Custom fields can help you store and organize additional data related to users. For example, if you need to gather data not collected in the system by default, add custom fields and use them to store any additional information required.

This page describes how to:

- Add custom fields to user objects
- Access the added fields from the Kentico.Membership ASP.NET Identity implementation

## Adding custom fields to users

Custom fields may store values such as text, date and time, and boolean values. To define custom fields:

1. Open the **Modules** application.
2. **Edit** (🖉) the **Membership** module.
3. Switch to the **Classes** tab.
4. **Edit** (🖉) the **User** class.
5. Switch to the **Fields** tab.
6. Create new fields based on your requirements using the field editor.
7. Click **Save**.

You have added custom fields for the user object in Kentico. In the administration interface, added custom fields are displayed on a separate **Custom fields** tab when editing users in the **Users** application.

## Accessing custom user fields in Kentico's ASP.NET Identity implementation

> **Hotfix 12.0.34 or newer required**
>
> The Kentico membership API used in this scenario is available after installing hotfix 12.0.34 or newer. The *UserManager*, *UserStore*, and *SignInManager* non-generic types existing in prior versions are still usable and work with the default *User* object.

Kentico MVC applications use an implementation of the ASP.NET Identity membership system and the OWIN standard (provided in the *Kentico.Membership* namespace, see Integrating Kentico membership for detailed information).

By default, the implementation uses the **Kentico.Membership.User** type to represent users. Objects of this type wrap *CMS.Membership.UserInfo* objects (which represent the user in Kentico), and facilitate the transfer of user data between the Identity implementation and internal Kentico logic.

If you defined any custom user fields that you wish to handle via *Kentico.Membership* types (e.g., when creating or updating user information, performing authentication or authorization, etc.), you need to ensure proper mapping between the *UserInfo* object and the *User* object.

1. Open the MVC project in Visual Studio.
2. Create a new class that inherits from *Kentico.Membership.User.*
3. Declare properties that correspond to the custom fields you specified in the administration interface.

   > Besides exposing custom fields added to the user object via the administration interface, you can also expose existing fields of the user object that are not accessible by default via the *Kentico.Membership.User* type. For example, the *MiddleName* user field or advanced user fields from the *UserSettings* class. See the sample **ExtendedUser** class below for an example.

4. Override the **MapFromUserInfo** and **MapToUserInfo** methods and:
   - Call the base implementation of the methods. This maps all properties of the *Kentico.Membership.User* class such as *FirstName* and *Email.*
   - Get and set values of the custom properties you wish to have available using **UserInfo.GetValue** and **UserInfo.SetValue**.

**Declaring an ExtendedUser class that extends the default Kentico.Membership.User object**

```
using CMS.Membership;

using Kentico.Membership;


namespace MembershipCustomization
{
    // Extends the default Kentico.Membership.User object
    public class ExtendedUser : User
    {
        // Exposes the existing 'MiddleName' property of the 'UserInfo'
object
        public string MiddleName
        {
            get;
            set;
        }


        // Property that corresponds to a custom field specified in the
administration interface
        public string CustomField
        {
            get;
            set;
        }


        // Ensures field mapping between Kentico's user objects and the
Kentico.Membership ASP.NET Identity implementation
        // Called when retrieving users from Kentico via Kentico.Membership.
KenticoUserManager<TUser>
        public override void MapFromUserInfo(UserInfo source)
        {
            // Calls the base class implementation of the MapFromUserInfo
method
            base.MapFromUserInfo(source);

            // Maps the 'MiddleName' property to the extended user object
            MiddleName = source.MiddleName;

            // Sets the value of the 'CustomField' property
            CustomField = source.GetValue<string>("CustomField", null);
        }


        // Ensures field mapping between Kentico's user objects and the
Kentico.Membership ASP.NET Identity implementation
        // Called when creating or updating users using Kentico.Membership.
KenticoUserManager<TUser>
        public override void MapToUserInfo(UserInfo target)
        {
            // Calls the base class implementation of the MapToUserInfo
method
            base.MapToUserInfo(target);

            // Maps the 'MiddleName' property to the target 'UserInfo'
```

```
object
            target.MiddleName = MiddleName;

            // Sets the value of the 'CustomField' custom user field
            target.SetValue("CustomField", CustomField);
        }
    }
}
```

5. In the Owin startup pipeline, register new or add additional *KenticoUserManager*, *KenticoUserStore*, and *KenticoSignInManager* types from the *Kentico.Membership* namespace. The types need to specify the extended user object as their generic parameter (*ExtendedUser* in this case). Register these generic types instead of the original *Manager* and *Store* types (which do not have the *Kentico* prefix).

```
using Owin;

using CMS.SiteProvider;

using Kentico.Membership;


public partial class Startup
{
        public void Configuration(IAppBuilder app)
        {
                // Registers Kentico.Membership Identity types with the
'ExtendedUser' user object
                app.CreatePerOwinContext(() => KenticoUserManager<ExtendedUser>.
Initialize(app, new KenticoUserManager<ExtendedUser>(new
KenticoUserStore<ExtendedUser>(SiteContext.CurrentSiteName))));
                app.CreatePerOwinContext<KenticoSignInManager<ExtendedUser>>
(KenticoSignInManager<ExtendedUser>.Create);

            ...
        }
}
```

6. When retrieving the registered types from the Owin context via **HttpContext.GetOwinContext().Get**, retrieve the type with the associated generic parameter, for example:

```
KenticoSignInManager<ExtendedUser> KenticoSignInManager = HttpContext.
GetOwinContext().Get<KenticoSignInManager<ExtendedUser>>();
```

   This ensures that you retrieve the type capable of working with the extended user object.

The *Kentico.Membership* Identity implementation is now able to work with the extended user object together with any additional properties and logic it contains.