

Kentico CMS 6.0 Controls

Table of Contents

Kentico CMS Controls	8
Overview	8
Configuring your project for Kentico CMS Controls	8
Using ASPX page templates	10
Transformations	12
Controls hierarchy	15
Generic Controls	16
Overview	16
UniView	17
Overview	17
Getting started	17
Configuration	21
Appearance and styling.....	22
Paging controls	23
Overview	23
Paging controls - common properties.....	23
DataPager	24
Overview	24
Getting started.....	24
Configuration.....	25
Appearance and styling.....	27
TemplateDataPager	28
Overview	28
Getting started.....	28
Configuration.....	31
Appearance and styling.....	31
UniPager	32
Overview	32
Getting started.....	33
Configuration.....	35
Structure.....	36
Appearance and styling.....	38
Implementing the IUniPageable interface.....	39
Basic Controls	42
Overview	42
Navigation	42
Overview	42
BasicTabControl	42
Overview	42
Getting started.....	43
Configuration.....	45
Appearance and styling.....	46
Listings and viewers	46
Overview	46
BasicCalendar	47

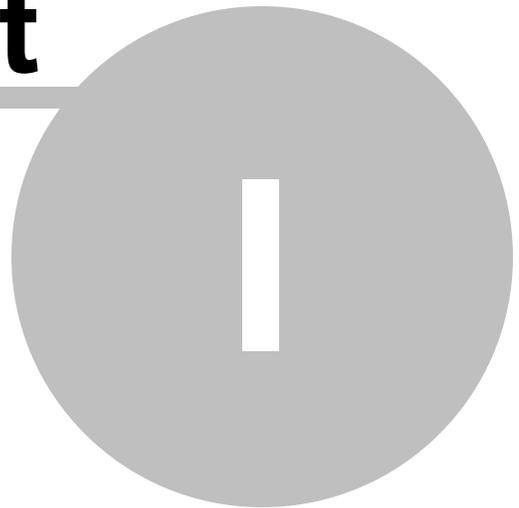
Overview	47
Getting started.....	47
Configuration.....	49
Appearance and styling.....	50
BasicDataGrid	50
Overview	50
Getting started.....	51
Configuration.....	52
Appearance and styling.....	53
BasicDataList	53
Overview	53
Getting started.....	54
Configuration.....	56
Appearance and styling.....	57
BasicRepeater.....	57
Overview	57
Getting started.....	58
Configuration.....	60
Appearance and styling.....	61
BasicUniView	61
Overview	61
Getting started.....	62
Configuration.....	64
Appearance and styling.....	64
CMS Controls.....	65
Overview	65
Path specification in controls and web parts	65
Caching	66
CMS controls - common properties	69
Navigation	72
Overview	72
Document menu settings.....	72
Using the CSSPrefix property.....	75
CMS navigation - common properties.....	76
CMSBreadCrumbs.....	77
Overview	77
Getting started.....	77
Configuration.....	78
Appearance and styling.....	79
CMSListMenu	80
Overview	80
Getting started.....	80
Configuration.....	81
Appearance and styling.....	83
General.....	83
Creating a horizontal drop-down menu using CSS styles.....	84
Creating a vertical drop-down menu using CSS styles.....	85
CMSMenu	86
Overview	86
Getting started.....	87
Configuration.....	89
Appearance and styling.....	91
CMSSiteMap	91
Overview	91

Getting started.....	92
Configuration.....	93
Appearance and styling.....	94
CMSTabControl.....	95
Overview.....	95
Getting started.....	95
Configuration.....	97
Appearance and styling.....	98
CMSTreeMenu.....	98
Overview.....	98
Getting started.....	98
Configuration.....	99
Appearance and styling.....	101
CMSTreeView.....	103
Overview.....	103
Getting started.....	104
Configuration.....	104
Appearance and styling.....	106
Listings and viewers.....	107
Overview.....	107
Standard listings and viewers.....	107
Overview.....	107
Using nested controls.....	107
Displaying related documents.....	110
CMSCalendar.....	115
Overview.....	115
Getting started.....	116
Configuration.....	117
Appearance and styling.....	117
CMSDataGrid.....	117
Overview.....	117
Getting started.....	118
Configuration.....	119
Appearance and styling.....	120
CMSDataList.....	120
Overview.....	120
Getting started.....	121
Configuration.....	122
Appearance and styling.....	123
CMSDocumentValue.....	123
Overview.....	123
Getting started.....	123
Configuration.....	124
CMSRepeater.....	124
Overview.....	124
Getting started.....	125
Configuration.....	126
Appearance and styling.....	127
CMSUniView.....	127
Overview.....	127
Getting started.....	128
Configuration.....	129
Using hierarchical transformations.....	133
Appearance and styling.....	134

CMSViewer.....	134
Overview.....	134
Getting started.....	135
Configuration.....	136
Appearance and styling.....	136
Listings and viewers with a custom query.....	136
Overview.....	136
Using control properties to set query clauses.....	137
CMS Custom query - common properties.....	138
QueryDataGrid.....	138
Overview.....	138
Getting started.....	139
Configuration.....	140
Appearance and styling.....	140
QueryDataList.....	140
Overview.....	140
Getting started.....	141
Configuration.....	142
Appearance and styling.....	143
QueryRepeater.....	143
Overview.....	143
Getting started.....	144
Configuration.....	145
Appearance and styling.....	146
QueryUniView.....	146
Overview.....	146
Getting started.....	147
Configuration.....	148
Appearance and styling.....	152
Edit mode buttons.....	153
Overview.....	153
CMSEditModeButtonAdd.....	153
Overview.....	153
Getting started.....	153
Configuration.....	154
Appearance and styling.....	155
CMSEditModeButtonEditDelete.....	155
Overview.....	155
Getting started.....	155
Configuration.....	157
Appearance and styling.....	157
Editable regions for ASPX page templates.....	157
Overview.....	157
CMSEditableImage.....	158
Overview.....	158
Getting started.....	158
Configuration.....	159
CMSEditableRegion.....	160
Overview.....	160
Getting started.....	160
Configuration.....	161
Appearance and styling.....	162
CMSPageManager.....	162
Overview.....	162

Getting started.....	163
Configuration.....	163
Appearance and styling.....	165
Search Controls	165
Overview	165
CMSSearchDialog.....	166
Overview	166
Getting started.....	166
Configuration.....	167
Structure.....	168
Appearance and styling.....	169
CMSSearchResults.....	170
Overview	170
Configuration.....	170
Structure.....	173
Appearance and styling.....	174
UI Controls.....	175
Overview	175
UniGrid	175
Overview	175
Getting started.....	175
Implementing custom functionality.....	178
Configuration	181
UniGrid definition.....	184
UniSelector	196
Overview	196
Getting started.....	196
Configuration	198

Part



Kentico CMS Controls

1 Kentico CMS Controls

1.1 Overview

Kentico CMS Controls are standard ASP.NET **server controls** that can be used in Visual Studio 2005, 2008 or 2010. You can place them in your user control .ascx files that implement [custom web parts](#), on ASPX page templates and pages that do not use the portal engine. Some of them can also be used outside of Kentico CMS.

Kentico CMS Controls work on the .NET 3.5 SP1 and 4.0 Frameworks.

Before you start using them, please make sure your project is configured as described in [Configuring your project for Kentico CMS Controls](#).

As mentioned, ASPX page templates are a common place to use controls. The [Using ASPX page templates](#) topic contains a guide describing how a new page template can be created and prepared to fully utilize CMS controls.

Many controls use transformations to customize the way they display data. More information about this can be found in the [Transformations](#) topic.

To learn more about the various types and inheritance hierarchy of Kentico CMS controls, please see the [Controls hierarchy](#) topic.



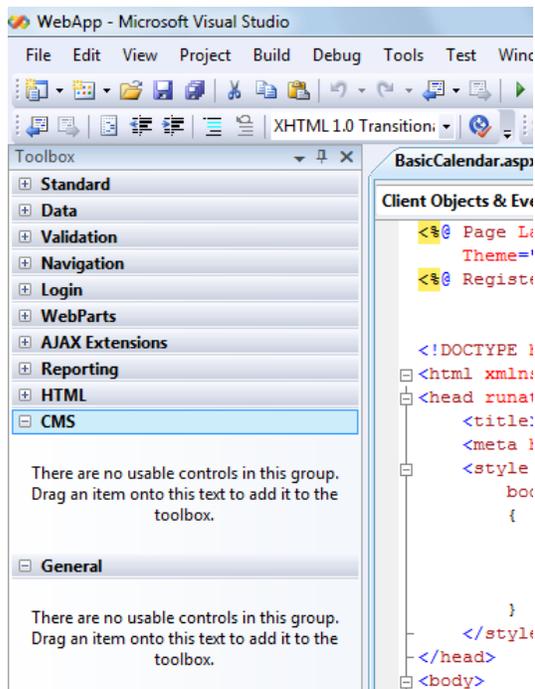
Corporate Site sample website required

The examples found in topics describing individual controls further in this guide assume that your Kentico CMS database contains data for the sample Corporate Site website.

1.2 Configuring your project for Kentico CMS Controls

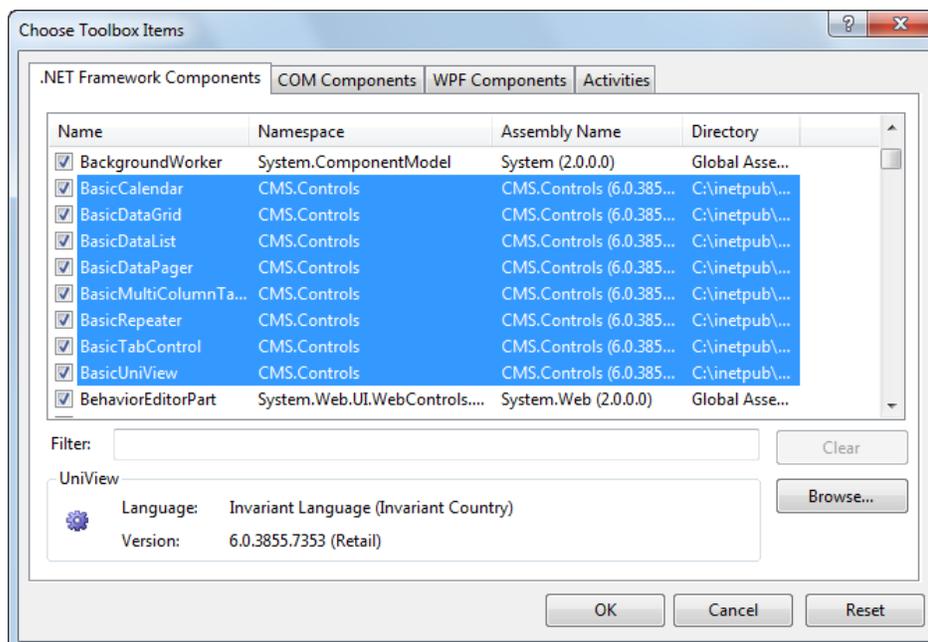
Before you start using Kentico CMS Controls in your ASP.NET project, you need to add the controls to the **Toolbox**:

1. Open the website project in Visual Studio and open an ASPX page.
2. Right-click the **Toolbox** and choose **Add tab** from the context menu.
3. Type the name of the new tab (e.g. *CMS*) and press Enter:

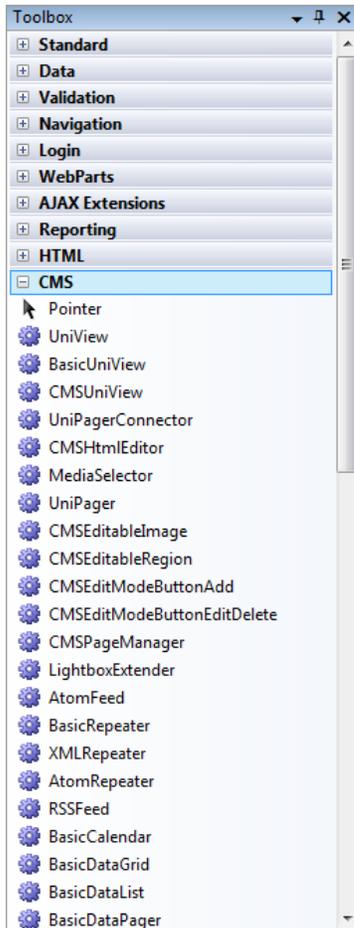


4. Right-click the new tab and choose **Choose Items...** from the context menu.

5. In the **Choose Toolbox Items** dialog, click **Browse** and locate the **CMS.Controls.dll** library in the **Bin** folder under your website. Click **Open** and then click **OK**.



6. The controls are now added to the **Toolbox**:



7. Now you can easily drag and drop the controls onto your Web forms.

1.3 Using ASPX page templates

Placing controls on ASPX page templates is one of their most common uses. For additional information about page templates, please refer to [Developer's Guide -> Development -> Web development overview -> ASPX page template development](#).

The following is a step-by-step tutorial showing how a new ASPX page template can be created and registered in the system.

1. Open your Kentico CMS web project in **Visual Studio** using **File -> Open -> Web Site...** in the menu.
2. Now right-click the **CMSTemplates/CorporateSite** folder in the Solution Explorer and select **Add New Item**.
3. Choose to create a new **Web Form** (.aspx page) and check the **Select master page** box. Click **Add**.
4. The **Select a Master Page** dialog appears. Choose a master page file and click **OK**. You can use the default ASPX master page in the **CMSTemplates/CorporateSite** folder named **root.master**.

5. Switch to the **Source** view of the newly created web form. Add the following line under the `<%@ Page %>` directive:

```
<%@ Register Assembly="CMS.Controls" Namespace="CMS.Controls" TagPrefix="cms" %>
```

You can now add any HTML code inside the `<asp:Content>` element, including any **CMS controls** and their definitions.

6. Switch to the code behind. You need to add a reference to the **CMS.UIControls** namespace:

[C#]

```
using CMS.UIControls;
```

7. The last step is to modify the class from which the page is inherited. Change the following code:

[C#]

```
public partial class CMSTemplates_CorporateSiteAspx_Example : System.Web.UI.Page
```

to this:

[C#]

```
public partial class CMSTemplates_CorporateSiteAspx_Example : TemplatePage
```

Now the page can be correctly used as a page template in Kentico CMS.

Please keep in mind that the name of the class must be identical to the value of the **Inherits** attribute of the `<%@ Page %>` directive on the ASPX page. This is case sensitive.

Registering the ASPX page as a page template

Now that we have created a new ASPX page, we need to register it in Kentico CMS as a page template so that it can be used.

8. Sign in to **Site Manager** (we recommend doing this on the sample Corporate Site when following this tutorial for the controls in this guide) and go to **Development -> Page templates**. Select a folder, for example **Corporate Site/Examples**, click  **New template** and enter a display and code name.

Click **OK**. Now use the **Select** button to choose the .aspx file created in the previous steps located in the `~/CMSTemplates/CorporateSite` folder.

Click  **Save**.

9. Now switch to the **Sites** tab, assign the page template to the websites where you wish for it to be

available using the **Add sites** button and click **OK**.

Creating a page based on the new page template

10. Go to **Kentico CMS Desk -> Content**. Create a **New** document of type **Page (menu item)**. Enter a **Page name** and select the **Use existing page template** option. Select the page template created in the previous steps and click  **Save** to create the page.

Now you have a page using an ASPX page template. Any changes made to the source .aspx file will now be automatically reflected by this page.

1.4 Transformations

Transformations are pieces of code that determine how Kentico CMS documents, or certain parts of them, are rendered by listing web parts and controls. They take raw data from the Kentico CMS database and *transform* it into the form you wish it to appear in. This makes them a crucial tool when displaying documents and document related data on the pages of your website.

Their functionality is very similar to that of templates used by standard ASP.NET list controls such as the Repeater, which can be defined within the tags of a control through various **ItemTemplate** properties. The main difference is that our transformations are stored separately in the database and can easily be used repeatedly. They are assigned to web parts or controls through the appropriate **TransformationName** properties. The use of transformations is supported by all web parts that display document data, as well as by those listing controls that are designed to work directly with Kentico CMS documents, such as those in the [CMS Controls -> Listings and viewers](#) section of this guide.

There are several different approaches that can be used to write transformations. You can choose how a transformation's code should be rendered by setting the appropriate type. The following are available:

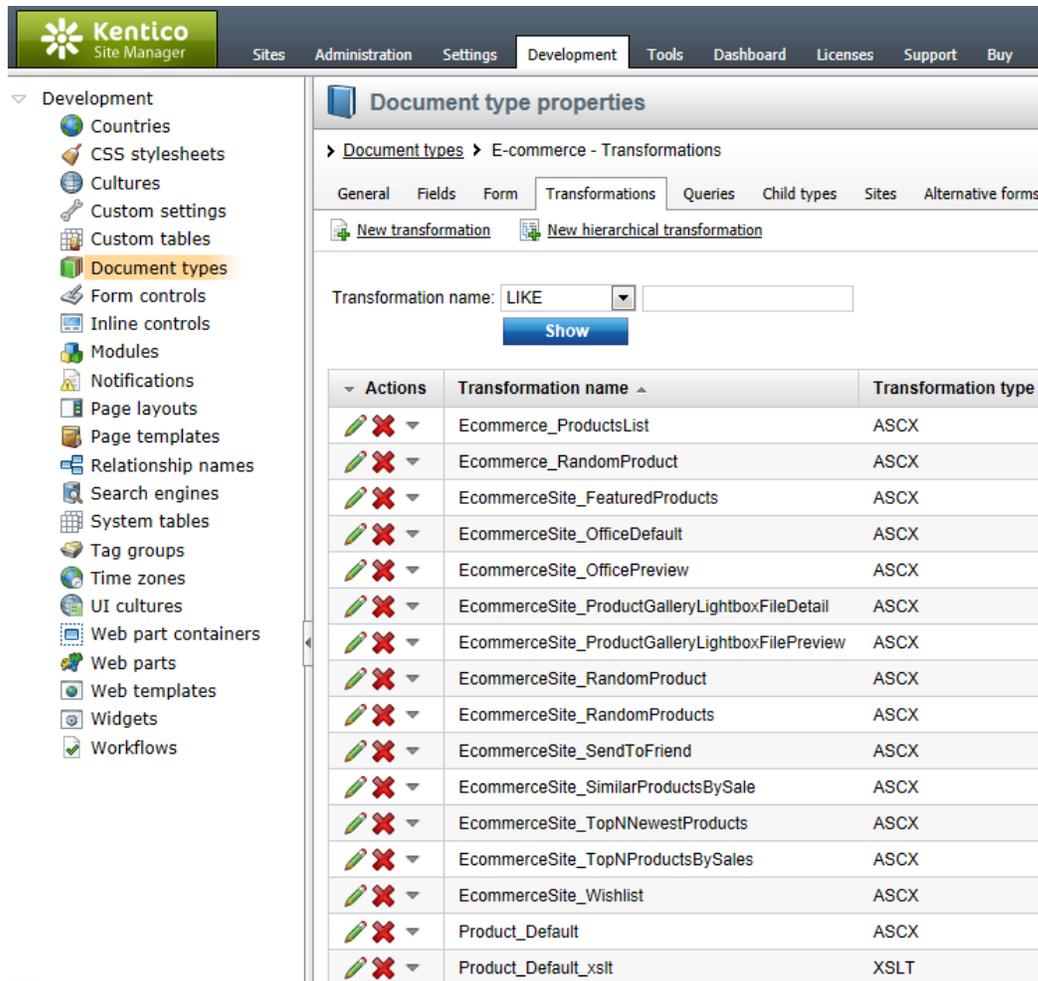
- **ASCX** - with this option, the code of the transformation will support ASCX markup, i.e. the same syntax that you would use to edit a standard web form or user control, including inline code, embedded controls, standard ASP.NET data binding expressions and special methods designed for use in transformations. Document fields can be accessed using expressions in format: `<%# Eval ("ColumnName") %>`
- **Text/XML** - the code will only be processed as basic HTML. This means that any ASCX markup, such as controls or inline code, will not be functional when the transformation is rendered. You may use Kentico CMS [Macro expressions](#) and methods to insert dynamic values into the content. Expressions in the following format allow you to easily get the values of the given document's fields: `{%ColumnName%}`
- **HTML** - works the same way as the **Text/XML** option, but editing is done through the [WYSIWYG editor](#). The rendered output of HTML code will be shown inside the editor.
- **XSLT** - this option can be selected for transformations that use XSL elements to render the data. The code must be in valid XML format.

Please note that for security reasons, the code of ASCX type transformations may only be edited by users who have the **Edit ASCX** code permission for the **Design** module. This permission can only be assigned by global administrators.

Since text-based transformations (**Text/XML** or **HTML** types) are only processed as basic HTML, they

cannot be used to compromise the security of the website. Another advantage of these transformation types is that they do not need to be compiled, which means they may be used and modified even if the Virtual path provider is not available, such as in a precompiled or medium trust environment.

Transformations are categorized under the document types or custom tables that they are supposed to display. They can be managed in the Kentico CMS administration interface at **Site Manager -> Development -> Document types** or **Custom tables -> ... Edit (✎) document type or custom table ... -> Transformations**. Some document types do not represent an object but serve only as a container for transformations and queries.



The screenshot shows the Kentico CMS Site Manager interface. The navigation menu on the left includes 'Development' with sub-items like 'Countries', 'CSS stylesheets', 'Cultures', 'Custom settings', 'Custom tables', 'Document types', 'Form controls', 'Inline controls', 'Modules', 'Notifications', 'Page layouts', 'Page templates', 'Relationship names', 'Search engines', 'System tables', 'Tag groups', 'Time zones', 'UI cultures', 'Web part containers', 'Web parts', 'Web templates', 'Widgets', and 'Workflows'. The main content area is titled 'Document type properties' and shows the 'E-commerce - Transformations' page. The 'Transformations' tab is selected, displaying a list of transformations. The 'Transformation name' field is set to 'LIKE' and a 'Show' button is visible. The table below lists the transformations:

Actions	Transformation name	Transformation type
✎ ✖	Ecommerce_ProductsList	ASCX
✎ ✖	Ecommerce_RandomProduct	ASCX
✎ ✖	EcommerceSite_FeaturedProducts	ASCX
✎ ✖	EcommerceSite_OfficeDefault	ASCX
✎ ✖	EcommerceSite_OfficePreview	ASCX
✎ ✖	EcommerceSite_ProductGalleryLightboxFileDetail	ASCX
✎ ✖	EcommerceSite_ProductGalleryLightboxFilePreview	ASCX
✎ ✖	EcommerceSite_RandomProduct	ASCX
✎ ✖	EcommerceSite_RandomProducts	ASCX
✎ ✖	EcommerceSite_SendToFriend	ASCX
✎ ✖	EcommerceSite_SimilarProductsBySale	ASCX
✎ ✖	EcommerceSite_TopNNewestProducts	ASCX
✎ ✖	EcommerceSite_TopNProductsBySales	ASCX
✎ ✖	EcommerceSite_Wishlist	ASCX
✎ ✖	Product_Default	ASCX
✎ ✖	Product_Default_xslt	XSLT

The sample sites include many transformations for all document types and you can modify them or write new transformations to suit any of your requirements.

For more information about transformations and document types, please refer to [Developer's Guide -> Development -> Document types and transformations](#).

Example

The code of an ASCX transformation used to display a list of products could look like this:

```

<div class="ProductPreview">
  <div class="ProductBox">
    <div class="ProductImage">
      <a href="<%# GetDocumentUrl() %>" style="display:block;">
<%# EcommerceFunctions.GetProductImage(Eval("SKUImagePath"), 180, Eval("SKUName"))
%>
      </a>
    </div>
    <a href="<%# GetDocumentUrl() %>">
      <span class="ProductTitle textContent">
        <%# HTMLEncode(ResHelper.LocalizeString(Convert.ToString(Eval
("SKUName")))) %>
      </span>
    </a>
    <div class="ProductFooter">
      <div class="productPrice"><%# EcommerceFunctions.GetFormattedPrice(Eval
("SKUPrice"), Eval("SKUDepartmentID"), null, 0, true, ValidationHelper.GetInteger
(Eval("SKUSiteID"), 0) == 0) %>
      </div>
    </div>
  </div>
</div>

```

When this transformation is assigned to a listing control or web part that has products (SKUs) in its data source, the output code of individual products will contain the values returned by the methods and data binding expressions, like in the following example:

```

<div class="ProductPreview">
  <div class="ProductBox">
    <div class="ProductImage">
      <a href="/KenticoCMS/Products/Smartphones/Apple-iPhone-3GS.aspx" style="
display:block;">

      </a>
    </div>
    <a href="/KenticoCMS/Products/Smartphones/Apple-iPhone-3GS.aspx">
      <span class="ProductTitle textContent">
        Apple iPhone 3GS
      </span>
    </a>
    <div class="ProductFooter">
      <div class="productPrice">$424.99
      </div>
    </div>
  </div>
</div>

```

The final output of this product on the website will then look like this:

**Please note**

The CSS stylesheet used by the page is applied to the output of the transformation. This example uses the classes from the default **Corporate Site** stylesheet.

1.5 Controls hierarchy

Kentico CMS Controls make use of the object-oriented nature of the .NET Framework and many of them are derived either from standard ASP.NET controls or from each other. This means that controls with similar functionality have many common properties and learning to use them is made easier due to this fact.

The following categories and controls are available:

- **Generic Controls** - these controls either serve as base controls that others are derived from or provide additional functionality, such as paging
 - [UniView](#)
 - **Paging controls**
 - [DataPager](#)
 - [TemplateDataPager](#)
 - [UniPager](#)
- **Basic Controls** - these controls do not use the Kentico CMS database or API and can be used with any type of bindable data; most of them are derived from either intrinsic ASP.NET control or Generic controls
 - **Navigation**
 - [BasicTabControl](#)
 - **Listings and viewers**
 - [BasicCalendar](#)
 - [BasicDataGrid](#)
 - [BasicDataList](#)

- [BasicRepeater](#)
- [BasicUniView](#)
- **CMS Controls** - these controls are designed to work exclusively with Kentico CMS documents and data; many of them are derived from Basic Controls with similar fundamental functionality
 - **Navigation**
 - [CMSBreadCrumbs](#)
 - [CMSListMenu](#)
 - [CMSMenu](#)
 - [CMSSiteMap](#)
 - [CMSTabControl](#)
 - [CMSTreeMenu](#)
 - [CMSTreeView](#)
 - **Listings and viewers**
 - **Standard listings and viewers**
 - [CMSCalendar](#)
 - [CMSDataGrid](#)
 - [CMSDataList](#)
 - [CMSDocumentValue](#)
 - [CMSRepeater](#)
 - [CMSUniView](#)
 - [CMSViewer](#)
 - **Listings and viewers with a custom query**
 - [QueryDataGrid](#)
 - [QueryDataList](#)
 - [QueryRepeater](#)
 - [QueryUniView](#)
 - **Edit mode buttons**
 - [CMSEditModeButtonAdd](#)
 - [CMSEditModeButtonEditDelete](#)
 - **Editable regions for ASPX page templates**
 - [CMSEditableImage](#)
 - [CMSEditableRegion](#)
 - [CMSPageManager](#)
 - **Search controls**
 - [CMSSearchDialog](#)
 - [CMSSearchResults](#)
- **UI Controls** - these controls are different from the others; they are **user** controls that are utilized in the interface of Kentico CMS, but can also be used for custom purposes
 - [UniGrid](#)
 - [UniSelector](#)

1.6 Generic Controls

1.6.1 Overview

The controls in this section have various functionality, and often serve as base controls that others are derived from or provide additional functionality, such as paging.

Available controls:

- [UniView](#)
- [Paging controls](#)

1.6.2 UniView

1.6.2.1 Overview

The UniView control is a universal templated data viewer with support for the displaying of hierarchical data from grouped data sources.



Grouped data sources

Grouped data sources are represented by the *GroupedDataSource* class, which takes a standard data source, such as a DataSet, and categorizes the data items into groups according to a specified column. It can also determine the hierarchical level of data items depending on the value of a given column.

An example of using this class with the UniView control can be found in the following topic.

UniView can be used with any bindable data source - it doesn't use Kentico CMS database or API.



Please note

If you want to display documents from Kentico CMS, please use the [CMSUniView](#) control that provides a more convenient way.

The following topics are available to help you familiarize yourself with the UniView control:

- [Getting started](#) - contains a step-by-step tutorial that allows you to learn the basics of using the control
- [Configuration](#) - describes and explains the properties that can be set for the control
- [Appearance and styling](#) - describes how the design of the control can be modified

1.6.2.2 Getting started

The following is a step-by-step tutorial that will show you how to display all pages (CMS.Menuitem documents) from the sample Corporate Site using the UniView control:

1. Create a new **Web form** somewhere in your website installation directory.
2. Switch to its **Design** tab, drag and drop the **UniView** control from the toolbox onto the form.
3. Add the code marked by the **UniView templates** comments between the `<cms:UniView>` tags. The overall code of the UniView control should look like this:

```

<cms:UniView ID="UniView1" runat="server">

    <%-- UniView templates
    ----- %>

    <ItemTemplate>
        <%# HTMLHelper.Encode( Convert.ToString(Eval("NodeAliasPath"))) %>
    >
    </ItemTemplate>
    <AlternatingItemTemplate>
        <font color="#999999"><%# HTMLHelper.Encode( Convert.ToString
        (Eval("NodeAliasPath"))) %>
        </font>
    </AlternatingItemTemplate>
    <SeparatorTemplate>
        </li>
        <li>
    </SeparatorTemplate>
    <HeaderTemplate>
        <ul>
        <li>
    </HeaderTemplate>
    <FooterTemplate>
        </li>
        </ul>
    </FooterTemplate>

    <%-- UniView templates
    ----- %>

</cms:UniView>

```

This sets the templates used when displaying the menu items. The control dynamically replaces the <% # ... %> tags with values of the currently displayed record. This is then repeated for every record in the data source.

4. Add the following references to the beginning of the web form code-behind:

[C#]

```

using System.Data;

using CMS.CMSHelper;
using CMS.GlobalHelper;

```

5. Now add the following code to the **Page_Load** method:

[C#]

```

//Create DataSet containing all menu item documents in the system

```

```
DataSet ds = TreeHelper.SelectNodes("/%", false, "CMS.MenuItem", "", "NodeLevel,
NodeOrder", -1, true);

//Check that the DataSet isn't empty
if (!DataHelper.DataSourceIsEmpty(ds))
{

    //Bind the DataSet to the UniView control
    this.UniView1.DataSource = ds;
    this.UniView1.DataBind();

}
```

This code reads documents from the database and provides them to the UniView control as a DataSet.

6. Save the changes to the web form and its code behind file. Now right-click it in the Solution explorer and select **View in Browser**. The resulting page should look like this:

- /Home
- /Services
- /Products
- /News
- /Partners
- /Community
- /Company
- /Media
- /Examples
- /Mobile
- /Other
- /Services/Web-design
- /Products/Smartphones
- /Partners/Silver-partners
- /Company/Careers
- /Community/Forums
- /Media/Images
- /Mobile/Home

As you can see, this displays all page (menu item) documents in the system, but a flat list is far from ideal for such a purpose. The following steps will show you how to modify the code to display a hierarchical structure.

7. Add another reference to the beginning of the web form code-behind:

[C#]

```
using CMS.SettingsProvider;
```

8. Modify the code of the **Page_Load** method to look like this:

[C#]

```
//Create DataSet containing all menu item documents in the system
DataSet ds = TreeHelper.SelectNodes("/%", false, "CMS.MenuItem", "", "NodeLevel,
NodeOrder", -1, true);

    //Check that the DataSet isn't empty
    if (!DataHelper.DataSourceIsEmpty(ds))
    {

        //Create GroupedDataSource from the ds DataSet
        GroupedDataSource gpd = new GroupedDataSource(ds, "NodeParentID",
"NodeLevel");

        //Set RelationColumnID property of the UniView control
        this.UniView1.RelationColumnID = "NodeID";

        //Bind the DataSet to the UniView control
        this.UniView1.DataSource = gpd;
        this.UniView1.DataBind();

    }
}
```

This code takes the DataSet containing the documents, groups them according to the *NodeID* of their parent document and determines their level in the hierarchy according to their *NodeLevel*.

9. Save the changes and refresh the page in your browser. It should now look like this:

- /Home
- /Services
 - /Services/Web-design
 - /Services/Web-development
 - /Services/Network-administration
- /Products
 - /Products/Smartphones
 - /Products/Laptops-and-Tablets
 - /Products/Software
 - /Products/E-Books
 - /Products/IT-Services
 - /Products/Membership
- /News
- /Partners
 - /Partners/Silver-partners
 - /Partners/Gold-partners
- /Community
 - /Community/Forums
 - /Community/Forums/Forum-search
 - /Community/Blogs
 - /Community/Events
 - /Community/Events/Events-list
 - /Community/Wiki
 - /Community/Wiki/My-contributions
- /Company
 - /Company/Careers
 - /Company/Offices
 - /Company/Donate-us
- /Media
 - /Media/Images
 - /Media/Videos
- /Examples
 - /Examples/My-home-page
 - /Examples/Development-models
 - /Examples/Development-models/ASPX
 - /Examples/Development-models/Portal-Engine

1.6.2.3 Configuration

The following properties of the UniView control can be set or used in the API:

Property Name	Description	Sample Value
AlternatingRange	Indicates how often the AlternatingItemTemplate should be used.	
AlternatingStartPosition	Indicates the item number from which the AlternatingItemTemplate should start being used.	
DataSource	The object from which the list of data items is retrieved.	
HideHeaderAndFooterForSingleItem	Indicates whether the content of the HeaderTemplate and FooterTemplate should be hidden if only a single item is displayed.	
HierarchicalDisplayMode	Sets the hierarchical display mode. <i>Inner</i> generates sub-levels inside the level above,	"Inner" "Separate"

	<i>Separate</i> generates sub-levels outside of the upper levels.	
OuterData	Data generated in the HeaderTemplate and FooterTemplate .	
PagerDataItem	Gets or sets the pager data item object.	
PagerForceNumberOfResults	If set, the DataSet containing paged items is not modified by the pager, but the pager itself behaves as if the amount of paged items were identical to this value. The value must be set to -1 for the property to be disabled.	
RelationColumnID	The name of the column that specifies the value of the current hierarchy level of items.	"NodeID"
SelectedItemColumnName	The name of the column that should be used for to find out which item is currently selected.	"DocumentID"
SelectedItemValue	The item whose column specified by the SelectedItemColumn property matches the value of this property will be designated as the currently selected item. Typically, you will need to insert a Macro expression in order to dynamically retrieve the appropriate value from the current context.	"{%currentpageinfo.documentid%}"
UseNearestItemForHeaderAndFooter	Indicates whether data from the nearest data item should be used for the header or footer item. This value is ignored if the OuterData property is set.	

1.6.2.4 Appearance and styling

The appearance of the UniView control is determined by the code of its templates and by certain other properties.

The following templates can be defined within the tags of the UniView control:

Template Name	Description	Sample Value
AlternatingItemTemplate	Code of the template used for alternating items.	<code><# HTMLHelper.Encode (Convert.ToString(Eval("NodeAliasPath"))) %></code>
FirstItemTemplate	Code of the template used for the first item on every level.	

FooterTemplate	Code of the template used for the footer.	<code></code> <code></code>
HeaderTemplate	Code of the template used for the header.	<code></code> <code></code>
ItemTemplate	Code of the template used for standard items.	<code><%# HTMLHelper. HTMLEncode(Convert. ToString(Eval("NodeAliasPath"))) %></code>
LastItemTemplate	Code of the template used for the last item on every level.	
SeparatorTemplate	Code of the template used for separating items.	<code></code> <code></code>
SingleItemTemplate	Code of the template used for displaying a single item.	

1.6.3 Paging controls

1.6.3.1 Overview

The controls in this section provide paging support to other controls that display data. This means they divide the displayed items into groups (pages) and provide an easy way to navigate between them.

The newest, most flexible and easiest to use is the UniPager control. It provides most of the functionality of the other two and more, so we recommend using it whenever possible.

Available controls:

- [DataPager](#)
- [TemplateDataPager](#)
- [UniPager](#)

1.6.3.2 Paging controls - common properties

All of the pager controls have the following properties in common:

Property Name	Description	Sample Value
CurrentPage	The current page number.	
MaxPages	Maximum number of pages that can be viewed.	
PageCount	The current number of pages (read only).	
PageSize	The number of displayed items per page.	

1.6.3.3 DataPager

1.6.3.3.1 Overview

The DataPager control can ensure paging for the following CMSControls:

- [CMSDataList](#)
- [CMSRepeater](#)
- [CMSSearchResults](#)
- [QueryDataList](#)
- [QueryRepeater](#)

This control doesn't need to be used separately, it is built into the above controls and can be enabled or disabled by using their **EnablePaging** property.

See also: [TemplateDataPager](#) - this is a paging control that can be used to customize the data paging format

	<p>Please note</p> <p>If possible, we recommend that you use the newer UniPager control instead.</p>
---	---

The following topics are available to help you familiarize yourself with the DataPager control:

- [Getting started](#) - contains a quick step-by-step tutorial that allows you to learn the basics of using the control
- [Configuration](#) - describes and explains the properties that can be set for the control
- [Appearance and styling](#) - describes and explains additional properties that can be used to customize the appearance of the control

1.6.3.3.2 Getting started

The following is a step-by-step tutorial that will show you how use the DataPager control with a CMSRepeater control that displays all pages (menu items) in the system:

1. Create a new **Web form** somewhere in your website installation directory.
2. Switch to its **Design** tab, drag and drop a **CMSRepeater** control from the toolbox onto the form and set its following properties:
 - **ClassNames:** cms.menuitem
 - **EnablePaging:** True

This tells the control which document types to read and enables the DataPager.

3. Switch to the **Source** tab and add the code marked by the **CMSRepeater templates** comments between the `<cms:CMSRepeater>` tags. The overall code of the CMSRepeater control should look like this:

```

<cms:CMSRepeater ID="CMSRepeater1" runat="server" ClassNames="cms.menuitem"
EnablePaging="true" >

    <%-- CMSRepeater templates
    ----- %>

    <ItemTemplate>
        <%# HTMLHelper.Encode( Convert.ToString(Eval("NodeAliasPath"))) %>
    >
    </ItemTemplate>
    <AlternatingItemTemplate>
        <font color="#999999"><%# HTMLHelper.Encode( Convert.ToString
(Eval("NodeAliasPath"))) %>
        </font>
    </AlternatingItemTemplate>
    <SeparatorTemplate>
        </li>
        <li>
    </SeparatorTemplate>

    <%-- CMSRepeater templates
    ----- %>

</cms:CMSRepeater>

```

This sets the templates used by the CMSRepeater to display the pages (menu items). The control dynamically replaces the <%# ... %> tags with values of the currently displayed record. This is then repeated for every record in the data source.

6. Save the changes to the web form. Now right-click it in the Solution explorer and select **View in Browser**. The resulting page should look like this:

```

/Home
/Services
/News
/Partners
/Company
/Forums
/Wiki
/Examples
/Blogs
/Events
Displaying results 1-10 (of 252)
|< < 1 - 2 - 3 - 4 - 5 - 6 - 7 - 8 - 9 - 10 > >|

```

1.6.3.3.3 Configuration

The properties of the DataPager control can be accessed through the **PagerControl** property of the paged controls, like for example:

[C#]

```

CMSRepeater1.PagerControl.BackText = "<-";

```

In addition to the properties from [Paging controls - common properties](#), the following properties can be set in your code behind files:

Property Name	Description	Sample Value
DataSource	Can be used to access the object of the pager's data source.	
BackNextDisplay	Back/Next display mode.	"Buttons" "Hyperlinks"
BackNextLocation	Back/Next location.	"Right" "Left" "Split" "None"
BackText	Back button/hyperlink text.	
FirstText	First button/hyperlink text.	
HideOnSinglePage	If true, the pager is hidden if only one page is displayed.	
IgnoreQueryString	Indicates whether querystring parameters should be ignored.	
InsertKeys	Adds keys to the querystring.	
InsertToUrl	Indicates whether inserting querystring keys is enabled.	
LabelText	Label text.	
LastText	Last text.	
NextText	Next button/hyperlink text.	
PagedData	Gets the data to be paged.	
PageNumbersDisplay	Page numbers display mode.	"Numbers" "Results"
PagerPosition	The position of the pager relative to the paged data.	"Bottom" "Top" "TopAndBottom"
PagingMode	Determines the type of the used paging parameter. It can either be passed through the URL (QueryString) or through postback (PostBack).	PostBack QueryString
QueryStringKey	Query parameter name for the page index.	"pagenumber"
RecordEnd	Index of the last record on the current page.	
RecordStart	Index of the first record on the current page.	
RemoveFromUrl	Indicates whether removing querystring keys is enabled.	

RemoveKeys	Removes keys from the querystring.	
ResultsFormat	Results text format.	"Displaying results {0}-{1} (of {2})"
ResultsLocation	Results location.	"Top" "Bottom" "None"
ShowFirstLast	Indicates whether first/last buttons should be displayed.	
ShowLabel	Indicates whether labels should be displayed.	
ShowPageNumbers	Indicates whether page numbers should be displayed.	
SliderSize	Slider size.	
TotalRecords	Total amount of data source records.	
UseSlider	Indicates whether the slider should be used.	

1.6.3.3.4 Appearance and styling

The appearance of the DataPager control can additionally be modified by its following properties and the CSS classes that they specify:

Property Name	Description	Sample Value
BackNextButtonStyle	Back/Next button style.	
BackNextLinkSeparator	Back/Next link separator.	
BackNextStyle	Back/Next style.	
ControlCssClass	CSS class of the pager control.	
LabelStyle	Label style.	
PageNumbersStyle	Page numbers style.	
PageNumbersSeparator	Page numbers separator.	","
PagerControlStyle	Pager control style.	
PagerHTMLAfter	HTML code to be rendered after the pager.	
PagerHTMLBefore	HTML code to be rendered before the pager.	
PagerNumberAreaClass	CSS class of the number area.	
ResultsStyle	Results style.	
SectionPadding	Section padding.	
SelectedClass	CSS class of the selected page.	

UnselectedClass	CSS class of unselected pages.	
-----------------	--------------------------------	--

1.6.3.4 TemplateDataPager

1.6.3.4.1 Overview

The TemplateDataPager control can be used to set a custom format for data paging. It can work with the same controls as the [DataPager](#). It automatically renders the list of numbers, but some code needs to be written to bind it to a control that ensures the displaying of content (e.g. CMSRepeater, CMSDataList or other).

