Binding classes represent relationships between other classes. Add a binding class to your custom modules if you need to store many-to-many relationships between a custom class and another class in the system (custom or default). For example, bindings allow you to assign objects to sites, users, etc.

When planning relationships between classes, consider the following alternatives to binding classes:

- Use reference fields (foreign keys) to define one-to-many relationships between standalone classes – each object of a class can have a relationship with one object of another class.
- Use parent-child relationships to define a hierarchy for classes.

Page bindings

We do NOT recommend creating bindings between custom classes and Kentico pages. Pages in Kentico are complex data structures that consist of multiple classes and have special logic for handling other factors, such as workflow.

Custom page bindings do not provide automatic support for many types of functionality that works for bindings between other classes, such as deleting of binding records when a related page is deleted.

Adding binding classes

You can create two types of binding classes:

- Binding class with its own primary key / identity column (recommended)
- Binding class with a compound primary key consisting of two or more foreign keys

Primary key with identity

Use the following steps to create a binding class with a dedicated identity column:

1. Open the **Modules** application and edit the custom module where you want to create the binding class.
2. Select the **Classes** tab and click **New class**.
3. Fill in the **Class display name** and **Class** identifier.
4. Click **Next**.
5. Confirm or change the **Primary key name** of the identity column.

   Important: Leave the **Is M:N table** option disabled.

6. Clear the **Include Guid field** and **Include LastModified field** checkboxes. These fields are not relevant for binding classes.
7. Click **Next**.
8. Leave the default primary key field. Click **New field** to create *two or more* standard fields for storing the IDs of the related objects:
   - **Field type**: Standard field
   - **Data type**: Integer number
   - **Required**: yes (checked)
   - **Reference to**: select the appropriate class
   - **Reference type**: Binding
   - **Display field in the editing form**: no (not checked)
9. Click **Save** for each field and then **Next**.
10. Click **Finish**.

The system creates the corresponding table in the database. Perform the following steps to prepare the class's code and finalize the database table:

1. Select the **Code** tab in the editing interface of the binding class.

2. Click **Save code**. The system generates an *Info* and *InfoProvider* class for the binding object.
3. Edit the *Info* class (for example in Visual Studio).
4. Navigate to the **TYPEINFO** field in the class's code.
5. Add the **IsBinding** property to the initializer of the new **ObjectTypeInfo** object and set the value to *true*. See Setting the type information for binding classes for details.

   Note: On web application installations, you need to manually include the *Info* and *InfoProvider* class files into **CMSApp** or the dedicated module project, and build the solution.

6. Manually set the class's ID fields as foreign keys in your database. See Create Foreign Key Relationships for more information.

The binding class is now ready. You can create relationships between objects using the API or prepare a user interface for this purpose. See the Example section to learn more.

Compound primary key

Use the following steps to create a binding class without its own identity column:

1. Open the **Modules** application and edit the custom module where you want to create the binding class.
2. Select the **Classes** tab and click **New class**.
3. Fill in the **Class display name** and **Class** identifier.
4. Click **Next**.
5. Enter the **Primary key name**. We recommend using the name of the identifier column of the first class in the relationship.
6. Enable **Is M:N table**.
7. Clear the **Include Guid field** and **Include LastModified field** checkboxes. These fields are not relevant for binding classes.
8. Click **Next**.
9. Set the following options for the default primary key field:
   - **Reference to**: select the first class in the relationship
   - **Reference type**: Binding
10. Click **Save**.
11. Click **New field** to create *one or more* additional fields for storing the IDs of the related objects:
    - **Field type**: Primary key (use the *Primary key* field type for all of the ID fields in Kentico — the result in the database is a compound primary key)
    - **Data type**: Integer number
    - **Required**: yes (checked)
    - **Reference to**: select the appropriate class
    - **Reference type**: Binding
    - **Display field in the editing form**: no (not checked)
12. Click **Save** for each field and then **Next**.
13. Click **Finish**.

The system creates the corresponding table in the database. Perform the following steps to prepare the class's code and finalize the database table:

1. Select the **Code** tab in the editing interface of the binding class.
2. Click **Save code**. The system generates an *Info* and *InfoProvider* class for the binding object.
3. Edit the *Info* class (for example in Visual Studio).
4. Navigate to the **TYPEINFO** field in the class's code.
5. Set the **idColumn** parameter to *null* in the **ObjectTypeInfo** constructor (the fourth parameter).
6. Add the **IsBinding** property to the initializer of the new **ObjectTypeInfo** object and set the value to *true*. See Setting the type information for binding classes for details.

