



When building the user interface for custom modules, you can either use the portal engine (as described in [Creating custom modules](#)) or develop pages manually as standard web forms.

With the manual approach of designing UI elements, you:

-  Cannot use the browser-based design interface and automatic features of the portal engine
-  Develop the interface in Visual Studio, which gives you full control over the content and logic of the pages

Development approaches for custom user interface pages

Before you start developing a user interface page that requires customization, carefully consider which of the following approaches best fits your needs:



- **Manual user interface pages that target a custom web form** - recommended for pages that do not fit into the patterns of the default UI templates or require heavy customization.
- **Pages with portal engine templates and extenders** - allows you to leverage the templates and automatic features of the portal engine and add custom code-behind logic. Suitable for pages that share a common pattern with other pages (for example object listing pages), but require one or more custom adjustments. See [Creating extenders for module interface pages](#) for more information about this approach.

Example

The following example demonstrates how to manually create a listing page for the sample *Company overview* module. See [Creating custom modules](#) to learn how to add the module and its *Office* class.

Adding the UI element

Start by creating a UI element:

1. Open the **Modules** application.
2. Edit () the **Company overview** module.
3. Switch to the **User interface** tab.
4. Select the **CMS -> Administration -> Custom** element in the tree.
5. Click **New element** (.
6. Enter the following values:
 - **Display name:** Manual office list
 - **Code name:** ManualOfficeList
 - **Module:** Company overview
 - **Element icon type:** Class
 - **Element icon CSS class:** icon-app-localization
 - **Type:** URL
 - **Target URL:** ~/CMSModules/CompanyOverview/OfficeListing.aspx
7. Click **Save**.

The system creates the new UI element. The position in the user interface tree under the **CMS -> Administration -> (Category)** section identifies the new element as an [application](#).

Creating the web form

Next, you need to build a web form that serves the URL of the UI element:

1. Open your project in Visual Studio (using the **WebSite.sln** or **WebApp.sln** file).
2. Expand the **CMSModules** folder and add a new folder that matches the module's code name (*CompanyOverview* for the sample module).
3. Right-click the new folder and select **Add -> Add New Item**.
4. Choose a **Web Form** with a master page.
5. For example, name the web form *OfficeListing.aspx* and click **Add**.



6. Select the **CMSMasterPages/UI/SimplePage.master** master page.
7. Set the **Theme** attribute to *Default* in the page declaration.
8. Register and place the UniGrid control into the page's content. See [UniGrid](#) to learn more.

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="OfficeListing.aspx.cs"
MasterPageFile="~/CMSMasterPages/UI/SimplePage.master" Inherits="
CMSModules_CompanyOverview_OfficeListing" Theme="Default" %>

<%@ Register src="~/CMSAdminControls/UI/UniGrid/UniGrid.ascx" tagname="UniGrid"
tagprefix="cms" %>
<%@ Register Namespace="CMS.UIControls.UniGridConfig" TagPrefix="ug" Assembly="CMS.
UIControls" %>

<asp:Content runat="server" ContentPlaceHolderID="plcContent" ID="plcOfficeList">

    <cms:UniGrid ID="officeList" runat="server" ObjectType="CompanyOverview.Office"
>

    <GridActions>
        <ug:Action Name="edit" Caption="$General.Edit$" FontIconClass="icon-
edit" FontIconStyle="allow" />
        <ug:Action Name="#delete" Caption="$General.Delete$" FontIconClass="
icon-bin" FontIconStyle="critical" Confirmation="$General.ConfirmDelete$" />
    </GridActions>

    <GridColumns>
        <ug:Column Source="OfficeDisplayName" Caption="Office name" Localize="
true">
            <Filter Type="Text" Size="200" />
        </ug:Column>
        <ug:Column Source="OfficeAddress" Caption="Address" Width="100%" />
    </GridColumns>

    <GridOptions DisplayFilter="true" />

</cms:UniGrid>

</asp:Content>
```

Edit the web form's code behind:

1. Make the class inherit from **CMSPage**.
2. Add the **UIElement** attribute above the class declaration to bind the page to the correct module and UI element. Specify the following parameters of the attribute:
 - *resourceName* – the code name of the module.
 - *elementName* – the code name of the UI element.
3. Add any required code behind logic.



```
using CMS.UIControls;
using CMS.Helpers;
using CMS.Base.Web.UI.ActionsConfig;

[UIElement("CompanyOverview", "ManualOfficeList")]
public partial class CMSModules_CompanyOverview_OfficeListing : CMSPage
{
    protected void Page_Load(object sender, EventArgs e)
    {
        // Creates and adds the "New office" button as a header action
        HeaderAction newOffice = new HeaderAction
        {
            Text = GetString("New office"),
            RedirectUrl = "~/CMSModules/CompanyOverview/NewOffice.
aspx"

        };

        CurrentMaster.HeaderActions.AddAction(newOffice);

        // Sets the URL for the edit action
        officeList.EditActionUrl = "~/CMSModules/CompanyOverview/EditOffice.
aspx";
    }
}
```

The sample code above:

- Binds the page to the *Company overview* module and the UI element created in the previous section
- Adds a *New office* header action and sets the URL of the edit button of the UniGrid (the new and edit pages are not implemented).

Users can access the page as a standard Kentico application. You can use a similar approach to add any required content or functionality to the Kentico administration interface.