	<p>Please note</p> <p>We recommend that you use the newer UniPager control, which can also be customized and is much easier to use, instead.</p>
---	---

The following topics are available to help you familiarize yourself with the TemplateDataPager control:

- [Getting started](#) - contains a step-by-step tutorial that allows you to learn how to use the control
- [Configuration](#) - describes and explains the properties that can be set for the control
- [Appearance and styling](#) - describes how the design of the control can be modified

1.6.3.4.2 Getting started

The following is a step-by-step tutorial that will show you how to add a custom pager to a CMSRepeater control that displays Smartphones (CMS.Smartphone documents) using the TemplateDataPager control:

1. Create a new **Web form** somewhere in your website installation directory.
2. Now add the following code to the page, inside the `<form>` element:

```
<ajaxToolkit:ToolkitScriptManager ID="manScript" runat="server" EnableViewState
="false" />
<table style="border: solid 1px #CCCCCC; margin-left: auto; margin-right: auto;">
  <tr>
    <td style="border-bottom: solid 1px #CCCCCC; padding: 10px; text-align:
center;">
      <cms:CMSRepeater ID="CMSRepeater1" runat="server" Path="/%" ClassNames
="CMS.Smartphone"
      TransformationName="Ecommerce.Transformations.Product_Default">
        </cms:CMSRepeater>
    </td>
  </tr>
  <tr>
    <td style="padding: 10px; background-color: #D9D9D9;">
      <cms:TemplateDataPager ID="TemplateDataPager1" runat="server">
        <NumberTemplate>
          <a href="?Page=<%# Eval("PageNumber") %>">
            <%# Eval("PageNumber") %>
          </a>
        </NumberTemplate>
      </cms:TemplateDataPager>
    </td>
  </tr>
</table>
```

```

</NumberTemplate>
<SelectedNumberTemplate>
  <b>
    <%# Eval("PageNumber") %>
  </b>
</SelectedNumberTemplate>
<SeparatorTemplate>
  -
</SeparatorTemplate>
<FirstItemTemplate>
  <a href="?Page=1">First</a>&nbsp;|&nbsp;&nbsp;
</FirstItemTemplate>
<LastItemTemplate>
  &nbsp;|&nbsp;&nbsp;<a href="?Page=<%# pageCount %>">Last</a>
</LastItemTemplate>
<PreviousItemTemplate>
  <a href="?Page=<%# previousPage %>">Previous</a> &nbsp;|&nbsp;&nbsp;
</PreviousItemTemplate>
<NextItemTemplate>
  &nbsp;|&nbsp;&nbsp;<a href="?Page=<%# nextPage %>">Next</a>
</NextItemTemplate>
</cms:TemplateDataPager>
</td>
</tr>
</table>

```

As you can see, the control uses a standard CMSRepeater control to display data. The pager format is specified using the templates defined between the tags of the `<cms: TemplateDataPager>` element.

The **ToolkitScriptManager** control included at the top is required by the transformation used to display smartphone documents. It is only there to ensure that the example is functional by itself and will usually be included on your website's master page.

3. Modify the code-behind so that it looks like this (the class name and type may be different):

[C#]

```

using CMS.GlobalHelper;

public partial class CMSControlsExamples_TemplateDataPager : System.Web.UI.Page
{
    public string pageCount = "1";
    public string previousPage = "1";
    public string nextPage = "";

    /// <summary>
    /// OnInit override
    /// </summary>
    /// <param name="e"></param>
    protected override void OnInit(EventArgs e)
    {
        // Disable repeater pager and databindbydefault
        CMSRepeater1.EnablePaging = false;
        CMSRepeater1.DataBindByDefault = false;
    }
}

```

```
        base.OnInit(e);
    }

    protected void Page_Load(object sender, EventArgs e)
    {
        // Get repeater datasource
        TemplateDataPager1.DataSource = CMSRepeater1.DataSource;

        // Set page size
        TemplateDataPager1.PageSize = 1;

        // Set current page from query string
        TemplateDataPager1.CurrentPage = ValidationHelper.GetInteger(Request.
QueryString["Page"], 1);

        // Get page number for last link
        pageCount = ((int)(TemplateDataPager1.PageCount - 1)).ToString();

        // Set default next page link
        nextPage = pageCount;

        // Set previous link
        if ((TemplateDataPager1.CurrentPage - 1) >= 1)
        {
            previousPage = ((int)(TemplateDataPager1.CurrentPage - 1)).ToString();
        }

        // Set next link
        if ((TemplateDataPager1.CurrentPage + 1) <= (TemplateDataPager1.PageCount
- 1))
        {
            nextPage = ((int)(TemplateDataPager1.CurrentPage + 1)).ToString();
        }

        // Set paged datasource to the repeater and databind it
        CMSRepeater1.DataSource = TemplateDataPager1.PagedData;
        if (!DataHelper.DataSourceIsEmpty(CMSRepeater1.DataSource))
        {
            CMSRepeater1.DataBind();
        }
    }
}
```

This code binds the TemplateDataPager to the CMSRepeater.

4. Save the changes to the web form. Now right-click it in the Solution explorer and select **View in Browser**. The resulting page should look like this:



1.6.3.4.3 Configuration

In addition to the properties from [Paging controls - common properties](#), the following properties of the TemplateDataPager control can be set or used in the API:

Property Name	Description	Sample Value
DataSource	Can be used to access the object of the pager's data source.	
NumberRepeater	Gets the repeater control used to display page numbers.	
PagedData	Gets the data to be paged.	
PagerPosition	The position of the pager relative to the paged data.	"Bottom" "Top" "TopAndBottom"
PagingMode	Determines the type of the used paging parameter. It can either be passed through the URL (QueryString) or through postback (PostBack).	"PostBack" "QueryString"
RecordEnd	Index of the last record on the current page.	
RecordStart	Index of the first record on the current page.	
TotalRecords	Total amount of data source records.	

1.6.3.4.4 Appearance and styling

The appearance of the TemplateDataPager control is determined by the code in its templates. The following are available:

Template Name	Description	Sample Value
FirstItemTemplate	Code of the template used for the link to the first page in the pager.	<code>First &nbsp; &nbsp;</code>

LastItemTemplate	Code of the template used for the link to the last page in the pager.	<code>&nbsp; &nbsp;<a href="?Page=<## pageCount %>">Last</code>
NextItemTemplate	Code of the template used for the link to the next page.	<code>&nbsp; &nbsp;<a href="?Page=<## nextPage %>">Next</code>
NumberTemplate	Code of the template used for page links in the pager. Use <code><## Eval("PageNumber") %></code> to get the current page number.	<code><a href="?Page=<## Eval("PageNumber") %>"><## Eval("PageNumber") %></code>
PreviousItemTemplate	Code of the template used for the link to the previous page.	<code><a href="?Page=<## previousPage %>">Previous &nbsp; &nbsp;&nbsp;&nbsp;</code>
SelectedNumberTemplate	Code of the template used for the number of the currently selected page.	<code><## Eval("PageNumber") %></code>
SeparatorTemplate	Code of the template used for the separator between page links in the pager.	-

1.6.3.5 UniPager

1.6.3.5.1 Overview

The UniPager is a universal paging control that can ensure paging for any control that implements the *IUniPageable* interface. This includes the following Basic and Generic controls:

- [BasicDataList](#)
- [BasicRepeater](#)
- [BasicUniView](#)
- [UniView](#)

As well as the following CMS controls:

- [CMSDataList](#)
- [CMSRepeater](#)
- [QueryDataList](#)
- [QueryRepeater](#)

In cases where the UniPager control is placed after the attached listing control, that control must bind its data later in the page life cycle than during the **Init** event, otherwise paging will not be applied. For the CMSDataList and CMSRepeater controls, this can easily be solved by setting their **DelayedLoading** property to *true*.

The QueryDataList and QueryRepeater controls do not have this property, however, you can ensure that paging is applied correctly by setting their **DataBindByDefault** property to *false* and manually calling their **Datbind()** method during the **Load** event:

[C#]

```
protected void Page_Load(object sender, EventArgs e)
{
    QueryRepeater1.DataBind();
}
```

The UniPager is also built into the [CMSUniView](#) and [QueryUniView](#) controls and can be enabled by their **EnablePaging** property.

The portal engine equivalent of the UniPager control is the **Listings and viewers -> Universal Pager** web part.

The following topics are available to help you familiarize yourself with the UniPager control:

- [Getting started](#) - contains a quick step-by-step tutorial that allows you to learn the basics of using the control
- [Configuration](#) - describes and explains the properties that can be set for the control
- [Structure](#) - contains a more advanced example of how the control can be configured and demonstrates what individual template properties affect
- [Appearance and styling](#) - describes how the design of the control can be modified
- [Implementing the IUniPageable interface](#) - contains a tutorial describing how a custom control that is pageable by the UniPager control can be created

1.6.3.5.2 Getting started

The following is a step-by-step tutorial that will show you how to add a simple pager to a CMSRepeater control that displays all pages (menu items) in the system:

1. Create a new **Web form** somewhere in your website installation directory.
2. Switch to its **Design** tab, drag and drop a **CMSRepeater** control from the toolbox onto the form and set its following properties:

- **ClassNames:** cms.menuitem
- **DelayedLoading:** True

3. Switch to the **Source** tab and add the code marked by the **CMSRepeater templates** comments between the `<cms:CMSRepeater>` tags. The overall code of the CMSRepeater control should look like this:

```
<cms:CMSRepeater ID="CMSRepeater1" runat="server" ClassNames="cms.menuitem"
DelayedLoading="true">

    <%-- CMSRepeater templates
    ----- %>

    <ItemTemplate>
        <%# HTMLHelper.Encode( Convert.ToString( Eval( "NodeAliasPath" ) ) ) %>
    >
</ItemTemplate>
```

```

<AlternatingItemTemplate>
  <font color="#999999"><%# HTMLHelper.Encode( Convert.ToString
(Eval("NodeAliasPath")) ) %>
  </font>
</AlternatingItemTemplate>
<SeparatorTemplate>
  </li>
  <li>
</SeparatorTemplate>

<%-- CMSRepeater templates
----- %>

</cms:CMSRepeater>

```

This sets the templates used by the CMSRepeater to display the pages (menu items). The control dynamically replaces the <%# ... %> tags with values of the currently displayed record. This is then repeated for every record in the data source.

4. Switch back to the **Design** tab, drag and drop a **UniPager** control from the toolbox onto the form one line below the CMSRepeater and set its **PageControl** property to *CMSRepeater1*.

5. Switch to the **Source** tab and add the code marked by the **UniPager templates** comments between the <cms:UniPager> tags. The overall code of the UniPager control should look like this:

```

<cms:UniPager ID="UniPager1" runat="server" PageControl="CMSRepeater1">

  <%-- UniPager templates
  ----- %>

  <PageNumbersTemplate>
    <a href="<%# Eval("PageURL") %>"><%# Eval("Page") %></a>
  </PageNumbersTemplate>

  <%-- UniPager templates
  ----- %>

</cms:UniPager>

```

This sets the minimum required template that enables the UniPager with a very simple design. Please see the [Structure](#) topic to learn about the more advanced templates.

6. Save the changes to the web form. Now right-click it in the Solution explorer and select **View in Browser**. The resulting page should look like this:

[/Home](#)
[/Services](#)
[/News](#)
[/Partners](#)
[/Company](#)
[/Forums](#)
[/Wiki](#)
[/Examples](#)
[/Blogs](#)
[/Events](#)
[1](#) [2](#) [3](#) [4](#) [5](#) [6](#) [7](#) [8](#) [9](#) [10](#)

1.6.3.5.3 Configuration

In addition to the properties from [Paging controls - common properties](#), the following properties of the UniPager control can be set or used in the API:

Property Name	Description	Sample Value
DataSourceItemsCount	The amount of items in the data source.	
DirectPageControlID	The ID of the control used for direct page changing.	
DisplayFirstLastAutomatically	If enabled, the first and last buttons of the pager will be displayed only when there is no other way of accessing the first or last page through the pager.	
DisplayPreviousNextAutomatically	If enabled, the previous and next buttons of the pager will be displayed only when there is no other way of accessing the previous or next page through the pager.	
EnvelopeTag	The current envelope tag.	
GroupSize	The amount of page links displayed in one group.	
HidePagerForSinglePage	If true, the pager is hidden if only one page is displayed.	
HTMLEnvelopeRenderingMode	The HTML envelope rendering mode for the current page.	"Always" "Never" "OnlyForUpdatePanel"
PageControl	The ID of the control to be paged.	
PagedControl	The object of the control to be paged.	
PagerMode	Determines the type of the used paging parameter. It can either be passed through the URL (QueryString) or through postback (PostBack).	"PostBack" "QueryString"
QueryStringKey	Name of the query string parameter that contains the current page number. This is	"pagenumber"

	useful if there are multiple UniPager controls on the same page.	
RelatedData	Custom data connected to the object.	

1.6.3.5.4 Structure

This topic shows an example of how the UniPager control looks when all of its template properties are defined. If you wish to create this example for yourself, please follow the tutorial in the [Getting started](#) topic up to and including step 4, then continue with the following steps:

1. Add the code marked by the **UniPager templates** comments between the `<cms:UniPager>` tags. The overall code of the UniPager control should look like this:

```
<cms:UniPager ID="UniPager1" runat="server" PageControl="CMSRepeater1">

    <!-- UniPager templates
    ----- -->

    <PageNumbersTemplate>
        <a href="<%# Eval("PageURL") %>"><%# Eval("Page") %></a>
    </PageNumbersTemplate>
    <CurrentPageTemplate>
        <strong><%# Eval("Page") %></strong>
    </CurrentPageTemplate>
    <PageNumbersSeparatorTemplate>
        &nbsp;-&nbsp;
    </PageNumbersSeparatorTemplate>
    <FirstPageTemplate>
        <a href="<%# Eval("FirstURL") %>">|&lt;</a>
    </FirstPageTemplate>
    <LastPageTemplate>
        <a href="<%# Eval("LastURL") %>">&gt;|</a>
    </LastPageTemplate>
    <PreviousPageTemplate>
        <a href="<%# Eval("PreviousURL") %>">&lt;</a>
    </PreviousPageTemplate>
    <NextPageTemplate>
        <a href="<%# Eval("NextURL") %>">&gt;</a>
    </NextPageTemplate>
    <PreviousGroupTemplate>
        <a href="<%# Eval("PreviousGroupURL") %>">...</a>
    </PreviousGroupTemplate>
    <NextGroupTemplate>
        <a href="<%# Eval("NextGroupURL") %>">...</a>
    </NextGroupTemplate>
    <DirectPageTemplate>
        Page
        <asp:TextBox ID="DirectPageControl" runat="server" Style="width:
25px;" />
        of
        <%# Eval("Pages") %>
    </DirectPageTemplate>
    <LayoutTemplate>
        <asp:Placeholder runat="server" ID="plcFirstPage"></asp:Placeholder>
        <asp:Placeholder runat="server" ID="plcPreviousPage"></asp:

```

```

Placeholder>&nbsp;
    <asp:Placeholder runat="server" ID="plcPreviousGroup"></asp:
Placeholder>
    <asp:Placeholder runat="server" ID="plcPageNumbers"></asp:
Placeholder>
    <asp:Placeholder runat="server" ID="plcNextGroup"></asp:Placeholder>
&nbsp;
    <asp:Placeholder runat="server" ID="plcNextPage"></asp:Placeholder>
    <asp:Placeholder runat="server" ID="plcLastPage"></asp:Placeholder>
    <br />
    <asp:Placeholder runat="server" ID="plcDirectPage"></asp:Placeholder
>
</LayoutTemplate>

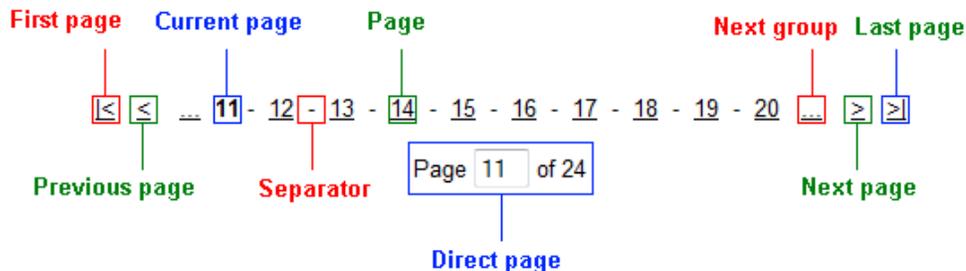
<!-- UniPager templates
----- %>

</cms:UniPager>

```

This sets all the templates of the UniPager control.

2. Save the changes to the web form. Now right-click it in the Solution explorer and select **View in Browser**. The resulting page should contain a pager like in the following diagram (without the descriptions), which shows the structure of the UniPager control. Individual areas are described below.



- **Layout Template** - determines the overall design of the displayed pager. To place the locations of individual templates into the layout, use Placeholder controls with their *ID* properties set exactly like in the example, e.g. *plcFirstPage* for the *FirstPageTemplate* etc. The content of individual pages is dependant on the control that the UniPager is ensuring paging for. The maximum amount of displayed items can be set through the UniPager control's *PageSize* property.
- **Page** - this area is defined by the code of the *PageNumbersTemplate*. It is usually used to display the general page links of the pager. The amount of displayed page links can be set by the UniPager control's *GroupSize* property.
- **Current page** - this area is defined by the code of the *CurrentPageTemplate*. It is usually used to display the number of the currently selected page.
- **Direct page** - this area is defined by the code of the *DirectPageTemplate*. It is usually used to display a control that allows direct switching between pages. The *ID* property of the used control must be set to *DirectPageControl* as in the example.
- **Separator** - this area is defined by the code of the *PageNumbersSeparatorTemplate*. It is placed between every page number in the pager.
- **First/Last page** - these areas are defined by the code of the *FirstPageTemplate* and *LastPageTemplate*. They are usually used to display links to the first and last page of the pager. If the UniPager control's *DisplayFirstLastAutomatically* property is set to *true*, this area is only shown

when there is no other way of accessing the first or last page through the pager.

- **Next/Previous page** - these areas are defined by the code of the *NextPageTemplate* and *PreviousPageTemplate*. They are usually used to display links to the next and previous page of the pager. If the UniPager control's *DisplayPreviousNextAutomatically* property is set to *true*, this area is only shown when there is no other way of accessing the previous or next page through the pager.
- **Next/Previous group** - these areas are defined by the code of the *NextGroupTemplate* and *PreviousGroupTemplate*. They are usually used to display links to the next and previous group of pages.

1.6.3.5.5 Appearance and styling

The appearance of the UniPager control is determined by the code of its templates and by certain other properties.

The following templates can be defined within the tags of the UniPager control. Please refer to the [Structure](#) topic to see what individual templates represent.

Template Name	Description	Sample Value
CurrentPageTemplate	Code of the template used for the current page in the pager. Use <code><%# Eval("Page") %></code> to get the current page number, <code><%# Eval("PageURL") %></code> to get page the URL or <code><%# Eval("PageLink") %></code> to get the page link.	<pre><%# Eval("Page") %></pre>
DirectPageTemplate	Code of the template used for direct page changing. Use a TextBox or DropDownList control with ID <i>DirectPageControl</i> to register the page change event.	<pre>Page <asp:TextBox ID ="DirectPageControl" runat="server" Style=" width: 25px;" /> of <%# Eval("Pages") %></pre>
FirstPageTemplate	Code of the template used for the link to the first page in the pager. Use <code><%# Eval("FirstURL") %></code> to get the link to the first page.	<pre><a href="<%# Eval ("FirstURL") %>"> &lt;</pre>
LastPageTemplate	Code of the template used for the link to the last page in the pager. Use <code><%# Eval("LastURL") %></code> to get the link to the last page.	<pre><a href="<%# Eval ("LastURL") %>">&gt; </pre>
LayoutTemplate	Code of the template used for the overall pager layout.	
NextGroupTemplate	Code of the template used for the link to the next group of pages. Use <code><%# Eval("NextGroupURL") %></code> to get the link to the next group.	<pre><a href="<%# Eval ("NextGroupURL") %>">... </pre>
NextPageTemplate	Code of the template used for the link to the next page. Use <code><%# Eval("NextURL") %></code> to get the link to the next page.	<pre><a href="<%# Eval ("NextURL") %>">&gt;</pre>

PageNumbersSeparatorTemplate	Code of the template used for the separator between page links in the pager.	<code>&nbsp;</code>
PageNumbersTemplate	Code of the template used for page links in the pager. Use <code><%=# Eval("Page") %></code> to get the current page number, <code><%=# Eval("PageURL") %></code> to get the URL of the current page or <code><%=# Eval("PageLink") %></code> to get the page link.	<code><a href="<%=# Eval("PageURL") %>"><%=# Eval("Page") %></code>
PreviousGroupTemplate	Code of the template used for the link to the previous group of pages. Use <code><%=# Eval("PreviousGroupURL") %></code> to get the link to the next group.	<code><a href="<%=# Eval("PreviousGroupURL") %>">...</code>
PreviousPageTemplate	Code of the template used for the link to the previous page. Use <code><%=# Eval("PreviousURL") %></code> to get the link to the next page.	<code><a href="<%=# Eval("PreviousURL") %>">&lt;</code>

1.6.3.5.6 Implementing the IUniPageable interface

The following is a step-by-step tutorial that will show you how to create a custom control that displays users, and have it implement the `IUniPageable` interface to allow it to be paged by the `UniPager` control:

1. Create a new **Web User Control** called `UniPageable_Repeater.ascx` inside a folder called `IUniPageableExample` in your website installation directory.

2. Now add the following code to the user control:

```
<asp:Repeater ID="Repeater1" runat="server">

    <ItemTemplate>

        <div>
            <%=# Eval("UserName") %>
        </div>

    </ItemTemplate>

</asp:Repeater>
```

This adds a standard .NET Repeater control that will be used to display user names.

3. Switch to the code behind of the control and add the following code into it. Please keep in mind that the name of the class will be different according to the location of your web user control.

[C#]

```
using CMS.SiteProvider;
using CMS.Controls;
```

```
public partial class IUniPageableExample_UniPageable_Repeater : System.Web.UI.
UserControl, IUniPageable
{
    // Private variable containing the value of the PagerForceNumberOfResults
    property
    private int mPagerForceNumberOfResults = -1;

    // Private variable used to contain the data source of the control
    private object dataSource = null;

    protected void Page_Load(object sender, EventArgs e)
    {
        // Loads all users from the database into the data source
        dataSource = UserInfoProvider.GetAllUsers();

        // Call page binding event
        if (OnPageBinding != null)
        {
            OnPageBinding(this, null);
        }

        // Assigns the data source to the encapsulated Repeater control
        Repeater1.DataSource = dataSource;
        Repeater1.DataBind();
    }

    /// <summary>
    /// Occurs when the control binds page data
    /// </summary>
    public event EventHandler<EventArgs> OnPageBinding;

    /// <summary>
    /// Occurs when the pager changes the page and the current PagerMode is set to
    postback
    /// </summary>
    public event EventHandler<EventArgs> OnPageChanged;

    /// <summary>
    /// Exposes the data object for the pager
    /// </summary>
    public object PagerDataItem
    {
        get
        {
            return dataSource;
        }
        set
        {
            dataSource = value;
            Repeater1.DataSource = value;
        }
    }

    /// <summary>
    /// If set, the DataSet containing paged items is not modified by the pager,
    /// but the pager itself behaves as if the amount of paged items were
```

```
identical to this value.
    /// By default this property is disabled (set to -1)
    /// </summary>
    public int PagerForceNumberOfResults
    {
        get
        {
            return mPagerForceNumberOfResults;
        }
        set
        {
            mPagerForceNumberOfResults = value;
        }
    }

    /// <summary>
    /// Evokes databinding for the control
    /// </summary>
    public void ReBind()
    {
        if (OnPageChanged != null)
        {
            OnPageChanged(this, null);
        }

        Repeater1.DataBind();
    }
}
```

This code causes the control to implement the IUniPageable interface and adds the implementation for all its required members.

4. Save the changes to both files. Now the newly created control is pageable by the UniPager control. Create a new **Web form** somewhere in your website installation directory, where this functionality will be demonstrated.

5. Add the following directive to the beginning of the code of the new web form to register the custom UniPageable_Repeater control:

```
<%@ Register src="~/IUniPageableExample/UniPageable_Repeater.ascx" tagname
="UniPageableRepeater"
tagprefix="aspl" %>
```

6. Now add the following code into the content area of the page (by default between the <div> tags inside the <form> element):

```
<aspl:UniPageableRepeater ID="UPRepeater1" runat="server" />

<cms:UniPager ID="UniPager1" runat="server" PageControl="UPRepeater1" PageSize
="5">

<PageNumbersTemplate>
```

```
<a href="<%# Eval("PageURL") %>"><%# Eval("Page") %></a>
</PageNumbersTemplate>

</cms:UniPager>
```

This adds the custom control you created in the previous steps and a UniPager that is assigned to page it.

7. Save the changes to the web form. Now right-click it in the Solution explorer and select **View in Browser**. The resulting page should display a list of user names with a simple pager like in the following image:

```
administrator
public
silver
gold
Andy
1 2 3 4
```

1.7 Basic Controls

1.7.1 Overview

Basic Controls are a set of controls that provide the same functionality as many standard ASP.NET controls, but extend them with some additional configuration options. Their main purpose is to provide a way to display external data that doesn't use Kentico CMS architecture.

The following control categories are available:

- [Navigation](#)
- [Listings and viewers](#)

1.7.2 Navigation

1.7.2.1 Overview

This section contains controls that provide basic navigation functionality.

Available controls:

- [BasicTabControl](#)

1.7.2.2 BasicTabControl

1.7.2.2.1 Overview

The BasicTabControl control displays a tab menu according to data provided by a two dimensional array. BasicTabControl doesn't use Kentico CMS database or API and can be used to navigate to pages outside of Kentico CMS websites.



	<p>Please note</p> <p>If you want to display a tab menu based on data from Kentico CMS, please use the CMSTabControl control.</p>
---	--

The following topics are available to help you familiarize yourself with the BasicTabControl control:

- [Getting started](#) - contains a step-by-step tutorial that allows you to learn the basics of using the control
- [Configuration](#) - describes and explains the properties that can be set for the control
- [Appearance and styling](#) - lists which CSS classes can be used with the control and how its appearance can be modified

1.7.2.2.2 Getting started

The following is a step-by-step tutorial that will show you how to display a simple tab menu using the BasicTabControl control:

1. Create a new **Web form** somewhere in your website installation directory.
2. Switch to its **Design** tab, drag and drop a **BasicTabControl** control from the toolbox onto the form.
3. Switch to the **Source** tab. The code of the BasicTabControl control should look like this:

```
<cms:BasicTabControl ID="BasicTabControl1" runat="server" />
```

Now add the following code between the tags of the `<head>` element:

```
<style type="text/css">

/* Tab menu class definitions */
.TabControlTable { FONT-SIZE: 14px; FONT-FAMILY: Arial,Verdana }
.TabControlRow { }
.TabControl { BORDER-RIGHT: black 1px solid; BORDER-TOP: black 1px solid; FONT-WEIGHT: bold; BACKGROUND: #e7e7ff; BORDER-LEFT: black 1px solid; CURSOR: hand; COLOR: black }
.TabControlSelected { BORDER-RIGHT: black 1px solid; BORDER-TOP: black 1px solid; FONT-WEIGHT: bold; BACKGROUND: #4a3c8c; BORDER-LEFT: black 1px solid; CURSOR: default; COLOR: white }
.TabControlLinkSelected { COLOR: white; TEXT-DECORATION: none }
.TabControlLink { COLOR: black; TEXT-DECORATION: none }
.TabControlLeft { WIDTH: 1px }
.TabControlRight { WIDTH: 0px }
.TabControlSelectedLeft { WIDTH: 1px }
.TabControlSelectedRight { WIDTH: 0px }

</style>
```

This sets the CSS styles that will modify the appearance of the tab menu. The BasicTabControl control renders tabs even without any CSS classes specified, but they are extremely basic and not very user

friendly. You can find out what individual CSS classes affect in the [Appearance and styling](#) topic.

The classes are defined in the <head> element only for this quick example, if you wish to use the control on a Kentico CMS website, it is recommended to define these classes in the used stylesheet in the administration interface at **Site Manager -> Development -> CSS stylesheets**.

4. Add the following code just after the <cms:BasicTabControl> element. It will display a stripe under the tabs.

```
<hr style="width:100%; height:2px; margin-top:0px;" />
```

5. Switch to the code behind of the page and add the following code to the **Page_Load** method:

[C#]

```
string[,] tabs = new string[3, 7];

tabs[0, 0] = "&nbsp;Home&nbsp;";
tabs[0, 1] = "alert('It is very simple!');";
tabs[0, 2] = "http://www.kentico.com";
tabs[1, 0] = "&nbsp;Features&nbsp;";
tabs[1, 2] = "http://www.kentico.com/free-cms-asp-net.aspx";
tabs[2, 0] = "&nbsp;Download&nbsp;";
tabs[2, 2] = "http://www.kentico.com/download/trial-version.aspx";
tabs[2, 3] = "Some tooltip";

BasicTabControll1.Tabs = tabs;
BasicTabControll1.SelectedTab = 0;
BasicTabControll1.UrlTarget = "_blank";
BasicTabControll1.UseClientScript = true;
```

[VB.NET]

```
Dim tabs(2, 6) As String

tabs(0, 0) = "&nbsp;Home&nbsp;"
tabs(0, 1) = "alert('It\'s very simple!');"
tabs(0, 2) = "http://www.kentico.com"
tabs(1, 0) = "&nbsp;Features&nbsp;"
tabs(1, 2) = "http://www.kentico.com/free-cms-asp-net.aspx"
tabs(2, 0) = "&nbsp;Download&nbsp;"
tabs(2, 2) = "http://www.kentico.com/download/trial-version.aspx"
tabs(2, 3) = "Some tooltip"

BasicTabControll1.Tabs = tabs
BasicTabControll1.SelectedTab = 0
BasicTabControll1.UrlTarget = "_blank"
BasicTabControll1.UseClientScript = True
```

This creates an array of tab items and assigns it to the BasicTabControl control. It also selects the first tab, sets the target frame to "_blank" and enables client script.

6. Save the changes to the web form. Now right-click it in the Solution explorer and select **View in Browser**. The resulting page should display a tab menu like this:

[Home](#) [Features](#) [Download](#)

1.7.2.2.3 Configuration

The following properties of the BasicTabControl control can be set or used in the API:

Property Name	Description	Sample Value
HideControlForZeroRows	Hides the control when no data is loaded. Default value is False.	
RenderedHTML	Allows you to get or set the HTML code rendered by the control. You need to set this property before the Render event - e.g. in the OnLoad event.	
RenderLinkTitle	Specifies if the TITLE tag of links should be rendered.	
SelectedTab	Index of the selected tab.	
SelectFirstItemByDefault	Indicates whether the first tab should be selected by default.	
TabControlIDPrefix	Prefix that will be used for all IDs in the HTML code rendered by the BasicTabControl. It's useful if you need to place multiple tab controls on the same page.	"FirstTab"
TabControlLayout	Specifies the layout of the tab control.	"Horizontal" "Vertical"
Tabs	A 2 dimensional string array where each row represents a tab and the columns represent the following: tabs[0, 0] = title tabs[0, 1] = OnClick JavaScript tabs[0, 2] = URL tabs[0, 3] = tooltip tabs[0, 4] = left image URL tabs[0, 5] = center image URL tabs[0, 6] = right image URL Please note: 1. The image URLs in columns 4, 5 and 6 are optional.	tabs[0, 0] = " Home "; tabs[0, 1] = "alert('It is very simple!');"; tabs[0, 2] = "http://www.kentico.com"; tabs[0, 3] = "Some tooltip"; tabs[0, 4] = "leftimage.gif"; tabs[0, 5] = "centerimage.gif"; tabs[0, 6] = "rightimage.gif";

	2. If you specify the center image URL, the image is displayed instead of the title.	
UrlTarget	If a URL is set for tab items, this property specifies the target frame for all URLs.	"_blank"
UseClientScript	Indicates if client script should be generated for each tab.	
UsePostBack	Indicates if postback should be fired when a tab is clicked.	
ZeroRowsText	Text to be shown when the control is hidden by the HideControlForZeroRows property.	"No records found."

1.7.2.2.4 Appearance and styling

The appearance of the BasicTabControl control is determined by the CSS classes it uses and by some of its properties.

You can use the following CSS classes to modify the design of the control:

Class Name	Description
TabControlTable	Table that contains the tabs (<TABLE> tag).
TabControlRow	Table row (<TR> tag).
TabControl	Tab item (<TD> tag).
TabControlSelected	Selected tab item (<TD> tag).
TabControlLink	Tab item link - if a URL is specified (<A> tag).
TabControlLinkSelected	Selected tab item link - if a URL is specified (<A> tag).
TabControlLeft	Left side of the tab item (<TD> tag).
TabControlRight	Right side of the tab item (<TD> tag).
TabControlSelectedLeft	Left side of the selected tab item (<TD> tag).
TabControlSelectedRight	Right side of the selected tab item (<TD> tag).

1.7.3 Listings and viewers

1.7.3.1 Overview

The controls in this section provide various ways to display data from a specified data source. Options include different types of lists, tables and calendars. Most of them are derived from standard ASP.NET list controls.

These controls can work with any bindable data source, not just Kentico CMS documents. They are also the best option for plain displaying of data from Kentico CMS data sources, because their performance

is better than that of the more complex CMS controls.

Available controls:

- [BasicCalendar](#)
- [BasicDataGrid](#)
- [BasicDataList](#)
- [BasicRepeater](#)
- [BasicUniView](#)

1.7.3.2 BasicCalendar

1.7.3.2.1 Overview

The BasicCalendar control allows you to display a calendar with events, news and other date-based documents specified by a data source. It is derived from the intrinsic ASP.NET Calendar control, which means it provides advanced formatting capabilities and it allows you to display additional information for appropriate days.

The BasicCalendar can be used with any bindable data source - it doesn't use Kentico CMS database or API.

	<p>Please note</p> <p>The CMSCalendar control provides a more convenient way to display data from the Kentico CMS Database in a calendar.</p>
--	--

The following topics are available to help you familiarize yourself with the BasicCalendar control:

- [Getting started](#) - contains a step-by-step tutorial that allows you to learn the basics of using the control
- [Configuration](#) - describes and explains the properties that can be set for the control
- [Appearance and styling](#) - describes how the design of the control can be modified

1.7.3.2.2 Getting started

The following is a step-by-step tutorial that will show you how to display a calendar that contains links to news items (CMS.News documents) on days when news items were released using the BasicCalendar control:

1. Create a new **Web form** somewhere in your website installation directory.
2. Switch to its **Design** tab, drag and drop a **BasicCalendar** control from the toolbox onto the form.
3. Switch to the **Source** tab and add the code marked by the **BasicCalendar templates** comments between the `<cms:BasicCalendar>` tags. The overall code of the BasicCalendar control should look like this:

```

<cms:BasicCalendar ID="BasicCalendar1" runat="server">
    <%-- BasicCalendar templates
    ----- %>

    <ItemTemplate>
        <br/>
        <a href='<%# ResolveUrl(CMS.CMSHelper.CMSContext.GetUrl(Convert.
        ToString(Eval("NodeAliasPath")), Convert.ToString(Eval
        ("DocumentUrlPath")))) %>'>
            <%# Eval("NewsTitle") %>
        </a>
    </ItemTemplate>

    <NoEventsTemplate>
        <br>
        No Event
    </NoEventsTemplate>

    <%-- BasicCalendar templates
    ----- %>

</cms:BasicCalendar>

```

This sets the template used to specify the layout of days with and without news releases. For days with news releases, the control dynamically replaces the <%# ... %> tags with values of the current news document from the data source.

4. Switch to the code behind of the page and add the following reference to the beginning of the code:

[C#]

```
using CMS.CMSHelper;
```

[VB.NET]

```
Imports CMS.CMSHelper
```

5. Now add the following code to the **Page_Load** method:

[C#]

```

BasicCalendar1.DataSource = TreeHelper.SelectNodes("/%", false, "cms.news", null,
"NewsReleaseDate", -1, true);

BasicCalendar1.DayField = "NewsReleaseDate";
BasicCalendar1.DataBind();

```

[VB.NET]

```

BasicCalendar1.DataSource = TreeHelper.SelectNodes("/%", False, "cms.news",
Nothing, "NewsReleaseDate", -1, True)

BasicCalendar1.DayField = "NewsReleaseDate"
BasicCalendar1.DataBind()

```

This retrieves all news items from the Kentico CMS database as a DataSet and assigns it as the data source of the BasicCalendar control. It also fills the **DayField** property, which tells the control which field it should get date/time values from.

6. Save the changes to the web form. Now right-click it in the Solution explorer and select **View in Browser**. The resulting page should display a calendar like this:

June 2011						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
29	30	31	1	2	3	4
No Event	No Event	No Event	No Event	No Event	No Event	No Event
5	6	7	8	9	10	11
New Consulting Services	No Event	No Event	No Event	Apple iPad 2 In Stock	No Event	No Event
12	13	14	15	16	17	18
No Event	No Event	No Event	No Event	No Event	Company Growth Exceeds Expectations	No Event
19	20	21	22	23	24	25
No Event	No Event	No Event	No Event	No Event	No Event	No Event
26	27	28	29	30	1	2
No Event	No Event	No Event	Community Website Section	No Event	No Event	No Event
3	4	5	6	7	8	9
No Event	No Event	No Event	No Event	No Event	No Event	No Event

1.7.3.2.3 Configuration

The following properties of the BasicCalendar control can be set or used in the API:

Property Name	Description	Sample Value
CustomTimeZone	A custom time zone to be used represented by a TimeZoneInfo object.	
DataMember	Name of the table when a DataSet is used as a DataSource.	
DataSource	Data source with calendar events - either a DataSet or DataTable object.	
DayField	Name of the field in the data source that contains the date/time value.	"NewsReleaseDate"
DayWithEventsStyle	Style of days that have an event.	
DisplayOnlySingleDayItem	Indicates whether only one item is displayed in a single day item.	
HideDefaultDayNumber	Indicates whether the day number should be displayed or if the day cell should be fully filled by the used template.	

RelatedData	Custom data connected to the object.	
TimeZone	Specifies the time zone type.	"Custom" "Inherit" "Server" "User" "WebSite"

As this control is inherited from the ASP.NET Calendar control, it also has all of its standard properties.

1.7.3.2.4 Appearance and styling

You can modify the appearance of the BasicCalendar control by setting the standard properties inherited from the ASP.NET Calendar control. You can find more details on particular properties in the .NET Framework documentation for the [Calendar](#) class.

A common way to set the appearance of this control is to assign a skin through the **SkinID** property. Skins can be defined in .skin files under individual themes in the **App_Themes** folder. More information can be found in the .NET [Skins and Themes](#) documentation.

The design of day cells is determined by the code of the templates defined within the tags of the BasicCalendar control. The following are available:

Template Name	Description	Sample Value
ItemTemplate	Template for displaying a day with an event.	
NoEventsTemplate	Template for displaying a day without any event.	

1.7.3.3 BasicDataGrid

1.7.3.3.1 Overview

The BasicDataGrid control allows you to display items from a data source in a customizable table. It is derived from the intrinsic ASP.NET DataGrid control, so it automatically provides paging and sorting support. You can use the standard DataGrid designer to set up BasicDataGrid style and behavior.

The BasicDataGrid can be used with any bindable data source - it doesn't use Kentico CMS database or API.

<p>Please note</p> <p>The CMSDataGrid control provides a more convenient way to display data from the Kentico CMS Database in a table.</p>

The following topics are available to help you familiarize yourself with the BasicDataGrid control:

- [Getting started](#) - contains a step-by-step tutorial that allows you to learn the basics of using the

control

- [Configuration](#) - describes and explains the properties that can be set for the control
- [Appearance and styling](#) - describes how the design of the control can be modified

1.7.3.3.2 Getting started

The following is a step-by-step tutorial that will show you how to display a table that contains laptops (CMS.Laptop documents) from the sample Corporate Site using the BasicDataGrid control:

1. Create a new **Web form** somewhere in your website installation directory.
2. Switch to its **Design** tab, drag and drop a **BasicDataGrid** control from the toolbox onto the form.
3. Right-click the BasicDataGrid on the form, select **AutoFormat...** and choose a scheme.
4. Right-click the BasicDataGrid on the form, select **Show Smart Tag** and then **Property Builder...**; the **BasicDataGrid1 Properties** dialog will be displayed.

On the **General** tab, check the **Allow sorting** box.

Now switch to the **Columns** tab, where you can specify the columns that will be displayed, and uncheck the **Create columns automatically at run time** box.

Add a new **Bound Column** from the **Available columns** list to the **Selected columns** list. Enter the following values into the appropriate fields:

- **Header text:** Name
- **Data Field:** LaptopName
- **Sort expression:** LaptopName

Add another **Bound column**. Enter the following values in the appropriate fields:

- **Header text:** Price
- **Data Field:** SKUPrice
- **Sort expression:** SKUPrice

Click **OK**.

5. Switch to the code behind of the page and add the following references to the beginning of the code:

[C#]

```
using System.Data;  
  
using CMS.CMSHelper;
```

[VB.NET]

```
Imports System.Data
```

```
Imports CMS.CMSHelper
```

6. Now add the following code to the **Page_Load** method:

[C#]

```
DataSet ds = TreeHelper.SelectNodes("/%", false, "CMS.Laptop", "", "LaptopName", -1, true);

BasicDataGrid1.DataSource = ds;
BasicDataGrid1.DataBind();
```

[VB.NET]

```
Dim ds As DataSet = TreeHelper.SelectNodes("/%", False, "CMS.Laptop", "", "LaptopName", -1, True)

BasicDataGrid1.DataSource = ds
BasicDataGrid1.DataBind()
```

This retrieves all CMS.Laptop documents from the Kentico CMS database as a DataSet and assigns it as the data source of the BasicDataGrid control.

7. Save the changes to the web form. Now right-click it in the Solution explorer and select **View in Browser**. The resulting page should display a table similar to the following (depending on the chosen scheme):

Name ▲	Price
Apple iPad 2	510.99
Apple MacBook Pro MC723LL/A	2199
Dell XPS 15z	1596.99
HP EliteBook 8440p WJ681AW	1899

1.7.3.3.3 Configuration

The following properties of the BasicDataGrid control can be set or used in the API:

Property Name	Description	Sample Value
DataBindByDefault	Indicates whether data binding should be performed by default during the init event.	
HideControlForZeroRows	Indicates whether the control should be hidden when no data is loaded. The default value is False .	

ProcessSorting	Indicates whether sorting should be processed in the DataView instead of sorting on the SQL level.	
RelatedData	Custom data connected to the object.	
SetFirstPageAfterSortChange	Indicates if the page index should be set to the first page when sorting is changed.	
SortAscending	Indicates whether sorting should be ascending. Default value is True.	
SortField	Gets or sets the sort field. It can be used for setting the default sort field.	"NewsReleaseDate"
ZeroRowsText	Text to be shown when the control is hidden by the HideControlForZeroRows property.	"No records found"

As this control is inherited from the ASP.NET DataGrid control, it also has all of its standard properties.

1.7.3.3.4 Appearance and styling

You can modify the appearance of the BasicDataGrid control by setting the standard properties inherited from the ASP.NET DataGrid control. You can find more details on particular properties in the .NET Framework documentation for the [DataGrid](#) class.

A common way to set the appearance of this control is to assign a skin through the **SkinID** property. Skins can be defined in .skin files under individual themes in the **App_Themes** folder. More information can be found in the .NET [Skins and Themes](#) documentation.

1.7.3.4 BasicDataList

1.7.3.4.1 Overview

The BasicDataList control allows you to display items from a data source in a list based on specified templates. It is derived from the intrinsic ASP.NET DataList control, so standard ASP.NET configuration techniques can be used to set up BasicDataList style and behaviour.

