The Kentico API allows you to cache data in custom code. To cache data in custom code, use the **CacheHelper.Cache** method.

This type of caching is applicable to any API result and is used internally by Kentico web parts. The same caching API is also available in macros.

We recommend caching in custom code if you aren't able to use higher levels of caching or for frequent API calls (used several times on a single page).

The following example shows the code behind of a custom user control that:

- Loads user data from the Kentico database (with caching)
- Displays the data using a BasicRepeater control

```
using System;
using System.Data;

using CMS.Helpers;
using CMS.Membership;

public partial class CachedUsers : System.Web.UI.Page
{
        protected void Page_Load(object sender, EventArgs e)
        {
                // Ensures loading and caching of data
                // Uses CacheSettings that cache the data for 10 minutes under the
cache key "customdatasource|..."
                // Automatically checks whether the given key is already in the cache
                DataSet data = CacheHelper.Cache(cs => LoadUsers(cs), new CacheSettings
(10, "customdatasource|" + UserBasicRepeater.ClientID));

                // Assigns the data to the BasicRepeater control
                UserBasicRepeater.DataSource = data;
                UserBasicRepeater.DataBind();
        }

        // Method that loads the required data
        // Called only if the data doesn't already exist in the cache
        private DataSet LoadUsers(CacheSettings cs)
        {
                // Loads all user accounts from the database
                DataSet result = UserInfoProvider.GetUsers();

                // Checks whether the data should be cached (based on the
CacheSettings)
                if (cs.Cached)
                {
                        // Sets a cache dependency for the data
                        // The data is removed from the cache if the objects
represented by the dummy key are modified (all user objects in this case)
                        cs.CacheDependency = CacheHelper.GetCacheDependency("cms.
user|all");
                }

                return result;
        }
}
```

The **Cache** method checks if the key specified by the **CacheSettings** object is in the cache:

- If yes, the method directly loads the data from the cache.
- If not, the code calls the custom private method (**LoadUsers**) with the *CacheSettings* as a parameter. The private method loads the data from the database, sets a cache dependency and saves the key into the cache for the specified number of minutes

You can use the caching API when handling data anywhere in your code.

Tip

If you do not need to access or set the **CacheSettings** in the method that loads the data, you can use a simplified version of the **Cache** method:

```
DataSet data = CacheHelper.Cache(LoadUsers, new CacheSettings(10, "customkey"));

private DataSet LoadUsers()
{
        ...
}
```

CacheSettings

When calling the **Cache** method, you need to specify a **CMS.Helpers.CacheSettings** object as a parameter. The settings configure the cache key that stores the data. If you set the same cache key name for multiple data loading operations, they share the same cached value.

You can work with the following properties of the *CacheSettings:*

CacheSettings property	Type	Description
CacheMinutes	int	The number of minutes for which the cache stores the loaded data. The default value is 10 minutes. We recommend using an interval of 1 to 60 minutes.
CacheDependency	CMSCacheDependency	Sets dependencies for the cache key (use the *CacheHelper.GetCacheDependency* method to get the dependency object). Make sure you always set cache dependencies, unless the *CacheMinutes* interval is set to an extremely short time or the cached content is predominantly static.
BoolCondition	bool	A boolean condition that must be fulfilled (true) in order to cache the loaded data.
Cached	bool	Indicates whether the data should be cached (based on the cache minutes and condition).
AllowProgressiveCaching	bool	Enables or disabled progressive caching, which ensures that multiple threads accessing the same code only load the data once and reuse the result.