The **Kentico.Membership** integration package allows you to set up registration functionality for the visitors of your MVC website. The system stores the user data in the shared Kentico database.

Before you start working on registration, you need to perform the following:

- Integrate the Kentico membership implementation into your MVC project
- Set up authentication functionality

Both scenarios are described on the Working with users on MVC sites page.

Compatibility with Kentico registration features

Kentico features related to user registration are NOT supported by default for registrations that occur on MVC sites. For example:

- The default Kentico functionality for email confirmation of registrations. However, you can use the integration API to set up your own solution – see the Adding email confirmation for registration section.
- Password policies - you need to define password requirements in your application's code (for example by configuring the PasswordValidator property during the initialization of *Kentico.Membership.UserManager*).
- Reserved user names (*Settings -> Security & Membership -> Reserved user names*)
- Approval of new user accounts by administrators (*Settings -> Security & Membership -> Registration requires administrator's approval*)
- Site prefixes for user names (*Settings -> Security & Membership -> Use site prefix for user names*)

One setting that does apply to registration on MVC sites is the **Password format**, which you can configure within Kentico in **Settings -> Security & Membership -> Passwords**. See Setting the user password format for more information. The integration API ensures that any users registered from your MVC site have their passwords stored in the format configured for the Kentico application.

Ensuring the correct password format

If your Kentico application uses custom salt values when generating password hashes, you also need to set the same values for the MVC application.

Check the *appSettings* section of your Kentico application's web.config for the following keys:

- **CMSPasswordSalt**
- **CMSUserSaltColumn** (obsolete key used only for backward compatibility)

If either of the keys are present, copy them to the web.config of your MVC project.

Setting up basic registration

Use the following approach to develop actions that allow visitors to register on your website:

Tip: To view the full code of a functional example directly in Visual Studio, download the Kentico MVC solution from GitHub and inspect the **LearningKit** project. You can also run the Learning Kit website after connecting the project to a Kentico database.

1. Create a new controller class in your MVC project or edit an existing one.
2. Prepare a property that gets an instance of the **Kentico.Membership.UserManager** class for the current request – call *HttpContext.GetOwinContext().Get<UserManager>()*.
3. Implement two registration actions – one basic GET action to display the registration form and a second POST action to handle the creation of new users when the form is submitted.
4. Perform the following steps within the POST action:
    a. Prepare a **Kentico.Membership.User** object based on the posted registration data.
    b. Call the **UserManager.CreateAsync** method to create the user in the Kentico database.

c. (Optional) If the registration is successful, sign in the user under the new account via the **SignInAsync** method of the current request's *SignInManager* instance (See Working with users on MVC sites for more information about the *SignInManager).*

Registration controller example

```
 »using System;
using System.Web;
using System.Web.Mvc;
using System.Threading.Tasks;

using Microsoft.AspNet.Identity;
using Microsoft.AspNet.Identity.Owin;

using Kentico.Membership;

using CMS.EventLog;

namespace LearningKit.Controllers
{
    public class RegisterController : Controller
    {
        /// <summary>
        /// Provides access to the Kentico.Membership.SignInManager instance.
        /// </summary>
        public SignInManager SignInManager
        {
            get
            {
                return HttpContext.GetOwinContext().Get<SignInManager>();
            }
        }

        /// <summary>
        /// Provides access to the Kentico.Membership.UserManager instance.
        /// </summary>
        public UserManager UserManager
        {
            get
            {
                return HttpContext.GetOwinContext().Get<UserManager>();
            }
        }

        /// <summary>
        /// Basic action that displays the registration form.
        /// </summary>
        public ActionResult Register()
        {
            return View();
        }

        /// <summary>
        /// Handles creation of new users when the registration form is submitted.
        /// Accepts parameters posted from the registration form via the
RegisterViewModel.
        /// </summary>
        [HttpPost]
        [ValidateAntiForgeryToken]
```

```
        [ValidateInput(false)]
        public async Task<ActionResult> Register(RegisterViewModel model)
        {
            // Validates the received user data based on the view model
            if (!ModelState.IsValid)
            {
                return View(model);
            }

            // Prepares a new user entity using the posted registration data
            Kentico.Membership.User user = new User
            {
                UserName = model.UserName,
                Email = model.Email,
                FirstName = model.FirstName,
                LastName = model.LastName,
                Enabled = true // Enables the new user directly
            };

            // Attempts to create the user in the Kentico database
            IdentityResult registerResult = IdentityResult.Failed();
            try
            {
                registerResult = await UserManager.CreateAsync(user, model.
Password);
            }
            catch (Exception ex)
            {
                // Logs an error into the Kentico event log if the creation of
the user fails
                EventLogProvider.LogException("MvcApplication",
"UserRegistration", ex);
                ModelState.AddModelError(String.Empty, "Registration failed");
            }

            // If the registration was not successful, displays the registration
form with an error message
            if (!registerResult.Succeeded)
            {
                foreach (string error in registerResult.Errors)
                {
                    ModelState.AddModelError(String.Empty, error);
                }
                return View(model);
            }

            // If the registration was successful, signs in the user and
redirects to a different action
            await SignInManager.SignInAsync(user, true, false);
            return RedirectToAction("Index", "Home");
        }
    }
}
```

