDiscountsService** interfaces to connect the Buy X Get Y functionality with the overall shopping cart and order calculation logic. However, customization of these interfaces is not required for typical scenarios. For such advanced cases, we recommend implementing your custom discount functionality as a separate step (*IShoppingCartCalculator* implementation) in the overall shopping cart calculation process. See Customizing the shopping cart calculation for more information.

> Back to the discount interface list

Order discounts

The system uses the **IOrderDiscountSource** interface to load discounts that apply to entire orders/purchases (Order discounts by default). Implementations must contain the **GetDiscounts** method, which provides the following parameters:

- A **CalculatorData** object holding data related to the shopping cart or order to which the discount is applied. See Customizing the shopping cart calculation for more information.
- A decimal value matching the total price of the order (before shipping costs, taxes and gift cards).

The *GetDiscounts* method must return an *IEnumerable* collection of **DiscountCollection** objects representing the applied discounts.

The E-commerce API uses the general **IDiscountApplicator** interface to connect the order discount source with the overall shopping cart and order calculation logic (see Customizing the shopping cart calculation to learn more).

Implementations of *IDiscountApplicator* must contain the *ApplyDiscounts* method, which performs the following functions:

- Processes parameters that specify a base price (*decimal*) and an *IEnumerable* collection of *DiscountCollection* objects (representing applied discounts).
- Returns a summary of the applied discounts as a *ValuesSummary* object containing discount name and value pairs.

Customization of this interface is not required or recommended for typical discount customization scenarios.

> Back to the discount interface list

Shipping discounts

The system uses the **IShippingDiscountSource** interface to load discounts that apply to shipping costs (Free shipping offers by default). Implementations must contain the following methods:

- **GetDiscounts** – returns an *IEnumerable* collection of *IDiscount* objects representing the applied discounts. Provides the following parameters:
  - A **CalculatorData** object holding data related to the shopping cart or order to which the shipping discount is applied. See Customizing the shopping cart calculation for more information.
  - A decimal value matching the total price of the order (before taxes and gift cards).
- **GetRemainingAmountForFreeShipping** – returns a decimal value indicating how much the order's price would need to increase to fulfill the conditions of a shipping discount. Return 0 if there is no valid shipping discount or if the shipping is already free. Provides the same parameters as the *GetDiscounts* method.

The E-commerce API uses the following additional interfaces to connect the shipping discount source with the overall shopping cart and order calculation logic (see Customizing the shopping cart calculation to learn more).

- **IDiscountApplicator** – general service for applying multiple discounts (also used for product-level discounts and order discounts). Implementations must contain the *ApplyDiscounts* method, which performs the following functions:
  - Processes parameters that specify a base price (*decimal*) and an *IEnumerable* collection of *DiscountCollection* objects (representing applied discounts).
  - Returns a summary of the applied discounts as a *ValuesSummary* object containing discount name and value pairs.
- **IShippingPriceService** – calculates the final shipping price. The default *IShippingPriceService* implementation gets shipping discounts using the registered *IShippingtDiscountSource*, and then creates the required shipping discount summary using the *IDiscountApplicator* implementation.

Customization of the *IDiscountApplicator* interface is not required or recommended for typical discount customization scenarios.

To learn more about customization of shipping cost customizations in general, refer to the Shipping-related customizing chapter.

> Back to the discount interface list

Gift cards

The system uses the **IGiftCardSource** interface to load gift cards that reduce the final price of orders (see Working with gift cards to learn about the default functionality).

> The **IGiftCardSource** interface is only intended for customization scenarios that utilize the default Kentico gift card objects (i.e. gift cards represented by the *GiftCardInfo* class in the API and managed in the *Gift cards* application of the administration interface).
>
> If you need to integrate gift cards from an external source or service, we recommend adding a custom step (*IShoppingCartCalculator* implementation) into the overall shopping cart calculation process. See Customizing the shopping cart calculation for more information.

Implementations of *IGiftCardSource* must contain the **GetGiftCards** method, which provides the following parameters:

- A **CalculatorData** object holding data related to the shopping cart or order to which the gift card is applied (see Customizing the shopping cart calculation). For example, use the *CalculatorData.Request.CouponCodes* property to get a collection of coupon codes that the customer added to the order.
- A decimal value matching the total price of the order (including other discounts, shipping costs and taxes).

The *GetGiftCards* method must return an *IEnumerable* collection of **GiftCardApplication** objects representing the applied gift cards.

> Back to the discount interface list

Example – Adding a custom product discount

The following example demonstrates how to implement a custom product-level discount. This customization applies all valid catalog and volume discounts from Kentico, and then additionally reduces the price of all products by 10% for customers who are registered users on the website.