The BasicDataList can be used with any bindable data source - it doesn't use Kentico CMS database or API.

Unlike the [BasicRepeater](#) control, the BasicDataList allows you to display data in several columns.

The portal engine equivalent of the BasicDataList control is the **Listings and viewers -> Basic datalist** web part.

	<p>Please note</p> <p>The CMSDataList control provides a more convenient way to display data from Kentico CMS.</p>
--	---

The following topics are available to help you familiarize yourself with the BasicDataList control:

- [Getting started](#) - contains a step-by-step tutorial that allows you to learn the basics of using the control
- [Configuration](#) - describes and explains the properties that can be set for the control
- [Appearance and styling](#) - describes how the design of the control can be modified

1.7.3.4.2 Getting started

The following is a step-by-step tutorial that will show you how to display a list of smartphones (CMS. Smartphone documents) from the sample Corporate Site using the BasicDataList control:

1. Create a new **Web form** somewhere in your website installation directory.
2. Switch to its **Design** tab, drag and drop a **BasicDataList** control from the toolbox onto the form and set its **RepeatColumns** property value to 3. This determines the number of columns that should be displayed.
3. Switch to the **Source** tab and add the code marked by the **BasicDataList template** comments between the `<cms:BasicDataList>` tags. The overall code of the BasicDataList control should look like this:

```
<cms:BasicDataList ID="BasicDataList1" runat="server" RepeatColumns="3">
  <%-- BasicDataList template
  ----- %>

  <itemtemplate>
    <div style="width: 250px">
      <h3>
        <%# Eval("SmartphoneName") %>
      </h3>
      <%# EcommerceFunctions.GetProductImage(Eval("SKUImagePath"), 200,
Eval("SmartphoneName")) %>
    </div>
  </itemtemplate>

  <%-- BasicDataList template
  ----- %>

</cms:BasicDataList>
```

This defines the template used by the BasicDataList to display items. The control dynamically replaces the `<%# ... %>` tags with values of the currently displayed record. This is then repeated for every record in the data source.

4. Switch to the code behind of the page and add the following references to the beginning of the code:

[C#]

```
using System.Data;
```

```
using CMS.CMSHelper;
```

[VB.NET]

```
Imports System.Data  
Imports CMS.CMSHelper
```

5. Now add the following code to the **Page_Load** method:

[C#]

```
DataSet ds = TreeHelper.SelectNodes("/%", false, "CMS.Smartphone", "",  
"SmartphoneName", -1, true);  
  
BasicDataList1.DataSource = ds;  
BasicDataList1.DataBind();
```

[VB.NET]

```
Dim ds As DataSet = TreeHelper.SelectNodes("/%", False, "CMS.Smartphone", "",  
"SmartphoneName", -1, True)  
  
BasicDataList1.DataSource = ds  
BasicDataList1.DataBind()
```

This retrieves all CMS.Smartphone documents from the Kentico CMS database as a DataSet and assigns it as the data source of the BasicDataList control.

6. Save the changes to the web form. Now right-click it in the Solution explorer and select **View in Browser**. The resulting page should display a list similar to the following:

Apple iPhone 3GS



BlackBerry Torch 9800



Motorola Atrix 4G



Apple iPhone 4



HTC Sensation



Samsung Google Nexus S



1.7.3.4.3 Configuration

The following properties of the BasicDataList control can be set or used in the API:

Property Name	Description	Sample Value
DataBindByDefault	Indicates whether data binding should be performed by default during the init event.	
HideControlForZeroRows	Indicates whether the control should be hidden when no data is loaded. The default value is False .	
PagerDataItem	Gets or sets the pager data item object.	
PagerForceNumberOfResults	If set, the DataSet containing paged items is not modified by the pager, but the pager itself behaves as if the amount of paged items were identical to this value. The value must be set to -1 for the property to be disabled.	
RelatedData	Custom data connected to the object.	
ZeroRowsText	Text to be shown when the control is hidden by the HideControlForZeroRows property.	"No records found."

As this control is inherited from the ASP.NET DataList control, it also has all of its standard properties.

1.7.3.4.4 Appearance and styling

You can modify the appearance of the BasicDataList control by setting the standard properties and templates inherited from the ASP.NET DataList control. You can find more details on particular properties in the .NET Framework documentation for the [DataList](#) class.

The following templates can be defined:

Template Name	Description	Sample Value
AlternatingItemTemplate	Code of the template applied to alternating items.	
EditItemTemplate	Code of the template applied to the item selected for editing.	
FooterTemplate	Code of the template used for the footer of the list.	
HeaderTemplate	Code of the template used for the header of the list.	
ItemTemplate	Code of the template applied to standard items.	<pre><div style="width: 100%"> <h3> <%# Eval("CellName") %> </h3> <%# EcommerceFunctions.GetProductImage(Eval("SKUImagePath"), 200) %> </div></pre>
SelectedItemTemplate	Code of the template applied to the selected item.	
SeparatorTemplate	Code of the template used for separating items.	

1.7.3.5 BasicRepeater

1.7.3.5.1 Overview

The BasicRepeater control allows you to display items from a data source in a list based on specified templates. It is derived from the intrinsic ASP.NET Repeater control, so standard ASP.NET configuration techniques can be used to set up BasicRepeater style and behaviour.

BasicRepeater can be used with any bindable data source - it doesn't use the Kentico CMS database or API.

The portal engine equivalent of the BasicRepeater control is the **Listings and viewers -> Basic repeater** web part.

	Please note
--	--------------------



The [CMSRepeater](#) control provides a more convenient way to display data from Kentico CMS.

The following topics are available to help you familiarize yourself with the BasicRepeater control:

- [Getting started](#) - contains a step-by-step tutorial that allows you to learn the basics of using the control
- [Configuration](#) - describes and explains the properties that can be set for the control
- [Appearance and styling](#) - describes how the design of the control can be modified

1.7.3.5.2 Getting started

The following is a step-by-step tutorial that will show you how to display a list of laptops (CMS.Laptop documents) from the sample Corporate Site using the BasicRepeater control:

1. Create a new **Web form** somewhere in your website installation directory.
2. Switch to its **Design** tab, drag and drop a **BasicRepeater** control from the toolbox onto the form.
3. Switch to the **Source** tab and add the code marked by the **BasicRepeater template** comments between the `<cms:BasicRepeater>` tags. The overall code of the BasicRepeater control should look like this:

```
<cms:BasicRepeater ID="BasicRepeater1" runat="server">
    <%-- BasicRepeater template
    ----- %>

    <itemtemplate>
        <h3>
            <%# Eval("LaptopName") %>
        </h3>
        <%# EcommerceFunctions.GetProductImage(Eval("SKUImagePath"), 200,
        Eval("LaptopName")) %>
    </itemtemplate>

    <%-- BasicRepeater template
    ----- %>

</cms:BasicRepeater>
```

This defines the template used by the BasicRepeater to display items. The control dynamically replaces the `<%# ... %>` tags with values of the currently displayed record. This is then repeated for every record in the data source.

4. Switch to the code behind of the page and add the following references to the beginning of the code:

[C#]

```
using System.Data;

using CMS.CMSHelper;
```

[VB.NET]

```
Imports System.Data

Imports CMS.CMSHelper
```

5. Now add the following code to the **Page_Load** method:

[C#]

```
DataSet ds = TreeHelper.SelectNodes("/%", false, "CMS.Laptop", "", "LaptopName", -1, true);

BasicRepeater1.DataSource = ds;
BasicRepeater1.DataBind();
```

[VB.NET]

```
Dim ds As DataSet = TreeHelper.SelectNodes("/%", False, "CMS.Laptop", "", "LaptopName", -1, True)

BasicRepeater1.DataSource = ds
BasicRepeater1.DataBind()
```

This retrieves all CMS.Laptop documents from the Kentico CMS database as a DataSet and assigns it as the data source of the BasicRepeater control.

6. Save the changes to the web form. Now right-click it in the Solution explorer and select **View in Browser**. The resulting page should display a list similar to the following:

Apple iPad 2**Apple MacBook Pro MC723LL/A****Dell XPS 15z**

1.7.3.5.3 Configuration

The following properties of the BasicRepeater control can be set or used in the API:

Property Name	Description	Sample Value
DataBindByDefault	Indicates whether data binding should be performed by default during the init event.	
HideControlForZeroRows	Indicates whether the control should be hidden when no data is loaded. The default value is False .	
PagerDataItem	Gets or sets the pager data item object.	
PagerForceNumberOfResults	If set, the DataSet containing paged items is not modified by the pager, but the pager itself behaves as if the amount of paged items were identical to this value. The value must be set to -1 for the property to be disabled.	
RelatedData	Custom data connected to the object.	
ZeroRowsText	Text to be shown when the control is hidden by the HideControlForZeroRows property.	"No records found."

As this control is inherited from the ASP.NET Repeater control, it also has all of its standard properties.

1.7.3.5.4 Appearance and styling

You can modify the appearance of the BasicRepeater control by setting the standard properties and templates inherited from the ASP.NET Repeater control. You can find more details on particular properties in the .NET Framework documentation for the [Repeater](#) class.

The following templates can be defined:

Template Name	Description	Sample Value
AlternatingItemTemplate	Code of the template applied to alternating items.	
FooterTemplate	Code of the template used for the footer of the list.	
HeaderTemplate	Code of the template used for the header of the list.	
ItemTemplate	Code of the template applied to standard items.	<pre><div style="width: 100%"> <h3> <%# Eval("CellName") %> </h3> <%# EcommerceFunctions. GetProductImage(Eval("SKUImagePath"), 200) %> </div></pre>
SeparatorTemplate	Code of the template used for separating items.	

1.7.3.6 BasicUniView

1.7.3.6.1 Overview

The BasicUniView control allows you to display items from a data source in a hierarchical structure based on specified templates. It is derived from the [UniView](#) control, but is extended to include several more properties.

BasicUniView can be used with any bindable data source - it doesn't use Kentico CMS database or API.

The portal engine equivalent of the BasicUniView control is the **Listings and viewers -> Basic universal viewer** web part.

Please note

If you want to display documents from Kentico CMS, please use the [CMSUniView](#) control that provides a more convenient way.

The following topics are available to help you familiarize yourself with the BasicUniView control:

- [Getting started](#) - contains a step-by-step tutorial that allows you to learn the basics of using the control
- [Configuration](#) - describes and explains the properties that can be set for the control
- [Appearance and styling](#) - describes how the design of the control can be modified

1.7.3.6.2 Getting started

The following is a step-by-step tutorial that will show you how to display all pages (CMS.Menuitem documents) from the sample Corporate Site in a hierarchical structure using the BasicUniView control:

1. Create a new **Web form** somewhere in your website installation directory.
2. Switch to its **Design** tab, drag and drop the **BasicUniView** control from the toolbox onto the form.
3. Add the code marked by the **BasicUniView templates** comments between the `<cms:BasicUniView>` tags. The overall code of the BasicUniView control should look like this:

```
<cms:BasicUniView ID="BasicUniView1" runat="server">

    <%-- BasicUniView templates
    ----- %>

    <ItemTemplate>
        <%# HTMLHelper.Encode(Convert.ToString(Eval("NodeAliasPath"))) %>
    </ItemTemplate>
    <AlternatingItemTemplate>
        <font color="#999999"><%# HTMLHelper.Encode(Convert.ToString(Eval(
        "NodeAliasPath"))) %>
        </font>
    </AlternatingItemTemplate>
    <SeparatorTemplate>
        </li>
        <li>
    </SeparatorTemplate>
    <HeaderTemplate>
        <ul>
        <li>
    </HeaderTemplate>
    <FooterTemplate>
        </li>
        </ul>
    </FooterTemplate>

    <%-- BasicUniView templates
    ----- %>

</cms:BasicUniView>
```

This sets the templates used when displaying the menu items. The control dynamically replaces the `<%# ... %>` tags with values of the currently displayed record. This is then repeated for every record in the data source.

4. Add the following references to the beginning of the web form code-behind:

[C#]

```
using System.Data;

using CMS.CMSHelper;
using CMS.GlobalHelper;
using CMS.SettingsProvider;
```

5. Now add the following code to the **Page_Load** method:

[C#]

```
//Create DataSet containing all menu item documents in the system
DataSet ds = TreeHelper.SelectNodes("/%", false, "CMS.MenuItem", "", "NodeLevel,
NodeOrder", -1, true);

//Check that the DataSet isn't empty
if (!DataHelper.DataSourceIsEmpty(ds))
{

//Create GroupedDataSource from the ds DataSet
GroupedDataSource gpd = new GroupedDataSource(ds, "NodeParentID",
"NodeLevel");

//Set RelationColumnID property of the UniView control
this.BasicUniView1.RelationColumnID = "NodeID";

//Bind the DataSet to the UniView control
this.BasicUniView1.DataSource = gpd;
this.BasicUniView1.DataBind();
}
```

This code reads documents from the database, saves them in a DataSet and then groups them according to the *NodeID* of their parent document and determines their level in the hierarchy according to their *NodeLevel*. This grouped data source is then provided to the BasicUniView control.

6. Save the changes to the web form and its code-behind file. Now right-click it in the Solution explorer and select **View in Browser**. The resulting page should look like this:

- /Home
- /Services
 - /Services/Web-design
 - /Services/Web-development
 - /Services/Network-administration
- /Products
 - /Products/Smartphones
 - /Products/Laptops-and-Tablets
 - /Products/Software
 - /Products/E-Books
 - /Products/IT-Services
 - /Products/Membership
- /News
- /Partners
 - /Partners/Silver-partners
 - /Partners/Gold-partners
- /Community
 - /Community/Forums
 - /Community/Forums/Forum-search
 - /Community/Blogs
 - /Community/Events
 - /Community/Events/Events-list
 - /Community/Wiki
 - /Community/Wiki/My-contributions
- /Company
 - /Company/Careers
 - /Company/Offices
 - /Company/Donate-us
- /Media
 - /Media/Images
 - /Media/Videos

1.7.3.6.3 Configuration

As it is inherited from the [UniView](#) control, the BasicUniView control has all of its properties (including templates). These can be found in the [UniView -> Configuration](#) topic. In addition, it has the following properties that can be set or used in the API:

Property Name	Description	Sample Value
DataBindByDefault	Indicates whether data binding should be performed by default during the init event.	
DataSource	The object from which the list of data items is retrieved.	
HideControlForZeroRows	Indicates whether the control should be hidden when no data is loaded. The default value is False .	
RelatedData	Custom data connected to the object.	
ZeroRowsText	Text to be shown when the control is hidden by the HideControlForZeroRows property.	"No records found."

1.7.3.6.4 Appearance and styling

The appearance of the BasicUniView control is determined by the code of its templates (inherited from the [UniView](#) control).

1.8 CMS Controls

1.8.1 Overview

CMS Controls are a set of controls that were designed to work within the Kentico CMS system. This means they are meant to be used only with data from Kentico CMS database and documents. They automatically perform many routine tasks, which makes them easier to work with, as common use does not require the writing of any additional code.

As they work with CMS documents, most of the controls in this section have a **Path** property that allows the selection of the documents that should be affected. The [Path specification in controls and web parts](#) topic explains the way this property should be filled.

To optimize the performance of the websites they are used on, CMS controls provide caching support. To learn more about this subject and how related configuration can be done, please see the [Caching](#) topic.

The following control categories are available:

- [Navigation](#)
- [Listings and viewers](#)
- [Edit mode buttons](#)
- [Editable regions for ASPX templates](#)
- [Search Controls](#)

1.8.2 Path specification in controls and web parts

Many web parts and controls use a **Path** property that allows you to specify which documents should be displayed. This is the **AliasPath** property of the document. You can use either an exact path or utilize special characters for specifying multiple selection or relative paths:

Using wildcard characters % and _

You can use % as a wildcard character for any number of characters, which allows you to select all documents under the specified site section.

Examples:

/ - only root
/% - all documents

/Products - only the *Products* document.
/Products/% - all documents under the *Products* document.

You can also use the *_* character (underscore) as a wildcard character for a single character.

Examples:

/Product_ - selects documents */ProductA*, */Product1*, etc.

Leaving the Path value empty

In many cases, the **Path value can be left empty**.

For navigation controls/web parts, such as CMSMenu/Drop-down menu, this causes the path to be set to all documents - /%.

In the case of listing controls/web parts, such as CMSRepeater/Repeater or CMSDataGrid/Grid, the path is set to display all child documents of the page containing the control or web part - *<current alias path>/%*.

Using a formatting string to get parts of the path

You can also use special expressions that **extract parts of the current path**, such as this.

Examples:

/{0}/{1}/% - all documents under the second document level of the current path

/{0}/{1}/Details - document *Details* under the second document level of the current path

Using relative paths

You can use relative path expressions to specify **sub-documents or parent documents**:

Examples:

./Product - document *Product* under the current path

../Product - document *Product* under the parent document of the current path

. - current path

.. - parent document of the current path

/% - all documents under the current path

../% - all documents under the parent document of the current path

1.8.3 Caching

What is Caching

Caching allows you to minimize the number of performed database queries. The server can store the data in the application's memory and the next time a user requests the content, the server will return it from the memory instead of performing a resource-intensive database query. Caching can improve the performance of your website significantly, depending on the specifics of your application.

The content **expires** after the specified time span and must be retrieved from the database again. Each cached item has its name and the cache memory is common for all pages in your web application.

Caching Support in Kentico CMS

You can manage the caching either by yourself in your code (please see the .NET Framework SDK documentation for more details) or you can leverage caching features of the following Kentico CMS Controls that are also used in CMS web parts:

- Controls in the [CMS Controls -> Navigation](#) section

- Controls in the [CMS Controls -> Listings and viewers](#) section
- [CMSPageManager](#)
- [CMSSearchResults](#)

All of these controls offer the following properties, that allow you to set up caching:

Property Name	Description	Sample Value
CacheDependencies	List of the cache keys on which the cached data depends.	cms.user all
CacheItemName	Name of the cache item the control will use.	"products_" & request.querystring("categoryid")
CacheMinutes	Number of minutes the retrieved content is cached for.	"10"

Cache Expiration Time

By setting the **CacheMinutes** property to a value higher than zero, the control starts to cache its source data. You can configure caching for all Kentico CMS content using the **Cache content (minutes)** setting in the **Site Manager -> Settings -> System -> Performance** section. If you set any particular value to the CacheMinutes property of a control/web part, it overrides the global settings. If you leave the value empty or set it to -1 (minus one), the global settings apply.

The caching mechanism of Kentico CMS Controls uses absolute expiration time instead of sliding expiration. It means that the cache item expires after a specified period of time regardless of if it was requested or not. It ensures that content is updated from the database at a regular interval.

Overriding the site-level caching settings

If you need to cache most of the content on your website, but still want to have a single control/web part that doesn't use cache, you can configure caching as described in the previous paragraph and set the value 0 (zero) to the **CacheMinutes** property of the particular control. It will override the site-level settings and disable caching for the single control/web part.

Cache Item Name

It's important to understand the **CacheItemName** property: Since the cache is common for all pages in your application, the cache item name should be unique not only for all pages, but also inside one page (in case you use several Kentico CMS Controls with caching on one page).

When you leave the **CacheItemName** property empty, the control automatically generates a name in this form: *URL including parameters/control ID*

If the content displayed on the page depends on some parameter, such as a URL parameter or the role of the current user, you need to adjust the **CacheItemName** value accordingly.

Example:

Your page `products.aspx` displays products according to the category that is passed through the URL parameter **category**. You will need to use code like this to ensure that the content will be cached "per

category":

[C#]

```
CMSDataGrid.CacheItemName = "products_grid1_" + Request.QueryString["category"];
```

[VB.NET]

```
CMSDataGrid.CacheItemName = "products_grid1_" & Request.QueryString("category")
```

Cache dependencies

Using the **CacheDependencies** property, you can specify which object changes cause the control/web part's cache to get cleared. Below, you can find a table showing which dummy cache keys get touched when some object gets changed, including some examples. By entering the appropriate dummy keys, one per line, you can specify that when the object gets changed, the cache gets cleared.

If you leave the property empty, default settings will be used. The default settings are configured for each control and include all possible object changes that the content of the web part could depend on.

Object	Touched keys	Sample values
Document (TreeNode)	node <sitename> <aliaspath> <culture> node <sitename> <aliaspath> nodeid <nodeid> nodeid <linkednodeid> nodes <sitename> <classname> all + for every parent node: node <sitename> <aliaspath> childnodes	node corporatesite /home en-us node corporatesite /home nodeid 12 nodeid 34 nodes corporatesite cms. menuitem all node sitename / childnodes
Any object (except documents)	<classname> all <classname> byid <id> <classname> byname <codename> <classname> byguid <guid>	cms.user all cms.user byid 53 cms.user byname administrator cms.user byguid 1ced44f3-f2fc- ...
Metafile	metafile <guid>	metafile 1ced44f3-f2fc- ...
Document attachment	attachment <guid>	attachment 1ced44f3-f2fc- ...
Forum attachment	forumattachment <guid>	forumattachment 1ced44f3-f2fc- ...
Avatar	avatarfile <guid>	avatarfile 1ced44f3-f2fc- ...
Media file	mediafile <guid> mediafile preview <guid>	mediafile 1ced44f3-f2fc- ... mediafile preview 1ced44f3-f2fc- ...

Page template	template <id>	template 12
CacheHelper .ClearFullPageCache	fullpage	fullpage

Example:

1. Let's presume that you have a control/web part displaying some information about users. Therefore, whenever some user gets their details modified, the control/web part's cache should be cleared. To ensure this, you need to enter **cms.user|all** into the property, which is the dummy key that would get touched whenever some user's details get changed.

2. Now let's presume that your control/web part is displaying information about one particular user - the administrator. Her user name is *administrator*, her ID is *53* and her GUID is something beginning with *1ced44f3-f2fc*. So if you want to have the cache cleared whenever this user's details are changed, you can use any of the following three keys that specify the user by the previously named properties:

- **cms.user|byid|53**
- **cms.user|byname|administrator**
- **cms.user|byguid|1ced44f3-f2fc-...**

1.8.4 CMS controls - common properties

The following properties provide configuration options for selecting Kentico CMS documents to most of the CMS Navigation and Standard listings and viewers controls. Please be aware that certain controls only use some of these properties:

Property Name	Description	Sample Value
CheckPermissions	Allows you to specify whether check permissions of the current user should be checked. If the value is <i>false</i> (default value) no permissions are checked. Otherwise, only nodes for which the user has read permission are selected.	
ClassNames	<p>Specifies which types of documents should be selected. Document types must be identified through their code names. Several values separated by semicolons (;) can be entered.</p> <p>The * wildcard can be used as a substitute for a random sequence of characters. For example <i>Product.*</i> would include the document types <i>Product.Camera</i>, <i>Product.CellPhone</i>, <i>Product.Computer</i> etc.</p> <p>If the property is left empty, controls retrieve all document types by default. In the case of menu and navigation controls, page (<i>cms.MenuItem</i>) documents are selected by</p>	"cms.news" or "cms.news;cms.article"

	<p>default.</p> <p>Please note: If all document types are loaded (empty value), only the common data columns from the <i>View_CMS_Tree_Joined</i> view will be available in the retrieved data. The specific fields of individual document types will not be included. This should be considered in the code of transformations, WHERE conditions, ORDER BY expressions etc.</p>	
CombineWithDefaultCulture	Indicates whether documents from the default culture version should be used if they are not available in the selected culture. This property is applied only if you do not set the TreeProvider property manually.	
CultureCode	Culture code of documents to be selected, such as en-us. If not specified, it's read from the user's session or the default value is used.	"en-us"
DataSource	Gets or sets a DataSet containing values used to fill the items of the control.	
FilterOutDuplicates	Indicates if duplicated (linked) documents should be filtered out from the data.	
MaxRelativeLevel	<p>Specifies the maximum number of content tree sub-levels from which content should be displayed. This number is relative, i.e. counted from the location of the page that the control is placed on.</p> <p>Entering -1 causes all child documents to be selected.</p>	
Path	<p>Path of the documents to be selected.</p> <p>See Path specification for details.</p>	See Path specification for examples.
SelectOnlyPublished	Indicates whether only published documents should be selected.	
TreeProvider	Tree provider instance used to access data. If no TreeProvider is assigned, a new TreeProvider instance is created automatically.	

CMS base - common properties:

The following properties provide basic configuration options to most of the CMS Navigation and Listings and viewers controls. Please be aware that certain controls only use some of these properties:

Property Name	Description	Sample Value
CacheDependencies	<p>List of the cache keys on which the cached data depends. When the cache item changes, the cache of the control is cleared. Each item (dependency) must be on one line.</p> <p>If you leave this property empty, default dependencies will be used.</p> <p>Please refer to the Caching topic to learn more.</p>	cms.user all
CacheItemName	<p>Name of the cache item the control will use.</p> <p>By setting this name dynamically, you can achieve caching based on a URL parameter or some other variable - simply enter the value of the parameter.</p> <p>If no value is set, the control stores its content in the item named "URL ControlID".</p> <p>Please refer to the Caching topic to learn more.</p>	"mycachename" + Request.QueryString["id"].ToString()
CacheMinutes	<p>Number of minutes the retrieved content is cached for.</p> <p>Zero indicates that the content will not be cached.</p> <p>-1 indicates that the site-level settings should be used.</p> <p>This parameter allows you to set up caching of content so that it doesn't have to be retrieved from the database each time a user requests the page.</p> <p>Please refer to the Caching topic to learn more.</p>	
ControlTagKey	Overrides the generation of the SPAN tag with a custom tag.	
FilterControl	Gets or sets the appropriate filter control used to limit the data read by this control.	
FilterName	Gets or sets the code name of the appropriate filter control used to limit the data read by this control.	
OrderBy	Gets or sets the ORDER BY clause of the	"NewsReleaseDate DESC"

	SQL statement.	
SelectedColumns	Database table columns that should be loaded with documents, separated by commas (,). Null or empty indicates that all columns should be selected.	
SiteName	Specifies the code name of the site to be used by the control.	
StopProcessing	Indicates if processing of the control should be stopped and the control should not retrieve or display any data.	
TopN	Specifies the maximum amount of rows that should be selected.	
WhereCondition	Gets or sets the WHERE clause of the SQL statement.	"ProductPrice > 100"

1.8.5 Navigation

1.8.5.1 Overview

The controls in this section provide functionality that helps users find their way around Kentico CMS websites. This includes various types of menus, site maps and other basic navigation tools.

These controls are used by the web parts in the **Navigation** category.

All Kentico CMS documents have settings that influence how they are displayed in menus, which affects the navigation controls in this section as well. Learn more about these settings in the [Document menu settings](#) topic.

Most of the controls in this section use CSS classes to modify their design and several of them contain the **CSSPrefix** property, which can be used to specify the class names. The [Using the CSSPrefix property](#) topic clarifies how this property works.

Available controls:

- [CMSBreadCrumbs](#)
- [CMSListMenu](#)
- [CMSMenu](#)
- [CMSSiteMap](#)
- [CMSTabControl](#)
- [CMSTreeMenu](#)
- [CMSTreeView](#)

1.8.5.2 Document menu settings

Various navigation related settings can be configured for individual Kentico CMS documents. This can be done in the **CMS Desk** interface at **Content -> Edit -> ... select document ... -> Properties -> Menu**.

These settings apply to CMS Navigation controls (and web parts) displaying the given document and they override any conflicting property settings of the controls unless a property is set to ignore them.

The following settings are available:

Basic properties

Menu caption	The name of the document as it's displayed in navigation. It may be different compared to the document name. If no value is entered, the document name is used.
Show in navigation	<p>Indicates if the document should be displayed by navigation controls and web parts (in menus).</p> <p>Please note: the document is displayed in the navigation if all of the following conditions are met:</p> <ol style="list-style-type: none"> 1. The Show in navigation box is checked. 2. The document is published. 3. The type of the document matches the document types configured in the appropriate navigation control (web part) - by default, only Page

	<p>(menu item) documents are displayed in navigation.</p> <p>4. If you turn on the Check permissions property of the menu control, the user must be allowed to read the given document so that it appears in the navigation controls.</p>
Show in site map	<p>Indicates if the document should be displayed by the CMSSiteMap control (Site map web part) and included in the in the website's Google Sitemap.</p> <p>Please note: the document is displayed in the site map if all of the following conditions are met:</p> <ol style="list-style-type: none"> 1. The Show in site map box is checked. 2. The document is published. 3. The type of the document matches the document types configured in the Site map control (web part) - by default, only Page (menu item) documents are displayed in navigation. 4. If you turn on the Check permissions property of the menu control, the user must be allowed to read the given document so that it appears in the navigation controls.

Menu actions

Standard behavior	The menu item redirects the user to the page as expected.
Inactive menu item	Clicking the menu item doesn't cause any action - the item is disabled. You can enter the alias path of the page which should be redirected to in the Redirect to URL field.
Javascript command	If you enter a JavaScript command, it will be run when this menu item is clicked. Example: <code>alert("hello");return false;</code>
URL redirection	The user is redirected to the target location when they try to access the given page. Example: <code>http://www.domain.com or ~/products.aspx</code>

Macro expressions can be used in the **URL redirection** and **JavaScript command** fields. These macros allow you to dynamically replace the given expression with specified values of the current menu item, such as alias path, id path, node name.

You only need to place macros in format `{%ColumnName%}` into the property field. For example, entering:

- `~/products.aspx?show=brand&aliaspath={%NodeAliasPath%}`

Into the **Redirect to URL** field of e.g. the `/MobileStore/Products/Nokia` document will cause users to be redirected to:

- `http://<domain>/products.aspx?show=brand&aliaspath=/MobileStore/Products/Nokia`

Please note: All apostrophes (') in the source data are escaped to \' so that they do not break JavaScript.

Menu design

The menu item design properties are available in three alternatives:

- standard design
- mouse-over design - style used when a user hovers their mouse over the menu item,
- highlighted design - style applied if the page represented by the menu item is currently selected.

These values override the settings of individual navigation controls (web parts) unless their **ApplyMenuDesign** property is set to *false*. The CSS styles defined in the CSS stylesheet are overridden as well.

Please note: some of the following properties may not be applied to the menu control depending on the menu control you are using.

Menu item style	Style definition of the menu item. Values can be entered the same way as when defining a CSS class in a stylesheet. <u>Sample value:</u> <i>color: orange; font-size: 140%</i>
Menu item CSS class	CSS class defined in the website's stylesheet. <u>Sample value:</u> <i>h1</i>
Menu item left image	Image that will be displayed next to the menu caption on the left side. Sample values as below.
Menu item image	Image that will be displayed in the menu instead of the menu caption. You can enter either an absolute URL or a relative path in the content tree. <u>Sample values:</u> <i>http://www.domain.com/image.gif</i> <i>~/Images-(1)/icon.aspx</i>
Menu item right image	Image that will be displayed next to the menu caption on the right side. Sample values as above.

1.8.5.3 Using the CSSPrefix property

The **CSSPrefix** property allows you to place multiple controls of the same type on the same page and differentiate the CSS classes that they use by adding a prefix to the class names. Additionally, it can also be used to **specify the style of menu sub-items at any chosen level**.

This property can be set for the following controls:

- [CMSListMenu](#)
- [CMSMenu](#)
- [CMSTreeMenu](#)

Example

Here's an example of how to specify various styles for particular menu levels:

1. First, you need to specify the list of prefixes for particular levels using the **CSSPrefix** property:

CSSPrefix = "MainMenu;MainMenuSubMenu;MainMenuOtherLevels"

As you can see, the prefixes used for individual levels are separated by semicolons and every prefix represents a lower level starting from the main one (level 0). The last defined prefix represents all sub-levels below it as well. If you only wish to differentiate the CSS classes used by multiple controls on the same page, one prefix is sufficient.

2. Now define the following styles with the specified prefixes:

.MainMenuCMSMenu
... for menu control

.MainMenuCMSMenuItem
.MainMenuCMSMenuItemMouseUp
... etc. for the first level of the menu (level 0)

.MainMenuSubMenuCMSMenuItem
.MainMenuSubMenuCMSMenuItemMouseUp
... etc. for the second level of the menu (level 1)

.MainMenuOtherLevelsCMSMenuItem
.MainMenuOtherLevelsCMSMenuItemMouseUp
... etc. for all underlying levels of the menu (level 2 and all remaining sub-levels)

1.8.5.4 CMS navigation - common properties

The following properties provide general configuration options to many of the CMS Navigation controls. Please be aware that certain controls only use some of these properties:

Property Name	Description	Sample Value
ApplyMenuDesign	Indicates whether the document menu settings for Menu design should be applied to this control. True by default.	
Columns	Contains the names of columns which should be loaded with the documents (menu items). Only the columns contained in the DefaultColumns property are loaded by default. If you need some other column to be loaded as well, please add its name into this property.	"DocumentPageTitle, DocumentPageKeywords"
CSSPrefix	Specifies the prefix of standard navigation control CSS classes. You can also use several values separated by a semicolon (;) for particular levels. Learn more at Using the CSSPrefix property .	"main;submenu1; submenu2"
HideControlForZeroRows	Indicates whether the control should be hidden when no data is loaded. Default value	

	is False.	
HighlightAllItemsInPath	Indicates whether all items in the unfolded path should be displayed as highlighted.	
SubmenuIndicator	Contains the path to an image that will be displayed next to every item that contains sub-items.	
UseAlternatingStyles	Indicates whether alternating styles should be used for even and odd items on the same menu level.	
UseItemImagesForHighlightedItem	Indicates whether the item image should be used if the highlighted image is not specified.	
WordWrap	Indicates whether text displayed by the control should use word wrapping or be replaced by 'nbsp' entities if it is too long.	
ZeroRowsText	Text to be shown when the control is hidden by the HideControlForZeroRows property.	"No records found."

1.8.5.5 CMSBreadCrumbs

1.8.5.5.1 Overview

The CMSBreadCrumbs control allows you to display the current user's position within a website in format:

Item 1 > Item 2 > Item 3

where **Item X** is the name of the document in the path.

The portal engine equivalent of the CMSBreadCrumbs control is the [Navigation -> Breadcrumbs](#) web part.

The following topics are available to help you familiarize yourself with the CMSBreadCrumbs control:

- [Getting started](#) - contains a quick step-by-step tutorial that allows you to learn the basics of using the control
- [Configuration](#) - describes and explains the properties that can be set for the control
- [Appearance and styling](#) - lists which CSS classes can be used with the control and how its appearance can be modified

1.8.5.5.2 Getting started

The following is a step-by-step tutorial that will show you how to display a user's current location within the structure of a website using the CMSBreadCrumbs control:

1. Create a new **Web form** and use it as a page template according to the guide in the [Using ASPX page templates](#) topic.

2. Switch to its **Design** tab and drag and drop a **CMSBreadCrumbs** control from the toolbox onto the form.
3. Save the changes to the web form. Now if you look at the page using the created template on some website, the user's position within the website will be displayed as is shown in the following image:

[Control Examples](#) > CMSBreadCrumbs

1.8.5.5.3 Configuration

The following properties of the CMSBreadCrumbs control can be set or used in the API:

All of the common properties from:

- [CMS controls - common properties](#)
- [CMS navigation - common properties](#)

In addition, the following properties are available:

Property Name	Description	Sample Value
BreadCrumbSeparator	Character(s) that separate the bread crumbs. You can use HTML code here.	">>"
BreadCrumbSeparatorRTL	Character(s) that separate the bread crumbs in RTL mode. You can use HTML code here.	"<<"
DefaultPath	Sets a default path that will be displayed in the breadcrumbs if no path is provided from the current page context.	"/home"
EncodeName	Indicates whether the items in the breadcrumbs should be HTML encoded.	
IgnoreShowInNavigation	The ShowInNavigation document property is ignored if this property is true.	
LoadDataAutomatically	Indicates whether data for the control should be loaded automatically. By default, the data is loaded automatically. If you set this property to false, you can enter a custom DataSet into the DataSource property and then call the <i>ReloadData(false)</i> method.	
RenderedHTML	Allows you to get or set the HTML code rendered by the control. You need to set this property before the Render event - e.g. in the OnLoad event.	
RenderLinkTitle	Specifies if the document name should be rendered as a TITLE tag of links (for better	

	accessibility).	
ShowCurrentItem	Indicates if the current (last) item should be displayed. True by default.	
ShowCurrentItemAsLink	Indicates if the current (last) item should be displayed as a link. False by default.	
StartingPath	Selects the document from which the path in the breadcrumbs will be displayed. Enter / (root document) to have the breadcrumbs display the full website path. For example, if you set this value to /News and go to /News/Events/2010, the breadcrumbs will display only Events -> 2010 .	"/products"
UrlTarget	Specifies the target frame for all links generated by the breadcrumbs.	"_blank"
UseRtlBehaviour	Indicates whether the bread crumbs should be rendered in the RTL direction for specific languages.	

Mentioned method:

Method Name	Description
ReloadData(bool forceLoad)	Reloads the data. If the forceLoad parameter is set to false and a value is assigned to the DataSource property, the properties of the CMSBreadCrumbs control are not used and only the data from the DataSource is loaded.

1.8.5.5.4 Appearance and styling

The appearance of the CMSBreadCrumbs control is determined by the CSS classes it uses and by some of its properties.

You can use the following CSS classes to modify the design of the control:

Class Name	Description
CMSBreadCrumbsLink	Link (A element) in the bread crumbs path.
CMSBreadCrumbsCurrentItem	Style of the last item representing the current location.

The recommended place to define these classes is in a stylesheet in the Kentico CMS administration interface at **Site Manager -> Development -> CSS stylesheets**. These stylesheets can be applied to individual documents (pages) that contain the control in **CMS Desk -> Content -> Edit -> Properties -> General -> CSS stylesheet**.

1.8.5.6 CMSListMenu

1.8.5.6.1 Overview

The CMSListMenu control allows you to create a large variety of menus. It renders and tags and the design depends only on your CSS style sheet. This menu control provides several advantages:

- It's based only on CSS styles which makes it highly configurable.
- It renders shorter HTML code than the [CMSMenu](#) control.
- It's fully XHTML compliant.
- The list-based menu is better accessible.
- You can create the drop-down menu using the list-based menu and CSS without almost any JavaScript.
- It automatically displays standard UL/LI listing with links if the browser does not support CSS styles so that the user can still navigate on the website.

However, **it requires advanced knowledge of CSS** as it doesn't render any specific layout by itself.

It allows you to display part of the CMS menu structure specified using its **Path**, **MaxRelativeLevel**, **ClassNames**, **CultureCode** and **WhereCondition** properties.

The portal engine equivalent of the CMSListMenu control is the [Navigation -> CSS list menu](#) web part.

The following topics are available to help you familiarize yourself with the CMSListMenu control:

- [Getting started](#) - contains a quick step-by-step tutorial that allows you to learn the basics of using the control
- [Configuration](#) - describes and explains the properties that can be set for the control
- [Appearance and styling](#) - describes and shows examples of how CSS styles can be used with the control

1.8.5.6.2 Getting started

The following is a step-by-step tutorial that will show you how to display a simple menu without any styles containing CMS content using the CMSListMenu control:

1. Create a new **Web form** somewhere in your website installation directory.
2. Switch to its **Design** tab, drag and drop a **CMSListMenu** control from the toolbox onto the form.
3. Switch to the **Source** tab. The code of the CMSListMenu control should look like this:

```
<cms:CMSListMenu ID="CMSListMenu1" runat="server" />
```

Now replace the DOCTYPE above the <HTML> element with the following one:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

4. Save the changes to the web form. Now right-click it in the Solution explorer and select **View in Browser**. The resulting page should display a UL/LI based menu like this:

- [Home](#)
- [Services](#)
 - [Web Design](#)
 - [Web Development](#)
 - [Network Administration](#)
- [Products](#)
 - [Smartphones](#)
 - [Laptops and Tablets](#)
 - [Software](#)
 - [E-Books](#)
 - [IT Services](#)
 - [Memberships](#)
- [News](#)
- [Partners](#)
 - [Silver Partners](#)
 - [Gold Partners](#)
- [Community](#)
 - [Forums](#)
 - [Forum search](#)
 - [Blogs](#)
 - [Events](#)
 - [Events list](#)
 - [Wiki](#)
 - [My contributions](#)
- [Company](#)
 - [Careers](#)
 - [Offices](#)
 - [Donate us](#)
- [Media](#)

To see how more advanced menus can be rendered using different CSS styles, continue this tutorial in one of the following topics:

- [Creating a horizontal drop-down menu using CSS styles](#)
- [Creating a vertical drop-down menu using CSS styles](#)

1.8.5.6.3 Configuration

The following properties of the CMSListMenu control can be set or used in the API:

All of the common properties from:

- [CMS controls - common properties](#)
- [CMS navigation - common properties](#)

In addition, the following properties are available:

Property Name	Description	Sample Value
DisplayHighlightedItemAsLink	Indicates whether the highlighted item should be displayed as a link.	
DisplayOnlySelectedPath	Indicates whether all sub-menus should be displayed or just the sub-menu under the highlighted (selected) item.	

EncodeMenuCaption	Indicates whether HTML encoding should be applied to the captions of menu items. Useful only in special cases where the names of the displayed documents contain HTML code.	
FirstItemCssClass	Specifies the CSS class for the first item on every menu level.	"ListMenuFirstItem"
HighlightedNodePath	Path of the item that should be highlighted as if it were selected. If you omit this value, the control automatically uses the current alias path from the aliaspath querystring parameter.	"/products/PDAs"
HoverCSSClassName	Name of the surrounding CSS class that is used to define styles for the hover effect if you want to render a drop-down menu.	"Horizontal"
ItemIdPrefix	Prefix placed before each item ID. You can use it to keep IDs unique if you have several CMSListMenu controls on the same page.	"submenu"
LastItemCssClass	Specifies the CSS class for the last item on every menu level.	"ListMenuLastItem"
LoadDataAutomatically	Indicates whether data for the control should be loaded automatically. By default, the data is loaded automatically. If you set this property to false, you can enter a custom DataSet into the DataSource property and then call the <i>ReloadData(false)</i> method.	
OnMouseOutScript	OnMouseOut script for menu items. You can use macro expressions here.	
OnMouseOverScript	OnMouseOver script for menu items. You can use macro expressions here.	
OrderBy	Gets or sets the ORDER BY clause of the SQL statement. Please be aware that it is necessary for the root of the displayed tree (or sub-tree) to be first in the resulting order, otherwise all documents may not be displayed correctly. This can be ensured by having the value of this property start with the NodeLevel column, such as in the sample value.	"NodeLevel, NodeOrder"
RenderCssClasses	Indicates whether CSS classes should be rendered for every element. If set to false, only the first and last item of a menu level will use a CSS class.	

RenderedHTML	Allows you to get or set the HTML code rendered by the control. You need to set this property before the Render event - e.g. in the OnLoad event.	
RenderImageAlt	Indicates whether the ALT attribute should be rendered for images used in the menu (for XHTML compatibility).	
RenderItemID	Indicates whether a unique ID should be rendered for every menu item.	
RenderLinkTitle	Specifies if the document name should be rendered as a TITLE tag of links (for better accessibility).	
UrlTarget	Specifies the target frame for all URLs.	"_blank"

Mentioned method:

Method Name	Description
ReloadData(bool forceLoad)	Reloads the data. If the forceLoad parameter is set to false and a value is assigned to the DataSource property, the properties of the CMSListMenu control are not used and only the data from the DataSource is loaded.