5. We recommend creating a view model for your registration action *(RegisterViewModel* in the example above). The view model allows you to:
    - Pass parameters from the registration form (user names, email address, password and confirmation field, etc.).

- Use data annotations to define validation and formatting rules for the registration data. See System.ComponentModel.DataAnnotations on MSDN for more information about the available data annotation attributes.

Registration view model example

```
 »using System.ComponentModel;
using System.ComponentModel.DataAnnotations;

public class RegisterViewModel
{
    [Required(ErrorMessage = "The User name cannot be empty.")]
    [DisplayName("User name")]
    [MaxLength(100, ErrorMessage = "The User name cannot be longer than 100
characters.")]
    public string UserName
    {
        get;
        set;
    }

    [DataType(DataType.EmailAddress)]
    [Required(ErrorMessage = "The Email address cannot be empty.")]
    [DisplayName("Email address")]
    [EmailAddress(ErrorMessage = "Invalid email address.")]
    [MaxLength(254, ErrorMessage = "The Email address cannot be longer than 254
characters.")]
    public string Email
    {
        get;
        set;
    }

    [DataType(DataType.Password)]
    [Required(ErrorMessage = "The Password cannot be empty.")]
    [DisplayName("Password")]
    [MaxLength(100, ErrorMessage = "The Password cannot be longer than 100
characters.")]
    public string Password
    {
        get;
        set;
    }

    [DataType(DataType.Password)]
    [DisplayName("Password confirmation")]
    [MaxLength(100, ErrorMessage = "The Password cannot be longer than 100
characters.")]
    [Compare("Password", ErrorMessage = "The entered passwords do not match.")]
    public string PasswordConfirmation
    {
        get;
        set;
    }

    [DisplayName("First name")]
    [Required(ErrorMessage = "The First name cannot be empty.")]
    [MaxLength(100, ErrorMessage = "The First name cannot be longer than 100
characters.")]
```

```
    public string FirstName
    {
        get;
        set;
    }

    [DisplayName("Last name")]
    [Required(ErrorMessage = "The Last name cannot be empty.")]
    [MaxLength(100, ErrorMessage = "The Last name cannot be longer than 100
characters.")]
    public string LastName
    {
        get;
        set;
    }
}
```

6. Design the user interface required for registration on your website:
    - Create a view for the *Register* action and display an appropriate registration form. We recommend using a strongly typed view based on your registration view model.
    - Add a registration button or link that targets the *Register* action into an appropriate location on your website.

Visitors can now register new user accounts on your MVC site. Upon successful registration, the system creates a new user in the connected Kentico database. The new user is automatically enabled in Kentico and assigned to the site where the registration occurred. The user can then sign in on the website.

Kentico matches sites to MVC applications based on the **Presentation URL** or **Domain name** set for sites in the **Sites** application.

Adding email confirmation for registration

The membership integration API allows you to set up a more advanced registration process that requires email confirmation (double opt-in).

Configuring the confirmation email settings

The confirmation emails are sent by the email engine provided by the Kentico API. Configure the email settings through the administration interface of the connected Kentico application:

- Set up SMTP servers in Kentico. See Configuring SMTP servers for more information.
- You can use the Kentico **Email queue** application to monitor the emails (if the email queue is enabled).
- Set the sender address for the confirmation emails in **Settings -> System -> No-reply email address**.