Note: On web application installations, you need to manually include the *Info* and *InfoProvider* class files into **CMSApp** or the dedicated module project, and build the solution.

7. Manually set the class's ID fields as foreign keys in your database. See Create Foreign Key Relationships for more information.

The binding class is now ready. You can create relationships between objects using the API or prepare a user interface for this purpose. See the Example section to learn more.

Setting the type information for binding classes

The **type information** is code that defines the general behavior and basic properties of classes in Kentico. You can configure the type information inside the *Info* class of a given Kentico class using an object of the **ObjectTypeInfo** type.

See the sections below to learn how to set the type information for binding classes.

Note: When you generate the *Info* class from the interface of the **Modules** application and have the fields set correctly for the binding class, the system configures *most* of the type information automatically.

Parameters of the ObjectTypeInfo constructor:

- **idColumn (4)** - the name of the identity column of the binding class. Must be *null* for binding classes that do not have a dedicated identity column (with a compound primary key).
- **siteIDColumn (10)** - for bindings to site objects, must contain the name of the class column that stores the ID of the related site. Set to *null* for bindings between non-site objects.
- **parentID (11)** - the name of the column that stores the ID of the first class in the relationship.
- **parentObjectType (12)** - the object type name of the first class in the relationship. You can get the value from the OBJECT_TYPE constant in the given class's *Info* class.

Properties in the ObjectTypeInfo initializer:

- **ModuleName** - the code name of the module that contains the binding class.
- **IsBinding** - always set to *true* for binding classes.
- **DependsOn** - List of **ObjectDependency** objects representing the classes in the relationship. Add an ObjectDependency for each class in the relationship except for the one specified via the *parent* parameters in the constructor. Not required for site bindings.

Examples

```
// Sample type information definition for a general binding class
// Relationship between custom offices and roles
public static ObjectTypeInfo TYPEINFO = new ObjectTypeInfo(typeof
(OfficeRoleInfoProvider), OBJECT_TYPE, "CompanyOverview.OfficeRole", "OfficeRoleID",
null, null, null, null, null, null, "OfficeID", "companyoverview.office")
{
        ModuleName = "CompanyOverview",
        IsBinding = true,
        DependsOn = new List<ObjectDependency>()
        {
                new ObjectDependency("RoleID", "cms.role", ObjectDependencyEnum.
Binding),
        },
};


// Sample type information definition for a site binding class
// Relationship between custom offices and sites
public static ObjectTypeInfo TYPEINFO = new ObjectTypeInfo(typeof
(OfficeSiteInfoProvider), OBJECT_TYPE, "CompanyOverview.OfficeSite", "OfficeSiteID",
null, null, null, null, null, "SiteID", "OfficeID", "companyoverview.office")
{
        ModuleName = "CompanyOverview",
        IsBinding = true
};
```

Example - Creating a binding class

The following example demonstrates how to create a binding class, including an editing interface. The sample binding class allows the creation of relationships between offices and users. Each relationship indicates that a user is a manager of a given office.

To follow the example, you first need to create the **Company overview** custom module according to the instructions in Creating custom modules.

Preparing the binding class

1. Open the **Modules** application and edit the **Company overview** module.
2. Select the **Classes** tab and click **New class**.
3. Fill in the class names:
    - **Class display name**: Office manager binding
    - **Class**: OfficeUser
4. Click **Next**.
5. In step 2, clear the **Include OfficeUserGuid field** and **Include OfficeUserLastModified field** checkboxes. These fields are not relevant for binding classes.
6. Click **Next**.
7. Create two fields for storing the IDs of the related offices and users. Click **New field**, set the properties, and click **Save** for each field:

    - **Field type**: Standard field
    - **Field name**: OfficeID
    - **Data type**: Integer number
    - **Required**: yes (checked)

- **Reference to**: Custom office
- **Reference type**: Binding
- **Display field in the editing form**: no (not checked)

- **Field type**: Standard field
- **Field name**: UserID
- **Data type**: Integer number
- **Required**: yes (checked)
- **Reference to**: User
- **Reference type**: Binding
- **Display field in the editing form**: no (not checked)

8. Click **Next**.
9. Click **Finish**.

Generate the code required for the binding class's API (including the type information):

1. Select the **Code** tab in the editing interface of the *Office manager binding* class.
2. Click **Save code**. The system generates an *Info* and *InfoProvider* class for the binding object in the *~/App_Code /CMSModules/CompanyOverview* folder.
3. Open your web project in Visual Studio and edit ***OfficeUserInfo.cs***.