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 (and any others that you may need in your custom code):
        - **CMS.Base.dll**
        - **CMS.Core.dll**
        - **CMS.DataEngine.dll**
        - **CMS.Ecommerce.dll**
        - **CMS.Globalization.dll**
        - **CMS.Helpers.dll**
        - **CMS.Membership.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 a custom implementation of the **IProductDiscountSource** interface:

1. Add a new class under the custom project, inheriting from the default **ProductDiscountSource** class (and implementing the *IProductDiscountSource* interface).

```
using System.Collections.Generic;

using CMS;
using CMS.Ecommerce;

// Registers the custom implementation of IProductDiscountSource
[assembly: RegisterImplementation(typeof(IProductDiscountSource), typeof
(CustomProductDiscountSource))]

public class CustomProductDiscountSource : ProductDiscountSource,
IProductDiscountSource
{
    /// <summary>
    /// Constructor with parameters that accept instances of services required by
the ProductDiscountSource base class.
    /// </summary>
    public CustomProductDiscountSource(ICatalogDiscountSource
catalogDiscountSource, IVolumeDiscountSource volumeDiscountSource,
ISiteMainCurrencySource mainCurrencySource, ICurrencyConverterFactory
currencyConverterFactory, IRoundingServiceFactory roundingServiceFactory)
        : base(catalogDiscountSource, volumeDiscountSource, mainCurrencySource,
currencyConverterFactory, roundingServiceFactory)
    {
    }

    /// <summary>
    /// Creates a collection of product-level discounts that apply to a specified
product.
    /// Additional data related to the price and purchase context is provided in
the PriceParameters.
    /// </summary>
    public override IEnumerable<DiscountCollection> GetDiscounts(SKUInfo sku,
decimal standardPrice, PriceParameters priceParams)
    {
        var discountGroups = new List<DiscountCollection>();

        // Adds the default Kentico catalog and volume discounts (using methods
from the ProductDiscountSource base class)
        AddCatalogDiscounts(discountGroups, sku, priceParams);
        AddVolumeDiscounts(discountGroups, sku, priceParams);

        // Adds a custom discount
        // The custom discount uses multiplicative stacking with any applied
catalog or volume discounts
        // (i.e. the custom discount is calculated from the price reduced by
other discounts)
        AddCustomDiscounts(discountGroups, sku, priceParams);

        return discountGroups;
    }

    /// <summary>
    /// Adds a custom discount for a specified product.
    /// </summary>
    private void AddCustomDiscounts(List<DiscountCollection> discountGroups,
SKUInfo sku, PriceParameters priceParams)
```

```
    {
        // Adds the discount only for customers who are registered users on the
website
        if (priceParams.User != null)
        {
            // Creates a 10% custom discount (using a method from the
ProductDiscountSource base class)
            IDiscount customDiscount = CreatePercentageProductDiscount("Discount
for registered users", 0.1m, priceParams);

            // Adds the custom discount to the collection of overall product
discounts
            var discountCollection = new DiscountCollection(new[] {
customDiscount });
            discountGroups.Add(discountCollection);
        }
    }
}
```

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

The registered **CustomProductDiscountSource** implementation extends the Kentico product-level discounts. The custom discount source retains the default catalog and volume discount functionality by inheriting from the **ProductDiscountSource** class (the default implementation of *IProductDiscountSource*). The added custom discount affects both price calculations during the checkout process and prices displayed in product catalogs on the live site (when using appropriate transformation methods).

Example – Using custom volume discounts

The following example demonstrates how to override the default volume discount functionality of Kentico. The example defines the volume discount rules directly in the code, but you can use the same basic approach to load the discount parameters from an external source or service.

1. Recreate or reuse the custom project from the previous example.
2. Add a new class under the custom project, implementing the *IVolumeDiscountSource* interface.
3. Save all changes and **Build** the custom project.

```
using CMS;
using CMS.Ecommerce;

// Registers the custom implementation of IVolumeDiscountSource
[assembly: RegisterImplementation(typeof(IVolumeDiscountSource), typeof
(CustomVolumeDiscountSource))]

public class CustomVolumeDiscountSource : IVolumeDiscountSource
{

    /// <summary>
    /// Returns an appropriate volume discount for a specified product (SKU) and unit
quantity.
    /// </summary>
    public VolumeDiscountInfo GetDiscount(SKUInfo sku, decimal quantity)
    {
        // No volume discount applies if the unit quantity is lower than 3
        if (quantity < 3)
        {
            return null;
        }
```

```
        // Defines a custom volume discount
        // The discount value is percentage-based, calculated according to the
purchased unit quantity
        var customVolumeDiscount = new VolumeDiscountInfo
        {
            VolumeDiscountSKUID = sku.SKUID,
            VolumeDiscountMinCount = 3,
            VolumeDiscountIsFlatValue = false,
            VolumeDiscountValue = GetDiscountValue(quantity)
        };

        return customVolumeDiscount;
    }

    /// <summary>
    /// Calculates the volume discount value based on the unit quantity.
    /// </summary>
    private decimal GetDiscountValue(decimal quantity)
    {
        // 5% discount if the unit quantity is at least 3, but below 5
        if (quantity < 5)
        {
            return 5m;
        }
        // 10% discount if the unit quantity is at least 5, but below 10
        if (quantity < 10)
        {
            return 10m;
        }
        // 15% discount if the unit quantity is at least 10, but below 15
        if (quantity < 15)
        {
            return 15m;
        }
        // 20% discount for 15 or more units
        return 20m;
    }
}
```

The registered **CustomVolumeDiscountSource** implementation fully replaces the default Kentico volume discounts. When a customer adds the required number of product units to their shopping cart, the system automatically applies the corresponding discount to the unit price. Any volume discounts defined in the Kentico administration interface no longer apply.