1.8.5.6.4 Appearance and styling

1.8.5.6.4.1 General

The appearance of the CMSListMenu control is determined by the CSS classes it uses and by some of its properties.

The following properties can be used to specify used CSS classes:

Property Name	Description
FirstItemCssClass	Specifies the CSS class for the first item on every menu level.
HoverCSSClassName	Name of the surrounding CSS class that is used to define styles for the hover effect if you want to render a drop-down menu.
LastItemCssClass	Specifies CSS class for the last item on every menu level.

You can also modify the design using the following CSS classes if the **RenderCssClasses** property is set to true:

Class Name	Description
CMSListMenuUL	UL element style

CMSListMenuLI	LI element style
CMSListMenuLink	A element style
CMSListMenuHighlightedLI	LI element style of a highlighted item

The recommended place to define these classes is in a stylesheet in the Kentico CMS administration interface at **Site Manager -> Development -> CSS stylesheets**. These stylesheets can be applied to individual documents (pages) that contain the control in **CMS Desk -> Content -> Edit -> Properties -> General -> CSS stylesheet**.

Please refer to [Using the CSSPrefix property](#) to learn how to add prefixes to these classes to customize items at any menu level.

The name of the CSS class used to render a drop-down menu must be identical to the value entered into the **HoverCSSClassName** property. Please be aware that this is case sensitive.

For examples of CSS styles that render a drop-down menu, please see the following topics:

- [Creating a horizontal drop-down menu using CSS styles](#)
- [Creating a vertical drop-down menu using CSS styles](#)

1.8.5.6.4.2 Creating a horizontal drop-down menu using CSS styles

This topic shows an example of how the CMSListMenu control can render a horizontal drop-down menu. If you wish to create this example for yourself, please follow the tutorial in the [Getting started](#) topic, then continue with the following steps:

1. Set the **HoverCSSClassName** property value to: *Horizontal*

The code of the CMSListMenu control should look like this:

```
<cms:CMSListMenu ID="CMSListMenu1" runat="server" HoverCSSClassName="Horizontal" />
```

2. Add the following style definitions inside the `<head>` element:

```
<style type="text/css" media="screen">

/* Horizontal menu class definitions*/
.Horizontal { BORDER-RIGHT: #c2c2c2 1px solid; BORDER-TOP: #c2c2c2 1px solid;
FONT-SIZE: 12px; FLOAT: left; BORDER-LEFT: #c2c2c2 1px solid; WIDTH: 100%; BORDER-
BOTTOM: #c2c2c2 1px solid; FONT-FAMILY: Arial; BACKGROUND-COLOR: #e2e2e2 }
.Horizontal UL { PADDING-RIGHT: 0px; PADDING-LEFT: 0px; PADDING-BOTTOM: 0px;
MARGIN: 0px; WIDTH: 100%; PADDING-TOP: 0px; LIST-STYLE-TYPE: none }
.Horizontal LI { BORDER-RIGHT: #e2e2e2 1px solid; PADDING-RIGHT: 0px; BORDER-TOP:
#e2e2e2 1px solid; DISPLAY: inline; PADDING-LEFT: 0px; FLOAT: left; PADDING-BOTTOM
: 0px; BORDER-LEFT: #e2e2e2 1px solid; PADDING-TOP: 0px; BORDER-BOTTOM: #e2e2e2
1px solid }
.Horizontal A { PADDING-RIGHT: 3px; DISPLAY: block; PADDING-LEFT: 3px; PADDING-
BOTTOM: 2px; MARGIN: 0px; WIDTH: 112px; COLOR: black; PADDING-TOP: 2px;
```

```

BACKGROUND-COLOR: #e2e2e2; TEXT-DECORATION: none }
.Horizontal A:hover { BACKGROUND: #808080 no-repeat right bottom; COLOR: white }
.Horizontal UL UL { Z-INDEX: 500; WIDTH: 120px; BORDER-BOTTOM: #c2c2c2 2px solid;
POSITION: absolute }
.Horizontal UL UL LI { CLEAR: left; DISPLAY: block; POSITION: relative }
.Horizontal UL UL UL { BORDER-RIGHT: #c2c2c2 2px solid; LEFT: 100%; BORDER-BOTTOM:
white 0px solid; TOP: -1px }
.Horizontal UL UL { DISPLAY: none }
.Horizontal UL LI:hover UL UL { DISPLAY: none }
.Horizontal UL UL LI:hover UL UL { DISPLAY: none }
.Horizontal UL LI:hover UL { DISPLAY: block }
.Horizontal UL UL LI:hover UL { DISPLAY: block }
.Horizontal UL UL UL LI:hover UL { DISPLAY: block }
</style>

```

This modifies the CSS style of the menu so that it displays a horizontal drop-down menu.

The classes are defined in the <head> element only for this quick example, if you wish to use the control on a Kentico CMS website, it is recommended to define these classes in the stylesheet used by the website or specific page via the administration interface in **Site Manager -> Development -> CSS stylesheets**.

3. Save the changes to the web form. Now right-click it in the Solution explorer and select **View in Browser**. The resulting page should display a menu like this:



1.8.5.6.4.3 Creating a vertical drop-down menu using CSS styles

This topic shows an example of how the CMSListMenu control can render a vertical drop-down menu. If you wish to create this example for yourself, please follow the tutorial in the [Getting started](#) topic, then continue with the following steps:

1. Set the **HoverCSSClassName** property value to: *Vertical*

The code of the CMSListMenu control should look like this:

```
<cms:CMSListMenu ID="CMSListMenu1" runat="server" HoverCSSClassName="Vertical" />
```

2. Add the following style definitions inside the <head> element:

```

<style type="text/css" media="screen">
/* Vertical menu class definitions*/

```

```

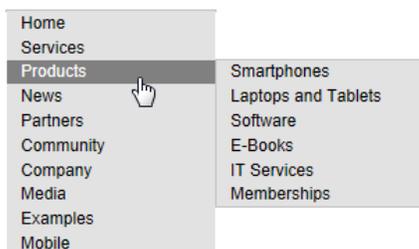
.Vertical { BORDER-RIGHT: #c2c2c2 1px solid; BORDER-TOP: #c2c2c2 1px solid; FONT-
SIZE: 12px; BORDER-LEFT: #c2c2c2 1px solid; WIDTH: 150px; BORDER-BOTTOM: #c2c2c2
1px solid; FONT-FAMILY: Arial; BACKGROUND-COLOR: #e2e2e2 }
.Vertical UL { PADDING-RIGHT: 0px; PADDING-LEFT: 0px; PADDING-BOTTOM: 0px; MARGIN:
0px; PADDING-TOP: 0px; LIST-STYLE-TYPE: none }
.Vertical LI { POSITION: relative; FLOAT: left; WIDTH: 100% }
.Vertical A { PADDING-RIGHT: 0px; BACKGROUND-POSITION: 0px 50%; DISPLAY: block;
PADDING-LEFT: 10px; PADDING-BOTTOM: 2px; MARGIN: 0px; WIDTH: 140px; COLOR: black;
PADDING-TOP: 2px; BACKGROUND-REPEAT: no-repeat; BACKGROUND-COLOR: #e2e2e2; TEXT-
DECORATION: none }
.Vertical A:hover { BACKGROUND: #808080 no-repeat 0px 50%; COLOR: white }
.Vertical UL UL { BORDER-RIGHT: #c2c2c2 1px solid; BORDER-TOP: #c2c2c2 1px solid;
Z-INDEX: 100; LEFT: 100%; BORDER-LEFT: #c2c2c2 1px solid; WIDTH: 100%; BORDER-
BOTTOM: #c2c2c2 1px solid; POSITION: absolute; TOP: -1px }
#Vertical1 UL { DISPLAY: none }
#Vertical1 LI:hover UL UL { DISPLAY: none }
#Vertical1 UL LI:hover UL UL { DISPLAY: none }
#Vertical1 LI:hover UL { DISPLAY: block }
#Vertical1 UL LI:hover UL { DISPLAY: block }
#Vertical1 UL UL LI:hover UL { DISPLAY: block }
</style>

```

This modifies the CSS style of the menu so that it displays a vertical drop-down menu.

The classes are defined in the <head> element only for this quick example, if you wish to use the control on a Kentico CMS website, it is recommended to define these classes in the stylesheet used by the website or specific page via the administration interface in **Site Manager -> Development -> CSS stylesheets**.

3. Save the changes to the web form. Now right-click it in the Solution explorer and select **View in Browser**. The resulting page should display a menu like this:



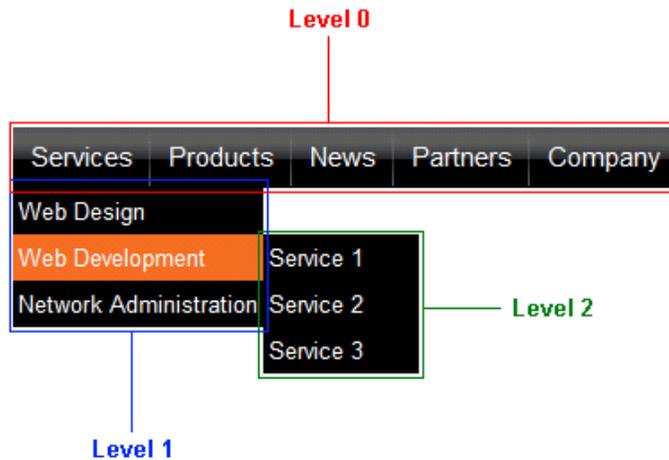
1.8.5.7 CMSMenu

1.8.5.7.1 Overview

The CMSMenu control allows you to display a multi-level DHTML menu based on data from Kentico CMS.

It allows you to display part of the CMS menu structure specified using its **Path**, **MaxRelativeLevel**, **ClassNames**, **CultureCode** and **WhereCondition** properties.

The following image is an example of how the CMSMenu control looks with the Horizontal **Layout**, including how menu item levels are rendered:



The portal engine equivalent of the CMSMenu control is the [Navigation -> Drop-down menu](#) web part.

The following topics are available to help you familiarize yourself with the CMSMenu control:

- [Getting started](#) - contains a step-by-step tutorial that allows you to learn the basics of using the control
- [Configuration](#) - describes and explains the properties that can be set for the control
- [Appearance and styling](#) - lists which CSS classes can be used with the control and how its appearance can be modified

1.8.5.7.2 Getting started

The following is a step-by-step tutorial that will show you how to display a simple DHTML menu containing CMS content using the CMSMenu control:

1. Create a new **Web form** somewhere in your website installation directory.
2. Switch to its **Design** tab, drag and drop a **CMSMenu** control from the toolbox onto the form and set its following properties:

- **Layout:** Horizontal
- **ClassNames:** CMS.MenuItem;CMS.Software
- **CSSPrefix:** Main;Sub

This tells the control to use a horizontal layout and which types of documents (pages) should be included. In this example, the menu will only display standard pages and software product types, but you can specify any other document types as needed. It also sets the CSS classes that should be used for the main menu and sub-menus. These classes will be defined in the next step. Please refer to [Using the CSSPrefix property](#) for more information.

3. Switch to the **Source** tab. The code of the CMSMenu control should look like this:

```
<cms:CMSMenu ID="CMSMenu1" runat="server" Layout="Horizontal" ClassNames="CMS.
MenuItem;CMS.Software"
CSSPrefix="Main;Sub" />
```

Now add the following code between the tags of the `<head>` element:

```
<style type="text/css">

/* horizontal menu - main menu */
.MainCMSMenu { BORDER-RIGHT: 0px; TABLE-LAYOUT: fixed; BORDER-TOP: 0px; BORDER-LEFT: 0px; WIDTH: 100px; BORDER-BOTTOM: 0px; BACKGROUND-COLOR: #b8baf6 }
.MainCMSMenuItem { PADDING-RIGHT: 15px; PADDING-LEFT: 5px; FONT-SIZE: 10pt; PADDING-BOTTOM: 2px; WIDTH: 100px; COLOR: black; PADDING-TOP: 2px; FONT-FAMILY: verdana }
.MainCMSMenuItemMouseUp { PADDING-RIGHT: 15px; PADDING-LEFT: 5px; FONT-SIZE: 10pt; PADDING-BOTTOM: 2px; WIDTH: 100px; COLOR: black; PADDING-TOP: 2px; FONT-FAMILY: verdana }
.MainCMSMenuItemMouseOver { PADDING-RIGHT: 15px; PADDING-LEFT: 5px; FONT-SIZE: 10pt; PADDING-BOTTOM: 2px; WIDTH: 100px; CURSOR: hand; COLOR: white; PADDING-TOP: 2px; FONT-FAMILY: verdana; BACKGROUND-COLOR: #4a3c8c }
.MainCMSMenuItemMouseDown { PADDING-RIGHT: 15px; PADDING-LEFT: 5px; FONT-SIZE: 10pt; PADDING-BOTTOM: 2px; WIDTH: 100px; COLOR: black; PADDING-TOP: 2px; FONT-FAMILY: verdana }
.MainCMSMenuHighlightedMenuItem { PADDING-RIGHT: 15px; PADDING-LEFT: 5px; FONT-SIZE: 10pt; PADDING-BOTTOM: 2px; WIDTH: 100px; COLOR: black; PADDING-TOP: 2px; FONT-FAMILY: verdana; BACKGROUND-COLOR: #ff7315 }
.MainCMSMenuHighlightedMenuItemMouseUp { PADDING-RIGHT: 15px; PADDING-LEFT: 5px; FONT-SIZE: 10pt; PADDING-BOTTOM: 2px; WIDTH: 100px; CURSOR: hand; COLOR: black; PADDING-TOP: 2px; FONT-FAMILY: verdana; BACKGROUND-COLOR: #ff7315 }
.MainCMSMenuHighlightedMenuItemMouseOver { PADDING-RIGHT: 15px; PADDING-LEFT: 5px; FONT-SIZE: 10pt; PADDING-BOTTOM: 2px; WIDTH: 100px; CURSOR: hand; COLOR: black; PADDING-TOP: 2px; FONT-FAMILY: verdana; BACKGROUND-COLOR: #ff7315 }
.MainCMSMenuHighlightedMenuItemMouseDown { PADDING-RIGHT: 15px; PADDING-LEFT: 5px; FONT-SIZE: 10pt; PADDING-BOTTOM: 2px; WIDTH: 100px; CURSOR: hand; COLOR: black; PADDING-TOP: 2px; FONT-FAMILY: verdana; BACKGROUND-COLOR: #ff7315 }

/* horizontal menu - sub-menus */
.SubCMSMenuItem { PADDING-RIGHT: 15px; PADDING-LEFT: 5px; FONT-SIZE: 10pt; BACKGROUND: #e7e7ff; PADDING-BOTTOM: 2px; WIDTH: 100px; COLOR: black; PADDING-TOP: 2px; FONT-FAMILY: verdana }
.SubCMSMenuItemMouseUp { PADDING-RIGHT: 15px; PADDING-LEFT: 5px; FONT-SIZE: 10pt; PADDING-BOTTOM: 2px; WIDTH: 100px; COLOR: black; PADDING-TOP: 2px; FONT-FAMILY: verdana }
.SubCMSMenuItemMouseOver { PADDING-RIGHT: 15px; PADDING-LEFT: 5px; FONT-SIZE: 10pt; PADDING-BOTTOM: 2px; WIDTH: 100px; CURSOR: hand; COLOR: white; PADDING-TOP: 2px; FONT-FAMILY: verdana; BACKGROUND-COLOR: green }
.SubCMSMenuItemMouseDown { PADDING-RIGHT: 15px; PADDING-LEFT: 5px; FONT-SIZE: 10pt; PADDING-BOTTOM: 2px; WIDTH: 100px; COLOR: black; PADDING-TOP: 2px; FONT-FAMILY: verdana }
.SubCMSMenuHighlightedMenuItem { PADDING-RIGHT: 15px; PADDING-LEFT: 5px; FONT-SIZE: 10pt; PADDING-BOTTOM: 2px; WIDTH: 100px; COLOR: black; PADDING-TOP: 2px; FONT-FAMILY: verdana; BACKGROUND-COLOR: #ff7315 }
.SubCMSMenuHighlightedMenuItemMouseUp { PADDING-RIGHT: 15px; PADDING-LEFT: 5px; FONT-SIZE: 10pt; PADDING-BOTTOM: 2px; WIDTH: 100px; CURSOR: hand; COLOR: black; PADDING-TOP: 2px; FONT-FAMILY: verdana; BACKGROUND-COLOR: #ff7315 }
.SubCMSMenuHighlightedMenuItemMouseOver { PADDING-RIGHT: 15px; PADDING-LEFT: 5px; FONT-SIZE: 10pt; PADDING-BOTTOM: 2px; WIDTH: 100px; CURSOR: hand; COLOR: black; PADDING-TOP: 2px; FONT-FAMILY: verdana; BACKGROUND-COLOR: #ff7315 }
.SubCMSMenuHighlightedMenuItemMouseDown { PADDING-RIGHT: 15px; PADDING-LEFT: 5px; FONT-SIZE: 10pt; PADDING-BOTTOM: 2px; WIDTH: 100px; CURSOR: hand; COLOR: black; PADDING-TOP: 2px; FONT-FAMILY: verdana; BACKGROUND-COLOR: #ff7315 }
```

```
</style>
```

This sets the CSS styles that will modify the appearance of the menu. The CMSMenu control renders a menu even without any CSS classes specified, but it is extremely basic and not very user friendly. You can find out what individual CSS classes affect in the [Appearance and styling](#) topic.

The classes are defined in the <head> element only for this quick example, if you wish to use the control on a Kentico CMS website, it is recommended to define these classes in the stylesheet used by the website or specific page via the administration interface in **Site Manager -> Development -> CSS stylesheets**.

4. Save the changes to the web form. Now right-click it in the Solution explorer and select **View in Browser**. The resulting page should look like this:

Home	Services	Products	News	Partners	Community	Company	Media
		Smartphones					
		Laptops and Tablets					
		Software	Microsoft Office Professional 2010				
		E-Books					
		IT Services					
		Memberships	Microsoft Windows 7 Ultimate				

1.8.5.7.3 Configuration

The following properties of the CMSMenu control can be set or used in the API:

All of the common properties from:

- [CMS controls - common properties](#)
- [CMS navigation - common properties](#)

In addition, the following properties are available:

Property Name	Description	Sample Value
Cursor	Determines the mouse cursor used when controlling the menu.	"Default" "Pointer"
EnableMouseUpDownClass	Indicates whether the menu should render different CSS classes for mouse-up and mouse-down events.	
EnableRTLBehaviour	Indicates whether a right-to-left layout should be used on documents displayed in a RTL culture.	
EncodeMenuCaption	Indicates whether HTML encoding should be applied to the captions of menu items. Useful only in special cases where the	

	names of the displayed documents contain HTML code.	
ExternalScriptPath	Path of the external .js file with skmMenu scripts. The default path is ~/cmsscripts/skmmenu.js.	"~/myscripts/skmmenu.js"
HighlightedMenuItem	This property can be used to get the currently highlighted menu item and set its CSS classes.	
HighlightedNodePath	Path of the item that should be highlighted as if it were selected. If you omit this value, the control automatically uses the current alias path from the aliaspath querystring parameter.	"/products/PDAs"
Layout	Determines the layout of the menu.	"Vertical" "Horizontal"
MenuControl	This property can be used to access the menu (skmMenu) control and its CSS classes and properties.	
Padding	Padding of the CMSMenu table.	
RenderImageAlt	Indicates whether the ALT attribute should be rendered for images used in the menu (for XHTML compatibility).	
RenderItemName	Indicates if the itemName attribute of menu items should be rendered in the output HTML code.	
SeparatorCssClass	CSS class of the separator cell (TD element) for the top menu level.	
SeparatorHeight	Height of the separator placed between menu items of the top menu level.	
SeparatorText	Text of the separator placed between menu items of the first menu level.	" "
Spacing	Spacing of the CMSMenu table.	

Method Name	Description
ReloadData(bool forceLoad)	<p>Reloads the data.</p> <p>If the forceLoad parameter is set to false and a value is assigned to the DataSource property, the properties of the CMSMenu control are not used and only the data from the DataSource is loaded.</p>

1.8.5.7.4 Appearance and styling

The appearance of the CMSMenu control is determined by the CSS classes it uses and by some of its properties.

You can also use the following CSS classes:

Class Name	Description
CMSSMenu	CSS class of the menu table.
CMSSMenuItem	CSS class of menu items.
CMSSMenuItemMouseDown	CSS class of menu items when the mouse button is down.
CMSSMenuItemMouseOver	CSS class of a menu item when a user moves the mouse cursor over it.
CMSSMenuItemMouseUp	CSS class of menu items when the mouse button is released.
CMSSMenuHighlightedMenuItem	CSS class of highlighted menu items.
CMSSMenuHighlightedMenuItemMouseDown	CSS class of highlighted menu items when the mouse button is down.
CMSSMenuHighlightedMenuItemMouseOver	CSS class of a highlighted menu item when a user moves the mouse cursor over it.
CMSSMenuHighlightedMenuItemMouseUp	CSS class of highlighted menu items when the mouse button is released.

The recommended place to define these classes is in a stylesheet in the Kentico CMS administration interface at **Site Manager -> Development -> CSS stylesheets**. These stylesheets can be applied to individual documents (pages) that contain the control in **CMS Desk -> Content -> Edit -> Properties -> General -> CSS stylesheet**.

Please refer to [Using the CSSPrefix property](#) to learn how to add prefixes to these classes to customize items at any menu level.

1.8.5.8 CMSSiteMap

1.8.5.8.1 Overview

The CMSSiteMap control allows you to display the whole navigation structure of a website or just its specified part. It reads CMS documents and renders their structure as a site map.

It allows you to display part of the CMS menu structure specified using its **Path**, **MaxRelativeLevel**, **ClassNames**, **CultureCode** and **WhereCondition** properties.

Only documents that have their **Show in sitemap** setting enabled in **CMS Desk -> Content -> Edit -> Properties -> Menu** will be displayed by this control.

The portal engine equivalent of the CMSSiteMap control is the [Navigation -> Site map](#) web part.

The following topics are available to help you familiarize yourself with the CMSSiteMap control:

- [Getting started](#) - contains a quick step-by-step tutorial that allows you to learn the basics of using the control
- [Configuration](#) - describes and explains the properties that can be set for the control
- [Appearance and styling](#) - describes and shows an example of how CSS styles can be used with the control

1.8.5.8.2 Getting started

The following is a step-by-step tutorial that will show you how to display a site map based on CMS content using the CMSSiteMap control:

1. Create a new **Web form** somewhere in your website installation directory.
2. Switch to its **Design** tab, drag and drop a **CMSSiteMap** control from the toolbox onto the form.
3. Switch to the **Source** tab. The code of the CMSSiteMap control should look like this:

```
<cms:CMSSiteMap ID="CMSSiteMap1" runat="server" />
```

4. Save the changes to the web form. Now right-click it in the Solution explorer and select **View in Browser**. The resulting page should look like this:

- [Home](#)
- [Services](#)
 - [Web Design](#)
 - [Web Development](#)
 - [Network Administration](#)
- [Products](#)
 - [Smartphones](#)
 - [Laptops and Tablets](#)
 - [Software](#)
 - [E-Books](#)
 - [IT Services](#)
 - [Memberships](#)
- [News](#)
- [Partners](#)
 - [Silver Partners](#)
 - [Gold Partners](#)
- [Community](#)
 - [Forums](#)
 - [Forum search](#)
 - [Blogs](#)
 - [Events](#)
 - [Events list](#)
 - [Wiki](#)
 - [My contributions](#)
- [Company](#)
 - [Careers](#)

Continue this tutorial in the [Appearance and styling](#) topic to see how CSS styles can be applied to the CMSSiteMap control.

1.8.5.8.3 Configuration

The following properties of the CMSSiteMap control can be set or used in the API:

All of the common properties from:

- [CMS controls - common properties](#)
- [CMS navigation - common properties](#)

In addition, the following properties are available:

Property Name	Description	Sample Value
ApplyMenuInactivation	If checked, documents in the site map that are set as Inactive menu items will not be rendered as links. This setting can be configured for individual documents in CMS Desk -> Content -> Edit -> Properties -> Menu .	
EncodeMenuCaption	Indicates whether HTML encoding should be applied to the captions of site map items. Useful only in special cases where the names of the displayed documents contain HTML code.	
LoadDataAutomatically	Indicates whether data for the control should be loaded automatically. This is performed by default. If you set this property to false, you can enter a custom DataSet into the DataSource property and then call the <i>ReloadData(false)</i> method.	
OrderBy	Gets or sets the ORDER BY clause of the SQL statement. Please be aware that it is necessary for the root of the displayed tree (or sub-tree) to be first in the resulting order, otherwise all documents may not be displayed correctly. This can be ensured by having the value of this property start with the NodeLevel column, such as in the sample value.	"NodeLevel, NodeOrder"
RenderedHTML	Allows you to get or set the HTML code rendered by the control. You need to set this property before the Render event - e.g. in the OnLoad event.	
RenderLinkTitle	If enabled, the site map will render document names as Title attributes (tooltips) for all links. This can improve the accessibility of	

	your website.	
UrlTarget	Specifies the target frame for all links in the site map.	"_blank"

Mentioned method:

Method Name	Description
ReloadData(bool forceLoad)	<p>Reloads the data.</p> <p>If the forceLoad parameter is set to false and a value is assigned to the DataSource property, the properties of the CMSSiteMap control are not used and only the data from the DataSource is used.</p>

1.8.5.8.4 Appearance and styling

The appearance of the CMSSiteMap control is determined by the CSS classes it uses and by some of its properties.

You can use the following CSS classes to modify the design of the control:

Class Name	Description
CMSSiteMapList	The UL element in the site map.
CMSSiteMapListItem	The LI element in the site map.
CMSSiteMapLink	Link (A element) in the site map.

The recommended place to define these classes is in a stylesheet in the Kentico CMS administration interface at **Site Manager -> Development -> CSS stylesheets**. These stylesheets can be applied to individual documents (pages) that contain the control in **CMS Desk -> Content -> Edit -> Properties -> General -> CSS stylesheet**.

Example

The following is an example of how CSS styles can be applied to a CMSSiteMap control. If you wish to create this example for yourself, please follow the tutorial in the [Getting started](#) topic, then continue with the following steps:

1. Add the following style definitions inside the `<head>` element:

```
<style type="text/css">

/* Site map class definitions */
.CMSSiteMapList { }
.CMSSiteMapListItem { list-style-type: square; }
.CMSSiteMapLink { color: #C34C17; text-decoration:none; }

</style>
```

This will modify the appearance of the site map.

2. Save the changes to the web form. Now right-click it in the Solution explorer and select **View in Browser**. The resulting page should look like this:

- Home
- Services
 - Web Design
 - Web Development
 - Network Administration
- Products
 - Smartphones
 - Laptops and Tablets
 - Software
 - E-Books
 - IT Services
 - Memberships
- News
- Partners
 - Silver Partners
 - Gold Partners
- Community
 - Forums
 - Forum search
 - Blog

1.8.5.9 CMSTabControl

1.8.5.9.1 Overview

The CMSTabControl control allows you to display a one-level tab menu based on data from Kentico CMS. It reads the specified documents and renders the menu according to their values.

It allows you to display part of the CMS menu structure specified using its **Path**, **MaxRelativeLevel**, **ClassNames**, **CultureCode** and **WhereCondition** properties.

It is derived from the [BasicTabControl](#) control, but it doesn't require any additional code to work and is extended to contain a set of common properties for CMS navigation controls.

The portal engine equivalent of the CMSTabControl control is the [Navigation -> Tab menu](#) web part.

The following topics are available to help you familiarize yourself with the CMSTabControl control:

- [Getting started](#) - contains a step-by-step tutorial that allows you to learn the basics of using the control
- [Configuration](#) - describes and explains the properties that can be set for the control
- [Appearance and styling](#) - describes and shows an example of how CSS styles can be used with the control

1.8.5.9.2 Getting started

The following is a step-by-step tutorial that will show you how to display a tab menu based on CMS content using the CMSTabControl control:

1. Create a new **Web form** somewhere in your website installation directory.

2. Switch to its **Design** tab, drag and drop a **CMSTabControl** control from the toolbox onto the form and set its following properties:

- **MaxRelativeLevel:** 1
- **SelectFirstItemByDefault:** True

This makes sure that only documents from the first level are displayed and that the first tab is selected by default.

3. Switch to the **Source** tab. The code of the CMSTabControl control should look like this:

```
<cms:CMSTabControl ID="CMSTabControl1" runat="server" MaxRelativeLevel="1"
SelectFirstItemByDefault="True" />
```

Now add the following code between the tags of the `<head>` element:

```
<style type="text/css">

/* Tab menu class definitions */
.TabControlTable { FONT-SIZE: 14px; FONT-FAMILY: Arial,Verdana }
.TabControlRow { }
.TabControl { BORDER-RIGHT: black 1px solid; BORDER-TOP: black 1px solid; FONT-
WEIGHT: bold; BACKGROUND: #e7e7ff; BORDER-LEFT: black 1px solid; CURSOR: hand;
COLOR: black }
.TabControlSelected { BORDER-RIGHT: black 1px solid; BORDER-TOP: black 1px solid;
FONT-WEIGHT: bold; BACKGROUND: #4a3c8c; BORDER-LEFT: black 1px solid; CURSOR:
default; COLOR: white }
.TabControlLinkSelected { COLOR: white; TEXT-DECORATION: none }
.TabControlLink { COLOR: black; TEXT-DECORATION: none }
.TabControlLeft { WIDTH: 1px }
.TabControlRight { WIDTH: 0px }
.TabControlSelectedLeft { WIDTH: 1px }
.TabControlSelectedRight { WIDTH: 0px }

</style>
```

This sets the CSS styles that will modify the appearance of the tab menu. The CMSTabControl control renders tabs even without any CSS classes specified, but they are extremely basic. You can find out what individual CSS classes affect in the [Appearance and styling](#) topic.

The classes are defined in the `<head>` element only for this quick example, if you wish to use the control on a Kentico CMS website, it is recommended to define these classes in the used stylesheet in the administration interface at **Site Manager -> Development -> CSS stylesheets**.

4. Add the following code just after the `<cms:CMSTabControl>` element. It will display a stripe under the tabs.

```
<hr style="width:100%; height:2px; margin-top:0px;" />
```

5. Save the changes to the web form. Now right-click it in the Solution explorer and select **View in Browser**. The resulting page should display a tab menu like this:

[Home](#) [Services](#) [Products](#) [News](#) [Partners](#) [Community](#) [Company](#) [Media](#) [Examples](#) [Mobile](#)

1.8.5.9.3 Configuration

As it is inherited from the [BasicTabControl](#) control, the CMSTabControl control has all of its properties. These can be found in the [BasicTabControl -> Configuration](#) topic.

In addition, it has the following properties that can be set or used in the API:

Those of the common properties that have meaning for a single level tab menu:

- [CMS controls - common properties](#)
- [CMS navigation - common properties](#)

As well as:

Property Name	Description	Sample Value
EncodeMenuCaption	Indicates whether HTML encoding should be applied to the captions of menu items. Useful only in special cases where the names of the displayed documents contain HTML code.	
HighlightedNodePath	Path of the item that should be highlighted as if selected. If you omit this value, the control automatically uses the current alias path from the aliaspath querystring parameter.	"/products/notebooks"
LoadDataAutomatically	Indicates whether data for the control should be loaded automatically. This is performed by default. If you set this property to false, you can enter a custom DataSet into the DataSource property and then call the <i>ReloadData(false)</i> method.	
RenderImageAlt	Indicates whether the ALT attribute should be rendered for images used in the menu (for XHTML compatibility).	

Mentioned method:

Method Name	Description
ReloadData(bool forceLoad)	Reloads the data. If the forceLoad parameter is set to false and a value is assigned to

	the DataSource property, the properties of the CMSTabControl control are not used and only the data from the DataSource is loaded.
--	---

1.8.5.9.4 Appearance and styling

The appearance of the CMSTabControl control is determined by the CSS classes it uses and by some of its properties.

It uses the same CSS classes as the BasicTabControl, which it inherits. These can be found at [BasicTabControl -> Appearance and styling](#).

The recommended place to define these classes is in a stylesheet in the Kentico CMS administration interface at **Site Manager -> Development -> CSS stylesheets**. These stylesheets can be applied to individual documents (pages) that contain the control in **CMS Desk -> Content -> Edit -> Properties -> General -> CSS stylesheet**.

1.8.5.10 CMSTreeMenu

1.8.5.10.1 Overview

The CMSTreeMenu control allows you to display a multi-level tree menu based on data from Kentico CMS.

It allows you to display part of the CMS menu structure specified using its **Path**, **MaxRelativeLevel**, **ClassNames**, **CultureCode** and **WhereCondition** properties.

The portal engine equivalent of the CMSTreeMenu control is the [Navigation -> Tree menu](#) web part.

The following topics are available to help you familiarize yourself with the CMSTreeMenu control:

- [Getting started](#) - contains a quick step-by-step tutorial that allows you to learn the basics of using the control
- [Configuration](#) - describes and explains the properties that can be set for the control
- [Appearance and styling](#) - describes and shows an example of how CSS styles can be used with the control

1.8.5.10.2 Getting started

The following is a step-by-step tutorial that will show you how to display a tree menu based on CMS content using the CMSTreeMenu control:

1. Create a new **Web form** and use it as a page template according to the guide in the [Using ASPX page templates](#) topic.
2. Switch to its **Design** tab, drag and drop a **CMSTreeMenu** control from the toolbox onto the form.
3. Switch to the **Source** tab. The code of the CMSTreeMenu control should look like this:

```
<cms:CMSTreeMenu ID="CMSTreeMenu1" runat="server" />
```

4. Save the changes to the web form. Now if you look at the page using the created template on the sample Corporate Site, an unstyled tree menu will be displayed.

Continue this tutorial in the [Appearance and styling](#) topic to see how CSS styles can be applied to the CMSTreeMenu control.

1.8.5.10.3 Configuration

The following properties of the CMSTreeMenu control can be set or used in the API:

All of the common properties from:

- [CMS controls - common properties](#)
- [CMS navigation - common properties](#)

In addition, the following properties are available:

Property Name	Description	Sample Value
CellPadding	Cell padding of the table representing the menu.	
CellSpacing	Cell spacing of the table representing the menu.	
CollapseSelectedNodeOnClick	Indicates whether the selected section of the menu should be collapsed when clicked.	
DisplayHighlightedItemAsLink	Indicates whether highlighted items should be displayed as a link.	
EncodeMenuCaption	Indicates whether HTML encoding should be applied to the captions of menu items. Useful only in special cases where the names of the displayed documents contain HTML code.	
GenerateAllSubItems	Indicates whether all sub-items should be generated	
GenerateIndentationInsideLink	Indicates whether indentation spaces should be generated inside hyperlinks (true) or outside (false). This applies only when you do not use images in the menu.	
GenerateOnlyOuterLink	Indicates whether only one outer link should be generated per each menu item.	
HighlightedNodePath	Path of the item that should be highlighted as if it were selected. If you omit this value, the control automatically uses the current alias path from the aliaspath querystring parameter.	"/products"
Indentation	Indentation of menu item levels. Number of	

	spaces that will be placed before each level of menu items.	
ItemIDPrefix	Prefix placed before each item ID. You can use it to keep IDs unique if you have several CMSTreeMenu controls on the same page.	"submenu"
LoadDataAutomaticaly	Indicates whether data for the control should be loaded automatically. By default, the data is loaded automatically. If you set this property to false, you can enter a custom DataSet into the DataSource property and then call the <i>ReloadData(false)</i> method.	
MenuItemImageUrl	URL address of the image that is displayed next to menu items. It may start with "~/" representing the virtual path of the current application.	
MenuItemOpenImageUrl	URL address of the image that is displayed next to open menu items. It may start with "~/" representing the virtual path of the current application.	
OnMouseOutScript	OnMouseOut script for menu items. You can use macro expressions here.	
OnMouseOverScript	OnMouseOver script for menu items. You can use macro expressions here.	
OrderBy	Gets or sets the ORDER BY clause of the SQL statement. Please be aware that it is necessary for the root of the displayed tree (or sub-tree) to be first in the resulting order, otherwise all documents may not be displayed correctly. This can be ensured by having the value of this property start with the NodeLevel column, such as in the sample value.	"NodeLevel, NodeOrder"
RenderedHTML	Allows you to get or set the HTML code rendered by the control. You need to set this property before the Render event - e.g. in the OnLoad event.	
RenderImageAlt	Indicates whether the ALT attribute should be rendered for images used in the menu (for XHTML compatibility).	
RenderLinkTitle	Specifies if the document name should be rendered as a TITLE tag of links (for better accessibility).	

RenderSubItems	Indicates whether sub-items should be rendered under the selected item.	
UrlTarget	Specifies the target frame for all URLs.	"_blank"

Mentioned method:

Method Name	Description
ReloadData(bool forceLoad)	<p>Reloads the data.</p> <p>If the forceLoad parameter is set to false and a value is assigned to the DataSource property, the properties of the CMSTreeMenu control are not used and only the data from the DataSource is loaded.</p>

1.8.5.10.4 Appearance and styling

The appearance of the CMSTreeMenu control is determined by the CSS classes it uses and by some of its properties.

You can use the following CSS classes to modify the design of the control:

Class Name	Description
CMSTreeMenuTable	The main table (TABLE element).
CMSTreeMenuItem	Tree menu item (TD element).
CMSTreeMenuItemAlt	Alternating style of the menu item (TD element). It's used only when you set the UseAlternatingStyles property to true.
CMSTreeMenuSelectedItem	Selected tree menu item (TD element).
CMSTreeMenuLink	Link (A element).
CMSTreeMenuLinkAlt	Alternating style of the link (A element). It's used only when you set the UseAlternatingStyles property to true.
CMSTreeMenuSelectedLink	Link of the selected item (A element).
CMSTreeMenuNestedTable	Nested table (TABLE element). It's used only when CollapseSelectedNodeOnClick is true.

The recommended place to define these classes is in a stylesheet in the Kentico CMS administration interface at **Site Manager -> Development -> CSS stylesheets**. These stylesheets can be applied to individual documents (pages) that contain the control in **CMS Desk -> Content -> Edit -> Properties -> General -> CSS stylesheet**.

Please refer to [Using the CSSPrefix property](#) to learn how to add prefixes to these classes to customize items at any menu level.

Example

The following is an example of how CSS styles can be applied to a CMSTreeMenu control. If you wish to create this example for yourself, please follow the tutorial in the [Getting started](#) topic, then continue with the following steps:

1. Set the following properties of the CMSTreeMenu control:

- **Indentation: 5**

This specifies the indentation used for lower menu levels.

The code of the control should now look like the following:

```
<cms:CMSTreeMenu ID="CMSTreeMenu1" runat="server" Indentation="5" />
```

Save the changes to the web form.

2. Now log in to the sample Corporate Site, go to **Site Manager -> Development -> CSS stylesheets**, edit (✎) the **Corporate Site** stylesheet and add the following classes:

```
/* Tree Menu design */
.CMSTreeMenuTable
{
    width: 100%;
}

.CMSTreeMenuTable a
{
    white-space: nowrap;
}

.subCMSTreeMenuTable
{
    margin-left: 10px;
}

.CMSTreeMenuTable a:hover
{
    text-decoration: underline;
}

.CMSTreeMenuItem
{
    background: #e7e7ff;
    padding: 3px 0px;
}

.CMSTreeMenuSelectedItem
{
    background: #4a3c8c;
    padding: 3px 0px;
    color: white;
}

.CMSTreeMenuLink
```

```
{  
    padding-left: 12px;  
    text-decoration: none;  
    color: #000000;  
}
```

This will modify the appearance of the tree menu. The created page should use this stylesheet by default.

3. Now if you look at the page using the created template on the website, the tree menu will have its appearance modified:



1.8.5.11 CMSTreeView

1.8.5.11.1 Overview

The CMSTreeView control allows you to display a multi-level tree menu based on data from Kentico CMS.

It allows you to display part of the CMS menu structure specified using its **Path**, **MaxRelativeLevel**, **ClassNames**, **CultureCode** and **WhereCondition** properties.

This control is derived from the intrinsic ASP.NET TreeView control and enhances it to automatically read CMS documents and adds a set of additional properties. Please see the ASP.NET documentation for more details on the properties, behavior and design of the TreeView control.

The portal engine equivalent of the CMSTreeView control is the [Navigation -> Tree view](#) web part.

The following topics are available to help you familiarize yourself with the CMSTreeView control:

- [Getting started](#) - contains a quick step-by-step tutorial that allows you to learn the basics of using the control
- [Configuration](#) - describes and explains the properties that can be set for the control
- [Appearance and styling](#) - describes and shows an example of how CSS styles can be used with the control

1.8.5.11.2 Getting started

The following is a step-by-step tutorial that will show you how to display a tree menu based on CMS content using the CMSTreeView control:

1. Create a new **Web form** somewhere in your website installation directory.
2. Switch to its **Design** tab, drag and drop a **CMSTreeView** control from the toolbox onto the form.
3. Switch to the **Source** tab. The code of the CMSTreeView control should look like this:

```
<cms:CMSTreeView ID="CMSTreeView1" runat="server">
</cms:CMSTreeView>
```

4. Save the changes to the web form. Now right-click it in the Solution explorer and select **View in Browser**. The resulting page should display a menu like this:



1.8.5.11.3 Configuration

The following properties of the CMSTreeView control can be set or used in the API:

Most of the common properties from:

- [CMS controls - common properties](#)
- [CMS navigation - common properties](#)

In addition, the following properties are available:

Property Name	Description	Sample Value
DisplayDocumentTypeImages	If true, the images used for items in the tree will be loaded according to the document type of the given item.	
DynamicBehaviour	If true, populate on demand will be enabled. In this case, sub-items will be loaded	

	<p>dynamically when their parent node is expanded.</p> <p>This is recommended if there is a large amount of nodes in the tree, or if the content depends on other dynamic parameters.</p>	
EncodeMenuCaption	<p>Indicates whether HTML encoding should be applied to the captions of menu items. Useful only in special cases where the names of the displayed documents contain HTML code.</p>	
ExpandAllOnStartup	<p>Indicates whether all nodes in the tree should be expanded by default.</p>	
ExpandCurrentPath	<p>Indicates whether all nodes along the path to the currently selected item should be expanded by default.</p>	
ExpandSubTree	<p>Indicates whether the sub-tree under the currently selected item should be expanded by default.</p>	
FixBrokenLines	<p>Indicates whether broken lines should be fixed.</p>	
HideRootNode	<p>Indicates whether the root node should be hidden.</p>	
HighlightSelectedItem	<p>Indicates whether the currently selected item should be highlighted.</p>	
IgnoreDocumentMenuAction	<p>Indicates whether the Menu actions document menu settings should be ignored. These can be set at CMS Desk -> Content -> ... select document ... -> Properties -> Menu -> Menu actions section.</p> <p>This is necessary if you wish to use an onClick Javascript action defined by the OnClickAction property.</p>	
InactivateAllItemsInPath	<p>If enabled, all nodes on the path of the currently selected document will be inactive (will not perform any action when clicked).</p>	
InactivateSelectedItem	<p>If enabled, the node of the currently selected document will be inactive.</p>	
InactiveNodeImage	<p>Indicates if the images displayed next to nodes should also perform the specified action for the given item when clicked.</p>	
InactiveRoot	<p>Indicates whether the root node should be inactive.</p>	

NodelImageUrl	Gets or sets the path to the image that is displayed next to regular nodes in the tree.	~/App_Themes/Site/Images/treeitem.gif
OnClickAction	Gets or sets the JavaScript command that will be executed when a user clicks on an item in the tree. If empty, the items in the tree will serve as links to the corresponding documents.	
OrderBy	Gets or sets the ORDER BY clause of the SQL statement. Please be aware that it is necessary for the root of the displayed tree (or sub-tree) to be first in the resulting order, otherwise all documents may not be displayed correctly. This can be ensured by having the value of this property start with the NodeLevel column, such as in the sample value.	"NodeLevel, NodeOrder"
RootImageUrl	Gets or sets the path to the image that is displayed next to the root node of the tree.	~/App_Themes/Site/Images/root.gif
RootText	Gets or sets the text caption of the root node in the tree. This text is displayed instead of the document name of the item in the root of the tree.	

