You can customize how the system creates, modifies or deletes orders. That is suitable especially when you want to process additional actions while working with orders, or when you want to completely replace how the system works with orders.

To customize processes that work with orders, you need to implement a custom provider that inherits from the **OrderInfoProvider** class:

1. Create a new class in your project in Visual Studio (for example **CustomOrderInfoProvider.cs**) that inherits from the **OrderInfoProvider** class and registers the custom InfoProvider:

   ```
   [assembly: RegisterCustomProvider(typeof(CustomOrderInfoProvider))]
   public class CustomOrderInfoProvider : OrderInfoProvider
   {
   }
   ```

2. Implement a method that overrides the default *OrderInfoProvider* method.
   - See the list of all virtual methods that you can override.
3. Save the file.

If you open your website, Kentico uses the modified *OrderInfoProvider* class with the modified methods.

List of virtual methods in the OrderInfoProvider

The following list does not contain virtual methods related to email notifications for orders.

GetOrderInfoInternal

Returns an order with a specified ID (*int orderId*).

```
protected virtual OrderInfo GetOrderInfoInternal(int orderId) { }
```

SetOrderInfoInternal

Creates or updates an order based on a specified order object (*OrderInfo orderObj*).

If you decide to completely override the order creation and update process without calling *base.SetOrderInfoInternal (orderObj)*, you also need to perform the following tasks

- Assign an order date
- Assign an order status
- Set the order as paid if necessary
- Save the order by calling *SetInfo(orderObj)*

To ensure the database's integrity, you can use transactions:

```
using (var tr = BeginTransaction())
{
        // ... your code ...

        tr.Commit();
}
```

```
protected virtual void SetOrderInfoInternal(OrderInfo orderObj) { }
```

See an example below.

DeleteOrderInfoInternal

Deletes an order based on a specified order object (*OrderInfo orderObj*).

To ensure the database's integrity, you can use transactions:

```
using (var tr = BeginTransaction())
{
        // ... your code ...

        tr.Commit();
}
```

```
protected virtual void DeleteOrderInfoInternal(OrderInfo orderObj) { }
```

GetOrdersInternal

Returns all orders on a site specified with its ID (*int siteId*).

```
protected virtual ObjectQuery<OrderInfo> GetOrdersInternal(int siteId) { }
```

GenerateInvoiceNumberInternal

Returns an invoice number generated from a specified shopping cart.

```
protected virtual string GenerateInvoiceNumberInternal(ShoppingCartInfo cart) { }
```

UpdateOrderStatusHistoryInternal

Updates an order status of a specified order (*OrderInfo orderObj*), which already contains the new order status, from a specified original order status *(int originalStatusId*). If the order is new (*bool newOrder*), the method does not check the original order status.

If the original and new statuses are not the same, the system sends an email notification about the change as configured.

```
protected virtual void UpdateOrderStatusHistoryInternal(OrderInfo orderObj, int
originalStatusId, bool newOrder) { }
```

ProcessOrderIsPaidChangeInternal

By default, the method is called when an order is paid (during the **SetOrderInfoInternal** method).

Processes order items of the membership and e-product product representations (see the **ProcessMembershipInternal** and **ProcessEProductInternal** methods) of a specified order (*OrderInfo oi*) and then sends an email notification if the order is paid.

```
protected virtual void ProcessOrderIsPaidChangeInternal(OrderInfo oi) { }
```

ProcessMembershipInternal

By default, the method is called when an order is paid (during the **ProcessOrderIsPaidChangeInternal** method).

Creates or updates a record regarding a specified user (*UserInfo ui*) and a specified membership (*SKUInfo skui*) in a specified order (*OrderInfo oi*) as a specified order item (*OrderItemInfo oii*). Specify to when the membership is supposed to be prolonged (*DateTime now*).

```
protected virtual void ProcessMembershipInternal(OrderInfo oi, OrderItemInfo oii,
SKUInfo skui, UserInfo ui, DateTime now)
```

ProcessEProductInternal

By default, the method is called when an order is paid (during the **ProcessOrderIsPaidChangeInternal** method).

Enables or disables downloading files of a specified e-product (*SKUInfo skui*) contained in a specified order (*OrderInfo order*) as a specified order item (*OrderItemInfo item*) from a specified date and time (*DateTime now*). Enabling or disabling is decided based on the payment status of the order.

```
protected virtual void ProcessEProductInternal(OrderInfo order, OrderItemInfo item,
SKUInfo skui, DateTime now)
```

Example – Adding extra credit after creating new orders

The following example demonstrates how to add extra credit to the credit account of every customer who creates an order with the total price higher than 1000 in the site's main currency.

```
using System;

using CMS;
using CMS.Ecommerce;
using CMS.SiteProvider;

[assembly: RegisterCustomProvider(typeof(CustomOrderInfoProvider))]

public class CustomOrderInfoProvider : OrderInfoProvider
{
    protected override void SetOrderInfoInternal(OrderInfo orderObj)
    {
        // Indicates whether the set object is a new order
        bool newOrder = ((orderObj != null) && (orderObj.OrderID <= 0));

        // Updates the order or creates a new order using the default API
        base.SetOrderInfoInternal(orderObj);

        // Adds extra credit for each new order with the total price higher than 1000
        if (newOrder && (orderObj.OrderTotalPriceInMainCurrency > 1000))
        {
            // Creates a new credit event
            CreditEventInfo extraCredit = new CreditEventInfo();

                        // Sets the credit event's general properties
            extraCredit.EventName = string.Format("Credit for your order: {0}",
orderObj.OrderID);
            extraCredit.EventDate = DateTime.Now;
            extraCredit.EventDescription = "Thank you for your order.";
            extraCredit.EventCustomerID = orderObj.OrderCustomerID;

            // Sets the credit event's value in the site main currency
            extraCredit.EventCreditChange = 10;

            // Sets credit as site credit or as global credit according to the settings
            extraCredit.EventSiteID = ECommerceHelper.GetSiteID(SiteContext.
CurrentSiteID, ECommerceSettings.USE_GLOBAL_CREDIT);

            // Saves the credit event
            CreditEventInfoProvider.SetCreditEventInfo(extraCredit);
        }
    }
}
```