You can access the current shopping cart using the **EcommerceContext.CurrentShoppingCart** property. This page describes how the process of retrieving the shopping cart works by default and how you can modify the default way of retrieving the shopping cart and working with it.

Default retrieval of the current shopping cart

By default, customizable services take care of the shopping cart and its processes. The **CurrentShoppingCartService** provides the main functionality of the current shopping cart. It consists of several other services:

- **CookieShoppingCartClientStorage** – stores the shopping cart's ID in the visitor's browser cookie.
- **ShoppingCartCache** – caches the shopping cart of the current visitor.
- **DefaultShoppingCartFactory** – creates and initializes new shopping carts (the *ShoppingCartInfo* object).
- **CurrentShoppingCartStrategy** – maintains rules for obtaining the current visitor's shopping cart.
- **DefaultShoppingCartRepository** – contains methods for basic operations with shopping carts.

See all relevant methods and properties in our API reference.

CurrentShoppingCartService

This service implements the **ICurrentShoppingCartService** interface and contains two methods, **GetCurrentShoppingCart** and **SetCurrentShoppingCart**.

GetCurrentShoppingCart

The **GetCurrentShoppingCart** method finds the most suitable shopping cart for the given visitor with the help of the other services. If no shopping cart is found, it creates a new shopping cart.

When getting a shopping cart, the method uses the following process:

1. Try to get the cached cart from the session (with the help of the **ShoppingCartCache**).
2. Try to get the ID of a shopping cart from the browser cookie (with the help of the **CookieShoppingCartClientStorage**).
    a. Retrieve the shopping cart from the database using the ID.

The method then checks whether the cart is from the same site and from the same user (with the help of the **CurrentShoppingCartStrategy**). If so, the process continues with the retrieved shopping cart. If the shopping cart is not from the same site, the method tries to get the latest saved shopping cart of the user and updates it to the current state of the store (discounts, taxes, etc.). If the retrieved shopping cart is from the same site but not from the same user, the method checks with the strategy whether the user can take over another user's shopping cart. If so, the system removes all personal information from the shopping cart. The strategy determines what information is considered personal.

If there is no suitable shopping cart to return at this point, the system creates a new shopping cart (with the help of the **DefaultShoppingCartFactory**) and assigns it to the user (see the **SetCurrentShoppingCart** method).

SetCurrentShoppingCart

The **SetCurrentShoppingCart** method assigns the shopping cart to the user. Specifically, it caches the cart using **ShoppingCartCache** and saves the cart's ID to the visitor's browser cookie using the **CookieShoppingCartClientStorage.**

CookieShoppingCartClientStorage

This service implements the **IShoppingCartClientStorage** interface and contains two methods, **GetCartGuid** and **SetCartGuid**.

The **GetCartGuid** method retrieves the shopping cart GUID from the visitor's browser cookie. The **SetCartGuid** method saves the shopping cart GUID to the visitor's browser cookie.

ShoppingCartCache

This service implements the **IShoppingCartCache** interface and contains two methods, **GetCart** and **StoreCart**.

The **GetCart** method retrieves the shopping cart from the current session's cookie. The **StoreCart** method saves the shopping cart to the current session' cookie.

DefaultShoppingCartFactory

This service implements the **IShoppingCartFactory** interface and contains the **CreateCart** method. The method creates a new instance of a shopping cart (the *ShoppingCartInfo* object) for the current user. It also pre-selects the preferred currency on the site.

CurrentShoppingCartStrategy

This service implements the **ICurrentShoppingCartStrategy** interface and contains a set of rules that decides about the suitability of shopping carts for the current user.

The **CartCanBeUsedOnSite** method decides whether the given shopping cart can be used on the given site. In the default implementation, the current site must match the cart's site.

Customize this method if you want to share shopping carts across multiple sites.

The **UserCanTakeOverCart** method decides whether the given user can take over an another user's given shopping cart. In the default implementation, the shopping cart can be taken over only if it has no owner or the owner is anonymous. The related **TakeOverCart** method then assigns the specified shopping cart to the specified user and its contact.

The **PreferStoredCart** method decides whether an older saved shopping cart should override the current content of the shopping cart. In the default implementation, the older shopping cart is loaded only if the current cart is empty.

The **AnonymizeShoppingCart** method removes personal information from the specified shopping cart. In the default implementation, it clears:

- The customer
- The selected shipping option
- The selected payment method and its possible custom data
- The entered addresses
- The applied discounts and coupon codes
- The shopping cart's note

Customize this method if you want to preserve the entered coupon codes even after a user signs in.

The **RefreshCart** method checks that the shopping cart does not contain invalid data. In the default implementation, the method checks whether the selected currency is available and enabled (and assigns the current site's main currency if not).

DefaultShoppingCartRepository

This service implements the **IShoppingCartRepository** interface and contains the basic actions needed to maintain the shopping carts of users.

The **GetUsersCart** method retrieves the most recent shopping cart stored for the specified user on the specified site.

The **DeleteUsersCart** method removes saved shopping carts from the specified site for the specified user.

The **GetCart** method retrieves a shopping cart with the specified ID or GUID from the database.

The **SetCart** method saves the specified shopping cart to the database.

The **DeleteCart** method removes the specified shopping cart from the database.

Example – Customizing retrieval of the current shopping cart

If you prefer different functionality for shopping cart management, you can customize the default approach. This example describes how to customize the shopping cart strategy. The customization process is the same for the other services.

1. Create a new class file in your project in Visual Studio, for example **CustomShoppingCartStrategy.cs**.
2. Make the class inherit from the **CurrentShoppingCartStrategy** class.
3. Register the custom class as the implementation of **ICurrentShoppingCartStartegy** by adding the **RegisterImplementation** assembly attribute above the class declaration.

```
using CMS;
using CMS.Ecommerce;

[assembly:RegisterImplementation(typeof(ICurrentShoppingCartStrategy), typeof
(CustomShoppingCartStrategy))]
public class CustomShoppingCartStrategy : CurrentShoppingCartStrategy
{
}
```

4. Override the methods you want to change. For example, override the **CartCanBeUsedOnSite** method so that the cart can be used across all sites.

```
public override bool CartCanBeUsedOnSite(ShoppingCartInfo cart,
SiteInfoIdentifier site)
{
        // Checks whether the cart is not null
    if (cart == null)
    {
                throw new ArgumentNullException("cart");
    }


        // Checks whether the site is not null
    if (site == null)
    {
        throw new ArgumentNullException("site");
    }


        // Returns true for all combinations of carts and sites
    return true;
}
```

5. Save the project.

The system now uses the new customized rules for accessing and assigning shopping carts. In the example, the same shopping cart is applicable for all stores within the same Kentico instance.