The system uses Info providers to work with individual types of objects. A provider class contains methods for creating, updating, getting, and deleting the data of the corresponding objects. By customizing the appropriate Info provider, you can change or extend the functionality of nearly any object in the system.

The following example customizes the **OrderInfoProvider**, which the E-commerce solution uses to manage product orders. The example adds a custom action when creating orders – for orders that fulfill certain conditions, credit is added to the given customer.

You can use the same general approach to customize any standard provider, helper or manager class in Kentico (i.e. those that inherit from the **AbstractInfoProvider**, **AbstractHelper** or **AbstractManager** class respectively).

Writing the custom provider

Start by preparing 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.SiteProvider.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 implementing the custom provider class:

1. Add a new class under the custom project. For example, name the class ***CustomOrderInfoProvider.cs***.
2. Modify the class's declaration so that it inherits from the **CMS.Ecommerce.OrderInfoProvider** class.

> Custom providers must always inherit from the original provider class. This ensures that the custom provider supports all of the default functionality and allows you to add your own customizations by overriding the existing members of the class.

3. Override the class's **SetOrderInfoInternal** method.
4. Register the custom provider by adding the **RegisterCustomProvider** assembly attribute above the class declaration (requires a reference to the **CMS** namespace). The attribute's parameter specifies the type of the custom provider class as a **System.Type** object.

```
using System;

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

// Registers the custom OrderInfoProvider
[assembly: RegisterCustomProvider(typeof(CustomOrderInfoProvider))]

public class CustomOrderInfoProvider : OrderInfoProvider
{
    /// <summary>
    /// Sets (updates or inserts) the specified order.
    /// </summary>
    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 of the
base class
        base.SetOrderInfoInternal(orderObj);

        // Adds extra credit for each new order with a 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);
        }
    }
}
```

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

The override of the **SetOrderInfoInternal** method ensures that customers who purchase orders with a total price higher than 1000 (in the site's main currency) receive extra customer credit. You can use the same approach to add any other types of custom actions.