1. Create a new controller class in your MVC project or edit an existing one.
2. Prepare a property that gets an instance of the **Kentico.Membership.UserManager** class for the current request – call *HttpContext.GetOwinContext().Get<UserManager>()*.
3. Implement two registration actions – one basic GET action to display the registration form and a second POST action to handle the creation of new users and sending of confirmation emails.
4. Perform the following steps within the POST action:
    - Prepare a **Kentico.Membership.User** object based on the posted registration data. Leave the user disabled.
    - Call the **UserManager.CreateAsync** method to create the user in the Kentico database.
    - Generate a token for the confirmation link by calling the **UserManager.GenerateEmailConfirmationTokenAsync** method.
    - Send the confirmation email by calling the **UserManager.SendEmailAsync** method. You need to specify the email subject and content through the parameters.

5. Add another action to the controller for handling of the confirmation requests. Call the **UserManager.ConfirmEmailAsync** method to verify and confirm user accounts.

> The **ConfirmEmailAsync** method requires the ID of the user and matching confirmation token as parameters. Pass the required values as query string parameters of the confirmation link within the content of the emails that you send to new users.

Controller example for registration with email confirmation

```
 »using System;
using System.Web;
using System.Web.Mvc;
using System.Threading.Tasks;

using Microsoft.AspNet.Identity;
using Microsoft.AspNet.Identity.Owin;

using Kentico.Membership;

using CMS.EventLog;

namespace LearningKit.Controllers
{
    public class EmailRegisterController : Controller
    {
        /// <summary>
        /// Provides access to the Kentico.Membership.UserManager instance.
        /// </summary>
        public UserManager UserManager
        {
            get
            {
                return HttpContext.GetOwinContext().Get<UserManager>();
            }
        }

        /// <summary>
        /// Basic action that displays the registration form.
        /// </summary>
        public ActionResult RegisterWithEmailConfirmation()
        {
            return View();
        }

        /// <summary>
        /// Creates new users when the registration form is submitted and sends
the confirmation emails.
        /// Accepts parameters posted from the registration form via the
RegisterViewModel.
        /// </summary>
        [HttpPost]
        [ValidateAntiForgeryToken]
        [ValidateInput(false)]
        public async Task<ActionResult> RegisterWithEmailConfirmation
(RegisterViewModel model)
        {
            // Validates the received user data based on the view model
            if (!ModelState.IsValid)
```

```
            {
                return View(model);
            }

            // Prepares a new user entity using the posted registration data
            // The user is not enabled by default
            Kentico.Membership.User user = new User
            {
                UserName = model.UserName,
                Email = model.Email,
                FirstName = model.FirstName,
                LastName = model.LastName
            };

            // Attempts to create the user in the Kentico database
            IdentityResult registerResult = IdentityResult.Failed();
            try
            {
                registerResult = await UserManager.CreateAsync(user, model.
Password);
            }
            catch (Exception ex)
            {
                // Logs an error into the Kentico event log if the creation of
the user fails
                EventLogProvider.LogException("MvcApplication",
"UserRegistration", ex);
                ModelState.AddModelError(String.Empty, "Registration failed");
            }

            // If the registration was not successful, displays the registration
form with an error message
            if (!registerResult.Succeeded)
            {
                foreach (var error in registerResult.Errors)
                {
                    ModelState.AddModelError(String.Empty, error);
                }
                return View(model);
            }

            // Generates a confirmation token for the new user
            // Accepts a user ID parameter, which is automatically set for the
'user' variable by the UserManager.CreateAsync method
            string token = await UserManager.GenerateEmailConfirmationTokenAsync
(user.Id);

            // Prepares the URL of the confirmation link for the user (targets
the "ConfirmUser" action)
            // Fill in the name of your controller
            string confirmationUrl = Url.Action("ConfirmUser", "EmailRegister",
new { userId = user.Id, token = token }, protocol: Request.Url.Scheme);

            // Creates and sends the confirmation email to the user's address
            await UserManager.SendEmailAsync(user.Id, "Confirm your new account",
                String.Format("Please confirm your new account by clicking <a
href=\"{0}\">here</a>", confirmationUrl));

            // Displays a view asking the visitor to check their email and
confirm the new account
```

```
            return View("CheckYourEmail");
        }

        /// <summary>
        /// Action for confirming new user accounts. Handles the links that users
click in confirmation emails.
        /// </summary>
        public async Task<ActionResult> ConfirmUser(int? userId, string token)
        {
            IdentityResult confirmResult;

            try
            {
                // Verifies the confirmation parameters and enables the user
account if successful
                confirmResult = await UserManager.ConfirmEmailAsync(userId.Value,
token);
            }
            catch (InvalidOperationException)
            {
                // An InvalidOperationException occurs if a user with the given
ID is not found
                confirmResult = IdentityResult.Failed("User not found.");
            }

            if (confirmResult.Succeeded)
            {
                // If the verification was successful, displays a view informing
the user that their account was activated
                return View();
            }

            // Returns a view informing the user that the email confirmation
failed
            return View("EmailConfirmationFailed");
        }
    }
}
```

