When performing operations with price values (for example calculation of taxes and discounts, currency conversions, etc.), the system rounds the result according to the number of decimal places allowed by the configuration of each currency.

By default, you can configure the type of price rounding via settings:

1. Open the **Store configuration** or **Multistore configuration** application.
2. Navigate to the **Store settings -> General** tab.
3. Select one of the **Price rounding** options:
   - **Mathematical** – standard rounding to the nearest number, with midpoint values rounded up (toward the nearest number that is away from zero).
   - **Truncate** – all values are rounded down (decimal numbers beyond the allowed limit are discarded).
   - **Financial** – standard rounding to the nearest number, with midpoint values rounded toward the nearest even number.
4. Click **Save**.

If the default options do not fulfill the requirements of your store, you need to customize the Kentico rounding functionality.

Customizing rounding globally

To customize the price rounding used throughout the entire system:

1. Open your Kentico project in Visual Studio.
2. Create a new class that implements the **IRoundingService** interface (found in the *CMS.Ecommerce* namespace).

> **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 the **Round** method according to your requirements.
4. Create a new class that implements the **IRoundingServiceFactory** interface (found in the *CMS.Ecommerce* namespace).
5. Implement the **GetRoundingService** method for the factory class.
6. Register your *IRoundingServiceFactory* implementation using the **RegisterImplementation** assembly attribute.

The customized rounding affects both price values displayed in product catalogs on the live site (when using appropriate transformation methods) and price calculations during the checkout process.

Example – Custom cash rounding

The following example demonstrates how to implement custom cash rounding of all price values. The example rounds values to the number of decimal places allowed by the configuration of each currency, with the last digit always rounded to the nearest multiple of **5**.

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 custom implementations of the **IRoundingService** and **IRoundingServiceFactory** interfaces:

1. Add a new class under the custom project, implementing the *IRoundingService* interface.

```
using System;

using CMS.Ecommerce;

public class CustomRoundingService : IRoundingService
{
    public decimal Round(decimal price, CurrencyInfo currency)
    {
        // Gets the number of allowed decimal places for the requested currency
        int numberOfDecimalPlaces = currency.CurrencyRoundTo;

        // Prepares a basic multiplier that ensures rounding to the required
number of decimal places
        decimal multiplier = Convert.ToDecimal(Math.Pow(10,
numberOfDecimalPlaces));

        // Prepares a "cash rounding" multiplier that ensures rounding to either
'5' or '0' for the last decimal place
        decimal cashRoundingMultiplier = multiplier / 5m;

        // Returns the rounded price values
        return Math.Round((price * cashRoundingMultiplier), MidpointRounding.
AwayFromZero) / cashRoundingMultiplier;
    }
}
```

2. Add a new class under the custom project, implementing the *IRoundingServiceFactory* interface.

```
using CMS;
using CMS.Ecommerce;

// Registers the custom implementation of IRoundingServiceFactory
[assembly: RegisterImplementation(typeof(IRoundingServiceFactory), typeof
(CustomRoundingServiceFactory))]

public class CustomRoundingServiceFactory : IRoundingServiceFactory
{
    // Provides an instance of the custom rounding service
    public IRoundingService GetRoundingService(int siteId)
    {
                // This basic sample does not parameterize the rounding service
based on the current site
                // Use the method's 'siteId' parameter if you need different
rounding behavior for each site
                return new CustomRoundingService();
    }
}
```

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

The registered **CustomRoundingServiceFactory** class provides an instance of **CustomRoundingService**. The customized rounding service ensures that all price values are rounded to the required number of decimal places, with the last digit being a multiple of 5.

Customizing rounding of tax values

The e-commerce API allows you to customize rounding specifically for tax values, without affecting rounding in other locations.

> **Note**
>
> The scenario described below is a customization of the default tax calculation functionality. It is not compatible with other custom implementations of the **ITaxEstimationService** or **ITaxCalculationService** interface. If you already have such customizations, you need to perform the required rounding of the tax results within the given implementations.
>
> For more details, see: Customizing tax calculation

1. Open your Kentico project in Visual Studio.
2. Create a new class that inherits from the **DefaultTaxEstimationService** class (found in the *CMS.Ecommerce* namespace).
3. Override the **RoundTax** method and implement your required rounding logic.
4. Use the **RegisterImplementation** assembly attribute to register the custom class as the implementation of the **ITaxEstimationService** interface.

The customization only changes the rounding of calculated tax values, while leaving the remaining tax functionality in its default state. The custom rounding applies both to tax values in checkout process calculations and when displaying tax values on the live site (using the appropriate transformation methods).

For all price values other than taxes, rounding is not affected and behaves according to the site's *Price rounding* setting (or the custom IRoundingService implementation if one exists).

Example – Rounding tax values up

The following example demonstrates how to customize the system to always round tax values up (even if the rounded fractional number is closer to the lower number). The example rounds values to the number of decimal places allowed by the configuration of each currency.

1. Recreate or reuse the custom project from the global rounding customization example.
2. Create a new class under the custom project, inheriting from the *DefaultTaxEstimationService* class.

```
using System;

using CMS;
using CMS.Ecommerce;

// Registers the custom implementation of ITaxEstimationService
[assembly: RegisterImplementation(typeof(ITaxEstimationService), typeof
(CustomTaxRoundingService))]

public class CustomTaxRoundingService : DefaultTaxEstimationService
{
    // Constructor that passes instances of required interface implementations to
the base DefaultTaxEstimationService constructor
    public CustomTaxRoundingService(ICountryStateTaxRateSource rateSource,
IRoundingServiceFactory roundingServiceFactory)
        : base(rateSource, roundingServiceFactory)
    {
    }

    protected override decimal RoundTax(decimal price, TaxEstimationParameters
parameters)
    {
        // Gets the number of allowed decimal places for the requested currency
        int numberOfDecimalPlaces = parameters.Currency.CurrencyRoundTo;

        // Rounds the tax value up to the requested number of decimal places
        decimal multiplier = Convert.ToDecimal(Math.Pow(10,
numberOfDecimalPlaces));
        return Math.Ceiling(price * multiplier) / multiplier;
    }
}
```

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

The customization ensures that values are always rounded up for tax calculation results that have more decimal places than allowed by the given currency. Other types of rounding are not affected.