The Kentico E-commerce Solution enables you to modify how the system calculates taxes. For example, you can:

- Integrate 3rd-party tax calculation services
- Create custom tax application rules according to your needs
- Customize how the system gets tax rates for countries and states
- Create custom logic for exempting users from taxes
- Customize how taxes affect prices displayed in product catalogs on the live site

To customize the Kentico tax functionality:

1. Open your Kentico project in Visual Studio.
2. Create new classes that implement one or more of the tax 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 (or use a *Factory* class to serve instances dynamically in certain cases).

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

Tax customization interfaces

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

Tax calculation

The system uses tax calculation to get exact tax numbers in the checkout process (shopping carts, orders). A slight delay is acceptable when producing tax calculation results, so implementations using external tax calculation services are suitable.

- **ITaxCalculationService**
- **ITaxCalculationServiceFactory** – serves a separate instance of a class implementing *ITaxCalculationService* for each site. Must be implemented to use custom *ITaxCalculationService* implementations.

Implementations of *ITaxCalculationService* must contain the **CalculateTaxes** method. Calculate the taxes using the data from the method's **TaxCalculationRequest** parameter, which provides the following properties:

- **Items** – a list of *TaxItem* objects representing all purchased items (each *TaxItem* provides the *SKUInfo* representation, quantity and total price).
- **ShippingPrice** – the price of the selected shipping option without tax.
- **Shipping** – *ShippingOptionInfo* representation of the selected shipping option.
- **TaxParameters** – a container providing the customer *(CustomerInfo)*, currency, date of purchase, shipping/billing address, and identifier of the site for which the taxes are calculated.
- **Discount** – the total value of all order discounts applied to the shopping cart or order (as a calculated decimal amount, not a percentage). Other types of discounts are included in the price of individual *TaxItem* objects.

The *CalculateTaxes* method must return a **TaxCalculationResult** object that contains:

- The final tax amount in the **TotalTax** property

> **Important – Hotfix 11.0.39**
>
> Certain payment gateways (e.g. PayPal) require separate tax values for the purchased items and the shipping costs, instead of a single overall tax value. After applying hotfix 11.0.39 or newer, the setter of the *TotalTax* property no longer works. Instead, you need to set the following properties of the *TaxCalculationResult* object:
>
> - **ItemsTax** – the final tax amount for all purchased items
> - **ShippingTax** – the final tax amount calculated for the shipping costs (if the shipping costs are not zero)

- A collection of tax name and value pairs providing details about the applied taxes in the **Summary** property (for example, provides data for tax summaries in invoices)

See: Example - Using an external service to calculate taxes

> **Note**: The default implementation in Kentico is the same for both tax calculation and tax estimation (see below).

Tax estimation

The system uses tax estimation to get taxes in product catalogs (for example when showing what portion of a product's price is tax) and in shipping option selectors. Implementations should produce results quickly – external tax calculation services are not recommended for the purpose of estimation.

- **ITaxEstimationService** – implementations must contain the following methods:
  - **GetTax** – returns a tax value (decimal) for a specified price, tax class (taxation type) and *TaxEstimationParameters* (address and currency).
  - **ExtractTax** – returns a tax value (decimal) for a price that already includes tax, based on a specified price, tax class (taxation type) and *TaxEstimationParameters* (address, currency, identifier of the site for which the taxes are estimated). By default, this method is used for sites that have the *Prices include tax* setting enabled.

Taxes in catalog prices

By default, the system displays prices in catalogs on the live site in the same way that price values are entered for products (with or without tax, depending on the site's **Prices include tax** setting).

You can customize how taxes affect catalog prices if the default options do not fulfill the needs of your store. For example, you can enter and store product prices without tax, but customize the system to display prices including tax on the live site.

- **ICatalogTaxCalculator** – implementations must contain the *ApplyTax* method, which processes a specified product and price based on tax parameters.
  - The supplied tax parameters contain the customer *(CustomerInfo*), currency, date of purchase, shipping/billing address, and identifier of the site for which the taxes are calculated.
  - The *ApplyTax* method must return the price that will be displayed in live site catalogs and set the applied tax value into an *out* parameter.
- **ICatalogTaxCalculatorFactory** – serves a separate instance of a class implementing *ICatalogTaxCalculator* for each site. Must be implemented to use custom *ICatalogTaxCalculator* implementations.