6. We recommend creating a view model for your registration action. See the basic registration section above for an example (*RegisterViewModel* class).
7. Design the user interface required for registration on your website:
    - Create a view for the *RegisterWithEmailConfirmation* action and display an appropriate registration form. We recommend using a strongly typed view based on your registration view model.
    - Add a registration button or link that targets the *RegisterWithEmailConfirmation* action into an appropriate location on your website.
    - Create a view with content that informs users about the need to confirm their newly registered account *(CheckYourEmail* view in the example).
    - Create views for the *ConfirmUser* action. Display information for users who click the confirmation link (for both successful and unsuccessful confirmation).

Visitors can now register new user accounts on your MVC site. Upon registration, the system creates a disabled user in the connected Kentico database (assigned to the site where the registration occurred) and sends a confirmation email to the submitted address. Upon successful confirmation, the new user becomes enabled and can sign in.

Updating user details from the MVC application

You can add a page to your MVC application, where your users can change their personal details.

1. Create a new controller class in your MVC project or edit an existing one.
2. Prepare a property that gets an instance of the **Kentico.Membership.UserManager** class for the current request – call *HttpContext.GetOwinContext().Get<UserManager>()*.
3. Implement two user detail actions – one basic GET action to display the form and a second POST action to handle the changing of user details.
4. Perform the following steps within the POST action:
   - Prepare a **Kentico.Membership.User** object based on the posted user data.
   - Call the **UserManager.FindByName** method to find out the current user in Kentico.
   - Call the **UserStore.UpdateAsync** method to save the changes to the database.

Controller example for editing user details

```
        /// <summary>
        /// Provides access to user related API which will automatically
save changes to the UserStore.
        /// </summary>
        public UserManager UserManager
        {
            get
            {
                return HttpContext.GetOwinContext().Get<UserManager>();
            }
        }

        /// <summary>
        /// Displays a form where user information can be changed.
        /// </summary>
        public ActionResult EditUser()
        {
            // Finds the user based on their current user name
            User user = UserManager.FindByName(User.Identity.Name);

            return View(user);
        }

        /// <summary>
        /// Saves the entered changes of the user details to the database.
        /// </summary>
        /// <param name="returnedUser">User that is changed.</param>
        [HttpPost]
        [ValidateAntiForgeryToken]
        [ValidateInput(false)]
        public async Task<ActionResult> EditUser(User returnedUser)
        {
            // Finds the user based on their current user name
            User user = UserManager.FindByName(User.Identity.Name);

            // Assigns the names based on the entered data
            user.FirstName = returnedUser.FirstName;
            user.LastName = returnedUser.LastName;

            // Saves the user details into the database
            UserStore userStore = new UserStore(SiteContext.
CurrentSiteName);
            await userStore.UpdateAsync(user);

            return RedirectToAction("Index", "Home");
        }
```

5. Instead of using the *User* object directly in your view, we recommend creating a view model for your action for editing user details.
6. Design the user interface required for editing user details on your website:
    - Create a view for the *EditUser* action and display an appropriate form. We recommend using a strongly typed view based on your view model.

Users can now change their personal details on your MVC site.