As this control is inherited from the ASP.NET TreeView control, it also has all of its standard properties.

1.8.5.11.4 Appearance and styling

You can adjust the appearance of the CMSTreeView control by setting the inherited standard properties of the ASP.NET TreeView control. You can find more details on particular properties in the .NET Framework documentation.

The design of the CMSTreeView control can additionally be modified by its following properties and the CSS classes that they specify:

Property Name	Description	Sample Value
InactiveItemClass	CSS class of inactive items.	
InactiveItemStyle	Style of inactive items.	
SelectedItemClass	CSS class of selected items.	
SelectedItemStyle	Style of selected items.	

The recommended place to define these classes is in a stylesheet in the Kentico CMS administration interface at **Site Manager -> Development -> CSS stylesheets**. These stylesheets can be applied to individual documents (pages) that contain the control in **CMS Desk -> Content -> Edit -> Properties -> General -> CSS stylesheet**.

1.8.6 Listings and viewers

1.8.6.1 Overview

The controls in this section provide various ways to display documents and their data from the Kentico CMS database.

Available categories:

- [Standard listings and viewers](#)
- [Listings and viewers with a custom query](#)

1.8.6.2 Standard listings and viewers

1.8.6.2.1 Overview

The controls in this section provide various ways to display document data from the Kentico CMS database. Options include several types of lists, tables, calendars and more.

Most of these controls inherit a corresponding control from the [Basic Listings and viewers](#) section and extend it by integrating a data source for Kentico CMS documents.

These controls also support the use of [Transformations](#).

If you wish to display more complex data structures containing multiple hierarchical levels and different document types, you can do so by nesting the controls within each other. For more information about this and an example, please refer to the [Using nested controls](#) topic. Alternatively, look into using the [CMSUniView](#) control in combination with [hierarchical transformations](#).

Documents in Kentico CMS can be connected to other documents through relationships. Learn more about how the controls in this section can work with relationships in the [Displaying related documents](#) topic.

Available controls:

- [CMSCalendar](#)
- [CMSDataGrid](#)
- [CMSDataList](#)
- [CMSDocumentValue](#)
- [CMSRepeater](#)
- [CMSUniView](#)
- [CMSViewer](#)

1.8.6.2.2 Using nested controls

A nested control is one that is defined within a transformation or template used by another control. When utilized with listing controls, this can be employed to display hierarchical data. The following controls may contain other nested controls:

- [CMSDataList](#)
- [CMSRepeater](#)

**Please note**

The [CMSUniView](#) control provides a way to display hierarchical data without the need for nested controls and using it instead is recommended, as it offers superior performance.

These controls have the **NestedControlsID** property, which ensures that the correct content **Path** is dynamically provided to nested controls.

The controls can be combined as required. Other controls, such as the [CMSDataGrid](#) for example, may be nested into one of the controls above, but cannot contain nested controls themselves.

If you need to dynamically set the properties of a nested control, it is necessary to set its **DelayedLoading** property to *True*.

The same approach can also be used for listing web parts when using the portal engine.

Example: Displaying a nested (hierarchical) repeater/datalist

This tutorial shows how you can use a hierarchical CMSRepeater/CMSDataList combination to display a list of product categories and a preview of products in each category from the products section of the sample Corporate Site:

1. Create a new **Web form** somewhere in your website installation directory.
2. Switch to its **Design** tab, drag and drop a **CMSRepeater** control from the toolbox onto the form and set its following properties:
 - **Path**: /products/%
 - **ClassNames**: cms.menuitem
 - **NestedControlsID**: CMSDataListNested
 - **OrderBy**: NodeOrder

This specifies the **Path** to the products section, tells the control to read pages (menu item documents) and specifies the ID of the nested control that will be used in the next step. It also orders the retrieved pages according to the order that they have in the CMS Desk content tree.

3. Switch to the **Source** tab and add the code marked by the **CMSRepeater template** comments between the `<cms:CMSRepeater>` tags. The overall code of the CMSRepeater control should look like this:

```
<ajaxToolkit:ToolkitScriptManager ID="manScript" runat="server" EnableViewState="false" />

<cms:CMSRepeater ID="CMSRepeater1" runat="server" Path="/products/%" ClassNames="CMS.MenuItem"
NestedControlsID="CMSDataListNested" OrderBy="NodeOrder" >

    <!-- CMSRepeater template
    ----- %>
```

```
<ItemTemplate>
<h1><%# Eval("DocumentName") %></h1>
<p>

<%-- Nested DataList
----- %>

<div class="ProductList" >

<cms:CMSDataList ID="CMSDataListNested" runat="server"
ClassNames="cms.smartphone;cms.laptop;cms.software;cms.ebook;cms.
itservice;cms.paidmembership"
TransformationName="CorporateSite.Transformations.ProductList"
RepeatColumns="6" >
</cms:CMSDataList>

</div>
<%-- Nested DataList
----- %>

</p>
</ItemTemplate>

<%-- CMSRepeater template
----- %>

</cms:CMSRepeater>
```

This defines the template used by the CMSRepeater to display items. As you can see, it contains a nested CMSDataList control that is configured to display all product types in two columns using the specified transformation and its **ID** is the same as the value that was entered into the **NestedControlsID** property of the main CMSRepeater. Please note that its **Path** property is not specified, as it is dynamically supplied by the main CMSRepeater control.

The same result could also be achieved by placing the CMSDataList into the code of a transformation, and assigning that transformation to the CMSRepeater through its **TransformationName** property.

The **ToolkitScriptManager** control included at the top is required by the transformation used to display the product documents. It is only there to ensure that this example is functional by itself and will usually be included on your website's master page.

4. Save the changes to the web form. Now right-click it in the Solution explorer and select **View in Browser**. The resulting page should display a hierarchical list as shown below (the image is from a page that uses the Corporate Site stylesheet, which styles the displayed products appropriately).

Smartphones

			
Apple iPhone 3GS	Apple iPhone 4	BlackBerry Torch 9800	HTC Sensation
\$424.99 	\$759.99 	\$459.99 	\$799.99 

Laptops and Tablets

			
Apple iPad 2	Apple MacBook Pro MC723LL/A	Dell XPS 15z	HP EliteBook 8440p WJ681AW
\$510.99 	\$2199.00 	\$1596.99 	\$1899.00 

1.8.6.2.3 Displaying related documents

If a Kentico CMS document is related to some other documents, these can be displayed using one of the following controls:

- [CMSCalendar](#)
- [CMSDataGrid](#)
- [CMSDataList](#)
- [CMSRepeater](#)
- [CMSUniView](#)
- [CMSViewer](#)

All of them have the following three properties that can be set in order to display related documents (besides other properties, such as **Path**, **ClassNames**, etc.):

Property Name	Description	Sample Value
---------------	-------------	--------------

RelatedNodesOnTheLeftSide	Indicates whether the related document is on the left or right side of the relationship.	
RelationshipName	Code name of the relationship.	"isrelatedto"
RelationshipWithNodeGUID	If set, only documents related to the document with the entered NodeGUID value will be selected. Entering the "11111111-1111-1111-1111-111111111111" value will select documents related to the current document.	"36f8c4bc-f702-4736-8a25-a82295668794"

More information about related documents can be found at [Developer's Guide -> Content management -> Document properties -> Related docs.](#)

Example

The following example shows how to create an ASPX page template that displays news items related to the current page by using the CMSRepeater control:

1. Now open the Kentico CMS web project in Visual Studio and create a new **Web form** called *RelationshipExample.aspx* in the **CMSTemplates/CorporateSite** folder and check the **Select master page** box. When the **Select a Master Page** dialog appears, choose the **root.master** default master page from the **CMSTemplates/CorporateSite** folder.

2. Switch to the **Source** view of the newly created web form and add the following line under the `<%@ Page %>` directive:

```
<%@ Register Assembly="CMS.Controls" Namespace="CMS.Controls" TagPrefix="cms" %>
```

3. Switch to the code behind. You need to add a reference to the **CMS.UIControls** namespace:

[C#]

```
using CMS.UIControls;
```

4. Modify the class from which the page is inherited. Change the following code:

[C#]

```
public partial class CMSTemplates_CorporateSiteAspx_RelationshipExample : System.Web.UI.Page
```

to this:

[C#]

```
public partial class CMSTemplates_CorporateSiteAspx_RelationshipExample :
    TemplatePage
```

Now the page can be correctly used as a page template in Kentico CMS.

Please keep in mind that the name of the class must be identical to the value of the **Inherits** attribute of the `<%@ Page %>` directive on the ASPX page. This is case sensitive.

5. Switch to its **Design** tab, drag and drop a **CMSRepeater** control from the toolbox onto the form and set its following properties:

- **ClassNames:** CMS.News
- **Path:** /News/%
- **TransformationName:** cms.news.preview
- **RelationshipName:** isrelatedto
- **RelationShipWithNodeGUID:** 11111111-1111-1111-1111-111111111111

This tells the control to read news documents, specifies the path to the News section of the sample Corporate Site, assigns the transformation that should be used to display the news documents and tells the control to display only documents that are in the *isrelatedto* relationship with the **currently** selected document.

The overall code of the CMSRepeater control will look like this:

```
<cms:CMSRepeater ID="CMSRepeater1" runat="server" ClassNames="CMS.News" Path="/
News/%"
TransformationName="cms.news.preview" RelationshipName="isrelatedto"
RelationshipWithNodeGUID="11111111-1111-1111-1111-111111111111" >
</cms:CMSRepeater>
```

6. Save the changes to the web form.

7. Now open the sample Corporate Site, go to **Site Manager -> Development -> Page templates**, select the **Corporate Site/Examples** category and create a new template. Enter the following properties:

- **Template display name:** Related news
- **Template code name:** RelatedNews

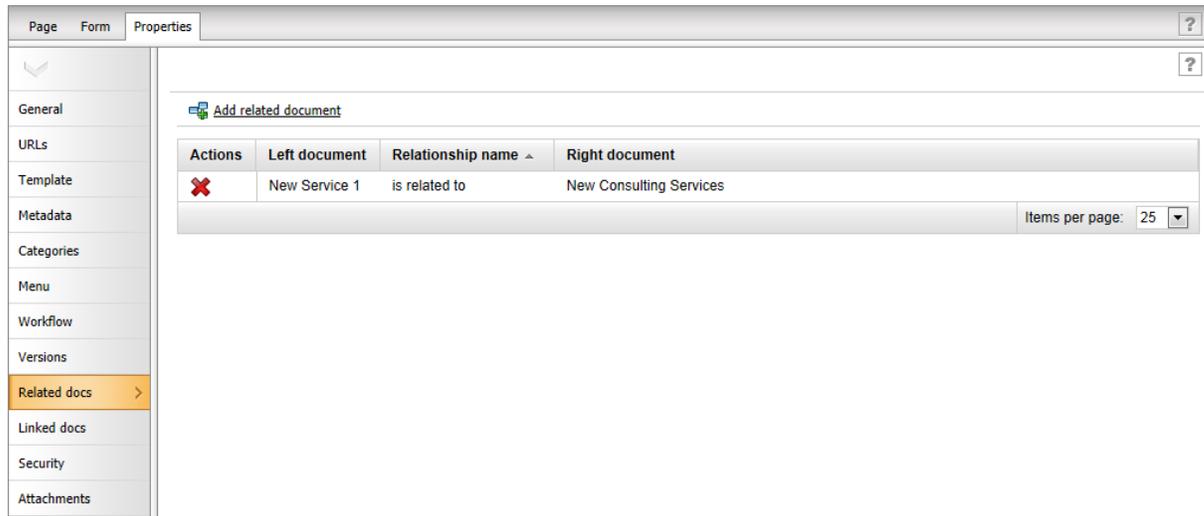
Click **OK**. Press the **Select** button next to the **File name** field and choose the **RelationshipExample.aspx** web form from the **CMSTemplates/CorporateSite** folder.

Click  **Save**.

8. Switch to the **Sites** tab, press **Add sites** and select **Corporate Site**.

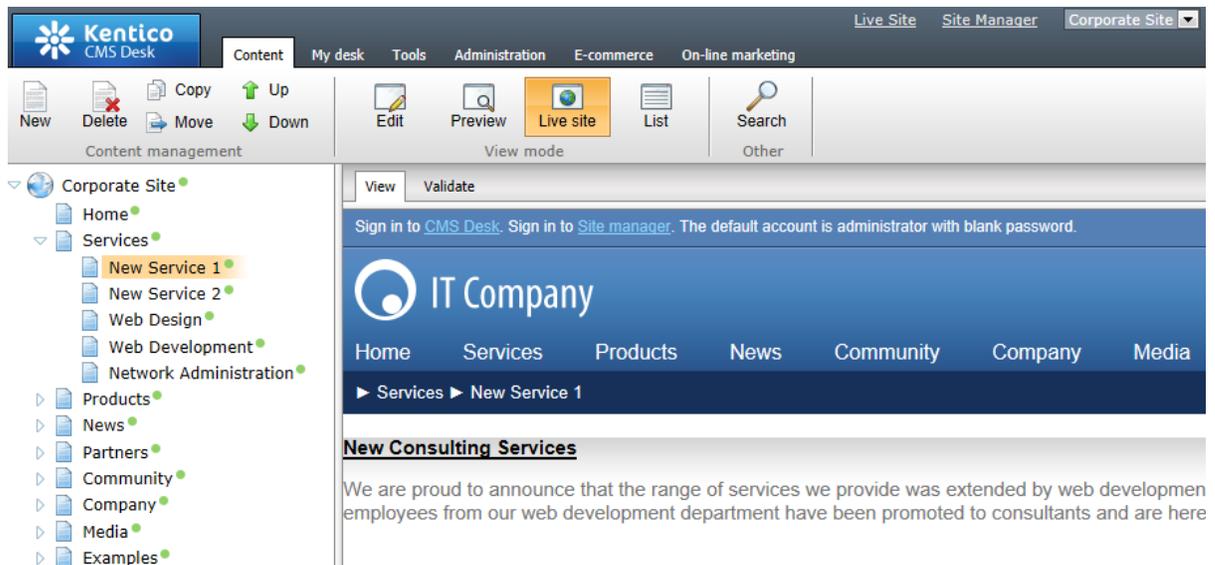
9. Switch to **CMS Desk**, select **Services** from the content tree and create a new **Page (menu item)** called under it, enter *New Service 1* into the **Page name** field and select the **Corporate Site -> Examples -> Related news** page template. Click  **Save**. Now switch to **Properties -> Related docs** tab of the new page and add a related document by using  **Add related document**, choose *is*

related to as the **Relationship name** and select the *News -> New Consulting Services* document as the **Right-side document**.



10. Repeat step 9, but call the new page *New Service 2* and select the *News -> Apple iPad2 In Stock* document as the **Right-side document** of the relationship.

11. Now select the **/Services/New Service 1** page from the content tree and switch to **Live site** mode. You will see that the CMSRepeater on the page template is displaying a preview of the news item related to this page:



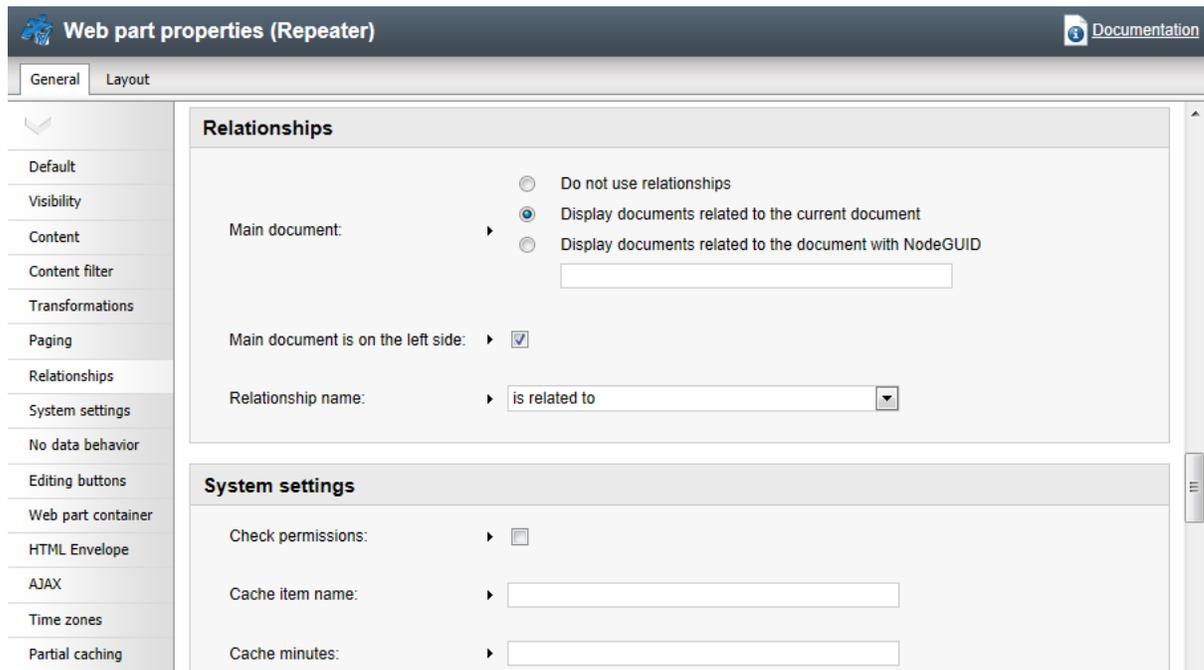
If you select **/Services/New Service 2** page, you will see the **Apple iPad2 In Stock** news item displayed in the same fashion.

Using web parts to display related documents

When using the portal engine, related documents can also be displayed by the following web parts from the **Listings and viewers** category:

- Calendar
- DataList
- Grid
- Related documents
- Repeater
- Universal document viewer
- Universal viewer
- XSLT viewer

This can be done by setting the properties in their **Relationships** property section as seen in the following image:



Example

The following example shows how to display news items related to a product by using the Repeater web part:

1. Open the sample Corporate Site, go to **CMS Desk -> Content**, select **Products -> Laptops and Tablets -> Apple iPad 2** and switch to **Properties -> Related docs**. Now add a related document by using **Add related document**, choose *is related to* as the **Relationship name** and select the *News -> Apple iPad2 In Stock* document as the **Right-side document**.
2. Switch to the **Design** tab and add (+) a **Listings and viewers -> Documents -> Repeater** web part to the **Main zone** web part zone.
3. The **Web part properties (Repeater)** dialog will appear. Set the following properties:
 - **Content -> Path:** /% (we want to display related news items from the whole website)
 - **Content filter -> Document types:** CMS.News
 - **Transformations -> Transformation:** CMS.News.Preview

- **Relationships -> Main document:** Select **Display documents related to the current document**
- **Relationships -> Main document is on the left side:** check the checkbox
- **Relationships -> Relationship name:** is related to
- **HTML Envelope -> Content before:** <h3>Related news:</h3>

Click **OK** to add the Repeater web part to the page and then drag it above the Image gallery web part.

4. Now switch to **Live site** mode. You will see the news item that was added as a related document in step 1 displayed below the product using the CMS.News.Preview transformation:

Processor type:	1GHz Dual-Core Apple A5
Display type:	LED
Display resolution:	1024x768
Graphics card:	PowerVR SGX543MP2
Memory size:	512
Memory type:	DDR2
Bluetooth:	Yes
WiFi:	Yes
Infraport:	No
Battery type:	25 Wh (90 kJ) Lithium-Ion Polymer
Operating system:	Mac OS

Related news:

Apple iPad 2 In Stock

Today, we have good news for all fans of the awesome Apple iPad. We are glad to announce that its new version, Apple iPad 2, is available in our web shop. Furthermore, we keep our reasonable pricing policy, providing the lowest price currently available on the Web.

Image gallery

1.8.6.2.4 CMSCalendar

1.8.6.2.4.1 Overview

The CMSCalendar control allows you to display a calendar with events, news and other date-based documents from the Kentico CMS database without the need to write any additional code.

The content is specified using its **Path**, **MaxRelativeLevel**, **ClassNames**, **CultureCode**, **WhereCondition** and **OrderBy** properties. Data is retrieved using the **SelectDocuments** query of the specified document type. These queries can be found at **Kentico CMS Site Manager -> Development -> Document types -> ... Edit specified document type ... -> Queries**.

The CMSCalendar is derived from the [BasicCalendar](#) control.

The portal engine equivalent of the CMSCalendar control is the [Listings and viewers -> Documents -> Calendar](#) web part.

The following topics are available to help you familiarize yourself with the CMSCalendar control:

- [Getting started](#) - contains a quick step-by-step tutorial that allows you to learn the basics of using the control
- [Configuration](#) - describes and explains the properties that can be set for the control
- [Appearance and styling](#) - describes how the design of the control can be modified

1.8.6.2.4.2 Getting started

The following is a step-by-step tutorial that will show you how to display a calendar that contains links to news items (CMS.News documents) on days when news items were released using the CMSCalendar control:

1. Create a new **Web form** somewhere in your website installation directory.
2. Switch to its **Design** tab, drag and drop a **CMSCalendar** control from the toolbox onto the form and set its following properties:

- **ClassNames:** cms.news
- **DayField:** NewsReleaseDate
- **TransformationName:** cms.news.calendarevent
- **NoEventTransformationName:** cms.news.calendarnoevent

This tells the control to read news documents, assigns the column it should get date/time values from and specifies the transformations that should be used to display days with and without news releases.

3. Switch to the **Source** tab. The code of the CMSCalendar control should look like this:

```
<cms:CMSCalendar ID="CMSCalendar1" runat="server" ClassNames="cms.news" DayField
="NewsReleaseDate"
TransformationName="cms.news.calendarevent" NoEventTransformationName="cms.news.
calendarnoevent" >
</cms:CMSCalendar>
```

There is no need to define templates for day items, since the transformation names have already been specified.

4. Save the changes to the web form. Now right-click it in the Solution explorer and select **View in Browser**. The resulting page should display a calendar like this:

June 2011						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
29 No event	30 No event	31 No event	1 No event	2 No event	3 No event	4 No event
5 New Consulting Services (6/5/2011 12:00:00 AM) <i>We are proud to announce that the range of services we provide was extended by web development consulting. The most experienced and skilled employees from our web development department have been promoted to consultants and are here to help you with your web development projects.</i>	6 No event	7 No event	8 No event	9 Apple iPad 2 In Stock (6/9/2011 12:00:00 AM) <i>Today, we have good news for all fans of the awesome Apple iPad. We are glad to announce that its new version, Apple iPad 2, is available in our web shop. Furthermore, we keep our reasonable pricing policy, providing the lowest price currently available on the Web.</i>	10 No event	11 No event

1.8.6.2.4.3 Configuration

As it is inherited from the [BasicCalendar](#) control, the CMSCalendar control has all of its properties (including templates). These can be found in the [BasicCalendar -> Configuration](#) topic. In addition, it has the following properties that can be set or used in the API:

In addition, it has most of the following properties that can be set or used in the API:

- [CMS controls - common properties](#)
- [Displaying related documents](#)

As well as:

Property Name	Description	Sample Value
DataSource	Gets or sets a DataSet containing calendar events used to populate the items within the control. This value is not required.	
NoEventTransformation Name	Name of the transformation applied to days without any events in format <i><document type code name>.<transformation name></i> .	"cms.news.CalendarNoEvent"
TransformationName	Name of the transformation applied to days with an event in format <i><document type code name>.<transformation name></i> .	"cms.news.CalendarEvent"

1.8.6.2.4.4 Appearance and styling

You can modify the appearance of the CMSCalendar control by setting the standard properties of the ASP.NET Calendar control (inherited through the BasicCalendar). You can find more details on particular properties in the .NET Framework documentation for the [Calendar](#) class.

A common way to set the appearance of this control is to assign a skin through the **SkinID** property. Skins can be defined in .skin files under individual themes in the **App_Themes** folder. More information can be found in the .NET [Skins and Themes](#) documentation.

The design of day cells can be determined by the transformations specified by its transformation name properties or by the code of the template properties inherited from the [BasicCalendar](#) control.

1.8.6.2.5 CMSDataGrid

1.8.6.2.5.1 Overview

The CMSDataGrid control allows you to display document data from the Kentico CMS database in a customizable table without the need to write any additional code.

The content is specified using its **Path**, **MaxRelativeLevel**, **ClassNames**, **CultureCode**, **WhereCondition** and **OrderBy** properties. Data is retrieved using the **SelectDocuments** query of the specified document type. These queries can be found at **Kentico CMS Site Manager -> Development -> Document types -> ... Edit specified document type ... -> Queries**.

	<p>Please note</p> <p>If you wish to display data using a custom query, please use the QueryDataGrid control.</p>
---	---

The CMSDataGrid is derived from the [BasicDataGrid](#) control.

The standard DataGrid designer can be used to set up CMSDataGrid style and behaviour.

The portal engine equivalent of the CMSDataGrid control is the [Listings and viewers -> Documents -> Grid](#) web part.

The following topics are available to help you familiarize yourself with the CMSDataGrid control:

- [Getting started](#) - contains a step-by-step tutorial that allows you to learn the basics of using the control
- [Configuration](#) - describes and explains the properties that can be set for the control
- [Appearance and styling](#) - describes how the design of the control can be modified

1.8.6.2.5.2 Getting started

The following is a step-by-step tutorial that will show you how to display a table that contains all products (CMS.Smartphone, CMS.Laptop, CMS.Software, CMS.Ebook, CMS.ITService and CMS.PaidMembership documents) from the products section of the sample Corporate Site using the CMSDataGrid control:

1. Create a new **Web form** somewhere in your website installation directory.
2. Switch to its **Design** tab, drag and drop a **CMSDataGrid** control from the toolbox onto the form and set its following properties:

- **ClassNames:** cms.smartphone;cms.laptop;cms.software;cms.ebook;cms.itservice;cms.paidmembership
- **Path:** /products/%

This tells the control which document types to read and specifies the **Path** to the products section (the default setting of /% would also display all products, but it is more effective to read only a section of the site).

3. Right-click the CMSDataGrid on the form, select **AutoFormat...** and choose a scheme.
4. Then right-click the control again, select **Show Smart Tag** and then **Property Builder...**; the **CMSDataGrid1 Properties** dialog will be displayed.

On the **General** tab, check the **Allow sorting** box.

Now switch to the **Columns** tab, where you can specify the columns that will be displayed, and uncheck the **Create columns automatically at run time** box.

Add a new **Bound Column** from the **Available columns** list to the **Selected columns** list. Enter the following values into the appropriate fields:

- **Header text:** Name
- **Data Field:** SKUName
- **Sort expression:** SKUName

Add another **Bound column** from the **Available columns** list to the **Selected columns** list. Enter the following values in the appropriate fields:

- **Header text:** Price
- **Data Field:** SKUPrice
- **Sort expression:** SKUPrice

Click **OK**.

4. Save the changes to the web form. Now right-click it in the Solution explorer and select **View in Browser**. The resulting page should display a table like this (depending on the chosen scheme):

<u>Name</u>	<u>Price ▼</u>
Apple MacBook Pro MC723LL/A	2199
HP EliteBook 8440p WJ681AW	1899
Dell XPS 15z	1596.99
HTC Sensation	799.99
Apple iPhone 4	759.99
Samsung Google Nexus S	599.99
Motorola Atrix 4G	529.98
Apple iPad 2	510.99
Microsoft Office Professional 2010	499.99
BlackBerry Torch 9800	459.99
Apple iPhone 3GS	424.99
Microsoft Windows 7 Ultimate	319.99
1 Hour of Web Development Consulting	150
Gold partner membership for 1 year	100
Silver partner membership for 1 year	60
Operating System Installation	49.99
Kentico CMS Information Sources	1

1.8.6.2.5.3 Configuration

As it is inherited from the [BasicDataGrid](#) control, the CMSDataGrid control has all of its properties. These can be found in the [BasicDataGrid -> Configuration](#) topic.

In addition, it has all of the following properties that can be set or used in the API:

- [CMS controls - common properties](#)
- [Displaying related documents](#)

As well as:

Property Name	Description	Sample Value
---------------	-------------	--------------

PageSize	The number of displayed items per page.	
SelectedItemTransformationName	Name of the transformation applied to the selected item in format <i><document type code name>.<transformation name></i> .	

1.8.6.2.5.4 Appearance and styling

You can modify the appearance of the CMSDataGrid control by setting the standard properties of the ASP.NET DataGrid control (inherited through the BasicDataGrid). You can find more details on particular properties in the .NET Framework documentation for the [DataGrid](#) class.

A common way to set the appearance of this control is to assign a skin through the **SkinID** property. Skins can be defined in .skin files under individual themes in the **App_Themes** folder. More information can be found in the .NET [Skins and Themes](#) documentation.

1.8.6.2.6 CMSDataList

1.8.6.2.6.1 Overview

The CMSDataList control allows you to display document data from the Kentico CMS database in a list based on transformations without the need to write any additional code.

The content is specified using its **Path**, **MaxRelativeLevel**, **ClassNames**, **CultureCode**, **WhereCondition** and **OrderBy** properties. Data is retrieved using the **SelectDocuments** query of the specified document type. These queries can be found at **Kentico CMS Site Manager -> Development -> Document types -> ... Edit specified document type ... -> Queries**.

	<p>Please note</p> <p>If you wish to display data using a custom query, please use the QueryDataList control.</p>
---	---

The CMSDataList is derived from the [BasicDataList](#) control.

Unlike the [CMSRepeater](#) control, the CMSDataList allows you to display data in several columns.

It supports nested controls, read more in the [Using nested controls](#) topic.

The portal engine equivalent of the CMSDataList control is the [Listings and viewers -> Documents -> Datalist](#) web part.

The following topics are available to help you familiarize yourself with the CMSDataList control:

- [Getting started](#) - contains a step-by-step tutorial that allows you to learn the basics of using the control
- [Configuration](#) - describes and explains the properties that can be set for the control
- [Appearance and styling](#) - describes how the design of the control can be modified

1.8.6.2.6.2 Getting started

The following is a step-by-step tutorial that will show you how to display a list that contains all smartphones, laptops and tablets (CMS.Smartphone and CMS.Laptop documents) from the products section of the sample Corporate Site using the CMSDataList control:

1. Create a new **Web form** somewhere in your website installation directory.
2. Switch to its **Design** tab, drag and drop a **CMSDataList** control from the toolbox onto the form and set its following properties:

- **ClassNames:** cms.smartphone;cms.laptop
- **OrderBy:** SKUName
- **Path:** /products/%
- **RepeatColumns:** 4
- **TransformationName:** ecommerce.transformations.Product_SimplePreview
- **SelectedItemTransformationName:** ecommerce.transformations.Product_Default

This tells the control which document types to read, sets the **OrderBy** value, specifies the **Path** to the products section (the default setting of /% would also display all products, but it is more effective to read only a section of the site), determines the amount of displayed columns and assigns the transformations that should be used to display products.

3. Switch to the **Source** tab. The code of the CMSDataList control should look like this:

```
<cms:CMSDataList ID="CMSDataList1" runat="server" ClassNames="cms.smartphone;cms.laptop"
  OrderBy="SKUName" Path="/products/%" RepeatColumns="4"
  TransformationName="ecommerce.transformations.Product_SimplePreview"
  SelectedItemTransformationName="ecommerce.transformations.Product_Default">
</cms:CMSDataList>
```

It's not necessary to define the standard **ItemTemplate** elements of the DataList control since the transformation names have already been specified.

4. Save the changes to the web form. Now right-click it in the Solution explorer and select **View in Browser**. The resulting page should display a list like this:



Apple iPad 2
Our price: \$510.99



Apple MacBook Pro MC723LL/A
Our price: \$2199.00



HP EliteBook 8440p WJ681AW
Our price: \$1899.00



Motorola Atrix 4G
Our price: \$529.98



Apple iPhone 3GS
Our price: \$424.99



BlackBerry Torch 9800
Our price: \$459.99



HTC Sensation
Our price: \$799.99



Samsung Google Nexus S
Our price: \$599.99



Apple iPhone 4
Our price: \$759.99



Dell XPS 15z
Our price: \$1596.99

1.8.6.2.6.3 Configuration

As it is inherited from the [BasicDataList](#) control, the CMSDataList control has all of its properties. These can be found in the [BasicDataList -> Configuration](#) topic.

In addition, it has all of the following properties that can be set or used in the API:

- [CMS controls - common properties](#)
- [Displaying related documents](#)

As well as:

Property Name	Description	Sample Value
AlternatingTransformationName	Name of the transformation applied to alternating items in format <i><document type code name>.<transformation name></i> .	
DelayedLoading	Indicates whether data should be loaded during the load event instead of the default init event.	
EnablePaging	Indicates whether the built-in DataPager control should be used to page the list. This property does not affect the UniPager control, which must be added separately if you wish to use it.	

NestedControlsID	IDs of nested controls (CMSRepeater, CMSDataList...), separated by semicolons.	"CMSRepeaterNested; CMSDataListNested"
PageSize	The number of displayed items per page.	
PagerControl	This property can be used to set or get the pager control and its properties.	
SelectedItemTransformationName	Name of the transformation applied to the selected item in format <i><document type code name>.<transformation name></i> .	
ShowEditDeleteButtons	Indicates if edit and delete buttons should automatically be shown for each displayed item in the edit mode of CMS Desk.	
TransformationName	Name of the transformation applied to standard items in format <i><document type code name>.<transformation name></i> .	

1.8.6.2.6.4 Appearance and styling

You can modify the appearance of the CMSDataList control by setting the standard properties of the ASP.NET DataList control (inherited through the BasicDataList). You can find more details on particular properties in the .NET Framework documentation for the [DataList](#) class.

The design of list items can be determined by the transformations specified by the **AlternatingTransformationName**, **TransformationName** and **SelectedItemTransformationName** properties or by the code of the template properties inherited from the standard ASP.NET DataList control.

1.8.6.2.7 CMSDocumentValue

1.8.6.2.7.1 Overview

This control allows you to display a specified value of the currently displayed Kentico CMS document. It can be useful if you need to display e.g. the current document name on the page.

This control can easily be placed into [ASPX page templates](#), [page layouts](#) or [transformation](#) code.

The following topics are available to help you familiarize yourself with the CMSDocumentValue control:

- [Getting started](#) - contains a quick example of how this control can be used
- [Configuration](#) - describes and explains the properties that can be set for the control

1.8.6.2.7.2 Getting started

The following tutorial will show you how to display the document name of the currently selected document using the CMSDocumentValue control:

1. Create a new **Web form** and use it as a page template according to the guide in the [Using ASPX page templates](#) topic.

2. Switch to its **Design** tab, drag and drop a **CMSDocumentValue** control from the toolbox onto the form and set its following properties:

- **AttributeName**: DocumentName
- **FormattingString**: Document name: {0}

This tells the control which document value to display and sets the format that should be used.

The code of the control will look like this:

```
<cms:CMSDocumentValue ID="CMSDocumentValue1" runat="server" AttributeName
="DocumentName" FormattingString="Document name: {0}" />
```

3. Save the changes to the web form. Now if you look at the page using the created template on some website, the name of the currently selected document will be displayed. This is only an example of how this control can be used and by itself isn't very useful. In a practical scenario, the code of the control from this example would be added to an existing ASPX page template, that has some other function, or to the code of a [page layout](#) when using the portal engine.

1.8.6.2.7.3 Configuration

The following properties of the CMSDocumentValue control can be set or used in the API:

Property Name	Description	Sample Value
AttributeName	Name of the field to be displayed.	"DocumentName"
ClassNames	List of document types for which the value should be displayed, separated by a semicolon (;).	"cms.article;cms.menuitem"
FormattingString	.NET formatting expression used for displaying the value.	"Name: {0}"
StopProcessing	Indicates if processing of the control should be stopped and the control should not retrieve or display any data.	

1.8.6.2.8 CMSRepeater

1.8.6.2.8.1 Overview

The CMSRepeater control allows you to display document data from the Kentico CMS database in a list based on transformations without the need to write any additional code.

The content is specified using its **Path**, **MaxRelativeLevel**, **ClassNames**, **CultureCode**, **WhereCondition** and **OrderBy** properties. Data is retrieved using the **SelectDocuments** query of the specified document type. These queries can be found at **Kentico CMS Site Manager -> Development -> Document types -> ... Edit specified document type ... -> Queries**.

Please note



If you wish to display data using a **custom query**, please use the [QueryRepeater](#) control.

The CMSRepeater is derived from the [BasicRepeater](#) control.

It supports nested controls, read more in the [Using nested controls](#) topic.

The portal engine equivalent of the CMSRepeater control is the [Listings and viewers -> Documents -> Repeater](#) web part.

The following topics are available to help you familiarize yourself with the CMSRepeater control:

- [Getting started](#) - contains a quick step-by-step tutorial that allows you to learn the basics of using the control
- [Configuration](#) - describes and explains the properties that can be set for the control
- [Appearance and styling](#) - describes how the design of the control can be modified

1.8.6.2.8.2 Getting started

The following is a step-by-step tutorial that will show you how to display a list of news items (CMS.News documents) using the CMSRepeater control:

1. Create a new **Web form** somewhere in your website installation directory.
2. Switch to its **Design** tab, drag and drop a **CMSRepeater** control from the toolbox onto the form and set its following properties:

- **ClassNames:** cms.news
- **OrderBy:** NewsReleaseDate DESC
- **TransformationName:** cms.news.preview
- **SelectedItemTransformationName:** cms.news.default

This tells the control which document types to read, sets the **OrderBy** value and assigns the transformations that should be used to display the news items.

3. Switch to the **Source** tab. The code of the CMSRepeater control should look like this:

```
<cms:CMSRepeater ID="CMSRepeater1" runat="server" ClassNames="cms.news"
OrderBy="NewsReleaseDate DESC" TransformationName="cms.news.preview"
SelectedItemTransformationName="cms.news.default">
</cms:CMSRepeater>
```

It's not necessary to define the standard **ItemTemplate** elements of the Repeater control since the transformation names have already been specified.

4. Save the changes to the web form. Now right-click it in the Solution explorer and select **View in Browser**. The resulting page should display a list like this:

[Community Website Section](#)

As a result of our continuous effort to improve our services, we have recently introduced the Community Website Section. We encourage our customers to provide feedback about the company from various communication channels and express your opinions and thoughts.

[Company Growth Exceeds Expectations](#)

Our company growth has reached astonishing 256% in the last financial year. It is not only thanks to our products but also to our loyal customers. Therefore, we would like to thank you for your loyalty and state a promise that we will continue to improve our services.

[Apple iPad 2 In Stock](#)

Today, we have good news for all fans of the awesome Apple iPad. We are glad to announce that it is now available on the Web. We are offering a special pricing policy, providing the lowest price currently available on the Web.

[New Consulting Services](#)

We are proud to announce that the range of services we provide was extended by web development services. Our consultants have been promoted to consultants and are here to help you with your web development projects.

1.8.6.2.8.3 Configuration

As it is inherited from the [BasicRepeater](#) control, the CMSRepeater control has all of its properties. These can be found in the [BasicRepeater -> Configuration](#) topic.

In addition, it has all of the following properties that can be set or used in the API:

- [CMS controls - common properties](#)
- [Displaying related documents](#)

As well as:

Property Name	Description	Sample Value
AlternatingTransformationName	Name of the transformation applied to alternating items in format <i><document type code name>.<transformation name></i> .	
DataSourceControl	Object of the data source control.	
DataSourceName	ID of the data source control.	
DelayedLoading	Indicates whether data should be loaded during the load event instead of the default init event.	
EnablePaging	Indicates whether the built-in DataPager control should be used to page the list. This property does not affect the UniPager control, which must be added separately if you wish to use it.	
ItemSeparator	Separator between displayed items.	"<hr/>"
NestedControlsID	IDs of nested controls (CMSRepeater, CMSDataList...), separated by semicolons.	"CMSRepeaterNested; CMSDataListNested"
PageSize	The number of displayed items per page.	

PagerControl	This property can be used to set or get the pager control and its properties.	
SelectedItemTransformationName	Name of the transformation applied to the selected item in format <i><document type code name>.<transformation name></i> .	
ShowEditDeleteButtons	Indicates if edit and delete buttons should automatically be shown for each displayed item in the edit mode of CMS Desk.	
TransformationName	Name of the transformation applied to standard items in format <i><document type code name>.<transformation name></i> .	

1.8.6.2.8.4 Appearance and styling

You can modify the appearance of the CMSRepeater control by setting the standard properties of the ASP.NET Repeater control (inherited through the BasicRepeater). You can find more details on particular properties in the .NET Framework documentation for the [Repeater](#) class.

The design of list items can be determined by the transformations specified by the **AlternatingTransformationName**, **TransformationName** and **SelectedItemTransformationName** properties or by the code of the template properties inherited from the standard ASP.NET Repeater control.

1.8.6.2.9 CMSUniView

1.8.6.2.9.1 Overview

The CMSUniView control allows you to display document data from the Kentico CMS database in a hierarchical structure based on transformations without the need to write any additional code. Using hierarchical transformations, this control can be used to display documents with a varying design depending on their type and hierarchical level.

The content is specified using its **Path**, **MaxRelativeLevel**, **ClassNames**, **CultureCode**, **WhereCondition** and **OrderBy** properties. Data is retrieved using the **SelectDocuments** query of the specified document type. These queries can be found at **Kentico CMS Site Manager -> Development -> Document types -> ... Edit specified document type ... -> Queries**.

The CMSUniView is derived from the [BasicUniView](#) control.

It supports nested controls, read more in the [Using nested controls](#) topic.

The portal engine equivalent of the CMSUniView control is the [Listings and viewers -> Documents -> Universal viewer](#) web part.

The following topics are available to help you familiarize yourself with the CMSUniView control:

- [Getting started](#) - contains a step-by-step tutorial that allows you to learn the basics of using the control
- [Configuration](#) - describes and explains the properties that can be set for the control
- [Using hierarchical transformations](#) - contains a quick step-by-step tutorial demonstrating how

hierarchical transformations can be used by this control to display several different document types.

