Data source web parts are responsible for loading data. You can display the retrieved data on the website by connecting the data source to a listing web part.

To develop custom data source web parts, you need to create two separate user controls:

1. The first control loads the data. This control must inherit from **CMSBaseDataSource** (found in the *CMS.DocumentEngine.Web.UI* namespace).
2. The second user control contains the first control in its markup and serves as the source file for the actual web part. Must inherit from **CMSAbstractWebPart**.

Example - Creating a user data source

The following example shows how to create a custom data source that loads user data from the Kentico database.

Open your web project in Visual Studio and create two user controls according to the sections below.

Note

Set the values of the *Inherits* attributes in the user control declarations and the class names according to the names and locations of your controls. If your Kentico project was installed as a web application, you need to rename the *CodeFile* attributes to *Codebehind*.

The code in the example assumes that the control is placed inside the **~/CMSGlobalFiles** folder (you may need to manually create this folder if you wish to use the default code). When developing your own controls, you can use another location and adjust the *Inherits* attribute and the corresponding class names as required.

DataSourceControl.ascx

```
<%@ Control Language="C#" AutoEventWireup="true" CodeFile="DataSourceControl.ascx.cs"
Inherits="CMSGlobalFiles_DataSourceControl" %>
<%--Leave the markup of the first user control empty--%>
```

DataSourceControl.ascx.cs

```
using System;

using CMS.DocumentEngine.Web.UI;
using CMS.Membership;
using CMS.FormEngine.Web.UI;

/// <summary>
/// User data source. Loads all users according to a specified Where condition.
/// </summary>
public partial class CMSGlobalFiles_DataSourceControl : CMSBaseDataSource
{
    /// <summary>
    /// Assigns a handler for the 'OnFilterChanged' event if there is a filter
attached to the data source.
    /// You do not need to implement this method if you do not plan to use filters
with the data source web part.
    /// </summary>
    protected override void OnInit(EventArgs e)
    {
        if (SourceFilterControl != null)
        {
            SourceFilterControl.OnFilterChanged += new ActionEventHandler
(DataFilter_OnFilterChanged);
        }

        base.OnInit(e);
    }

    /// <summary>
    /// Handles the OnFilterChanged event.
    /// You do not need to implement this method if you do not plan to use filters
with the data source web part.
    /// </summary>
    void DataFilter_OnFilterChanged()
    {
        // Clears the old data
        InvalidateLoadedData();

        // Raises the change event
        this.RaiseOnFilterChanged();
    }

    /// <summary>
    /// Loads the data of the data source.
    /// </summary>
    protected override object GetDataSourceFromDB()
    {
        // Initializes the data properties according to the filter settings
        if (SourceFilterControl != null)
        {
            SourceFilterControl.InitDataProperties(this);
        }

                // Loads the data. The WhereCondition and OrderBy properties are
inherited from the parent class.
                DataSource = UserInfoProvider.GetUsersDataWithSettings()
                                                                                .Where
(WhereCondition)
                                                                                .
OrderBy(OrderBy)
                                                                                .
TypedResult;

        return DataSource;
    }
}
```

DataSourceWebPart.ascx

```
<%@ Control Language="C#" AutoEventWireup="true" CodeFile="~/CMSGlobalFiles
/DataSourceWebPart.ascx.cs" Inherits="CMSGlobalFiles_DataSourceWebPart" %>

<%@ Register Src="~/CMSGlobalFiles/DataSourceControl.ascx" TagPrefix="cms" TagName="
DataSourceControl" %>

<cms:DataSourceControl ID="srcUsers" runat="server" />
```