Note: On web application installations, the system generates the files in the **Old_App_Code** folder. You need to manually include the files into the **CMSApp** project and build the solution.

4. Navigate to the **TYPEINFO** field in the class's code.
5. Add the **IsBinding** property to the initializer of the new **ObjectTypeInfo** object and set the value to *true*:

```
public static ObjectTypeInfo TYPEINFO = new ObjectTypeInfo(typeof
(OfficeUserInfoProvider), OBJECT_TYPE, "CompanyOverview.OfficeUser",
"OfficeUserID", null, null, null, null, null, null, "OfficeID", "companyoverview.
office")
{
        ModuleName = "CompanyOverview",
        IsBinding = true,
        DependsOn = new List<ObjectDependency>()
        {
                new ObjectDependency("UserID", "cms.user", ObjectDependencyEnum.
Binding),
        },
};
```

6. Save the change (build the project on web application installations).

Finally, create a resource string for displaying the binding class's object type name:

1. In the Kentico administration interface, open the **Localization** application.
2. On the **Resource strings** tab, click **New string**.
3. Enter the following **Key**: *ObjectType.CompanyOverview_OfficeUser* (the general format is *ObjectType.<class code name with an underscore>*)
4. Type the following text for the English version of the key: *Office manager binding*
5. Click **Save**.

The system uses the resource string in the administration interface, for example when selecting classes.

Creating foreign keys for the database table

You need to manually set the ID columns of your custom binding classes as foreign keys in the corresponding database table.

In this example, create foreign keys for the **OfficeID** and **UserID** columns in the **CompanyOverview_OfficeUser** table. Add relationships with the corresponding columns in the **CompanyOverview_Office** and **CMS_User** tables respectively.

You can either use SQL Server Management Studio or execute an SQL script. See Create Foreign Key Relationships for more information.

Building a binding management interface

The following steps show how to create a custom page in the administration interface that allows users to set relationships between objects:

1. In the **Modules** application, edit the **Company overview** module.
2. Select the **User interface** tab.
3. Expand the **CMS -> Administration -> Custom** element in the UI element tree.

Adding tabs to the office editing interface:

1. Select **Company overview** in the UI element tree.
2. Click **New element** (+).
3. Set the following properties for the element:
   - **Display name**: Edit office tabs
   - **Code name**: EditOfficeTabs (**Important**: The code name of elements for editing objects under listings must always start with the ***Edit*** keyword)
   - **Module**: Company overview
   - **Display breadcrumbs**: yes
   - **Page template**: Vertical tabs
4. Click **Save**.

Moving the original office editing element under the tabs:

1. Select the **Edit office** element (from the example in Creating custom modules).
2. Change the element's names and move it under the new tabs element:
   - **Display name**: General
   - **Code name**: GeneralEditOffice
   - **Parent element**: Edit office tabs
3. Click **Save**.
4. Open the **Properties** tab and disable the **Display breadcrumbs** property.
5. Click **Save**.

Adding the binding management element:

1. Select the **Edit office tabs** element in the tree.
2. Click **New element** (+).
3. Set the following properties for the element:
   - **Display name**: Managers
   - **Code name**: ManagersEditOffice
   - **Module**: Company overview
   - **Page template**: Edit bindings
4. Click **Save**.
5. Switch to the **Properties** tab and set the following values in the **Binding** category:
   - **Binding object type**: Office manager binding
   - **Target object type**: User
   - **Where condition**: OfficeID = {% UIContext.ObjectID %}