- [Appearance and styling](#) - describes how the design of the control can be modified

1.8.6.2.9.2 Getting started

The following is a step-by-step tutorial that will show you how to display all documents from the sample Corporate Site in a hierarchical structure using the CMSUniView control:

1. Create a new **Web form** somewhere in your website installation directory.
2. Switch to its **Design** tab, drag and drop a **CMSUniView** control from the toolbox onto the form and set its **LoadHierarchicalData** property to *true*.
3. Switch to the **Source** tab and add the code marked by the **CMSUniView templates** comments between the `<cms:CMSUniView>` tags. The overall code of the CMSUniView control should look like this:

```
<cms:CMSUniView ID="CMSUniView1" runat="server" LoadHierarchicalData="True">

    <!-- CMSUniView templates
    ----- %>

    <ItemTemplate>
        <%# HTMLHelper.Encode(Convert.ToString(Eval("NodeAliasPath"))) %>
    </ItemTemplate>
    <AlternatingItemTemplate>
        <font color="#999999"><%# HTMLHelper.Encode(Convert.ToString(Eval(
        "NodeAliasPath"))) %>
        </font>
    </AlternatingItemTemplate>
    <SeparatorTemplate>
        </li>
        <li>
    </SeparatorTemplate>
    <HeaderTemplate>
        <ul>
        <li>
    </HeaderTemplate>
    <FooterTemplate>
        </li>
        </ul>
    </FooterTemplate>
    <FirstItemTemplate>
        <font color="Blue"><%# HTMLHelper.Encode(Convert.ToString(Eval(
        "NodeAliasPath"))) %>
        </font>
    </FirstItemTemplate>
    <LastItemTemplate>
        <font color="Blue"><%# HTMLHelper.Encode(Convert.ToString(Eval(
        "NodeAliasPath"))) %>
        </font>
    </LastItemTemplate>

    <!-- CMSUniView templates
    ----- %>
```

```
</cms:CMSUniView>
```

As you can see, this example uses **ItemTemplates** to determine the design of the displayed documents. An alternative way of doing this is to set the **TransformationName** properties of the CMSUniView, which causes the control to load the transformations from the database. A list of these properties can be found in the [Configuration](#) topic.

4. Save the changes to the web form. Now right-click it in the Solution explorer and select **View in Browser**. The resulting page should display a hierarchical list like this:

- /
- /Home
- /Services
 - /Services/Web-design
 - /Services/Web-development
 - /Services/Network-administration
- /Products
 - /Products/Smartphones
 - /Products/Smartphones/Apple-iPhone-3GS
 - /Products/Smartphones/Apple-iPhone-4
 - /Products/Smartphones/BlackBerry-Torch-9800
 - /Products/Smartphones/HTC-Sensation
 - /Products/Smartphones/Motorola-Atrix-4G
 - /Products/Smartphones/Samsung-Google-Nexus-S
 - /Products/Laptops-and-Tablets
 - /Products/Laptops-and-Tablets/Apple-iPad-2
 - /Products/Laptops-and-Tablets/Apple-MacBook-Pro-MC723LL-A
 - /Products/Laptops-and-Tablets/Dell-XPS-15z
 - /Products/Laptops-and-Tablets/HP-EliteBook-WJ681AW
 - /Products/Laptops-and-Tablets/Dell-XPS-15x-with-iPhone
 - /Products/Software
 - /Products/Software/Test-software
 - /Products/Software/Microsoft-Windows-7-Ultimate
 - /Products/Software/Microsoft-Windows-7-installation-package
 - /Products/E-Books
 - /Products/E-Books/Kentico-CMS-Information-Sources
 - /Products/IT-Services
 - /Products/IT-Services/Test-service
 - /Products/IT-Services/Operating-System-Installation
 - /Products/Membership
 - /Products/Membership/Silver-partner-membership-for-1-year
 - /Products/Membership/Gold-partner-membership-for-1-year
- /News
 - /News/New-Consulting-Services
 - /News/Apple-iPad-2-In-Stock
 - /News/Company-Growth-Exceeds-Expectations
 - /News/Community-Website-Section

1.8.6.2.9.3 Configuration

As it is inherited from the [BasicUniView](#) control, which is in turn derived from the [UniView](#) control, the CMSUniView control has all of their properties. These can be found in the [BasicUniView -> Configuration](#) and [UniView -> Configuration](#) topics.

In addition, it has all of the following properties that can be set or used in the API:

- [CMS controls - common properties](#)

- [Displaying related documents](#)

As well as:

Property Name	Description	Sample Value
DataSourceControl	Object of the data source control.	
DataSourceName	ID of the data source control.	
DelayedLoading	Indicates whether data should be loaded during the load event instead of the default init event.	
EnablePaging	Indicates whether the built-in UniPager control should be used to for paging. If you wish to use paging, please be sure to set LoadHierarchicalData to false, as hierarchical data cannot be paged.	
ItemSeparatorValue	Separator between displayed items. This property has greater priority than the separator set in the transformation given in the SeparatorTransformationName property.	"<hr/>"
LoadHierarchicalData	Indicates whether data should be bound in the default format (flat structure) or changed to a hierarchical grouped dataset.	
NestedControlsID	IDs of nested controls (CMSRepeater, CMSDataList...), separated by semicolons.	"CMSRepeaterNested; CMSDataListNested"
PageSize	The number of displayed items per page.	
PagerControl	Gets the current UniPager control used for paging.	
PagerPosition	The position of the pager relative to the paged data.	"Bottom" "Top" "TopAndBottom"
ShowEditDeleteButtons	Indicates if edit and delete buttons should automatically be shown for each displayed item in the edit mode of CMS Desk.	
UseHierarchicalOrder	Indicates whether the default hierarchical order value should be used. The order is used only if LoadHierarchicalData is set to true. The default order value is "NodeLevel, NodeOrder". The value of the OrderBy property is added to the end of the OrderBy expression.	

Specifying transformations

The following properties can be filled to define which transformations should be used by the control. All transformations are specified in format:

`<document type code name>.<transformation name>`

The design of the listed items can alternatively be set by defining the **ItemTemplate** elements inherited from the UniView control between the tags of the CMSUniView control.

Property Name	Description	Sample Value
AlternatingTransformationName	Name of the transformation applied to items that have an even position in the listing order. Every level in the hierarchy has its own separate alternation pattern.	
FirstTransformationName	Name of the transformation applied to the first item on every level in the hierarchy. Only works for levels that contain more than one item.	
FooterTransformationName	Name of the transformation rendered at the end of every level (after the last item on the level). Can be used to close encapsulating elements from the <i>Header</i> .	
HeaderTransformationName	Name of the transformation rendered at the beginning of every level (before the first item on the level). Provides a convenient way to visually separate or style individual levels.	
HierarchicalTransformationName	Name of the used hierarchical transformation . Please see the Using hierarchical transformations topic for an example of how this property can be used.	
LastTransformationName	Name of the transformation applied to the last item on every level in the hierarchy. Only works for levels that contain more than one item.	
SelectedFooterItemTransformationName	Name of the transformation used for the footer of selected items.	
SelectedHeaderItemTransformationName	Name of the transformation used for the header of selected items.	
SelectedItemTransformationName	Name of the transformation applied to the currently selected item (i.e. the document that is being viewed).	
SeparatorTransformationName	Name of the transformation rendered between items. If hierarchical data is loaded, the separator is	

	placed only between items on the same level (i.e. not between a parent item and its child).	
SingleTransformationName	Name of the transformation applied in cases where there is only one item on a level in the hierarchy.	
TransformationName	Name of the transformation applied to all displayed items that are not covered by a specialized transformation type (e.g. alternating items, first items etc.).	

Configuring the pager

The CMSUniView control has a built-in [UniPager](#) control which can be enabled by the **EnablePaging** property. The following templates can be defined within the tags of the CMSUniView control to determine the design of the pager. Please refer to the [UniPager -> Structure](#) topic to see what individual templates represent.

Template Name	Description	Sample Value
PagerCurrentPageTemplate	Code of the template used for the current page in the pager. Use <code><%=# Eval("Page") %></code> to get the current page number, <code><%=# Eval("PageURL") %></code> to get page the URL or <code><%=# Eval("PageLink") %></code> to get the page link.	<code><%=# Eval("Page") %></code>
PagerDirectPageTemplate	Code of the template used for direct page changing. Use a TextBox or DropDownList control with ID <i>directPageControl</i> to register the page change event.	Page <code><asp:TextBox ID="DirectPageControl" runat="server" Style="width: 25px;" /></code> of <code><%=# Eval("Pages") %></code>
PagerFirstPageTemplate	Code of the template used for the link to the first page in the pager. Use <code><%=# Eval("FirstURL") %></code> to get the link to the first page.	<code><a href="<%=# Eval("FirstURL") %>">&lt;/a></code>
PagerLastPageTemplate	Code of the template used for the link to the last page in the pager. Use <code><%=# Eval("LastURL") %></code> to get the link to the last page.	<code><a href="<%=# Eval("LastURL") %>">&gt;&lt;/a></code>
PagerLayoutTemplate	Code of the template used for the overall pager layout.	
PagerNextGroupTemplate	Code of the template used for the link to the next group of pages. Use <code><%=# Eval("NextGroupURL") %></code> to get the link to the next group.	<code><a href="<%=# Eval("NextGroupURL") %>">...</code>
PagerNextPageTemplate	Code of the template used for the link to the next page. Use <code><%=# Eval("NextURL") %></code> to	<code><a href="<%=# Eval("NextURL") %>">&gt;&lt;/a></code>

	get the link to the next page.	
PagerPageNumbersSeparatorTemplate	Code of the template used for the separator between page links in the pager.	<code>&nbsp;</code>
PagerPageNumbersTemplate	Code of the template used for page links in the pager. Use <code><%=# Eval("Page") %></code> to get the current page number, <code><%=# Eval("PageURL") %></code> to get the URL of the current page or <code><%=# Eval("PageLink") %></code> to get the page link.	<code><a href="<%=# Eval("PageURL") %>"><%=# Eval("Page") %></code>
PagerPreviousGroupTemplate	Code of the template used for the link to the previous group of pages. Use <code><%=# Eval("PreviousGroupURL") %></code> to get the link to the next group.	<code><a href="<%=# Eval("PreviousGroupURL") %>">...</code>
PagerPreviousPageTemplate	Code of the template used for the link to the previous page. Use <code><%=# Eval("PreviousURL") %></code> to get the link to the next page.	<code><a href="<%=# Eval("PreviousURL") %>">&lt;</code>

1.8.6.2.9.4 Using hierarchical transformations

The following is a step-by-step tutorial that will show you how the CMSUniView control can use a hierarchical transformation to display a hierarchical list of job openings (CMS.Job documents), offices (CMS.Office documents) and their categories (CMS.Menuitem documents) from the sample Corporate Site:

1. Create a new **Web form** somewhere in your web site installation directory.
2. Switch to its **Design** tab, drag and drop a **CMSUniView** control from the toolbox onto the form and set its following properties:
 - **ClassNames:** CMS.Office;CMS.Job;CMS.Menuitem
 - **Path:** /Company/%
 - **LoadHierarchicalData:** true
 - **HierarchicalTransformationName:** CMS.Job.HierarchicalJobsCareer

This tells the control which document types to read, specifies the **Path** that should be used, indicates that data should be bound in a hierarchical grouped dataset and assigns the hierarchical transformation that should be used to display the list. The hierarchical transformation contains individual transformations for all three document types. This is how it is defined in the CMS:

Actions	Type	Document types ^	Level	Transformation name
 	Item transformation	CMS.Job	All	CMS.Job.Preview
 	Item transformation	CMS.Office	All	CMS.Office.Simple
 	Header item transformation	All	All	CMS.Menuitem.ListItemHeader
 	Footer item transformation	All	All	CMS.Menuitem.ListItemFooter
 	Item transformation	CMS.Menuitem	All	CMS.Menuitem.ListItem
 	Current item transformation	CMS.Office	All	CMS.Office.Default
 	Current item transformation	CMS.Job	All	CMS.Job.Default

Items per page: 25

3. Switch to the **Source** tab. The code of the CMSUniView control should look like this:

```
<cms:CMSUniView ID="CMSUniView1" runat="server" ClassNames="CMS.Office;CMS.Job;
CMS.MenuItem"
Path="/Company/%" LoadHierarchicalData="true" HierarchicalTransformationName="CMS.
Job.HierarchicalJobsCareer" >
</cms:CMSUniView>
```

4. Save the changes to the web form. Now right-click it in the Solution explorer and select **View in Browser**. The resulting page should look like this:

Careers

[Web Developer](#)

We are currently looking for skilled web developers to join our web development team in the London Office. themselves in work on demanding complex projects for top grade customers.

Location:

[London Office](#)

[Web Designer](#)

We are currently looking for skilled web designers to empower our London Office web design team. The position is looking for a job where they can use their creativity together with their technical skills.

Location:

[London Office](#)

Offices

[London Office](#)



Old Street
London, England EC1Y 1BE

[New York Office](#)



Third Avenue
New York, NY 10022-7601

1.8.6.2.9.5 Appearance and styling

The appearance of the CMSUniView control is determined by the transformations specified by its **TransformationName** properties or by the code of the **ItemTemplates** inherited from the [UniView](#) control and by certain other properties. These are described in more detail in the [Configuration](#) topic.

1.8.6.2.10 CMSViewer

1.8.6.2.10.1 Overview

The CMSViewer control allows you to display document data from the Kentico CMS database based on XSLT transformations without the need to write any additional code.

The content is specified using its **Path**, **MaxRelativeLevel**, **ClassNames**, **CultureCode**, **WhereCondition** and **OrderBy** properties. Data is retrieved using the **SelectDocuments** query of the

specified document type. These queries can be found at **Kentico CMS Site Manager -> Development -> Document types -> ... Edit specified document type ... -> Queries.**

The portal engine equivalent of the CMSViewer control is the [Listings and viewers -> Documents -> XSLT viewer](#) web part.

The following topics are available to help you familiarize yourself with the CMSViewer control:

- [Getting started](#) - contains a quick step-by-step tutorial that allows you to learn the basics of using the control
- [Configuration](#) - describes and explains the properties that can be set for the control
- [Appearance and styling](#) - describes how the design of the control can be modified

1.8.6.2.10.2 Getting started

The following is a step-by-step tutorial that will show you how to use the CMSViewer control to display a specific news item (CMS.News document) using an XSLT transformation:

1. Create a new **Web form** somewhere in your website installation directory.

2. Switch to its **Design** tab, drag and drop a **CMSViewer** control from the toolbox onto the form and set its following properties:

- **ClassNames:** CMS.News
- **Path:** /News/New-Consulting-Services
- **TransformationName:** cms.news.default_xslt

This tells the control which document types to read, specifies the **Path** to the news document and assigns the transformation that should be used.



Please note

The document types entered into the **ClassNames** property must be identical (it is **case sensitivite**), to the value of the **match** property of the `<xsl:template>` element of the specified xslt transformation.

3. Switch to the **Source** tab. The code of the CMSViewer control should look like this:

```
<cms:CMSViewer ID="CMSViewer1" runat="server" ClassNames="CMS.News"
Path="/News/New-Consulting-Services" TransformationName="cms.news.default_xslt" />
```

4. Save the changes to the web form. Now right-click it in the Solution explorer and select **View in Browser**. The resulting page should look like this:

NewsID: 1
 News Title: New Consulting Services
 News Summary:

We are proud to announce that the range of services we provide was extended by web development have been promoted to consultants and are here to help you with your web development projects.

News Text:

We are proud to announce that the range of services we provide was extended by web development from our web development team. With extensive professional background and hundreds of successful or forthcoming web development projects. No matter how complicated your projects are or how tight success.

Release Date: 2011-06-05T00:00:00+02:00

1.8.6.2.10.3 Configuration

The following properties of the CMSViewer control can be set or used in the API:

All of the common properties from:

- [CMS controls - common properties](#)
- [Displaying related documents](#)

In addition, the following properties are available:

Property Name	Description	Sample Value
HideControlForZeroRows	Hides the control when no data is loaded. Default value is False.	
SelectedItemTransformationName	Name of the XSLT transformation applied to the selected item in format <i><document type code name>.<transformation name></i> .	
TransformationName	Name of the used XSLT transformation in format <i><document type code name>.<transformation name></i> .	"cms.news.default_xslt"
ZeroRowsText	Text to be shown when the control is hidden by the HideControlForZeroRows property.	

1.8.6.2.10.4 Appearance and styling

The appearance and design of the displayed content is completely driven by the XSLT transformation specified in the **TransformationName** property.

1.8.6.3 Listings and viewers with a custom query

1.8.6.3.1 Overview

The controls in this section provide various ways to display document data from the Kentico CMS database.

Each of these controls inherits a corresponding control from the [Basic Listings and viewers](#) section and extends it to easily work with Kentico CMS documents and read and use predefined queries from the system.

The functionality of these controls is very similar to that of the controls in the [Standard listings and viewers](#) section, the main difference between them is that these controls use full database queries assigned through their **QueryName** property to determine the content they display. Only queries stored in the *CMS_Query* table of the Kentico CMS database can be selected, so if you wish to use a custom query, you must create it there or edit an existing one. The queries can be managed through the administration interface at **Site Manager -> Development -> Document types / System tables / Custom tables -> ... Edit (✎) object ... -> Queries**.

These controls also support the use of [Transformations](#).

Since the controls in this section use pre-defined queries to read data, certain steps must be taken to keep the functionality of the usual properties that set query clauses, such as **OrderBy** etc. This must be taken into consideration when writing custom queries. For more information, please refer to the [Using control properties to set query clauses](#) topic.

Available controls:

- [QueryDataGrid](#)
- [QueryDataList](#)
- [QueryRepeater](#)
- [QueryUniView](#)

1.8.6.3.2 Using control properties to set query clauses

If you wish to maintain the functionality of the control properties that set query clauses when writing a custom query to be used by the controls in this section, you need to make sure it contains the appropriate macro expressions that are dynamically replaced with the values of the corresponding properties at run-time. The following table maps the query clause properties to their respective macro expression:

Property	Expression
OrderBy	##ORDERBY##
SelectedColumns	##COLUMNS##
TopN	##TOPN##
WhereCondition	##WHERE##

So a query that selects page (menu item) documents would look like this:

```
SELECT ##TOPN## ##COLUMNS## FROM View_CONTENT_MenuItem_Joined WHERE (##WHERE##)
ORDER BY ##ORDERBY##
```

Dynamic insertion of WhereCondition parameters

The controls in this section also support dynamically inserted parameters into the **WhereCondition** property:

You can use context macro expressions that are resolved at run-time, such as the following:

Expression	Description
{%currentaliaspath%}	Alias path of the current page.
{%currentculturecode%}	Culture code of the user's preferred content culture.
{%currentsiteid%}	SiteID value of the current site.

1.8.6.3.3 CMS Custom query - common properties

All of the Listings and viewers controls with a custom query have the following properties in common:

Property Name	Description	Sample Value
PageSize	The number of displayed items per page.	
QueryName	Name of the used query in format <document type code name>. <transformation name>.	"CMS.Menuitem. selectdocuments"
QueryParameters	Gets or sets an array with query parameters.	

1.8.6.3.4 QueryDataGrid

1.8.6.3.4.1 Overview

The QueryDataGrid control displays document data from the Kentico CMS database in a customizable table without the need to write any extra code. Additionally, it allows you to specify the query used to retrieve data through its **QueryName** property. Only queries stored in the *CMS_Query* table of the Kentico CMS database can be selected, so if you wish to use a custom query, you must create it there or edit an existing one. The queries can be managed through the administration interface at **Site Manager -> Development -> Document types / System tables / Custom tables -> ... Edit  object ... -> Queries**.

The QueryDataGrid is derived from the [BasicDataGrid](#) control.

The standard DataGrid designer can be used to set up QueryDataGrid style and behaviour.

Please refer to the [Using control properties to set query clauses](#) topic to find information about using properties such as **WhereCondition** with this control.

The portal engine equivalent of the QueryDataGrid control is the **Listings and viewers -> Grid with custom query** web part.

	Please note
--	--------------------



If you only wish to display data from a specific part of the content tree, please consider using the [CMSDataGrid](#) control instead.

The following topics are available to help you familiarize yourself with the QueryDataGrid control:

- [Getting started](#) - contains a step-by-step tutorial that allows you to learn the basics of using the control
- [Configuration](#) - describes and explains the properties that can be set for the control
- [Appearance and styling](#) - describes how the design of the control can be modified

1.8.6.3.4.2 Getting started

The following is a step-by-step tutorial that will show you how to display a table that contains the latest news items (CMS.News documents) from the sample Corporate Site using the QueryDataGrid control:

1. Create a new **Web form** somewhere in your website installation directory.
2. Switch to its **Design** tab, drag and drop a **QueryDataGrid** control from the toolbox onto the form and set its **QueryName** property to `cms.news.selectlatest`. This assigns the query that should be used to retrieve news documents and tells the control to ensure sorting.
3. Right-click the QueryDataGrid on the form, select **AutoFormat...** and choose a scheme.
4. Right-click the QueryDataGrid on the form, select **Show Smart Tag** and then **Property Builder...**; the **QueryDataGrid1 Properties** dialog will be displayed.

On the **General** tab, check the **Allow sorting** box.

Now switch to the **Columns** tab, where you can specify the columns that will be displayed, and uncheck the **Create columns automatically at run time** box.

Add a new **Bound Column** from the **Available columns** list to the **Selected columns** list. Enter the following values into the appropriate fields:

- **Header text:** News Title
- **Data Field:** NewsTitle
- **Sort expression:** NewsTitle

Add another **Bound column** from the **Available columns** list to the **Selected columns** list. Enter the following values in the appropriate fields:

- **Header text:** Release Date
- **Data Field:** NewsReleaseDate
- **Sort expression:** NewsReleaseDate

Click **OK**.

5. Save the changes to the web form. Now right-click it in the Solution explorer and select **View in Browser**. The resulting page should display a table similar to the following (depending on the chosen scheme):

<u>News Title ▲</u>	<u>Release Date</u>
Apple iPad 2 In Stock	6/9/2011 12:00:00 AM
Community Website Section	6/29/2011 12:00:00 AM
Company Growth Exceeds Expectations	6/17/2011 12:00:00 AM

1.8.6.3.4.3 Configuration

As it is inherited from the [BasicDataGrid](#) control, the QueryDataGrid control has all of its properties. These can be found in the [BasicDataGrid -> Configuration](#) topic.

In addition, it has all of the following properties that can be set or used in the API:

- [CMSBase - common properties](#)
- [CMS Custom query - common properties](#)

1.8.6.3.4.4 Appearance and styling

You can modify the appearance of the QueryDataGrid control by setting the standard properties of the ASP.NET DataGrid control (inherited through the BasicDataGrid). You can find more details on particular properties in the .NET Framework documentation for the [DataGrid](#) class.

A common way to set the appearance of this control is to assign a skin through the **SkinID** property. Skins can be defined in .skin files under individual themes in the **App_Themes** folder. More information can be found in the .NET [Skins and Themes](#) documentation.

1.8.6.3.5 QueryDataList

1.8.6.3.5.1 Overview

The QueryDataList control displays document data from the Kentico CMS database in a list based on transformations without the need to write any extra code. Additionally, it allows you to specify the query used to retrieve data through its **QueryName** property. Only queries stored in the *CMS_Query* table of the Kentico CMS database can be selected, so if you wish to use a custom query, you must create it there or edit an existing one. The queries can be managed through the administration interface at **Site Manager -> Development -> Document types / System tables / Custom tables -> ... Edit (✎) object ... -> Queries**.

The QueryDataList is derived from the [BasicDataList](#) control.

Unlike the [QueryRepeater](#) control, the QueryDataList allows you to display data in several columns.

Please refer to the [Using control properties to set query clauses](#) topic to find information about using properties such as **WhereCondition** with this control.

The portal engine equivalent of the QueryDataList control is the **Listings and viewers -> Datalist with custom query** web part.

Please note



If you only wish to display data from a specific part of the content tree, please consider using the [CMSDataList](#) control instead.

The following topics are available to help you familiarize yourself with the QueryDataList control:

- [Getting started](#) - contains a quick step-by-step tutorial that allows you to learn the basics of using the control
- [Configuration](#) - describes and explains the properties that can be set for the control
- [Appearance and styling](#) - describes how the design of the control can be modified

1.8.6.3.5.2 Getting started

The following is a step-by-step tutorial that will show you how to display a list of all smartphones (CMS.Smartphone documents) from the sample Corporate Site using the QueryDataList control:

1. Create a new **Web form** somewhere in your website installation directory.
2. Switch to its **Design** tab, drag and drop a **QueryDataList** control from the toolbox onto the form and set its following properties:

- **QueryName:** cms.smartphone.selectdocuments
- **RepeatColumns:** 3
- **TransformationName:** ecommerce.transformations.Product_SimplePreview

This assigns the query that should be used to retrieve smartphone documents, determines the amount of displayed columns and assigns the transformation that should be used.

3. Switch to the **Source** tab. The code of the QueryDataList control should look like this:

```
<cms:QueryDataList ID="QueryDataList1" runat="server" QueryName="cms.smartphone.  
selectdocuments"  
RepeatColumns="3" TransformationName="ecommerce.transformations.  
Product_SimplePreview" >  
</cms:QueryDataList>
```

It's not necessary to define the standard **ItemTemplate** elements of the DataList control since the transformation names have already been specified.

4. Save the changes to the web form. Now right-click it in the Solution explorer and select **View in Browser**. The resulting page should display a list like this:



Apple iPhone 4

Our price: \$759.99



Samsung Google Nexus S HTC Sensation

Our price: \$599.99



Our price: \$799.99



Apple iPhone 3GS

Our price: \$424.99



Motorola Atrix 4G

Our price: \$529.98



BlackBerry Torch 9800

Our price: \$459.99

1.8.6.3.5.3 Configuration

As it is inherited from the [BasicDataList](#) control, the QueryDataList control has all of its properties. These can be found in the [BasicDataList -> Configuration](#) topic.

In addition, it has all of the following properties that can be set or used in the API:

- [CMSBase - common properties](#)
- [CMS Custom query - common properties](#)

As well as:

Property Name	Description	Sample Value
AlternatingTransformationName	Name of the transformation applied to alternating items in format <i><document type code name>.<transformation name></i> .	
EnablePaging	Indicates whether the built-in DataPager control should be used to page the list. This property does not affect the UniPager control, which must be added separately if you wish to use it.	
IsSelected	Indicates whether the current data source contains the selected item.	
PagerControl	This property can be used to set or get the pager control and its properties.	
SelectedDatabaseColumnName	Gets or sets the column name that should be used to select items.	
SelectedItemTransformationName	Name of the transformation applied to the selected item in format <i><document type code name>.<transformation name></i> .	

SelectedQueryStringKeyName	Gets or sets the query string key name. The presence of the key in a query string indicates, that some item should be selected. The item is determined by the value of the query string key.	
SelectedValidationType	Gets or sets the validation type used for the query string key that determines which item is selected.	"int" "guid" "string"
ShowEditDeleteButtons	Indicates if edit and delete buttons should automatically be shown for each displayed item in the edit mode of CMS Desk.	
TransformationName	Name of the transformation applied to standard items in format <i><document type code name>.<transformation name></i> .	

1.8.6.3.5.4 Appearance and styling

You can modify the appearance of the QueryDataList control by setting the standard properties of the ASP.NET DataList control (inherited through the BasicDataList). You can find more details on particular properties in the .NET Framework documentation for the [DataList](#) class.

The design of list items can be determined by the transformations specified by the **AlternatingTransformationName**, **TransformationName** and **SelectedItemTransformationName** properties or by the code of the template properties inherited from the standard ASP.NET DataList control.

1.8.6.3.6 QueryRepeater

1.8.6.3.6.1 Overview

The QueryRepeater control displays document data from the Kentico CMS database in a list based on transformations without the need to write any extra code. Additionally, it allows you to specify the query used to retrieve data through its **QueryName** property. Only queries stored in the *CMS_Query* table of the Kentico CMS database can be selected, so if you wish to use a custom query, you must create it there or edit an existing one. The queries can be managed through the administration interface at **Site Manager -> Development -> Document types / System tables / Custom tables -> ... Edit (✎) object ... -> Queries**.

The QueryRepeater is derived from the [BasicRepeater](#) control.

If you only wish to display data from a specific part of the content tree, please consider using the [CMSRepeater](#) control instead.

Please refer to the [Using control properties to set query clauses](#) topic to find information about using properties such as **WhereCondition** with this control.

The portal engine equivalent of the QueryRepeater control is the **Listings and viewers -> Repeater with custom query** web part.

**Please note**

If you only wish to display data from a specific part of the content tree, please consider using the [CMSRepeater](#) control instead.

The following topics are available to help you familiarize yourself with the QueryRepeater control:

- [Getting started](#) - contains a quick step-by-step tutorial that allows you to learn the basics of using the control
- [Configuration](#) - describes and explains the properties that can be set for the control
- [Appearance and styling](#) - describes how the design of the control can be modified

1.8.6.3.6.2 Getting started

The following is a step-by-step tutorial that will show you how to display a list of all laptops (CMS.Laptop documents) from the sample Corporate Site using the QueryRepeater control:

1. Create a new **Web form** somewhere in your website installation directory.
2. Switch to its **Design** tab, drag and drop a **QueryRepeater** control from the toolbox onto the form and set its following properties:

- **QueryName:** cms.laptop.selectdocuments
- **TransformationName:** ecommerce.transformations.Product_SimplePreview

This assigns the query that should be used to retrieve laptop documents and the transformations that should be used to display them.

3. Switch to the **Source** tab. The code of the QueryRepeater control should look like this:

```
<cms:QueryRepeater ID="QueryRepeater1" runat="server" QueryName="cms.laptop.  
selectdocuments"  
TransformationName="ecommerce.transformations.Product_SimplePreview" >  
</cms:QueryRepeater>
```

It's not necessary to define the standard **ItemTemplate** elements of the Repeater control since the transformation names have already been specified.

4. Save the changes to the web form. Now right-click it in the Solution explorer and select **View in Browser**. The resulting page should display a list like this:



Apple iPad 2
Our price: \$510.99



Apple MacBook Pro MC723LL/A
Our price: \$2199.00



Dell XPS 15z
Our price: \$1596.99



HP EliteBook 8440p WJ681AW
Our price: \$1899.00

1.8.6.3.6.3 Configuration

As it is inherited from the [BasicRepeater](#) control, the QueryRepeater control has all of its properties. These can be found in the [BasicRepeater -> Configuration](#) topic.

In addition, it has all of the following properties that can be set or used in the API:

- [CMSBase - common properties](#)
- [CMS Custom query - common properties](#)

As well as:

Property Name	Description	Sample Value
AlternatingTransformationName	Name of the transformation applied to alternating items in format <i><document type code name>.<transformation name></i> .	
EnablePaging	Indicates whether the built-in DataPager control should be used to page the list. This property does not affect the UniPager control, which must be added separately if you wish to use it.	
IsSelected	Indicates whether the current data source contains the selected item.	

ItemSeparator	Separator between displayed items.	"<hr/>"
PagerControl	This property can be used to set or get the pager control and its properties.	
SelectedDatabaseColumnName	Gets or sets the column name that should be used to select items.	
SelectedItemTransformationName	Name of the transformation applied to the selected item in format <i><document type code name>.<transformation name></i> .	
SelectedQueryStringKeyName	Gets or sets the query string key name. The presence of the key in a query string indicates, that some item should be selected. The item is determined by the value of the query string key.	
SelectedValidationType	Gets or sets the validation type used for the query string key that determines which item is selected.	"int" "guid" "string"
ShowEditDeleteButtons	Indicates if edit and delete buttons should automatically be shown for each displayed item in the edit mode of CMS Desk.	
TransformationName	Name of the transformation applied to standard items in format <i><document type code name>.<transformation name></i> .	

1.8.6.3.6.4 Appearance and styling

You can modify the appearance of the QueryRepeater control by setting the standard properties of the ASP.NET Repeater control (inherited through the BasicRepeater). You can find more details on particular properties in the .NET Framework documentation for the [Repeater](#) class.

The design of list items can be determined by the transformations specified by the **AlternatingTransformationName**, **TransformationName** and **SelectedItemTransformationName** properties or by the code of the template properties inherited from the standard ASP.NET Repeater control.

1.8.6.3.7 QueryUniView

1.8.6.3.7.1 Overview

The QueryUniView control displays document data from the Kentico CMS database in a hierarchical structure based on transformations without the need to write any additional code. Using hierarchical transformations, this control can be used to display documents with a varying design depending on their type and hierarchical level. Please see the [Using hierarchical transformations](#) topic under the CMSUniView control for an example of how this works.

Additionally, it allows you to specify the query used to retrieve data through its **QueryName** property. Only queries stored in the *CMS_Query* table of the Kentico CMS database can be selected, so if you wish to use a custom query, you must create it there or edit an existing one. The queries can be

managed through the administration interface at **Site Manager -> Development -> Document types / System tables / Custom tables -> ... Edit (✎) object ... -> Queries.**

The QueryUniView is derived from the [BasicUniView](#) control.

Please refer to the [Using control properties to set query clauses](#) topic to find information about using properties such as **WhereCondition** with this control.

The portal engine equivalent of the QueryUniView control is the **Listings and viewers -> Universal viewer with custom query** web part.

	<p>Please note</p> <p>If you only wish to display data from a specific part of the content tree, please consider using the CMSUniView control instead.</p>
---	---

The following topics are available to help you familiarize yourself with the QueryUniView control:

- [Getting started](#) - contains a step-by-step tutorial that allows you to learn the basics of using the control
- [Configuration](#) - describes and explains the properties that can be set for the control
- [Appearance and styling](#) - describes how the design of the control can be modified

1.8.6.3.7.2 Getting started

The following is a step-by-step tutorial that will show you how to display all pages (CMS.MenuItem documents) from the sample Corporate Site in a hierarchical structure using the QueryUniView control:

1. Create a new **Web form** somewhere in your website installation directory.
2. Switch to its **Design** tab, drag and drop a **QueryUniView** control onto the form and set its following properties:

- **LoadHierarchicalData**: true
- **QueryName**: CMS.MenuItem.selectdocuments
- **IDColumnName**: NodeID
- **ParentIDColumnName**: NodeParentID
- **LevelColumnName**: NodeLevel
- **TransformationName**: CMS.MenuItem.ListItem
- **HeaderTransformationName**: CMS.MenuItem.ListItemHeader
- **FooterTransformationName**: CMS.MenuItem.ListItemFooter

This tells the control to load data in a hierarchical grouped dataset, assigns the query that should be used to retrieve the page documents, sets the column names that should be used to determine the hierarchy structure of the pages and assigns the transformations that should be used to display them.

3. Switch to the **Source** tab. The code of the QueryUniView control should look like this:

```
<cms:QueryUniView ID="QueryUniView1" runat="server" LoadHierarchicalData="true"
QueryName="CMS.MenuItem.selectdocuments" IDColumnName="NodeID" ParentIDColumnName
="NodeParentID"
LevelColumnName="NodeLevel" TransformationName="CMS.MenuItem.ListItem"
HeaderTransformationName="CMS.MenuItem.ListItemHeader"
FooterTransformationName="CMS.MenuItem.ListItemFooter" >
</cms:QueryUniView>
```

It's not necessary to define the standard **ItemTemplate** elements of the UniView control since the transformation names have already been specified.

4. Save the changes to the web form. Now right-click it in the Solution explorer and select **View in Browser**. The resulting page should display a hierarchical list like this:

- Home
- Services
 - Web Design
 - Web Development
 - Network Administration
- Products
 - Smartphones
 - Laptops and Tablets
 - Software
 - E-Books
 - IT Services
 - Memberships
- News
- Partners
 - Silver Partners
 - Gold Partners
- Company
 - Careers
 - Offices
 - Donate Us
- Community
 - Blogs
 - Forums
 - Forum search
 - Events
 - Events list
 - Wiki
 - My contributions
- Media
 - Images
 - Videos

1.8.6.3.7.3 Configuration

As it is inherited from the [BasicUniView](#) control, which is in turn derived from the [UniView](#) control, the QueryUniView control has all of their properties. These can be found in the [BasicUniView ->](#)

[Configuration](#) and [UniView -> Configuration](#) topics.

In addition, it has all of the following properties that can be set or used in the API:

- [CMSBase - common properties](#)
- [CMS Custom query - common properties](#)

As well as:

Property Name	Description	Sample Value
DelayedLoading	Indicates whether data should be loaded during the load event instead of the default init event.	
EnablePaging	Indicates whether the built-in UniPager control should be used to for paging. If you wish to use paging, please be sure to set LoadHierarchicalData to false, as hierarchical data cannot be paged.	
IDColumnName	Gets or sets the name of the column that is used as a unique identifier for the displayed objects. It is used in a parent-child relationship with the column specified in the ParentIDColumnName property.	"NodeID"
IsSelected	Indicates whether the current data source contains the selected item.	
ItemSeparatorValue	Separator between displayed items. This property has greater priority than the separator set in the transformation given in the SeparatorTransformationName property.	"<hr/>"
LevelColumnName	Gets or sets the name of the column that is used to determine the hierarchical level of items.	"NodeLevel"
LoadHierarchicalData	Indicates whether data should be bound in default format or changed to a hierarchical grouped dataset.	
PagerControl	Gets the current UniPager control used for paging.	
PagerPosition	The position of the pager relative to the paged data.	"Bottom" "Top" "TopAndBottom"
ParentIDColumnName	Gets or sets the name of the column that contains the unique identifier of an object's parent. It is used in a parent-child relationship with the column specified in the IDColumnName property.	"NodeParentID"

SelectedDatabaseColumnName	Gets or sets the column name that should be used to identify the currently selected item.	"DocumentID"
SelectedQueryStringKeyName	Gets or sets the query string key name. The presence of this key in a query string indicates that some item should be selected. The item is determined by the value of the query string key.	
SelectedValidationType	Gets or sets the validation type used for the query string key that determines which item is selected.	"int" "guid" "string"
ShowEditDeleteButtons	Indicates if edit and delete buttons should automatically be shown for each displayed item in the edit mode of CMS Desk.	
UseHierarchicalOrder	Indicates whether the default hierarchical order value should be used. The order is used only if LoadHierarchicalData is set to true. The default order value is "NodeLevel, NodeOrder". The value of the OrderBy property is added to the end of the OrderBy expression. Enabling this property is only recommended when displaying documents.	

Specifying transformations

The following properties can be filled to define which transformations should be used by the control. All transformations are specified in format:

<document type code name>.<transformation name>

The design of the listed items can alternatively be set by defining the **ItemTemplate** elements inherited from the UniView control between the tags of the QueryUniView control.

Property Name	Description	Sample Value
AlternatingTransformationName	Name of the transformation applied to items that have an even position in the listing order. Every level in the hierarchy has its own separate alternation pattern.	
FirstTransformationName	Name of the transformation applied to the first item on every level in the hierarchy. Only works for levels that contain more than one item.	
FooterTransformationName	Name of the transformation rendered at the end of every level (after the last item on the level). Can be used to close encapsulating	

	elements from the <i>Header</i> .	
HeaderTransformationName	Name of the transformation rendered at the beginning of every level (before the first item on the level). Provides a convenient way to visually separate or style individual levels.	
HierarchicalTransformationName	Name of the used hierarchical transformation . Please see the Using hierarchical transformations topic for an example of how this property can be used.	
LastTransformationName	Name of the transformation applied to the last item on every level in the hierarchy. Only works for levels that contain more than one item.	
SeparatorTransformationName	Name of the transformation rendered between items. If hierarchical data is loaded, the separator is placed only between items on the same level (i.e. not between a parent item and its child).	
SingleTransformationName	Name of the transformation applied in cases where there is only one item on a level in the hierarchy.	
TransformationName	Name of the transformation applied to all displayed items that are not covered by a specialized transformation type (e.g. alternating items, first items etc.).	

Configuring the pager

The QueryUniView control has a built-in [UniPager](#) control which can be enabled by the **EnablePaging** property. The following templates can be defined within the tags of the QueryUniView control to determine the design of the pager. Please refer to the [UniPager -> Structure](#) topic to see what individual templates represent.

Template Name	Description	Sample Value
PagerCurrentPageTemplate	Code of the template used for the current page in the pager. Use <code>< %# Eval("Page") % ></code> to get the current page number, <code>< %# Eval("PageURL") % ></code> to get page the URL or <code>< %# Eval("PageLink") % ></code> to get the page link.	<code>< %# Eval("Page") % ></code>
PagerDirectPageTemplate	Code of the template used for direct page changing. Use a TextBox or DropDownList control with ID <i>directPageControl</i> to register the page change event.	Page <code><asp:TextBox ID="DirectPageControl" runat="server" Style="</code>

		<code>width: 25px;" /> of <%# Eval("Pages") %></code>
PagerFirstPageTemplate	Code of the template used for the link to the first page in the pager. Use <code><%# Eval("FirstURL") %></code> to get the link to the first page.	<code><a href="<%# Eval("FirstURL") %>">&lt;</code>
PagerLastPageTemplate	Code of the template used for the link to the last page in the pager. Use <code><%# Eval("LastURL") %></code> to get the link to the last page.	<code><a href="<%# Eval("LastURL") %>">&gt;</code>
PagerLayoutTemplate	Code of the template used for the overall pager layout.	
PagerNextGroupTemplate	Code of the template used for the link to the next group of pages. Use <code><%# Eval("NextGroupURL") %></code> to get the link to the next group.	<code><a href="<%# Eval("NextGroupURL") %>">...</code>
PagerNextPageTemplate	Code of the template used for the link to the next page. Use <code><%# Eval("NextURL") %></code> to get the link to the next page.	<code><a href="<%# Eval("NextURL") %>">&gt;</code>
PagerPageNumbersSeparatorTemplate	Code of the template used for the separator between page links in the pager.	<code>&nbsp;</code>
PagerPageNumbersTemplate	Code of the template used for page links in the pager. Use <code><%# Eval("Page") %></code> to get the current page number, <code><%# Eval("PageURL") %></code> to get the URL of the current page or <code><%# Eval("PageLink") %></code> to get the page link.	<code><a href="<%# Eval("PageURL") %>"><%# Eval("Page") %></code>
PagerPreviousGroupTemplate	Code of the template used for the link to the previous group of pages. Use <code><%# Eval("PreviousGroupURL") %></code> to get the link to the next group.	<code><a href="<%# Eval("PreviousGroupURL") %>">...</code>
PagerPreviousPageTemplate	Code of the template used for the link to the previous page. Use <code><%# Eval("PreviousURL") %></code> to get the link to the next page.	<code><a href="<%# Eval("PreviousURL") %>">&lt;</code>

1.8.6.3.7.4 Appearance and styling

The appearance of the QueryUniView control is determined by the transformations specified by its **TransformationName** properties or by the code of the **ItemTemplates** inherited from the **UniView** control and by certain other properties. These are described in more detail in the [Configuration](#) topic.

1.8.7 Edit mode buttons

1.8.7.1 Overview

The controls in this section display buttons that allow Kentico CMS documents to quickly and easily be created, edited or deleted. These buttons are only displayed in the **Edit** mode of **CMS Desk**.

Available controls:

- [CMSEditModeButtonAdd](#)
- [CMSEditModeButtonEditDelete](#)

1.8.7.2 CMSEditModeButtonAdd

1.8.7.2.1 Overview

The CMSEditModeButtonAdd control displays a button that is shown in the **Edit** mode of **CMS Desk** and allows content editors to add a new document when they click it. It provides an intuitive way of creating new documents.