Custom implementations of the catalog tax calculator only impact the prices of products displayed on the live site (when using the appropriate e-commerce transformation methods). This type of customization does not affect the calculation of total prices during the checkout process.

See: Example - Adjusting catalog prices based on taxes

Address used in tax calculations

You can customize how the system gets the address used in tax calculations. For example, this type of customization allows you to determine whether the system calculates taxes based on the billing or shipping address (or another custom address). Note that this type of customization overrides the system's **Apply taxes based** on setting.

- **ITaxAddressService** – implementations must contain the *GetTaxAddress* method, which returns an address for tax calculation based on a specified billing address, shipping address, tax class (taxation type) and customer.
- **ITaxAddressServiceFactory** – serves a separate instance of a class implementing *ITaxAddressService* for each site. Must be implemented to use custom *ITaxAddressService* implementations.

Tax type per product or shipping option

You can customize how the system retrieves Tax classes for products or shipping options if you do not wish to assign the tax classes manually in the administration interface (or if you wish to override the tax class assignments in a custom way).

- **ITaxClassService** – implementations must contain two overloads of the *GetTaxClass* method, which return a tax class object (taxation type) for a specified product or shipping option.

Tax rates per country or state

If you do not wish to use the default configuration options of Tax classes, you can customize how the system loads tax rate values for countries and states.

- **ICountryStateTaxRateSource** – implementations must contain the *GetRate* method, which returns a tax rate for a specified tax class (taxation type), country and state.

Tax exemptions for customers

You can customize the system to create tax exemptions for certain types of customers.

> **Default tax exemptions**
>
> By default, customers who have a **Tax registration ID** specified in their *Company details* are exempt from Tax classes with the **Zero tax if tax ID is specified** property enabled.
>
> Registering a custom *ICustomerTaxClassService* implementation overrides the default tax exemption.

- **ICustomerTaxClassService** – implementations must contain the *GetTaxClass* method, which determines whether taxes apply to a specified customer.

See: Example - Create tax exemptions for customers

Example – Using an external service to calculate taxes

The following example demonstrates how to customize the system to integrate with a 3rd-party tax calculation service. In this case, the tax values are calculated manually within the custom code, but you can replace the basic calculations with calls to the API of a dedicated tax calculation service.

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 **ITaxCalculationService** and **ITaxCalculationServiceFactory** interfaces:

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

> **Note**
>
> The following example works for projects with hotfix 11.0.39 or newer applied. On older versions, you need to set the *TotalTax* property of the *TaxCaclulationResult* object (instead of the newer *ItemsTax* and *ShippingTax*).

```
using System.Linq;

using CMS.Ecommerce;
using CMS.Globalization;

public class CustomTaxCalculationService : ITaxCalculationService
{
        /// <summary>
        /// Calculates all taxes for a given purchase.
        /// </summary>
    public TaxCalculationResult CalculateTaxes(TaxCalculationRequest taxRequest)
    {
        var result = new TaxCalculationResult();

        // Sums up the total price of all purchased items
        decimal itemsTotal = taxRequest.Items.Sum(item => item.Price);

        // Subtracts the overall order discount value from the total price
        itemsTotal = itemsTotal - taxRequest.Discount;

        // Calculates the tax for the purchased items
        // The example only provides a very basic "calculation" based on the
country in the billing address
        // You can use any type of custom calculation, e.g. call the API of a 3rd-
party tax calculation service that returns the calculated taxes
        result.ItemsTax = CalculateCustomTax(itemsTotal, taxRequest.TaxParameters.
BillingAddress);

        // Adds the tax value to the returned tax summary
        // The name parameter describes the type of the applied tax (may be
returned by the external tax calculation service)
        // Taxes added under the same name via the 'Sum' method are automatically
summed up into the existing tax summary item
```

```
        result.Summary.Sum("Custom tax", result.ItemsTax);

        // Adds a basic shipping tax based on the shipping price
        decimal shippingTaxRate = 0.15m;
        result.ShippingTax = taxRequest.ShippingPrice * shippingTaxRate;
        result.Summary.Sum("Custom shipping tax", result.ShippingTax);

        return result;
    }

    /// <summary>
    /// Calculates the tax for a specified price. Replace this method with any
type of custom calculation.
    /// </summary>
    private decimal CalculateCustomTax(decimal price, IAddress billingAddress)
    {
        // Gets a base tax rate, with a different value if the billing address is
in the USA
        decimal taxRate = 0.15m;
        if ((billingAddress != null))
        {
            // Gets the 3-letter country code of the country in the billing
address
            var country = CountryInfoProvider.GetCountryInfo(billingAddress.
AddressCountryID);
            string countryCode = country?.CountryThreeLetterCode;

            if (countryCode == "USA")
            {
                taxRate = 0.1m;
            }
        }

        return (price * taxRate);
    }
}
```