DataSourceWebPart.ascx.cs

```
using CMS.PortalEngine.Web.UI;
using CMS.Helpers;

public partial class CMSGlobalFiles_DataSourceWebPart : CMSAbstractWebPart
{
    /// <summary>
    /// Gets or sets the value of the WhereCondition web part property.
    /// </summary>
    public string WhereCondition
    {
        get
        {
            return ValidationHelper.GetString(this.GetValue("WhereCondition"), "");
        }
        set
        {
            this.SetValue("WhereCondition", value);
            srcUsers.WhereCondition = value;
        }
    }

    /// <summary>
    /// Gets or sets the value of the OrderBy web part property.
    /// </summary>
    public string OrderBy
    {
        get
        {
            return ValidationHelper.GetString(this.GetValue("OrderBy"), "");
        }
        set
        {
            this.SetValue("OrderBy", value);
            srcUsers.OrderBy = value;
        }
    }

    /// <summary>
    /// Gets or sets the value of the FilterName web part property.
    /// </summary>
    public string FilterName
    {
        get
        {
            return ValidationHelper.GetString(this.GetValue("FilterName"), "");
        }
        set
        {
            this.SetValue("FilterName", value);
            srcUsers.SourceFilterName = value;
        }
    }

    /// <summary>
    /// Content loaded event handler.
    /// </summary>
    public override void OnContentLoaded()
    {
        base.OnContentLoaded();
        SetupControl();
    }

    /// <summary>
    /// Initializes the properties of the internal data source control.
    /// </summary>
    protected void SetupControl()
    {
        if (this.StopProcessing)
        {
            // Does nothing if the web part is disabled
        }
        else
        {
            this.srcUsers.WhereCondition = this.WhereCondition;
            this.srcUsers.OrderBy = this.OrderBy;

            // Sets the inner data source's name according to the web part's 'Web part
control ID' property.
            // This allows listing web parts to connect to the inner control.
            // The identifier property is named 'FilterName' because data sources
inherit the property from
            // a base class shared with filter objects.
            this.srcUsers.FilterName = (string)this.GetValue("WebPartControlID");

            // Connects the attached filter, as specified by the 'Filter name'
property of the web part
            this.srcUsers.SourceFilterName = this.FilterName;
        }
    }
}
```

If your Kentico project was installed as a web application, you must **Build** the project.

Registering the web part

Register the **DataSourceWebPart.ascx** control as a new web part in Kentico.

1. Open the **Web parts** application.
2. Create a new web part. See Creating new web parts for details.
3. Type *Custom user data source* as the web part's **Display name**.
4. Set the **Type** of the web part to *Data source* on the **General** tab.
5. On the **Properties** tab, define the following properties for the web part:

    - **Field name**: OrderBy
    - **Data type**: Text
    - **Size**: 500
    - **Field caption**: SQL Order by clause
    - **Form control**: Order by

    - **Field name**: WhereCondition
    - **Data type**: Text
    - **Size**: 500
    - **Field caption**: SQL Where condition
    - **Form control**: Where condition

    - **Field name**: FilterName
    - **Data type**: Text
    - **Size**: 200

- **Field caption**: Filter name
- **Form control**: Text box

Result

The custom data source web part is now ready and you can try out its functionality.

Open the **Pages** application, create a new page and add the following web parts on the **Design** tab:

Web part	Instructions
Custom user data source	Type *UsersFilter* into the **Filter name** property.
Users filter	Leave the default property values. The **Web part control ID** is set to *UsersFilter* by default, which connects the filter to the custom data source.
Basic repeater	1. Enter the ID of the Custom user data source web part into the **Data source name** property: *CustomUserDataSource* 2. Set an appropriate **Transformation name** for displaying users, for example: *Community.Transformations.MembersList*

The page displays a list of all users in the system. You can modify the list by configuring the *SQL Where condition* and *SQL Order by clause* properties of the data source. Visitors can dynamically narrow down the list directly on the page using the connected filter.

Example – Implementing data source caching

This example builds on the previous one to add caching functionality for the retrieved data. Cached data does not need to be retrieved every time the page is requested, which increases the website performance and lowers database overhead.