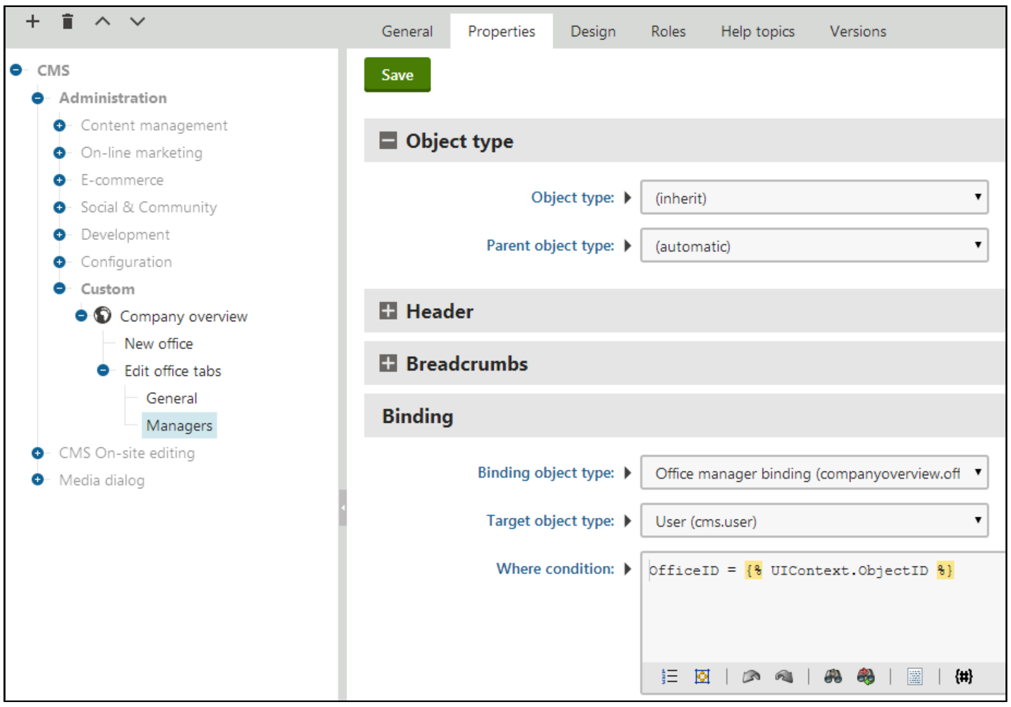
The following properties define the **Office – User** binding:

- **Object type** - the first class in the relationship (in this case, the element inherits the ***Custom office*** object type from the parent elements)
- **Binding object type** - the binding class itself (***Office manager binding***)
- **Target object type** - the second class in the relationship (***User*** in this case)

The **Where condition** ensures that only the relationships of the currently edited office object are displayed.

6. Click **Save**.

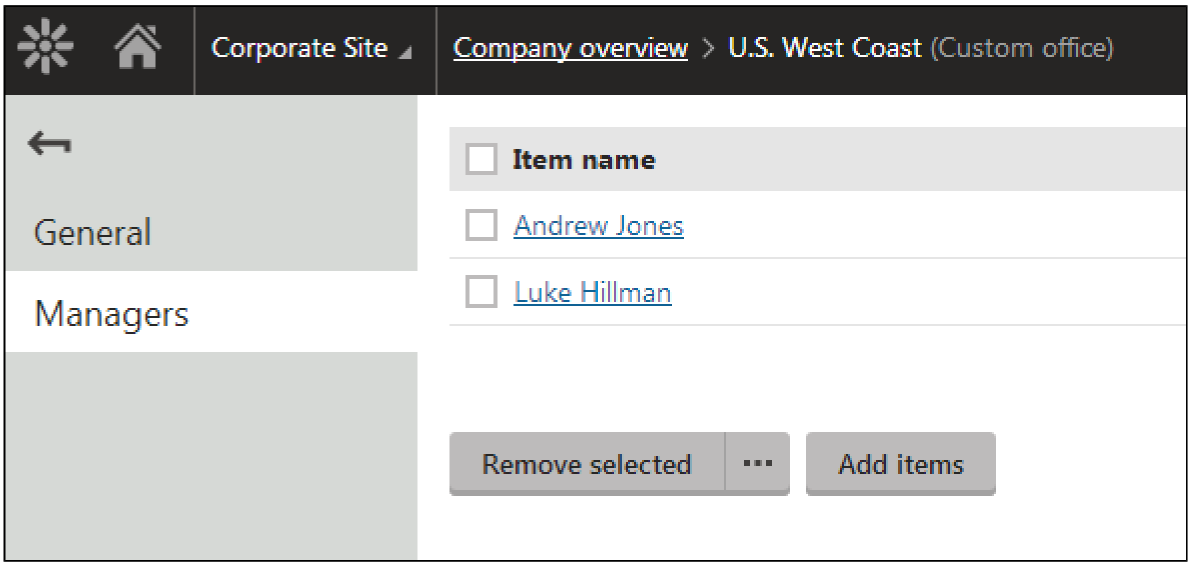
The binding editing interface is now ready.



Object type: (inherit)
Parent object type: (automatic)
Binding object type: Office manager binding (companyoverview.offi
Target object type: User (cms.user)
Where condition:

```
OfficeID = {% UIContext.ObjectID %}
```

If you open the **Company overview** application in the **Custom** category and edit an office, you can manage the office-user bindings on the **Managers** tab.


Corporate Site
Company overview > U.S. West Coast (Custom office)
General
Managers
Item name
Andrew Jones
Luke Hillman
Remove selected
...
Add items