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

```
using CMS;
using CMS.Ecommerce;

// Registers the custom implementation of ITaxCalculationServiceFactory
[assembly: RegisterImplementation(typeof(ITaxCalculationServiceFactory), typeof
(CustomTaxCalculationServiceFactory))]

public class CustomTaxCalculationServiceFactory : ITaxCalculationServiceFactory
{
        // Provides an instance of the custom tax calculation service
        public ITaxCalculationService GetTaxCalculationService(int siteId)
        {
        // This basic sample does not parameterize the tax calculation service
based on the current site
        // Use the method's 'siteId' parameter if you need different tax
calculation logic for each site
                return new CustomTaxCalculationService();
        }
}
```

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

The registered **CustomTaxCalculationServiceFactory** class provides an instance of **CustomTaxCalculationService**. The tax calculation in all shopping carts (and orders) now uses the custom logic. The customizations do not affect tax estimations in product catalogs.

Example – Creating tax exemptions for customers

The following example demonstrates how to create a customization that adds tax exemptions for certain types of customers.

1. Recreate or reuse the custom project from the custom tax calculation example.
2. Create a new class under the custom project, implementing the *ICustomerTaxClassService* interface.
3. Save all changes and **Build** the custom project.

```
using CMS;
using CMS.Ecommerce;

// Registers the custom implementation of ICustomerTaxClassService
[assembly: RegisterImplementation(typeof(ICustomerTaxClassService), typeof
(CustomCustomerTaxClassService))]

public class CustomCustomerTaxClassService : ICustomerTaxClassService
{
        /// <summary>
        /// Returns a CustomerTaxClass object for the specified customer to determine
whether they are subject to tax.
        /// The default value is CustomerTaxClass.Taxable.
        /// </summary>
        public CustomerTaxClass GetTaxClass(CustomerInfo customer)
        {
                // Creates a tax exemption for customers who have a value filled in
within one of the 'Company details' fields
                if ((customer != null) && (customer.CustomerHasCompanyInfo))
                {
                        return CustomerTaxClass.Exempt;
                }
                else
                {
                        // All other customers are subject to tax
                        return CustomerTaxClass.Taxable;
                }
        }
}
```

Example – Adjusting catalog prices based on taxes

The following example demonstrates how to customize the system to store product prices without tax, but display prices on the live site with tax already included.

The example assumes that your site is configured to not include tax in prices:

1. Open the **Store configuration** or **Multistore configuration** application in the Kentico administration interface.
2. Navigate to the **Store settings -> General** tab.
3. In the **Taxes** section, make sure that:
   - you have a **Default country** set (and that your tax classes have values assigned for the given country)
   - the **Prices include tax** setting is disabled
4. Click **Save** if necessary.