To implement data source caching for the previous example, you need to:

1. Add the following code to the **DataSourceWebPart.ascx.cs** file:

DataSourceWebPart.ascx.cs

```
public partial class CMSGlobalFiles_DataSourceWebPart : CMSAbstractWebPart
{

...

    /// <summary>
    /// Gets or sets the cache item name.
    /// </summary>
    public override string CacheItemName
    {
        get
        {
            return base.CacheItemName;
        }
        set
        {
            base.CacheItemName = value;
            srcUsers.CacheItemName = value;
        }
    }

    /// <summary>
    /// Cache dependencies, each cache dependency on a new line.
    /// </summary>
    public override string CacheDependencies
    {
        get
        {
            return ValidationHelper.GetString(base.CacheDependencies, srcUsers.
CacheDependencies);
        }
        set
        {
            base.CacheDependencies = value;
            srcUsers.CacheDependencies = value;
        }
    }

        /// <summary>
    /// Gets or sets the cache minutes.
    /// </summary>
    public override int CacheMinutes
    {
        get
        {
            return base.CacheMinutes;
        }
        set
        {
            base.CacheMinutes = value;
            srcUsers.CacheMinutes = value;
        }
    }

...

}
```

2. Replace the **SetupControl()** method implementation (in **DataSourceWebPart.ascx.cs**) with the following:

```
protected void SetupControl()
{
    if (this.StopProcessing)
    {
        // Does nothing if the web part is disabled
    }
    else
    {
        this.srcUsers.WhereCondition = this.WhereCondition;
        this.srcUsers.OrderBy = this.OrderBy;

        // Sets the inner data source's name according to the web part's 'Web
part control ID' property.
        // This allows listing web parts to connect to the inner control.
        // The identifier property is named 'FilterName' because data sources
inherit the property from
        // a base class shared with filter objects.
        this.srcUsers.FilterName = (string)this.GetValue("WebPartControlID");

        // Connects the attached filter, as specified by the 'Filter name'
property of the web part
        this.srcUsers.SourceFilterName = this.FilterName;

        this.srcUsers.CacheItemName = this.CacheItemName;
        this.srcUsers.CacheDependencies = this.CacheDependencies;
        this.srcUsers.CacheMinutes = this.CacheMinutes;
    }
}
```

3. (Optional) In **DataSourceControl.ascx.cs**, override the **GetDefaultCacheKey()** and **GetDefaultCacheDependencies()** m ethods, generating the cache item name and cache dependencies, with your custom logic. The code sample below shows the default implementation of these methods for the *UsersDataSource* web part.

```
using CMS.Helpers;

...

/// <summary>
/// Gets default cache key.
/// </summary>
protected override object[] GetDefaultCacheKey()
{
    return new object[] { "customuserdatasource", CacheHelper.BaseCacheKey,
ClientID, WhereCondition, OrderBy};
}


/// <summary>
/// Gets the default cache dependencies for the data source.
/// </summary>
public override string GetDefaultCacheDependencies()
{
    // Get default dependencies
    string result = base.GetDefaultCacheDependencies();

    if (result != null)
    {
        result += "\n";
    }

    result += "cms.user|all";

    return result;
}
```

4. If your Kentico project was installed as a web application, **Build** the project.
5. Open the *Custom user data source* web part in the **Web parts** application and switch to the **Properties** tab.
6. Click **...** next to the **New field** button and select **New category**.
7. Enter *Data caching* as the **Category caption** and move the category below the existing properties.
8. Add the following fields under the category:

    - **Field name**: CacheItemName
    - **Data type**: Text
    - **Size**: 500
    - **Field caption**: Cache item name
    - **Form control**: Text box

    - **Field name**: CacheMinutes
    - **Data type**: Integer number
    - **Field caption**: Cache minutes
    - **Form control**: Text box

    - **Field name**: CacheDependencies
    - **Data type**: Long text
    - **Field caption**: Cache dependencies
    - **Form control**: Cache dependencies

9. Insert the web part onto a page and set the values of the newly created fields to, for example, the following:

- **CacheItemName**: CustomUserDataSourceCache
- **CacheMinutes**: 10
- **CacheDependencies**: cms.user|all

The data is now set to cache for a duration of 10 minutes (controlled by the **CacheMinutes** field) and any changes to **cms.user** ob jects automatically invalidate the cache (controlled by the **CacheDependencies** field).