The following topics are available to help you familiarize yourself with the CMSEditModeButtonAdd control:

- [Getting started](#) - contains a quick step-by-step tutorial that allows you to learn the basics of using the control
- [Configuration](#) - describes and explains the properties that can be set for the control
- [Appearance and styling](#) - lists which CSS classes can be used with the control

1.8.7.2.2 Getting started

The following tutorial will show you how to display a button, that creates new news documents, on an ASPX page template using the CMSEditModeButtonAdd control:

1. Create a new **Web form** and prepare it to be an ASPX page template according to the guide found in the [Using ASPX page templates](#) topic.
2. Switch to its **Design** tab, drag and drop a **CMSEditModeButtonAdd** control from the toolbox onto the form and set its following properties:

- **ClassName:** CMS.News
- **Path:** /News/

This tells the control what type of documents it should create and sets the path to the document under which they should be added.

The code of the control will look like this:

```
<cms:CMSEditModeButtonAdd ID="CMSEditModeButtonAdd1" runat="server" Path="/News/"  
ClassName="CMS.News" />
```

3. Switch to the code behind of the page and make sure it has the following references at the beginning of the code:

[C#]

```
using CMS.UIControls;
using CMS.CMSHelper;
using CMS.PortalEngine;
using CMS.GlobalHelper;
```

4. Now add the following code to the **Page_Load** method:

[C#]

```
// Register edit mode buttons script
if (CMSContext.ViewMode != ViewModeEnum.LiveSite)
{
    ScriptHelper.RegisterClientScriptBlock(this, typeof(string), ScriptHelper.
EDIT_DOCUMENT_SCRIPT_KEY, ScriptHelper.EditDocumentScript);
}
```

This is necessary if you wish to use the control individually on an ASPX page template.

5. Save the changes to the web form. Now if you look at the page using the created template on some website in the **Edit** mode of **CMS Desk**, a button like the one in the following image will be displayed:



If you press the button, it will redirect you to the form used to create new news documents under the / News page.

1.8.7.2.3 Configuration

The following properties of the CMSEditModeButtonAdd control can be set or used in the API:

Property Name	Description	Sample Value
ClassName	Document type that specifies the type of the document that should be created.	"cms.article"
Path	Alias path of the parent document under which the new document should be created. If omitted, the document is added under the currently selected document.	"/whitepapers"
StopProcessing	Indicates if processing of the control should be stopped and the control should not retrieve or display any data.	
Text	Custom caption of the button. If not set, the default text "Add new" is displayed.	"Add new article"

1.8.7.2.4 Appearance and styling

The appearance of the CMSEditModeButtonAdd control is determined by the CSS class it uses.

You can use the following CSS class to modify the design of the control:

Class Name	Description
CMSEditModeButtonAdd	CSS style of the <A> element.

The recommended place to define this class is in a stylesheet in the Kentico CMS administration interface at **Site Manager -> Development -> CSS stylesheets**. These stylesheets can be applied to individual documents (pages) that contain the control in **CMS Desk -> Content -> Edit -> Properties -> General -> CSS stylesheet**.

1.8.7.3 CMSEditModeButtonEditDelete

1.8.7.3.1 Overview

The CMSEditModeButtonEditDelete control displays a pair of buttons that are shown in the **Edit** mode **CMS Desk** and allow content editors to edit or delete documents when they click it. It provides an intuitive way of editing/deleting documents.

Many CMS listing controls (and web parts), such as the CMS Repeater (Repeater) have the **ShowEditDeleteButtons (Show Edit and Delete buttons)** property, which causes this control to automatically be shown next to every displayed document. The path of these controls will automatically be set to that of the corresponding displayed document.

The following topics are available to help you familiarize yourself with the CMSEditModeButtonEditDelete control:

- [Getting started](#) - contains a quick step-by-step tutorial that allows you to learn the basics of using the control
- [Configuration](#) - describes and explains the properties that can be set for the control
- [Appearance and styling](#) - lists which CSS classes can be used with the control

1.8.7.3.2 Getting started

The following tutorial will show you how to display a pair of buttons, that can edit or delete a specific news document, on an ASPX page template using the CMSEditModeButtonEditDelete control:

1. Create a new **Web form** and prepare it to be an ASPX page template according to the guide found in the [Using ASPX page templates](#) topic.
2. Switch to its **Design** tab, drag and drop a **CMSEditModeButtonEditDelete** control from the toolbox onto the form and set its **Path** property to `/News/New-Consulting-Services`.

This sets the path to the document that the control should edit or delete.

The code of the control will look like this:

```
<cms:CMSEditModeButtonEditDelete ID="CMSEditModeButtonEditDelete1" runat="server"
Path="/News/New-Consulting-Services" />
```

3. Switch to the code behind of the page and make sure it has the following references at the beginning of the code:

[C#]

```
using CMS.UIControls;
using CMS.CMSHelper;
using CMS.PortalEngine;
using CMS.GlobalHelper;
```

4. Now add the following code to the **Page_Load** method:

[C#]

```
// Register edit mode buttons script
if (CMSContext.ViewMode != ViewModeEnum.LiveSite)
{
    ScriptHelper.RegisterClientScriptBlock(this, typeof(string), ScriptHelper.
EDIT_DOCUMENT_SCRIPT_KEY, ScriptHelper.EditDocumentScript);
}
```

This is necessary if you wish to use the control individually on an ASPX page template.

5. Save the changes to the web form. Now if you look at the page using the created template on some website in the **Edit** mode of **CMS Desk**, a pair of buttons as seen in the following image will be displayed:



If you press the **Edit** button, it will redirect you to the form used to edit the */News/Your-first-news* document. If you click the **Delete** button, the same document will be deleted.

Use in transformations

If you wish to use this control in the code of a [transformation](#), you can do so by adding code similar to the following:

```
<cms:CMSEditModeButtonEditDelete runat="server" id="btnEditDelete" Path='<%# Eval
("NodeAliasPath") %>' />
```

The path will automatically be set to that of the currently transformed document.

1.8.7.3.3 Configuration

The following properties of the CMSEditModeButtonEditDelete control can be set or used in the API:

Property Name	Description	Sample Value
DeleteText	Custom caption of the delete button. If not set, the default text "Delete" is displayed.	"Delete article"
EditText	Custom caption of the edit button. If not set, the default text "Edit" is displayed.	"Edit article"
Path	Alias path of the document to be edited/ deleted.	"/whitepapers/myfirstpaper"

1.8.7.3.4 Appearance and styling

The appearance of the CMSEditModeButtonEditDelete control is determined by the CSS classes it uses.

You can use the following CSS classes to modify the design of the control:

Class Name	Description
CMSEditModeButtonEdit	CSS style of the edit button <A> element.
CMSEditModeButtonDelete	CSS style of the delete button <A> element.

The recommended place to define these classes is in a stylesheet in the Kentico CMS administration interface at **Site Manager -> Development -> CSS stylesheets**. These stylesheets can be applied to individual documents (pages) that contain the control in **CMS Desk -> Content -> Edit -> Properties -> General -> CSS stylesheet**.

1.8.8 Editable regions for ASPX page templates

1.8.8.1 Overview

The controls in this section allow the creation of editable regions that can be used by editors to fill in content. They provide editing dialogs in the **Edit** mode of **CMS Desk** and once some content is entered into them, it is displayed on the live site. They are only compatible with Kentico CMS [ASPX page templates](#).

Please note:

If you wish to have this kind of functionality on [Portal engine templates](#), use the **Editable text** and **Editable image** web parts instead.

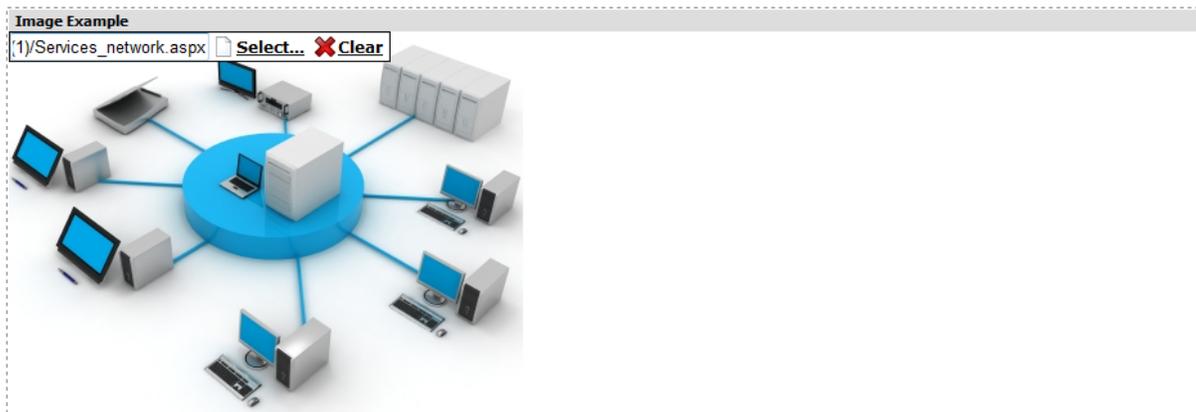
Available controls:

- [CMSEditableImage](#)
- [CMSEditableRegion](#)
- [CMSPageManager](#)

1.8.8.2 CMSEditableImage

1.8.8.2.1 Overview

The CMSEditableImage control displays an editable region in the **Edit** mode of CMS Desk that allows content editors to select an image. The chosen image is then displayed without the selection interface on the live version of the website.



Please note

This control is compatible only with ASPX page templates. On portal engine pages, use the [Text -> Editable image](#) web part instead of this control

This control requires either the [CMSPageManager](#) or CMSPortalManager control to regulate the flow of data to/from the editable region, it doesn't communicate with the database directly. There only has to be one Manager control for any number of CMSEditableImage controls.

The following topics are available to help you familiarize yourself with the CMSEditableImage control:

- [Getting started](#) - contains a quick step-by-step tutorial that allows you to learn the basics of using the control
- [Configuration](#) - describes and explains the properties that can be set for the control

1.8.8.2.2 Getting started

The following is a step-by-step tutorial that will show you how to display a region which can be used to select and display images using the CMSEditableImage control:

1. Create a new **Web form** in your website installation directory under the **CMSTemplates/CorporateSite** folder and check the **Select master page** box.

2. The **Select a Master Page** dialog appears. Choose the folder **CMSTemplates/CorporateSite** and choose the **root.master** file and click **OK**.
3. Follow the remaining steps of the guide found in the [Using ASPX page templates](#) topic.
4. Switch to the **Design** tab and drag and drop a **CMSEditableImage** control from the toolbox onto the form and set its **ImageTitle** property to *Image Example*.
5. Switch to the **Source** tab. The code of the CMSEditableImage control should look like this:

```
<cms:CMSEditableImage ID="CMSEditableImage1" runat="server" ImageTitle="Image Example" />
```

It's not necessary to add the CMSPageManager control onto the web form, as there is already a CMSPortalManager control on the *root.master* master page.

6. Save the changes to the web form. Now if you look at the page using the created template on some website, it will display an image selection dialog in the **Edit** mode of **CMS Desk** similar to the following:



Now if you select an image and press the **Save** button, the image will be displayed on the live site version of the page without the selection dialog.

1.8.8.2.3 Configuration

The following properties of the CMSEditableImage control can be set or used in the API:

Property Name	Description	Sample Value
AlternateText	ALT text of the image displayed on the website.	
DisplaySelectorTextBox	Indicates whether a textbox with the image path should be displayed in Edit mode.	
ImageControl	Returns the instance of the used Image control.	
ImageCSSClass	Sets or gets the name of the image CSS class.	
ImageHeight	Image height in pixels - the image will be resized to this height.	
ImageStyle	Sets or gets the style of the image.	
ImageTitle	Title displayed above the image in Edit mode.	
ImageWidth	Image width in pixels - the image will be resized to this width.	

1.8.8.3 CMSEditableRegion

1.8.8.3.1 Overview

The CMSEditableRegion control displays an editable region in the **Edit** mode of CMS Desk that allows website editors to enter a wide range of content. It then displays its content without the editing interface on the live version of the website.



Please note

This control is compatible only with ASPX page templates. On portal engine pages, use the [Text -> Editable text](#) web part instead of this control.

This control requires either the [CMSPageManager](#) or CMSPortalManager control to regulate the flow of data to/from the editable region, it doesn't communicate with the database directly. There only has to be one Manager control for any number of CMSEditableRegion controls.

The following topics are available to help you familiarize yourself with the CMSEditableRegion control:

- [Getting started](#) - contains a quick step-by-step tutorial that allows you to learn the basics of using the control
- [Configuration](#) - describes and explains the properties that can be set for the control
- [Appearance and styling](#) - lists which CSS classes can be used with the control and how its appearance can be modified

1.8.8.3.2 Getting started

The following is a step-by-step tutorial that will show you how to display an editable region, which works as an HTML editor, using the CMSEditableRegion control:

1. Create a new **Web form** in your website installation directory under the **CMSTemplates/CorporateSite** folder and check the **Select master page** box.
2. The **Select a Master Page** dialog appears. Choose the folder **CMSTemplates/CorporateSite** and choose the **root.master** file and click **OK**.
3. Follow the remaining steps of the guide found in the [Using ASPX page templates](#) topic.
4. Switch to the **Design** tab and drag and drop a **CMSEditableRegion** control from the toolbox onto the form and set its following properties:
 - **RegionTitle**: Region Example
 - **DialogHeight**: 200
 - **RegionType**: HTML editor
 - **HTMLAreaToolBarLocation**: In

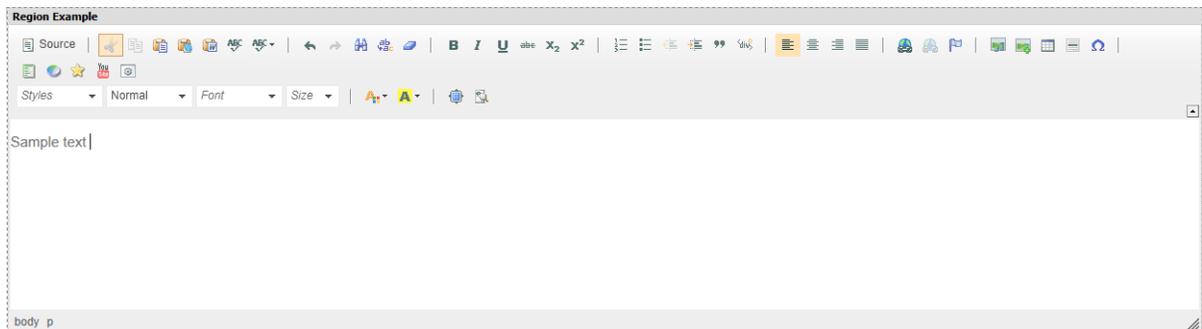
This specifies the height of the editable region, tells the control that the region should act as an HTML editor and that the toolbar of the editor should be displayed directly above the region.

5. Switch to the **Source** tab. The code of the CMSEditableRegion control should look like this:

```
<cms:CMSEditableRegion ID="CMSEditableRegion1" runat="server" RegionTitle="Region
Example"
DialogHeight="200" RegionType="HTMLEditor" HtmlAreaToolBarLocation="In" />
```

It's not necessary to add the CMSPageManager control onto the web form, as there is already a CMSPortalManager control on the *root.master* master page.

6. Save the changes to the web form. Now if you look at the page using the created template on some website, it will display an editable region in the **Edit** mode of **CMS Desk** similar to the following:



Now if you enter some content and press the **Save** button, it will be displayed on the live site version of the page without the editor.

1.8.8.3.3 Configuration

The following properties of the CMSEditableRegion control can be set or used in the API:

Property Name	Description	Sample Value
DialogHeight	Height of the displayed region in pixels.	
DialogWidth	Width of the displayed region in pixels.	
HtmlAreaToolBar	Name of the HTML editor toolbar set. This is only used if the RegionType property is set to <i>HtmlEditor</i> .	"Default"
HtmlAreaToolBarLocation	Determines the location of the HTML editor toolbar. This is only used if the RegionType property is set to <i>HtmlEditor</i> .	"In" for inline - directly above the region "Out:CKToolBar" for shared - at the top of the page
InheritContent	Indicates if the content of the editable region should be inherited from the parent page (menu item) document.	
MaxLength	Maximum length of the content (in number of	

	characters).	
MinLength	Minimum length of the content (in number of characters).	
RegionTitle	Title displayed about the editable region in editing mode.	
RegionType	Type of server control which is displayed in the editable region. It can be a textbox, textarea or HTML editor.	"HtmlEditor" "TextArea" "TextBox"
WordWrap	Indicates whether text displayed by the control should use word wrapping if the text area RegionType is selected.	

Please note: the remaining properties are used internally by the system and should not be modified manually.

1.8.8.3.4 Appearance and styling

The appearance of the CMSEditableRegion control is determined by the CSS classes it uses and by some of its properties.

You can use the following CSS classes to modify the design of the control:

Class Name	Description
CMSEditableRegionEdit	Style of the main <TABLE> element.
CMSEditableRegionTitle	Style of the <TD> element containing the error title.
CMSEditableRegionError	Style of the <TD> element containing the error message.

The recommended place to define these classes is in a stylesheet in the Kentico CMS administration interface at **Site Manager -> Development -> CSS stylesheets**. These stylesheets can be applied to individual documents (pages) that contain the control in **CMS Desk -> Content -> Edit -> Properties -> General -> CSS stylesheet**.

1.8.8.4 CMSPageManager

1.8.8.4.1 Overview

The CMSPageManager control is required for pages with editable regions, as it manages the flow of data to/from the [CMSEditableRegion](#) and [CMSEditableImage](#) controls. It ensures that the content of editable regions is loaded from and saved to the database. It also displays the "Save" dialog.

Please note

The CMSPageManager control is obsolete. It is still supported for the purposes of backward compatibility, but we recommend using the CMSPortalManager instead, which provides all of the same functionality and more. Further information can be found

in [Developer's Guide -> .. -> Creating ASPX master pages](#).

Data Source

Content is loaded from the nearest Page (menu item) document in the alias path specified through the URL or through the **DefaultPageAliasPath** property. The content is stored in the following format:

```
<content>
<region id="ID of the CMSEditableRegion control related to this content section">
<!CDATA[ content of the editable region ]>
</region>
<region id="...">
<!CDATA[ content of the editable region ]>
</region>
</content>
```

The following topics are available to help you familiarize yourself with the CMSEditableRegion control:

- [Getting started](#) - describes the basics of using the control
- [Configuration](#) - describes and explains the properties that can be set for the control
- [Appearance and styling](#) - lists which CSS classes can be used with the control

1.8.8.4.2 Getting started

To use the CMSPageManager control, simply drag and drop it from the toolbox onto an ASPX page template that contains CMSEditableImage or CMSEditableRegion controls. A common place to have the control is on the [master page](#), as this allows it to manage editable regions on all page templates that use this master page.

1.8.8.4.3 Configuration

The following properties of the CMSPageManager control can be set or used in the API:

Property Name	Description	Sample Value
CacheDependencies	List of the cache keys on which the cached data depends. When the cache item changes, the cache of the control is cleared. Each item (dependency) must be on one line. If you leave this property empty, default dependencies will be used.	cms.user all
CacheItemName	Name of the cache item the control will use. By setting this name dynamically, you can achieve caching based on a URL parameter or some other variable - simply enter the	"homepage_pagemanager"

	<p>value of the parameter.</p> <p>If no value is set, the control stores its content in the item named "URL ControlID".</p>	
CacheMinutes	<p>Number of minutes the retrieved content is cached for.</p> <p>Zero indicates that the content will not be cached. -1 indicates that the site-level settings should be used.</p> <p>This parameter allows you to set up caching of content so that it doesn't have to be retrieved from the database each time a user requests the page.</p>	
CheckPermissions	<p>Allows you to specify whether to check permissions of the current user. If the value is 'false' (default value) no permissions are checked. Otherwise, only nodes for which the user has read permission can be selected.</p>	
CMSEditableControls	<p>Returns an array containing the managed editable controls (CMSEditableImage or CMSEditableRegion).</p>	
CombineWithDefaultCulture	<p>Indicates if the results should be combined with default language versions in case the translated version is not available. This property is applied only if you do not set the TreeProvider property manually.</p>	
CultureCode	<p>Culture code of documents to be selected, such as en-us. If not specified, it's read from the user's session or the default value is used.</p>	"en-us"
DefaultPageAliasPath	<p>Default path that is used if no alias path is provided in the query string or through a friendly URL.</p>	"/home"
ErrorMessage	<p>Gets or sets the error message string.</p>	
InfoMessage	<p>Gets or sets the information message string.</p>	
IsAuthorized	<p>Is true if the current user is authorized for the current document.</p>	
PageAliasPath	<p>The alias path of the current page.</p>	
PreserveContent	<p>Allows you to specify whether the content of non-existing or not visible regions should be preserved when the content is saved.</p>	

SaveChanges	Is true if the current changes to the page should be saved.	
SiteName	Specifies the site code name.	
TagKey	Overrides the generation of the SPAN tag with a custom tag.	
TreeProvider	Tree provider instance used to access data. If no TreeProvider is assigned, a new TreeProvider instance is created automatically.	
ViewMode	Gets or sets the current page mode.	"Edit" "Preview" "LiveSite"

1.8.8.4.4 Appearance and styling

The appearance of the CMSPageManager control is determined by the CSS classes it uses.

You can use the following CSS classes to modify the design of the control:

Class Name	Description
CMSPageManagerError	Style of the error label.
CMSPageManagerLabel	Style of the standard label.
CMSPageManagerTextLink	Style of the link.
CMSPageManagerTDLabel	Style of the TD element that contains text with save confirmation message.

The recommended place to define these classes is in a stylesheet in the Kentico CMS administration interface at **Site Manager -> Development -> CSS stylesheets**. These stylesheets can be applied to individual documents (pages) that contain the control in **CMS Desk -> Content -> Edit -> Properties -> General -> CSS stylesheet**.

1.8.9 Search Controls

1.8.9.1 Overview

The controls in this section provide the capability to search through the content of the Kentico CMS database. They use the SQL search engine, which uses standard SQL queries to search for a given expression. It runs a separate query for each document type. You can find the search query for a given document type in **Site Manager -> Development -> Document types -> ... select document type ... -> Queries -> Edit** (✎) the **searchtree** query.

Common fields (such as document name) are searched using the **cms.root.searchdocuments** query.

Uploaded files are searched using the **cms.root.searchattachments** query. If you want to search uploaded files, you need to configure the system as described at [Developer's Guide -> Installation and deployment -> Additional configuration tasks -> Configuration of full-text search in files](#). In this case, the files are searched using the Microsoft SQL Server Search Engine.

**Please note**

Current Kentico CMS versions contain Smart search capability. The Smart search engine is index based and has significantly better performance than the SQL search engine.

To learn more about it, please refer to [Developer's Guide -> Modules -> Smart search](#).

Available controls:

- [CMSSearchDialog](#)
- [CMSSearchResults](#)

1.8.9.2 CMSSearchDialog

1.8.9.2.1 Overview

The CMSSearchDialog control allows users to enter expressions that are then searched for in the Kentico CMS database. The user can also (optionally) specify the search scope (where to search) and search mode (how to search).

CMSSearchDialog can easily be used with the [CMSSearchResults](#) control to display search results. Both controls can be used separately.

The CMSSearchDialog control is not connected to any data source - it only communicates with the user. The search method in the CMSSearchResults dialog uses the pre-defined **searchtree** queries stored in document type definitions. It combines all results and returns them as one table.

The portal engine equivalent of the CMSSearchDialog control is the **Full-text search -> SQL Search dialog** web part.

The following topics are available to help you familiarize yourself with the CMSSearchDialog control:

- [Getting started](#) - contains a step-by-step tutorial that allows you to learn the basics of using the control
- [Configuration](#) - describes and explains the properties that can be set for the control
- [Structure](#) - continues the tutorial from Getting started, shows a more advanced example of how the control can be configured and demonstrates what individual template properties affect
- [Appearance and styling](#) - lists which CSS classes can be used with the control and how its appearance can be modified

1.8.9.2.2 Getting started

The following is a step-by-step tutorial that will show you how to create a working search dialog using the CMSSearchDialog and CMSSearchResults controls:

1. Create a new **Web form** somewhere in your website installation directory.
2. Switch to its **Design** tab, drag and drop a **CMSSearchDialog** control from the toolbox onto the form.

3. Now drag and drop a **CMSSearchResults** control from the toolbox onto the form below the CMSSearchDialog and set its **CMSSearchDialogID** property to *CMSSearchDialog1*.

The code of the two controls should look like this:

```
<cms:CMSSearchDialog ID="CMSSearchDialog1" runat="server" />

<cms:CMSSearchResults ID="CMSSearchResults1" runat="server" CMSSearchDialogID
="CMSSearchDialog1">
</cms:CMSSearchResults>
```

4. Save the changes to the web form. Now right-click it in the Solution explorer and select **View in Browser**. The resulting page will contain a functional search dialog that will allow you to search the sample Corporate Site:

Search for:

[Web Designer](#)
Path: /Company/Careers/Web Designer

[Web Developer](#)
Path: /Company/Careers/Web Developer

[Offices](#)
Path: /Company/Offices

[Offices](#)
Path: /Examples/Webparts/Listings and viewers/Universal viewer/Offices

[New York Office](#)
Path: /Company/Offices/New York Office

[London Office](#)
Path: /Company/Offices/London Office

[Prague office](#)
Path: /Examples/Webparts/Listings and viewers/Universal viewer/Offices/Prague office

[Sydney office](#)
Path: /Examples/Webparts/Listings and viewers/Universal viewer/Offices/Sydney office

Continue this tutorial in the [Structure](#) topic to see how to create a more advanced search dialog.

1.8.9.2.3 Configuration

The following properties of the CMSSearchDialog control can be set or used in the API:

Property Name	Description	Sample Value
CustomQueryStringData	Gets or sets a custom query string which is placed after search querystring data, do not use & or ? at the start of the custom query string.	
SearchExpression	Entered word(s) to be searched for.	
SearchMode	Search mode - any word, all words or exact phrase.	"AllWords" "AnyWord" "ExactPhrase"
SearchScope	Indicates whether all content or only the current section should be searched.	"SearchAllContent" "SearchCurrentSection"

ShowSearchMode	Indicates whether search mode settings should be displayed.	
ShowSearchScope	Indicates whether search scope settings should be displayed.	
StopProcessing	Indicates if processing of the control should be stopped and the control should not retrieve or display any data.	

CMSSearchDialog Events

Event Name	Description
DoSearch	Occurs when a user submits the dialog.

The CMSSearchDialog contains the following controls. Refer to the [Structure](#) topic to see what individual controls represent. They can be accessed in your code behind through the following properties:

Property Name	Description
SearchButton	Search submit button.
SearchForLabel	SearchFor label control.
SearchForTextBox	SearchFor textbox.
SearchModeLabel	SearchMode label.
SearchModeList	SearchMode drop-down list.
SearchScopeLabel	SearchScope label.
SearchScopeList	SearchScope drop-down list.

1.8.9.2.4 Structure

This topic shows an example of how the CMSSearchDialog control looks when its **ShowSearchMode** and **ShowSearchScope** properties are enabled. If you wish to create this example for yourself, please follow the tutorial in the [Getting started](#) topic, then continue with the following steps:

1. Make sure that the following properties of the CMSSearchDialog are set:

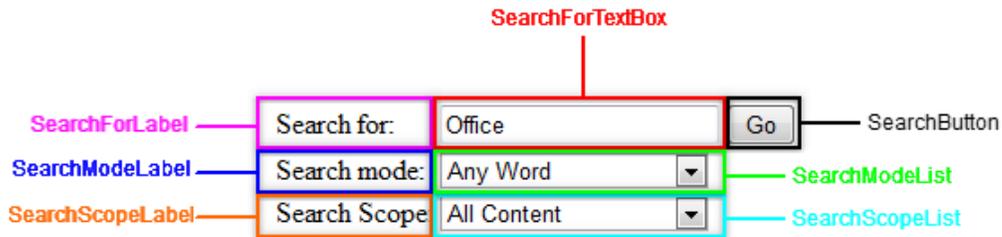
- **ShowSearchMode**: True
- **ShowSearchScope**: True

The code of the CMSSearchDialog control should now look like this:

```
<cms:CMSSearchDialog ID="CMSSearchDialog1" runat="server"
    ShowSearchMode="True" ShowSearchScope="True" />
```

2. Save the changes to the web form. Now right-click it in the Solution explorer and select **View in**

Browser. The resulting page should look like the following diagram (without the descriptions), which shows the structure of the CMSSearchDialog control:



Individual areas correspond with the controls the dialog is composed of.

1.8.9.2.5 Appearance and styling

The appearance of the CMSSearchDialog control is determined by the CSS classes it uses and by some of its properties.

You can use the following CSS classes to modify the design of the control:

Class Name	Description
CMSSearchDialogSearchButton	CSS class of the search button.
CMSSearchDialogSearchForLabel	CSS class of the "Search for:" label.
CMSSearchDialogSearchForTextBox	CSS class of the search expression text box.
CMSSearchDialogSearchModeDropDownList	CSS class of the search mode drop down list.
CMSSearchDialogSearchModeLabel	CSS class of the "Search mode:" label.
CMSSearchDialogSearchScopeDropDownList	CSS class of the search scope drop down list.
CMSSearchDialogSearchScopeLabel	CSS class of the "Search Scope:" label.

The recommended place to define these classes is in a stylesheet in the Kentico CMS administration interface at **Site Manager -> Development -> CSS stylesheets**. These stylesheets can be applied to individual documents (pages) that contain the control in **CMS Desk -> Content -> Edit -> Properties -> General -> CSS stylesheet**.

The design can also be modified in your code behind files by configuring individual controls contained in the CMSSearchDialog. The properties that can be used to access them are listed in the [Configuration](#) topic.

1.8.9.3 CMSSearchResults

1.8.9.3.1 Overview

The CMSSearchResults control is used to display search results according to parameters provided from the [CMSSearchDialog](#) control.

It can also receive search results using the CMS.TreeEngine.TreeProvider.Search() method.

The displayed search results can be formatted using the built-in [DataPager](#) control.

The portal engine equivalent of the CMSSearchResults control is the **Full-text search -> SQL Search results** web part.

The following topics are available to help you familiarize yourself with the CMSSearchResults control:

- [CMSSearchDialog -> Getting started](#) - contains a quick step-by-step tutorial that allows you to learn the basics of using the control
- [Configuration](#) - describes and explains the properties that can be set for the control
- [Structure](#) - demonstrates what individual templates and transformations affect
- [Appearance and styling](#) - describes how the design of the control can be modified

1.8.9.3.2 Configuration

The following properties of the CMSSearchResults control can be set or used in the API:

Property Name	Description	Sample Value
CacheDependencies	List of the cache keys on which the cached data depends. When the cache item changes, the cache of the control is cleared. Each item (dependency) must be on one line. If you leave this property empty, default dependencies will be used.	cms.user all
CacheItemName	Name of the cache item the control will use. By setting this name dynamically, you can achieve caching based on a URL parameter or some other variable - simply enter the value of the parameter. If no value is set, the control stores its content in the item named "URL ControlID".	"mycachename" + Request.QueryString["id"].ToString()
CacheMinutes	Number of minutes the retrieved content is cached for. Zero indicates that the content will not be cached.	

	<p>-1 indicates that the site-level settings should be used.</p> <p>This parameter allows you to set up caching of content so that it doesn't have to be retrieved from the database each time a user requests the page.</p>	
CheckPermissions	Allows you to specify whether to check permissions of the current user. If the value is 'false' (default value) no permissions are checked. Otherwise, only nodes for which the user has read permission are selected.	
ClassNames	Specifies which document type should be selected. Several values separated by a semicolon can be entered.	"cms.news" or "cms.news;cms.article"
CMSSearchDialogID	You can use this property to specify the ID of the source CMSSearchDialog control that provides search parameters.	"CMSSearchDialog1"
CombineWithDefaultCulture	Indicates whether documents from the default culture version should be used if they are not available in the selected culture. This property is applied only if you do not set the TreeProvider property manually.	
CultureCode	Culture code of documents to be selected, such as en-us. If not specified, it's read from the user's session or the default value is used.	"en-us"
DataSource	Gets or sets a DataSet containing values used to fill the items of the control.	
EnablePaging	Enables the paging of search results. True by default.	
FilterControl	Gets or sets the appropriate filter control used to limit the data read by this control.	
FilterName	Gets or sets the code name of the appropriate filter control used to limit the data read by this control.	
FilterOutDuplicates	Indicates if duplicated (linked) documents should be filtered out from the data.	
IgnoreTransformations	Indicates whether the TransformationName property should be ignored and the templates for direct access (described further below) used instead.	
NoResultsLabel	The label control that should be displayed when there are no results.	

OrderBy	ORDER BY part of the SQL statement.	"NewsReleaseDate DESC"
PagerControl	DataPager object used for the paging of search results.	
QueryStringKey	Name of the query string parameter that contains the current page number (if paging is used).	"pagenumber"
SearchExpression	Word(s) to be searched for.	
SearchMode	Search mode - any word, all words or exact phrase.	"AllWords" "AnyWord" "ExactPhrase"
SearchScope	Indicates whether all content or only the current section should be searched.	"SearchAllContent" "SearchCurrentSection"
SelectOnlyPublished	Indicates whether only published documents should be selected.	
SiteName	Specifies the site code name.	
StopProcessing	Indicates if processing of the control should be stopped and the control should not retrieve or display any data.	
TagKey	Overrides the generation of the SPAN tag with a custom tag.	
TransformationName	Name of the transformation applied to displayed search results in format <i><document type code name></i> . <i><transformation name></i> . The default transformation is cms.root.searchresults .	"cms.searchresults"
WhereCondition	WHERE clause used for the SQL search queries.	" DocumentModifiedWhen > '1/1/2007' "
OrderBy	ORDER BY clause used for the SQL search queries.	"DocumentModifiedWhen DESC"

The CMSSearchResults control accepts the following querystring (URL) parameters:

Parameter Name	Description	Sample Value
searchtext	Searched text.	products
searchmode	Search mode.	allwords exactphrase anyword (default value)

1.8.9.3.3 Structure

This topic shows an example of how the CMSSearchResults control can be configured. If you wish to create this example for yourself, please follow the tutorial in the [CMS Search Dialog -> Getting started](#) topic, then continue with the following steps:

1. Add the code marked by the **CMSSearchResults templates** comments between the `<cms:CMSSearchResults>` tags. The overall code of the CMSSearchResults control should look like this:

```
<cms:CMSSearchResults ID="CMSSearchResults1" runat="server" CMSSearchDialogID
="CMSSearchDialog1">

    <%-- CMSSearchResults templates
    ----- %>

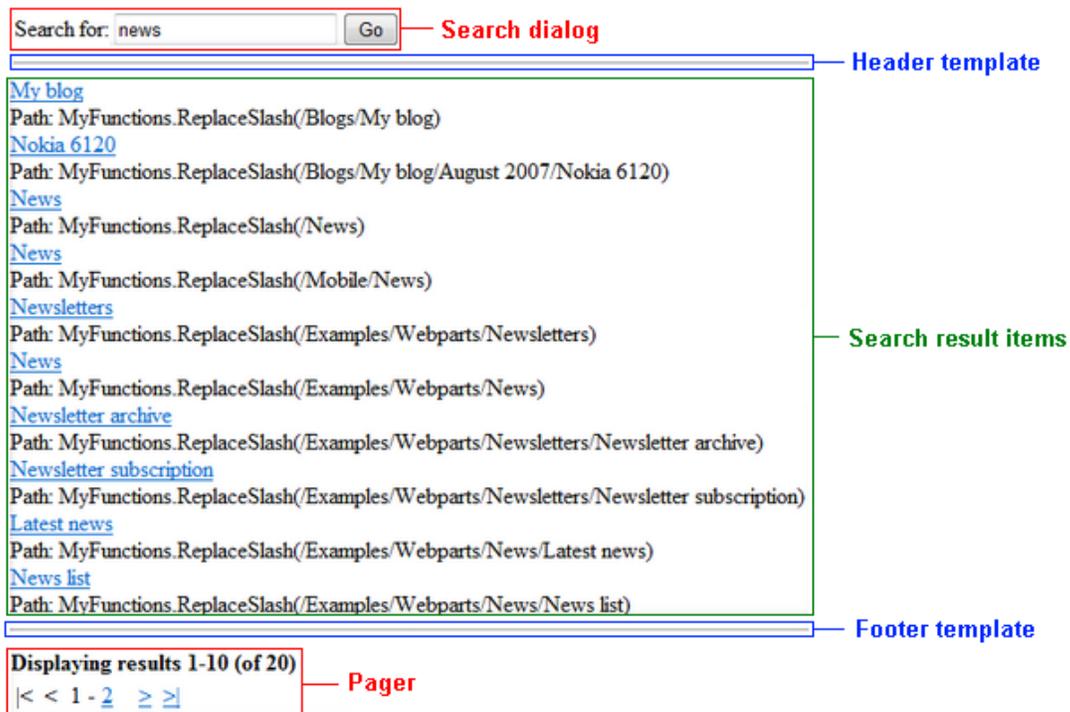
    <HeaderTemplate>
    <hr />
    </HeaderTemplate>

    <FooterTemplate>
    <hr />
    </FooterTemplate>

    <%-- CMSSearchResults templates
    ----- %>

</cms:CMSSearchResults>
```

2. Save the changes to the web form. Now right-click it in the Solution explorer and select **View in Browser**. The resulting page should look like the following diagram (without the descriptions), which shows the structure of the CMSSearchResults control. Individual areas are described below.



- **SearchDialog** - the [CMSSearchDialog](#) control specified by the *CMSSearchDialogID* property.
- **HeaderTemplate** - this area is defined by the code between the `<HeaderTemplate>` tags.
- **Search Result Items** - this area is used to display the search results. It is defined by the transformation specified by the *TransformationName* property (*cms.root.searchresults* by default) or by the code between the `<ItemTemplate tags>` if the *IgnoreTransformations* property is enabled.
- **FooterTemplate** - this areas is defined by the code between the `<FooterTemplate>` tags.
- **Pager** - the built-in [DataPager](#) control, which is used for the paging of search results unless the *EnablePaging* property is set to *false*. It can be accessed through the *PagerControl* property.

1.8.9.3.4 Appearance and styling

The appearance of the `CMSSearchResults` control is determined by the transformation specified in its **TransformationName** property or by its templates.

The following templates can be defined:

Property Name	Description	Sample Value
FooterTemplate	Code of the template used for the footer.	<code><hr /></code>
HeaderTemplate	Code of the template used for the header.	<code><hr /></code>
ItemTemplate	Code of the template applied to search result items.	

1.9 UI Controls

1.9.1 Overview

UI Controls are **user** controls that provide the functionality of standard user interface elements, but offer a higher degree of versatility and customization, and also support additional useful features. Many examples of these controls can be found in the administration interface of Kentico CMS.

Available controls:

- [UniGrid](#)
- [UniSelector](#)

1.9.2 UniGrid

1.9.2.1 Overview

The UniGrid is a **user** control that can be used to display data in a highly customizable and flexible table. It also supports many additional features such as paging, sorting, filtering, row selection and action buttons. It is used extensively in the user interface of Kentico CMS.

Although this control may only be used within a Kentico CMS project, it can display data from an external data source.

Please be aware that using the UniGrid control beyond its most basic functions requires some knowledge of coding and Kentico CMS API.

The following topics are available to help you familiarize yourself with the UniGrid control:

- [Getting started](#) - contains a step-by-step tutorial that allows you to learn the basics of using the control.
- [Implementing custom functionality](#) - contains a tutorial showing how custom modifications can be added via code.
- [Configuration](#) - describes and explains the properties that can be set for the control.
- [UniGrid definition](#) - describes the format of the special configuration options used to determine the structure and behaviour of the UniGrid control.

