Content personalization is an on-line marketing technique that creates pages with different content depending on the circumstances in which they are viewed. For example, you can build pages that offer special content for different types of visitors or dynamically change for each user according to the actions they performed on the website.

When developing MVC websites, you cannot create personalization variants of web parts, widgets or zones. Instead, you need to write personalization conditions within the code of your MVC site's controllers, views or other code. The conditions leverage the visitor segmentation tools that marketers define within the Kentico application – Contact groups or Personas.

Prerequisite

Before defining content personalization conditions, you first need to set up Tracking of contacts on your MVC website.

Note: The **Settings -> On-line marketing -> Enable content personalization** setting does not have any effect on content personalization conditions written in an MVC website's code.

Personalizing content based on contact groups

We recommend using Contact groups to personalize your MVC site's content:

1. Define contact groups to segment your website's visitors (in the **Contact groups** application within Kentico).
2. Use the contact tracking API to get the current contact in your MVC site's code.
3. Create conditions to display different content for contacts who belong to different contact groups.
4. Call one of the following extension methods for the current contact object within the conditions:

    - **IsInContactGroup** – evaluates whether the contact belongs to one specific contact group.
    - **IsInAnyContactGroup** – evaluates whether the contact belongs to *at least one* of the specified contact groups.
    - **IsInAllContactGroups** – evaluates whether the contact belongs to *all* of the specified contact groups.

    All of the methods return a boolean value that you can use in your content conditions.

    The methods accept one or more string parameters, which must match the **code names** of contact groups. To find the code names, edit contact groups in the *Contact groups* application in Kentico.

The system segments your website's visitors into contact groups, for example according to performed activities. Your MVC site then displays personalized content depending on the contact groups to which the current visitor (contact) belongs.

Razor view example

```
    @using CMS.ContactManagement
    @using Kentico.ContactManagement

    @{
        // Gets the current contact
        IContactTrackingService contactService = new ContactTrackingService();
        ContactInfo currentContact = contactService.GetCurrentContactAsync(User.
Identity.Name).Result;
    }

    @if (currentContact != null)
    {
        // Displays personalized content for contacts belonging to the
"YoungCustomers" contact group
        if (currentContact.IsInContactGroup("YoungCustomers"))
        {
            <text>Hiya @currentContact.ContactFirstName</text>
        }
        else
        {
            <text>Hello</text>
        }
    }
```

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.

Personalizing content for personas

If you use Personas to segment your site's visitors, you can create conditions and serve different content for different personas.

To check whether a contact is assigned to a persona:

1. Get the contact's persona by calling the **GetPersona()** extension method for the contact object (requires a reference to the **CMS.Personas** namespace).
2. Create a condition that checks the code name of the persona (or any other persona property).

```
using System;
using CMS.ContactManagement;
using CMS.Personas;
using Kentico.ContactManagement;
```

Example

```
            // Gets the current contact
            IContactTrackingService contactService = new ContactTrackingService();
            ContactInfo currentContact = contactService.GetCurrentContactAsync(User.
Identity.Name).Result;

            // Gets the code name of the current contact's persona
            string currentPersonaName = currentContact.GetPersona()?.PersonaName;

            // Checks whether the current contact is assigned to the "EarlyAdopter"
persona
            if (String.Equals(currentPersonaName, "EarlyAdopter", StringComparison.
InvariantCultureIgnoreCase))
            {
                // Serve personalized content for the "EarlyAdopter" persona
            }
```

We recommend using a dependency injection container to initialize service instances. When configuring the lifetime scope for *ContactTrackingService*, create a shared instance (singleton) for all requests.

Using output caching for personalized pages

When using ASP.NET output caching to improve the performance of your MVC website, you need to take additional steps to ensure that personalized pages display the correct content. Because personalized pages serve different content based on conditions, you need to adjust your application to cache separate output versions according to the personalization criteria.

The following steps describe how to cache multiple versions of personalized output using custom strings:

1. Edit your MVC site's controller classes that serve personalized content.
2. Set the **VaryByCustom** property for the **OutputCache** attributes of the appropriate action methods to a custom string.

Example

```
 »using System.Web.Mvc;
using CMS.ContactManagement;
using Kentico.ContactManagement;

namespace LearningKit.Controllers
{
    public class PersonalizationController : Controller
    {
        /// <summary>
        /// Gets the current contact, if contact tracking is enabled for the
connected Kentico instance.
        /// </summary>
        private ContactInfo CurrentContact
        {
            get
            {
                IContactTrackingService contactService = new
ContactTrackingService();
                return contactService.GetCurrentContactAsync(User.Identity.Name).
Result;
            }
        }

        /// <summary>
        /// Displays a page with a personalized greeting.
        /// The content depends on whether the current contact belongs to the
"YoungCustomers" contact group.
        /// Caches the action's output for 10 minutes, with different cache
versions defined by the "contactdata" custom string.
        /// The "contactdata" string ensures separate caching for each
combination of the following contact variables: contact groups, persona, gender
        /// </summary>
        [OutputCache(Duration = 600, VaryByCustom = "contactdata")]
        public ActionResult PersonalizedGreeting()
        {
            return View(CurrentContact);
        }
    }
}
```

3. Edit your project's **Global.asax** file and override the GetVaryByCustomString method.

    - The method accepts the string that you set in the *VaryByCustom* property of *OutputCache* attributes as the *arg* string parameter.
    - For each custom string, you need to define the caching requirements – return a string value containing all variables that you wish to use to vary the output cache.

    ```
    using System;
    using System.Linq;
    using System.Web;

    using CMS.ContactManagement;
    using Kentico.ContactManagement;
    ```

Example

```
        public override string GetVaryByCustomString(HttpContext context,
string arg)
        {
            // Defines caching requirements based on the on-line marketing
data of the current contact
            if (arg == "contactdata")
            {
                // Gets the current contact
                ContactInfo currentContact = new ContactTrackingService().
GetExistingContactAsync().Result;

                if (currentContact != null)
                {
                    // Ensures separate caching for each combination of the
following contact variables: contact groups, persona, gender
                    // Note: This does NOT define separate caching for
every contact
                    return String.Format("ContactData={0}|{1}|{2}",
                        String.Join("|", currentContact.ContactGroups.Select
(c => c.ContactGroupName).OrderBy(x => x)),
                        currentContact.ContactPersonaID,
                        currentContact.ContactGender);
                }
            }

            return base.GetVaryByCustomString(context, arg);
        }
```

Important

The "contactdata" example above defines a broad set of cache variables that ensure separate caching for most types of personalization conditions. This may be unnecessary for your website. For optimal caching, you need to prepare output cache variables tailored according to the personalization conditions that you use on the site or individual pages.

For example, pages that only check personas in their personalization condition only require the persona identifier in the cache variables – cached content can be shared for contacts belonging to different contact groups. On the other hand, strongly personalized pages with dynamic content may require separate cache versions for every contact.

You always need to balance website performance, memory usage on the server, and the risk of displaying incorrect cached content.

The application now caches separate versions of action output based on your custom variables. This ensures that visitors do not receive incorrect cached content when viewing personalized pages.

Using client-side only output caching

An alternative approach that you can consider is to set the **Location** property of your **OutputCache** attributes to ***System.Web.UI.OutputCacheLocation.Client***. This disables output caching on the server.

With only client-side caching, each client device caches its own personalized content and you do not need to define custom cache variables. This scenario is easier to set up for heavily personalized pages, but the performance gains are usually lower than with optimally configured output caching for both the server and client sides.