Continue by creating custom implementations of the **ICatalogTaxCalculator** and **ICatalogTaxCalculatorFactory** interfaces:

1. Open your Kentico solution in Visual Studio.
2. Recreate or reuse the custom project from the custom tax calculation example.
3. Add a new class under the custom project, implementing the *ICatalogTaxCalculator* interface.

```
using CMS.Ecommerce;

public class CustomCatalogTaxCalculator : ICatalogTaxCalculator
{
    private readonly ITaxEstimationService estimationService;
    private readonly ITaxClassService taxClassService;
    private readonly ITaxAddressService addressService;
```

```
    /// <summary>
    /// The constructor parameters accept instances of tax-related services from
CustomCatalogTaxCalculatorFactory.
    /// The services are used to get the appropriate tax values before applying
the catalog customizations.
    /// </summary>
    public CustomCatalogTaxCalculator(ITaxEstimationService estimationService,
ITaxClassService taxClassService, ITaxAddressService addressService)
    {
        this.estimationService = estimationService;
        this.taxClassService = taxClassService;
        this.addressService = addressService;
    }

    /// <summary>
    /// Processes a specified product and price, makes any required adjustments
based on taxes,
    /// and returns the price value to be displayed in live site catalogs.
    /// The applied tax value must be returned using the 'tax' out parameter.
    /// The customization assumes that prices are without tax (i.e. the 'Prices
include tax' setting is false).
    /// </summary>
    public decimal ApplyTax(SKUInfo sku, decimal price, TaxCalculationParameters
parameters, out decimal tax)
    {
        tax = 0;

        // Gets the tax class (taxation type) assigned to the product
        TaxClassInfo taxClass = taxClassService.GetTaxClass(sku);

        if (taxClass != null)
        {
            // Uses the registered ITaxAddressService implementation to get the
appropriate address for the tax calculation
            IAddress address = addressService.GetTaxAddress(parameters.
BillingAddress, parameters.ShippingAddress, taxClass, parameters.Customer);

            // Uses the registered ITaxEstimationService implementation to get
the tax value
            var estimationParams = new TaxEstimationParameters
            {
                Currency = parameters.Currency,
                Address = address,
                                SiteID = parameters.SiteID
                        };
            tax = estimationService.GetTax(price, taxClass, estimationParams);
        }

        // Returns the price to be displayed in live site catalog, with the tax
value added
        return price + tax;
    }
}
```

4. Add a new class under the custom project, implementing the *ICatalogTaxCalculatorFactory* interface.

```
using CMS;
using CMS.Ecommerce;

// Registers the custom implementation of ICatalogTaxCalculatorFactory
[assembly: RegisterImplementation(typeof(ICatalogTaxCalculatorFactory), typeof
(CustomCatalogTaxCalculatorFactory))]

public class CustomCatalogTaxCalculatorFactory : ICatalogTaxCalculatorFactory
{
    private readonly ITaxEstimationService estimationService;
    private readonly ITaxClassService taxClassService;
    private readonly ITaxAddressServiceFactory addressServiceFactory;

    /// <summary>
    /// Constructor.
    /// The system automatically supplies instances of the ITaxEstimationService,
ITaxClassService and ITaxAddressServiceFactory implementations.
    /// </summary>
    public CustomCatalogTaxCalculatorFactory(ITaxEstimationService
estimationService, ITaxClassService taxClassService, ITaxAddressServiceFactory
addressServiceFactory)
    {
        this.estimationService = estimationService;
        this.taxClassService = taxClassService;
        this.addressServiceFactory = addressServiceFactory;
    }

    // Provides an instance of the custom catalog tax calculator
    public ICatalogTaxCalculator GetCalculator(int siteId)
    {
        // This basic sample does not parameterize the tax calculation service
based on the current site
        // Use the method's 'siteId' parameter if you need different tax
calculation logic for each site
        return new CustomCatalogTaxCalculator(estimationService, taxClassService,
addressServiceFactory.GetTaxAddressService(siteId));
    }
}
```

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

The registered **CustomCatalogTaxCalculatorFactory** class provides an instance of **CustomCatalogTaxCalculator** and prepares instances of services needed to get the tax values. The customized catalog tax calculator ensures that tax values are added to product prices displayed on the live site (when using the appropriate e-commerce transformation methods). The customization does not affect the calculation of total prices during the checkout process.