1.9.2.2 Getting started

The following is a step-by-step tutorial that will show you how to display a list of all users from the Kentico CMS Database in a table and implement a simple action button using the UniGrid user control:

1. Create a new **Web form** called *User_UniGrid.aspx* somewhere in your website installation directory.
2. Add the following directive to the beginning of the page code to register the UniGrid control:

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

3. Modify the `<%@ Page %>` directive at the top of the code as in the following example:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="User_UniGrid.aspx.cs"
Inherits="UniGridExample_User_UniGrid" Theme="Default" %>
```

The **Theme** attribute was added with its value set to `"Default"`, which specifies the default theme used to style the UniGrid control. Please keep in mind that the value of the **Inherits** attribute depends on the location of the web form, so the example above will not match your code exactly.

4. Now add the following code into the content area of the page (by default between the `<div>` tags inside the `<form>` element):

```
<ajaxToolkit:ToolkitScriptManager ID="manScript" runat="server" EnableViewState
="false" />
<asp:Label runat="server" ID="lblInfo" EnableViewState="false" Visible="false" />

<cms:UniGrid ID="UserGrid" runat="server" GridName="User_UniGrid.xml" OrderBy
="UserName" />
```

This adds a standard label control, that will be used to display information messages, and the UniGrid control itself. The label is not necessary for the functioning of the UniGrid, but it can be very convenient, for example to display error messages.

The **ToolkitScriptManager** control is required by the UniGrid control. It is only there to ensure that the example is functional by itself and will usually be included on your website's master page, so you do not have to add it in real-world scenarios.

5. Now create a new **XML file** called `User_UniGrid.xml` in the same location as the web form. It will be used as the configuration file for the UniGrid control and as you can see, it is already specified in the **GridName** property. Now copy the following into the XML file and save it:

```
<?xml version="1.0" encoding="utf-8" ?>
<grid>
  <actions>
    <action name="edit" caption="$General.Edit$" icon="Edit.png" />
  </actions>

  <columns>
    <column source="UserName" caption="$general.username$" width="100%" />
  </columns>

  <query name="cms.user.selectall" columns="UserID, UserName" />
</grid>
```

The basic example above defines only a single action (edit) and one column containing user names without any additional configuration settings. This example uses a query to retrieve the user data. For more details and a full account of the settings that can be set through the UniGrid's XML configuration file, please see the [UniGrid definition](#) topic.

6. Switch to the code behind of the `User_UniGrid.aspx` web form and add the following code into it.

Please keep in mind that the name of the class will be different according to the location of your web form.

[C#]

```
using CMS.SiteProvider;
using CMS.GlobalHelper;

public partial class UniGridExample_User_UniGrid : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        // Assigns a handler for the OnAction event
        UserGrid.OnAction += userGrid_OnAction;
    }

    /// <summary>
    /// Handles the UniGrid's OnAction event.
    /// </summary>
    protected void userGrid_OnAction(string actionName, object actionArgument)
    {
        //Defines the code used to implement the edit action
        if (actionName == "edit")
        {
            //Sets an integer to the value of the actionArgument argument (UserID)
            int userId = ValidationHelper.GetInteger(actionArgument, 0);

            //Gets a UserInfo object of the user with the given ID
            UserInfo ui = UserInfoProvider.GetUserInfo(userId);

            //If user exists
            if (ui != null)
            {
                //Sets the information label to display the full name of the
                edited user
                lblInfo.Visible = true;
                lblInfo.Text = "Edited user: " + HTMLHelper.Encode(ui.
                FullName);
            }
        }
    }
}
```

This code demonstrates how the task that should be performed when an action is used can be implemented. The parameters of **OnAction** event handlers are explained below:

- **string actionName** - is used to identify which action raised the event; this example only has one action, but the UniGrid control often contains more in real scenarios. The name passed into this parameter is defined in the configuration XML file in the **name** attribute of individual **<action>** elements.
- **object actionArgument** - is used to pass the value of a data source column from the UniGrid row for which the action was used. The used column can be specified in the configuration XML file in the **commandargument** attribute of individual **<action>** elements, otherwise the first column in the data source is used by default.

This example only displays the full name of the "edited" user in the label above the UniGrid when the edit

button is clicked, but any required action can be implemented in a similar fashion.

7. Save the changes to all files. Now right-click the web form in the Solution explorer and select **View in Browser**. The resulting page should display a table containing user names and edit action buttons. If you click one of the edit buttons, the full name of the user on the same row will be displayed above the grid, similar to the following:

Edited user: Andrew Jones

Actions	User name ^
	administrator
	Andy
	BradS
	cmsdesigner
	cmsdeskadministrator
	cmseditor
	cmsmarketingmanager
	cmsreader
	gold
	LukeH
	public
	SeanG
	silver

Items per page: 25

1.9.2.3 Implementing custom functionality

This tutorial follows up on the one from the [Getting started](#) topic and will demonstrate how custom functionality can be added to action buttons and columns using the handler of the **OnExternalDataBound** event:

1. Open the web form from the previous tutorial as well as its code behind and XML configuration files.
2. Modify the XML configuration file like in the following code:

```
<?xml version="1.0" encoding="utf-8" ?>
<grid>
  <actions>
    <action name="edit" caption="$General.Edit$" icon="Edit.png"
externalsourcename="edit_modify" />
  </actions>

  <columns>
    <column source="UserName" caption="$general.username$" width="100%"
externalsourcename="user_modify" />
  </columns>

  <query name="cms.user.selectall" columns="UserID, UserName" />
</grid>
```

This defines the **externalsourcename** attributes used to identify the action or column in the **OnExternalDataBound** handler, where the required functionality can be implemented.

3. Switch to the code behind file, and add the sections marked in the following code:

[C#]

```
using System.Data;
using CMS.SiteProvider;
using CMS.GlobalHelper;

public partial class UniGridExample_User_UniGrid : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        // Assigns a handler for the OnAction event
        UserGrid.OnAction += userGrid_OnAction;

        // Assigns a handler for the OnExternalDataBound event
        UserGrid.OnExternalDataBound += userGrid_OnExternalDataBound;
    }

    /// <summary>
    /// Handles the Unigrd's OnExternalDataBound event.
    /// </summary>
    protected object userGrid_OnExternalDataBound(object sender, string
sourceName, object parameter)
    {
        switch (sourceName)
        {
            //Custom code for the edit action
            case "edit_modify":

                //Gets the value of the UserName column from the current data row
                string userName = ValidationHelper.GetString(((DataRowView)((
GridViewRow)parameter).DataItem).Row["UserName"], "");

                //If the user is the administrator
                if (userName == "administrator")
                {
                    //Gets the ImageButton object of the edit action that is
being processed
                    ImageButton button = ((ImageButton)sender);

                    //Disables the button and changes its icon
                    button.ImageUrl = "~/App_Themes/Default/Images/Design/
Controls/UniGrid/Actions/Editdisabled.png";
                    button.Enabled = false;
                }

                break;

            //Custom code for the UserName column
            case "user_modify":
```

```

        //Returns modified user names to be displayed in the UniGrid
        return Convert.ToString(parameter) + " - modified";
    }

    return parameter;
}

/// <summary>
/// Handles the UniGrid's OnAction event.
/// </summary>
protected void userGrid_OnAction(string actionName, object actionArgument)
{
    //Defines the code used to implement the edit action
    if (actionName == "edit")
    {
        //Sets an integer to the value of the actionArgument parameter (UserID)
        int userId = ValidationHelper.GetInteger(actionArgument, 0);

        //Gets a UserInfo object of the user with the given ID
        UserInfo ui = UserInfoProvider.GetUserInfo(userId);

        //If user exists
        if (ui != null)
        {
            //Sets the information label to display the full name of the
            edited user

            lblInfo.Visible = true;
            lblInfo.Text = "Edited user: " + HTMLHelper.Encode(ui.FullName);
        }
    }
}
}

```

This code demonstrates how custom functionality can be handled for actions and columns. The parameters of **OnExternalDataBound** event handlers are explained below:

- **object sender** - is used to pass the ImageButton object of the current action. For columns, it contains a DataRowView of the current row.
- **string sourceName** - is used to identify the action or column for which the functionality is implemented. The name passed into this parameter is defined in the UniGrid's definition in the **externalsourcename** attribute of individual **<action>** or **<column>** elements.
- **object parameter** - is used to pass the value in the current row of a column. For actions, it contains a DataRowView of the current row.

This example modifies the **edit** action to be disabled for the UniGrid row containing the user named *administrator*, and also alters the values displayed in the **UserName** column. Any custom functionality required for actions or columns can be implemented in a similar fashion.

4. Save the changes to all files. Now right-click the web form in the Solution explorer and select **View in Browser**. The resulting page should display a table just like in the example before, but all the values in the **User name** column will be modified, and the edit action for the administrator user will be grayed out and won't be functional:

Actions	User name ^
	administrator - modified
	Andy - modified
	BradS - modified
	cmsdesigner - modified
	cmsdeskadministrator - modified
	cmseditor - modified
	cmsmarketingmanager - modified
	cmsreader - modified
	gold - modified
	LukeH - modified
	public - modified
	SeanG - modified
	silver - modified

Items per page: 25 ▾

1.9.2.4 Configuration

The following properties of the UniGrid control can be set or used in the API:

Property Name	Description	Sample Value
Columns	Specifies the columns that should be loaded from the data source specified in the DataSource property. By default, the values of the first column are passed as the actionArgument parameter of the OnAction event handler. This can be overridden in the definition by specifying a column name in the commandargument attribute of individual <action> elements.	
CompleteWhereCondition	Can be used to get the used WHERE clause including any modifications applied by the filter.	
DataSource	Can be used to gets or set an object (DataSet or DataTable) containing the data to be displayed by the UniGrid control.	
DelayedReload	If enabled, data will not be loaded automatically during the Load event of the page and the ReloadData() method must be called manually instead.	
FilterDirectoryPath	Path to the control (.ascx file) that should be used instead of the default filter. The default relative path is <i>~/CMSAdminControls/UI/UniGrid/Filters/</i> .	
FilteredZeroRowsText	Text to be shown when no rows are displayed after the filter is applied.	
FilterLimit	Determines the minimum amount of rows that must be displayed in the UniGrid before a filter is shown. The default value is read from the CMSDefaultListingFilterLimit web.config key.	
GridName	Contains the name of an external XML file that defines the structure and behaviour of the UniGrid control. For more information, please refer to the UniGrid definition topic.	

GridView	Can be used to access the GridView control encapsulated by the UniGrid.	
HideControlForZeroRows	Indicates whether the control should be hidden when no rows are loaded. The control is not hidden if the filter causes zero rows to be displayed.	
ImageDirectoryPath	Path to the directory that contains images used by the control. The default value is <code>~/App_Themes/Default/Images/Design/Controls/UniGrid/Actions</code> .	
NamedColumns	<p>Gets a dictionary mapping custom names to <i>DataControlField</i> objects that represent the columns of the UniGrid.</p> <p>The names of columns can be specified in the UniGrid's definition through the name attribute of individual <column> elements.</p> <p>This can be used to access the grid's columns in your code.</p> <p>For example:</p> <p>[C#]</p> <pre>UniGrid.NamedColumns["column1"].Visible = false;</pre> <p>When executed, this code would hide the column named <i>column1</i>.</p>	
ObjectType	<p>Can be used to define the data class of the objects that should be loaded as the data source and displayed by the UniGrid control. A list of all available data classes and related information can be found in the <i>CMS_Class</i> database table.</p> <p>Alternatively, the same can be defined in the UniGrid's definition through the <objecttype> element as described in the following topic.</p> <p>Please note that this approach is not supported for classes representing document types (i.e. those whose value in the ClassesDocumentType column is 1). In these cases, you can load the required data by specifying an appropriate query through the Query property.</p>	
OrderBy	The ORDER BY clause used to determine how the UniGrid rows are sorted when the page is first loaded.	
Pager	Can be used to access the UniGridPager control used for paging.	
PageSize	This setting can be used to override the default values offered by the page size selection drop-down list. Values must be separated by commas.	"10,20,##ALL##"

	<p>The ##ALL## macro can be used as a value to indicate that all rows should be displayed.</p> <p>The default value is "25,50,100,##ALL##".</p>	
Query	<p>Can be used to specify the name of the query that should be used to retrieve data from the Kentico CMS database to be displayed by the UniGrid control. The name is entered in format <code><class name>.<query name></code>.</p> <p>Alternatively, the same can be defined in the definition through the <query> element as described in the following topic.</p>	"cms.user.selectallview"
SelectedItems	Gets (as an ArrayList) or sets the currently selected rows from the UniGrid.	
ShowActionsMenu	Indicates whether the header of the actions column should contain a context menu that provides the option to export the data displayed in the grid into various other formats (Excel, CSV or XML).	
ShowObjectMenu	<p>Indicates if an action providing a context menu with object actions should automatically be added to the displayed grid. This requires the data source of the UniGrid to be an object type, specified either through the <code><objecttype></code> element or the ObjectType property.</p> <p>This menu provides options that can be used to Export, Backup, Restore or Destroy individual listed objects. Some types of objects may not have all menu options available.</p> <p>This action is not added if there is another action specified that has a contextmenu attribute or in cases where there are no actions at all defined for the grid.</p> <p>The default value is <i>true</i>.</p>	
SortDirect	The ORDER BY clause reflecting the current row sorting being used by the UniGrid.	
TopN	Specifies the maximum amount of rows that should be selected.	
WhereCondition	Can be used to get the used WHERE clause without modifications applied by the filter.	
ZeroRowsText	Text to be shown when the control is hidden by the HideControlForZeroRows property.	

The following events of the UniGrid control are available:

Event Name	Description
OnAction	Occurs when one of the actions of the control is used. The name of the

	given action is passed as a parameter to the handlers of the event. An example of how it is used can be found in the tutorial found in the Getting started topic.
OnExternalDataBound	Occurs after data is loaded. It is used to implement a custom design or functionality for UniGrid columns, including the action column. An example of how it is used can be found in the Implementing custom functionality topic.
OnBeforeDataReload	This event can be used to perform any actions before the <i>ReloadData()</i> method is executed.
OnAfterDataReload	This event can be used to perform any actions after the <i>ReloadData()</i> method is executed.

1.9.2.5 UniGrid definition

Many configuration options that determine the behavior, design and content of the UniGrid control must be specifically defined. This can either be done either in an external XML configuration file, which is then assigned to the control through its **GridName** property, or directly within the definition of the control in the ASPX markup of the page or user control where the UniGrid is placed.

When using an external XML file, it must be organized according to the structure shown below (some elements are optional):

```
<?xml version="1.0" encoding="utf-8" ?>
<grid>

  <actions>
    <action />
    <separator />
    ...
  </actions>

  <columns>
    <column>
      <tooltip />
      <filter />
      ...
    </column>
    ...
  </columns>

  <objecttype />

  <query>
    <parameter />
    ...
  </query>

  <pager>
    <key name="DefaultPageSize" value="10" />
    ...
  </pager>

```

```
<options>
  <key name="DisplayFilter" value="true" />
  ...
</options>

</grid>
```



Please note

If you use an external XML configuration file to specify the UniGrid's definition, the names of elements and their attributes used must be written in **lower case** to be recognized correctly, since it is case sensitive.

To define the UniGrid directly in the ASPX markup, it is first necessary to register the following namespace at the start of the code (in addition to the UniGrid control):

```
<%@ Register Namespace="CMS.UIControls.UniGridConfig" TagPrefix="ug" Assembly
="CMS.UIControls" %>
```

Then you can simply add elements under the control according to the following structure:

```
<cms:UniGrid runat="server" ID="UniGrid" ... >

  <GridActions>
    <ug:Action />
    <ug:ActionSeparator />
    ...
  </GridActions>

  <GridColumnms>
    <ug:Column>
      <Tooltip />
      <Filter />
      ...
    </ug:Column>
    ...
  </GridColumnms>

  <PagerConfig DisplayPager="true" ... />

  <GridOptions DisplayFilter="true" ... />

</cms:UniGrid>
```

When using this approach, the data source of the control must be specified directly through the UniGrid's properties (**Query**, **ObjectType** or **DataSource**). An advantage of this option is that you may use the IntelliSense in Visual Studio to help find the appropriate elements and attributes.

Individual elements that can be defined for the UniGrid and their attributes are described below:

- [<actions>](#)
- [<columns>](#)
- [<objecttype>](#)
- [<query>](#)
- [<pager>](#)
- [<options>](#)

<actions> (<GridActions>):

This element is used to define a column that contains various possible actions (e.g. Edit, Delete, View...) represented by icons for every row of the UniGrid. Individual actions must be defined by child **<action>** elements.

The following attributes of the **<actions>** element are available:

Attribute Name	Description	Sample Value
cssclass	Specifies the name of a CSS class from the assigned stylesheet to be used to style the appearance of the actions column.	"UniGridCustomActionsColumn"
parameters	A list of columns separated by semicolons that will be usable as parameters in the onclick or menuparameter attributes of child <action> elements .	"AttachmentGUID; AttachmentFormGUID"
showheader	Indicates whether the header of the actions column should be displayed. The default value is true.	
width	Determines the width of the actions column in the UniGrid.	"30%" "100px"

This element may contain **<action>** and **<separator>** child elements.

<action> (<ug:Action>):

This element is used to define individual actions. The implementation of individual actions is handled during the **OnAction** event of the UniGrid control. Any advanced features of individual action buttons, such as defining when a button should be functional, can be implemented in the handler of the **OnExternalDataBound** event.

The following attributes are available:

Attribute Name	Description	Sample Value
caption	Specifies the text used as the tooltip of the image defined in the icon attribute. You can enter the name of a resource string if you start and end it with the \$ character.	"\$General.Delete\$"
commandargument	The name of the column whose value should	

	<p>be passed as the actionArgument parameter of the OnAction event handler.</p> <p>If not defined, the first column of the data source will be used.</p>	
confirmation	<p>The text used in a JavaScript confirmation for the action. Most commonly used as a confirmation for delete type actions. You can enter the name of a resource string if you start and end it with the \$ character.</p>	"\$General.ConfirmDelete\$"
contextmenu	<p>The relative path to a control (.ascx file) that implements a context menu for the action. Controls created for this purpose must inherit from the CMS.ExtendedControls.CMSContextMenuControl class.</p>	"~/CMSAdminControls/UI/UniGrid/Controls/ObjectMenu.ascx"
externalsourcename	<p>Name of the action that is passed as the sourceName parameter of the OnExternalDataBound event handler.</p>	"deletefile"
icon	<p>Name of the image that should be used as the icon of the action. The image must be located in the folder defined by the ImageDirectoryPath property of the UniGrid.</p>	"delete.png"
menuparameter	<p>Contains an array of parameters passed to the control implementing the action's context menu (the path to this control must be specified in the contextmenu attribute). These parameters may be retrieved in the control's code using the GetContextMenuParameter JavaScript function.</p> <p>The columns defined in the parameters attribute of the <actions> element may be entered as parameters using the following expressions:</p> <p>{0} - first parameter {1} - second parameter</p> <p>and so forth.</p>	"new Array('cms.site', '{0}')"
mousebutton	<p>Specifies which mouse button causes the action's context menu to appear (if a context menu is enabled via the contextmenu attribute).</p> <p>If not defined, both mouse buttons open the context menu.</p>	"left" "right"

name	Name of the action. This is passed to the handler of the OnAction event as the actionName parameter.	"delete"
onclick	The JavaScript OnClick function for the given action. It may use the columns defined in the parameters attribute of the <actions> element as parameters, which can be called by using the following expressions: {0} - first parameter {1} - second parameter and so forth.	"alert('{0}');"
Action security		
modulename	This attribute (and the two listed below) may be specified to leverage the security model of Kentico CMS to make the action usable only by a limited group of users. Enter the code name of the module related to the action. You can find information about modules, their permissions and UI elements in the Site Manager -> Development -> Modules interface.	"cms.ecommerce"
permissions	Sets the code name of the permission that users must have to be allowed to perform the action. The permissions must belong to the module specified in the modulename attribute.	"modifyorders"
uielements	If specified, users will need to be allowed to view the given UI element in order to perform the action. The given user interface element must belong to the module specified in the modulename attribute.	"orders.general"
hideifnotauthorized	Indicates if the action should be hidden for users who are not allowed to perform it (as defined by the attributes above).	

Default object menu action

If your UniGrid control uses an object type data source (specified either through the **<objecttype>** definition element or the **ObjectType** property), then an action providing a context menu will automatically be added to the displayed grid.

This menu provides options that can be used to Export, Backup, Restore or Destroy

the listed objects. Some types of objects may not have all menu options available.

This does not occur if you manually specify another action with a **contextmenu** attribute or in cases where there are no actions at all defined for the grid. You can also disable this action by setting the **ShowObjectMenu** property of the UniGrid to *false*.

<separator> (<ug:ActionSeparator>):

This element is used to define a separator between actions. The following attribute is available for it:

Attribute Name	Description	Sample Value
text	Text to be generated in the Literal control between actions.	"&nbsp;"

<columns> (<GridColumns>):

This element represents the main section of the UniGrid. The <columns> element itself has no attributes as each column can have its own settings. Individual columns are defined by child <column> elements.

<column> (<ug:Column>):

This element is used to define columns. Any advanced functionality of the cells in the given column can be implemented in the handler of the **OnExternalDataBound** event.

The following attributes are available for it:

Attribute Name	Description	Sample Value
action	Can be used to set the name of an action that will be performed when the content of this column's cells is clicked. An action with this name must be defined for the UniGrid via the name attribute of an <action> element.	
allowsorting	Indicates whether the column can be used to sort the rows of the UniGrid.	
caption	Specifies the text used as the header for the column. You can enter the name of a resource string if you start and end it with the \$ character.	"\$general.name\$"
commandargument	The name of the column whose value should be passed as the actionArgument parameter of the OnAction event handler	

	<p>when the action specified via the action attribute is used.</p> <p>If not defined, the first column of the data source will be used.</p>	
cssclass	Specifies the name of a CSS class from the assigned stylesheet to be used to style the appearance of the given column.	"UniGridCustomColumn"
externalsourcename	<p>Sets a name for the column that will be passed as the sourceName parameter of the OnExternalDataBound event handler. Used for implementing custom functionality in the cells of the given column.</p> <p>You can use the following values to call built-in functions of the UniGrid to format the content of the column without having to write any code:</p> <ul style="list-style-type: none"> • #yesno - can be set for columns with a source that uses the bit (boolean) data type. The values will be displayed as <i>Yes</i> (colored green) or <i>No</i> (colored red). • #sitename - converts site ID (integer) values into the appropriate site display name for each row. • #sitenameorglobal - converts site ID values into the appropriate site display name for each row. If a record is not related to a specific site (i.e. the site ID is <i>null</i>), then the given cell will display (<i>global</i>) as its value. • #countryname - converts ID (integer) values into the display name of the Country object with the given ID. 	
href	If a URL is entered here, a link to this URL is generated around the content of the cells in this column. Macros {0}, {1}, ... can be used to access parameters defined by the parameters attribute.	"~/page.aspx"
icon	Name of an image that should be added into the column cells after the loaded data. The image must be located in the folder defined by the ImageDirectoryPath property of the UniGrid.	"edit.png"
istext	Indicates whether the content of the column is of type Text or nText. This is used to generate a special OrderBy clause of the query, so it must be set if sorting is enabled	

	for the column.	
localize	Indicates whether localization should be enabled for string values in the column.	
maxlength	Sets the maximum number of characters that can be displayed in the column's cells. The last 3 characters will be replaced by periods.	
name	Can be used to set a custom name for the column, which will be used in the column dictionary accessible through the NamedColumns property of the UniGrid control.	
parameters	Names of the columns used as parameters of the URL generated by the Href attribute. Separated by semicolons.	
source	Name of the column from the data source of the UniGrid that is used as the source for the content of this column. The special macro ##ALL## can be used to specify all columns.	
sort	Used to define the column name to be used for sorting if the ##ALL## macro is used in the source attribute.	
style	The style used for the entire column.	"padding:10px"
visible	Indicates whether the column should be visible.	
width	Determines the width of the column.	"20%" "200px"
wrap	Indicates whether word wrapping is used in the column.	

The column element may contain child **<tooltip>** and **<filter>** elements.

<tooltip>:

When this element is added, a tooltip is displayed when the mouse hovers over the content of the cells in this column. If an icon is present in the cell, the tooltip is displayed over the icon instead of the text. The content of the tooltip can be defined and configured by the following attributes:

Attribute Name	Description	Sample Value
encode	Indicates whether the output of the tooltip should be encoded.	

externalsourcename	Name used in the OnExternalDataBound event for changing the appearance of the tooltip. This can be used to create complex tooltips including images, panels etc.	
source	Name of the column from the data source of the UniGrid that is used as the source of the tooltip.	
width	Determines the width of the tooltip.	

<filter>:

When this element is added, the given column will be used in the UniGrid filter. The following attributes are available to configure the filter:

Attribute Name	Description	Sample Value
format	Can be used to define a custom WHERE clause format to be generated by the default filter. The following expressions can be used: {0} - is resolved into the column name {1} - is resolved into the operator selected in the drop-down list of the default filter {2} - is resolved into the value entered into the textbox of the default filter	" [{0}] {1} {2} "
size	Determines the maximum amount of characters that can be entered into the textbox of the default filter. Available for <i>Text</i> , <i>Integer</i> and <i>Double</i> filter types . The default value is 1000.	
source	Name of the column used in the WHERE clause generated by the filter.	
path	Path to the control (.ascx file) that should be used instead of the default filter for the column. If filled, the type attribute is ignored. The default relative path is ~/CMSAdminControls/UI/UniGrid/Filters/.	
type	The filter type that should be created for the given column.	"Text" "Bool" "Integer" "Double"

<objecttype>:

This element can be used to define the data class of the objects that should be loaded as the data source and displayed by the UniGrid control. A list of all data classes and related information can be found in the *CMS_Class* database table. Please note that this approach is not supported for classes

representing document types (those whose value in the **ClassIsDocumentType** column is 1).

If this element isn't used, a data source must be retrieved by means of the **<query>** element or assigned through the UniGrid control's **DataSource** property before its *ReloadData()* method is called. Alternatively, the **ObjectType** property of the UniGrid control can be used for the same purpose.

The following attributes can be used to define the object type:

Attribute Name	Description	Sample Value
columns	Names of the columns that should be retrieved separated by commas. If empty, all columns will be retrieved. By default, the values of the first column are passed as the actionArgument parameter of the OnAction event handler. This can be overridden for actions by specifying a column name in the commandargument attribute of individual <action> elements.	
name	Code name of the used data class.	"cms.user"

<query>:

This element can be used to specify the system query that will retrieve data from the Kentico CMS database to be displayed by the UniGrid control. If it isn't used, an external data source must be assigned through the UniGrid control's **DataSource** property before its *ReloadData()* method is called. Alternatively, the **Query** property of the Unigrig control can be used for the same purpose.

The following attributes can be used to define the query:

Attribute Name	Description	Sample Value
columns	Names of the columns that should be retrieved by the query separated by commas. If empty, all database columns will be retrieved. By default, the values of the first column are passed as the actionArgument parameter of the OnAction event handler. This can be overridden for actions by specifying a column name in the commandargument attribute of individual <action> elements.	
name	Code name of the used system query in format <class name>.<query name> .	"cms.site.selectsitelist"

The query element may contain **<parameter>** child elements:

<parameter>:

This element can be used to define the value of a parameter inside the specified query.

The following attributes must be filled to define the parameter:

Attribute Name	Description	Sample Value
name	Name of the parameter. Parameters are placed into queries using the following syntax: @<parameter name> For example, if the specified query looked like this: <code>SELECT TOP @customTop FROM CMS_User</code> Then entering <i>customTop</i> into this attribute would cause the value of this element to be used by the query instead of the @customTop expression.	
type	The type of the parameter.	"String" "Int" "Double" "Bool"
value	The value of the parameter.	

<pager> (<PagerConfig>):

This element is used to define the behaviour of the UniGrid pager. You can either add the settings as child <key> elements in the XML configuration file, or as attributes of the <PagerConfig> element when defined directly in the code. The following are available:

Key name	Description	Sample Value
DisplayPager	Indicates if a pager should be included below the UniGrid. <i>True</i> by default.	<code><key name="DisplayPager" value="false" /></code>
DefaultPageSize	Defines the default amount of rows displayed on one UniGrid page. The value must be one of the options offered by the page size selection drop-down list. These values are defined by the PageSizeOptions key.	<code><key name="DefaultPageSize" value="10" /></code>
PageSizeOptions	This setting can be used to override the default values offered by the page size selection drop-down list. Values must be separated by commas. The ##ALL## macro can be used as a value to indicate that all rows should be displayed.	<code><key name="PageSizeOptions" value="10,20,##ALL##" /></code>

	The default value is "25,50,100,##ALL##".	
ShowDirectPageControl	Indicates whether a drop-down list used for direct page selection should be displayed.	<code><key name="ShowDirectPageControl" value="true" /></code>
ShowFirstLastButtons	Indicates whether the buttons that link to the first and last page should be displayed.	<code><key name="ShowFirstLastButtons" value="false" /></code>
ShowPageSize	Indicates whether the page size selection drop-down list should be displayed.	<code><key name="ShowPageSize" value="false" /></code>
ShowPreviousNextButtons	Indicates whether the buttons that link to the previous and next page should be displayed.	<code><key name="ShowPreviousNextButtons" value="false" /></code>
ShowPreviousNextPageGroup	Indicates whether the buttons that link to the next group of page links should be displayed.	<code><key name="ShowPreviousNextPageGroup" value="false" /></code>
VisiblePages	Determines the amount of displayed page links in one group.	<code><key name="VisiblePages" value="5" /></code>

<options> (<GridOptions>):

This element is used to define additional settings and special features of the UniGrid control. You can either add the settings as child **<key>** elements in the XML configuration file, or as attributes of the *<GridOptions>* element when defined directly in the code. The following are available:

Key name	Description	Sample Value
DisplayFilter	Indicates whether a filter should be displayed above the UniGrid. If the amount of displayed rows is lower than the value of the FilterLimit key, the filter will be hidden despite this setting.	<code><key name="DisplayFilter" value="true" /></code>
FilterLimit	Determines the minimum amount of rows that must be displayed in the UniGrid before a filter is shown. The default value is read from the CMSDefaultListingFilterLimit web.config key.	<code><key name="FilterLimit" value="10" /></code>
ShowSelection	Indicates whether a column allowing the selection of rows should be displayed on the left of the UniGrid. This can be used to perform mass actions affecting multiple rows. The selected rows can be accessed through the SelectedItems property of the UniGrid.	<code><key name="ShowSelection" value="true" /></code>
SelectionColumn	Name of the column used as an item in the array of selected rows which can be accessed through the SelectedItems property of the UniGrid. By default the first	<code><key name="SelectionColumn" value="SiteName" /></code>

	column in the data source is used.	
ShowSortDirection	Determines if an arrow showing the sorting direction should be displayed next to the header of the column used for sorting.	<code><key name="ShowSortDirection" value="false" /></code>

1.9.3 UniSelector

1.9.3.1 Overview

The UniSelector is a **user** control that can be used to make a selection from a list of objects of a specified data class, such as users, sites etc. Several different selection modes are supported, as is extensive customization. This control can be found in many places within the user interface of Kentico CMS.

Another advantage provided by the UniSelector over standard selection controls, such as the DropDownList, is greater performance and scalability, since it is optimized to handle very large amounts of objects.

Please be aware that using the UniSelector control beyond its most basic functions requires some knowledge of coding and Kentico CMS API, because when a selection is made, the control only stores the values of the selected objects. Any additional functionality, such as database changes, must be implemented in the handlers of the control's events or using the **Click** event of a Button control used for confirmation. A basic example can be found in the next topic.

The following topics are available to help you familiarize yourself with the UniSelector control:

- [Getting started](#) - contains a step-by-step tutorial that allows you to learn the basics of using the control
- [Configuration](#) - describes the properties and events available for the control

1.9.3.2 Getting started

The following is a step-by-step tutorial that will show you how to use the UniSelector user control to allow the selection of users from the system and perform a basic task with the selected user:

1. Create a new **Web form** called *User_UniSelector.aspx* somewhere in your website installation directory.
2. Add the following directive to the beginning of the page code to register the UniSelector control:

```
<%@ Register src="~/CMSAdminControls/UI/UniSelector/UniSelector.ascx" tagname="UniSelector" tagprefix="cms" %>
```

3. Modify the `<%@ Page %>` directive at the top of the code as in the following example:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="User_UniSelector.aspx.cs" Inherits="UniSelectorExample_User_UniSelector" Theme="Default" %>
```

The **Theme** attribute was added with its value set to *"Default"*, which specifies the default theme used to

style the UniSelector control. Please keep in mind that the value of the **Inherits** attribute depends on the location of the web form, so the example above will not match your code exactly.

4. Now add the following code into the content area of the page (by default between the **<div>** tags inside the **<form>** element):

```
<ajaxToolkit:ToolkitScriptManager ID="manScript" runat="server" EnableViewState="false" />
<table>
  <tr>
    <td>
      <td>
        <cms:UniSelector ID="UserSelector" runat="server" ObjectType="cms.user" SelectionMode="SingleDropDownList" ReturnColumnName="UserName" />
      </td>
    </tr>
    <tr>
      <td>
        <asp:Button runat="server" ID="OKButton" onclick="OKButton_Click" CssClass="SubmitButton" Text="OK" />
      </td>
    </tr>
    <tr>
      <td>
        <asp:Label runat="server" ID="lblButton" Visible="false" />
      </td>
    </tr>
  </table>
```

This adds the UniSelector control, which is configured to allow the selection of user objects from a drop-down list and to use the content of the *UserName* column in its value. For more information about the available properties of the control, please refer to the [Configuration](#) topic. The code also contains a Button and Label control, organized in a basic table layout, which will be used to demonstrate how a basic task can be performed with the value of the UniSelector.

The **ToolkitScriptManager** control is required by the UniSelector control. It is only there to ensure that the example is functional by itself and will usually be included on your website's master page, so you do not have to add it in real-world scenarios.

5. Switch to the code behind of the **User_UniSelector.aspx** web form and add the following code into it. Please keep in mind that the name of the class will be different according to the location of your web form.

[C#]

```
using CMS.GlobalHelper;

public partial class UniSelectorExample_User_UniSelector : System.Web.UI.Page
{
    /// <summary>
    /// Handles the Click event of the submit button.
    /// </summary>
    protected void OKButton_Click(object sender, EventArgs e)
```

```

{
    //Assigns the value of the UniSelector control to be displayed by the Label
    lblButton.Visible = true;
    lblButton.Text = ValidationHelper.GetString(UserSelector.Value, null);
}
}

```

This code causes the user name of the selected user to be displayed when the button on the page is clicked. This code would also work if the UniSelector control used a **SelectionMode** that allowed the selection of multiple users, the user names would all be displayed separated by semicolons.

This example only serves as a demonstration and the selection has no permanent effect, however, any required functionality, such as changes in the database, can be implemented using the Kentico CMS API in a similar fashion. Another option is to use handlers of the UniSelector events listed in the [Configuration](#) topic.

6. Save the changes to both files. Now right-click the web form in the Solution explorer and select **View in Browser**. The resulting page should display a drop-down list containing user names and an **OK** button like in the following image:

A screenshot of a web form. It features a dropdown menu with the text "(none)" and a small downward arrow on the right. Below the dropdown is a green button with the text "OK" in white.

If you select a user and click the button, the user name of the user will be displayed below:

A screenshot of a web form. The dropdown menu now shows "Andrew Jones" with a downward arrow. Below the dropdown is a green button with the text "OK" in white. Underneath the button, the name "Andy" is displayed.

1.9.3.3 Configuration

The following properties of the UniSelector control can be set or used in the API:

Property Name	Description	Sample Value
AdditionalColumns	Contains the names of columns that should be loaded with the objects of the specified data class in addition to those required by default.	
AllowAll	Indicates whether the selector allows the <i>all</i> value.	
AllowEditTextBox	Indicates whether the value of the TextBox displayed in <i>SingleTextBox</i> or <i>MultipleTextBox</i> SelectionMode can be manually edited.	
AllowEmpty	Indicates whether the selector allows an	

	<p>empty value.</p> <p>If enabled, the <i>(none)</i> value is available in <i>SingleDropDownList SelectionMode</i> and the Clear button is displayed in <i>SingleTextBox</i> and <i>MultipleTextBox</i> mode.</p> <p>When an empty value is used, the Value of the control is by default <i>0</i> in <i>SingleDropDownList SelectionMode</i> or an empty string in the remaining modes.</p>	
AllRecordValue	Contains the value used when the <i>(all)</i> item is selected in <i>SingleDropDownList SelectionMode</i> . The default value is <i>-1</i> .	
ButtonImage	Can be used to enter a path to an image. If specified, the selection button is displayed as a LinkButton using this image. Only applies if the SelectionMode is <i>SingleButton</i> or <i>MultipleButton</i> .	"~/App_Themes/Default/Images/SampleImage.png"
CacheMinutes	<p>Number of minutes that the content of the control is cached for, so that it doesn't have to be retrieved from the database each time a user requests the page.</p> <p>Zero indicates that caching will not be used.</p> <p>-1 indicates that the site-level settings should be used.</p> <p>Please refer to the Caching topic to learn more.</p>	
DialogWindowHeight	Determines the default height of the opened selection window.	
DialogWindowName	Can be used to specify the name of the selection window to prevent conflicts between multiple UniSelector controls.	
DialogWindowWidth	Determines the default width of the opened selection window.	
DisplayNameFormat	<p>Used to modify the format of the display names of objects in the selection list.</p> <p>To correctly display values dependant on individual objects, macro expressions in format <i>{%ColumnName%}</i> must be used here. The columns required by the used macros are loaded automatically.</p>	"{%FullName%}, {%Email%}"
EditItemPageUrl	Can be used to specify the URL of a custom	

	<p>page that handles the editing of the selected object. If a value is entered, an edit button that links to the specified URL is displayed. Only available for <i>SingleTextBox</i> and <i>SingleDropDownList</i> SelectionMode.</p> <p>The URL may contain macros in format <code>##<ITEM>ID##</code>, which will be resolved into the value of the selected object's ID column. For example, <code><url>?userid=##USERID##</code> would contain the ID of the currently selected user for a UniSelector set to use the <code>cms.user</code> ObjectType.</p>	
EditWindowName	Can be used to specify the name of the object editing window to prevent conflicts between multiple UniSelector controls.	
EmptyReplacement	Contains a string that is used in the selection list as a replacement value for objects whose display name column is empty.	"N/A"
Enabled	Indicates whether the control is enabled.	
EnabledColumnName	Can be used to specify the name of the column that determines if the selected object is enabled.	
FilterControl	Path to the filter control (.ascx file; must inherit from the <code>CMSAbstractBaseFilterControl</code> class) that will be used for custom filtering of objects in the selection window.	"~/CMSFormControls/Filters/CustomFilter.ascx"
GridName	Path to the XML configuration file of the UniGrid control used to display and select objects in <i>Multiple SelectionMode</i> .	
IconPath	Can be used to enter the path to the image used in the title of the selection window.	
ItemsPerPage	Can be used to set the maximum amount of displayed selected items per page in <i>Multiple SelectionMode</i> .	
LocalizeItems	Indicates whether localization macros should be resolved in the control.	
MaxDisplayedItems	Determines the maximum amount of items displayed in the list in <i>SingleDropDownList SelectionMode</i> if the number of selectable objects is higher than the value of the MaxDisplayedTotalItems property. The remaining objects can be selected in a dialog opened through the (<i>more...</i>) list	

	<p>option.</p> <p>The default value is 25.</p>	
MaxDisplayedTotalItems	<p>If the total number of selectable objects is lower than the value of this property, all of them will be available in the list in <i>SingleDropDownList</i> SelectionMode. If there are more items, the length of the list will match the value of the MaxDisplayedItems property and the <i>(more..)</i> option will be included.</p> <p>The default value is 50.</p> <p>You can also set this value globally for all UniSelectors in your project through the CMSSelectorMaxDisplayedTotalItems key that can be added to the <i><appSettings></i> section of your web.config.</p>	
NewItemPageUrl	<p>Can be used to specify the URL of a custom page that handles the creation of new objects. If a value is entered, a new button that links to the specified URL is displayed. Only available for <i>SingleTextBox</i> SelectionMode.</p>	
NoneRecordValue	<p>Contains the value used when the <i>(none)</i> item is selected in <i>SingleDropDownList</i> SelectionMode. The default value is 0.</p>	
ObjectType	<p>Specifies the data class of the objects to be selected.</p>	"cms.user"
OrderBy	<p>Contains the ORDER BY clause used to determine the order of objects. Also affects the order in the selection window.</p>	
RemoveConfirmation	<p>Can be used to specify the text displayed in the confirmation message displayed when removing selected items from the UniSelector. Entering an empty string disables the confirmation message.</p>	
ResourcePrefix	<p>Determines the prefix that is used in the full names of resource strings (Keys) containing the labels of the various interface elements displayed by the UniSelector. This can be used to assign custom strings to the control.</p> <p>Custom strings can be created at CMS Site Manager -> Development -> UI cultures -> ... edit (✎) a UI culture ... -> Strings. The keys of these strings must use the</p>	"mycustom"

	<p>following format: <ResourcePrefix>.<string name></p> <p>The following string names are available for the UniSelector:</p> <ul style="list-style-type: none">• additems - text caption of the add items button used to open the selection window in <i>Multiple</i> mode• all - name of the list item representing the selection of all available objects in <i>SingleDropDownList</i> mode• clear - text caption of the clear button used in <i>TextBox</i> modes• edit - text caption of the edit button used in <i>SingleTextBox</i> and <i>SingleDropDownList</i> mode• empty - name of the list item representing an empty selection in <i>SingleDropDownList</i> mode• itemname - header text of the column containing the names of objects in the selection window and the UniGrid displaying selected objects in <i>Multiple</i> mode• moreitems - name of the list item that opens the selection window if the maximum amount of list items is exceeded in <i>SingleDropDownList</i> mode• new - text caption of the new button used in <i>SingleTextBox</i> mode• newitem - name of the list item that opens the new item page in <i>SingleDropDownList</i> mode• nodata - text message displayed in <i>Multiple</i> mode if no objects are selected and the ZeroRowsText property is not defined• pleaseselectitem - text of the JavaScript alert displayed when the edit button is used when no object is selected• removeall - text caption of the button used to deselect all objects in <i>Multiple</i> mode• removeselected - text caption of the button used to deselect the specified objects in <i>Multiple</i> mode• select - text caption of the select button used to open the selection window in <i>TextBox</i> and <i>Button</i> modes• selectitem - text of the labels associated with the action elements used by the	
--	--	--

	UniSelector; this is also used to set the title of the selection window	
ReturnColumnName	<p>Specifies the name of the column used for the values of selected objects by the UniSelector. If empty, the ID column is used.</p> <p>To ensure correct functionality of the control, the column must be a unique identifier for the given object type.</p>	
SelectionMode	<p>Determines the design of the selection dialog displayed by the control. The value of this property affects the behaviour of many of the other properties of the UniSelector control.</p> <p>The following modes are available:</p> <ul style="list-style-type: none"> • SingleTextBox - consists of a button that allows the selection of one object and a TextBox displaying the selected value. • MultipleTextBox - consists of a button that allows the selection of multiple objects and a TextBox displaying the selected values. • SingleDropDownList - displays a drop-down list containing objects. If necessary, the selection window can be opened by selecting (<i>more items ...</i>) from the list. • Multiple - consists of a UniGrid control displaying the selected objects and buttons that can be used to add or remove them. • SingleButton - consists of a button that allows the selection of one object. • MultipleButton - consists of a button that allows the selection of multiple objects. 	<p>"SingleTextBox" "MultipleTextBox" "SingleDropDownList" "Multiple" "SingleButton" "MultipleButton"</p>
SpecialFields	<p>Can be used to get or set a two dimensional string array that contains custom items to be displayed in <i>SingleDropDownList</i> SelectionMode. The first value in the array is the name of the item in the list, the second represents the value of that item when it is selected.</p>	
UseDefaultNameFilter	<p>Indicates whether the default name filter should be used in the selection window. Can be used to disable the default filter if a custom filter is specified through the</p>	

	FilterControl property.	
Value	Can be used to get or set the selected value of the control. The column that is used for the values of selected objects can be specified in the ReturnColumnName property.	
ValuesSeparator	Specifies the character used to separate selected values in the case of multiple selection. A semicolon (" ; ") is used by default.	
WhereCondition	Contains the WHERE clause used for the list of objects available for selection.	
ZeroRowsText	Can be used to specify the text displayed when no objects are selected in <i>Multiple SelectionMode</i> .	

The following events of the UniSelector control are available:

Event Name	Description
OnItemsSelected	Occurs when an object or objects are selected in <i>SingleButton</i> and <i>MultipleButton Selection Mode</i> . This event is not raised in other modes.
OnSelectionChanged	Occurs when the set of selected objects is changed. The event is not raised in <i>SingleButton</i> or <i>MultipleButton SelectionMode</i> and may not always be triggered in <i>TextBox</i> modes depending on how the selection is changed. This event is usually used to perform tasks with selected objects in <i>Multiple</i> mode without the need for a confirmation button.

Index

- B -

Basic controls - Listings and viewers

BasicCalendar 47
BasicDataGrid 50
BasicDataList 53
BasicRepeater 57
BasicUniView 61

Basic controls - Navigation

BasicTabControl 42

- C -

CMS controls - Buttons

CMSEditModeButtonAdd 153
CMSEditModeButtonEditDelete 155

CMS controls - Editable regions

CMSEditableImage 158
CMSEditableRegion 160
CMSPageManager 162

CMS controls - Listings and viewers

CMSCalendar 115
CMSDataGrid 117
CMSDataList 120
CMSDocumentValue 123
CMSRepeater 124
CMSUniView 127
CMSViewer 134

CMS controls - Listings and viewers - Custom query

QueryDataGrid 138
QueryDataList 140
QueryRepeater 143
QueryUniView 146

CMS controls - Navigation

CMSBreadCrumbs 77
CMSListMenu 80
CMSMenu 86
CMSSiteMap 91
CMSTabControl 95
CMSTreeMenu 98
CMSTreeView 103

CMS controls - Search

CMSSearchDialog 166

CMSSearchResults 170

- G -

Generic controls

UniView 17

Generic controls - Pagers

DataPager 24
TemplateDataPager 28
UniPager 32

- U -

UI controls

UniGrid 175
UniSelector 196