Kentico MVC websites respond to incoming requests using standard ASP.NET routing. This means that page URLs are fully determined by the routes that you register into your MVC application's routing table.

To ensure that the Kentico administration application knows the correct live site URLs of pages, set URL patterns for your content only pages types. The URL patterns need to match your MVC application's route templates.

Identifying pages based on a page alias

You can give content editors control over a part of the page URL by adding parameters into URL patterns. This allows you to include suitable human-readable keywords in URLs, and improve the site's SEO.

By default, content only pages in Kentico provide a page alias field for this purpose. The system automatically defines a page alias for every content only page, typically as a URL-safe version of its name in the content tree. Internally, the page alias is stored in the *NodeAlias* field (column) of each page and you can retrieve it using the *{%NodeAlias%}* macro expression (for example when specifying URL patterns of page types).

Since page aliases are controlled by content editors, they can change over time. Also, page aliases are only guaranteed to be unique within a given sub-section level of the content tree (under the same parent page). For example, only pages stored directly in the */Articles* section will have a unique page alias among each other. However, if one page is stored under */Articles/March* and another page is stored under */Articles/May*, they can both have the same page alias. This is why you may not want to use page aliases as the only identifier when displaying pages.

The **recommended practice** for identifying pages with a page alias value in the URL is to combine it with another unique identifier, such as the **NodeGUID**. This ensures that conflicts cannot occur and URLs remain valid even if a page's alias changes.

> **Note**: *NodeGUID* values provide more stable URLs and links than *NodeIDs*, which may differ when hosting the same website on multiple instances (for example in environments that use content staging).

Typically, a page alias will be part of a macro expression that makes up the URL pattern of content only page types. For example, a URL pattern could look like the following: */Articles/{%NodeGUID%}/{%NodeAlias%}*

Identifying multilingual content when using page aliases

Page aliases remain the same for pages in different cultures. To be able to distinguish pages in different cultures, you can include a language prefix in your URL patterns (for example using the *{%DocumentCulture%}* macro), and/or create a custom page type field and use it instead of the page alias.

Retrieving content only pages based on a page alias

If you use both a *NodeGUID* and page alias parameter in your URLs, use the *NodeGUID* identifier to retrieve specific pages in your code. The generated page type code provides the required method by default.

```
public static DocumentQuery<Article> GetArticle(Guid nodeGuid, string cultureName,
string siteName)
{
    return GetArticles().OnSite(siteName).Culture(cultureName).WhereEquals("NodeGUID",
nodeGuid);
}
```

For URLs that only contain a page alias parameter (*NodeAlias*), you need to make sure the page alias is always unique. You can, to a certain degree, ensure this uniqueness by limiting how content editors are allowed to create new pages:

- Allowed page type restrictions – by defining which page types users can place under the current page type.
- Page scope restrictions – by specifying which page types users can use when creating new pages under specified paths.

To retrieve pages using only the page alias:

```
public static DocumentQuery<Article> GetArticle(string pageAlias)
{
    return GetArticles().WhereEquals("NodeAlias", pageAlias).TopN(1);
}
```

You can also use the full *NodeAliasPath* to retrieve pages (for example */Articles/March/On-Roasts* for an article named *On Roasts*). This approach can be useful in scenarios where you have a highly structure content tree, and can build the alias path in your code according to other URL parameters. The generated page type code provides a method for retrieving pages based on the alias path by default.

```
public static DocumentQuery<Article> GetArticle(string nodeAliasPath, string
cultureName, string siteName)
{
    return GetArticles().OnSite(siteName).Culture(cultureName).Path(nodeAliasPath);
}
```

See also: Retrieving content on MVC sites

Configuring MVC application to display pages based on a page alias

The following example shows how you can configure your MVC application to display articles that have a URL based on a page alias, without using any additional identifiers. Note that the examples built on our MVC Demo site, which uses patterns like repositories to retrieve content and other implementation specifics.

1. In your application's RouteConfig.cs file, create a route for accessing articles based on a page alias. For example:

```
public class RouteConfig
    {
        public static void RegisterRoutes(RouteCollection routes)
        {
                ...
                        route = routes.MapRoute(
                         name: "ArticleByPageAlias",
                         url: "{culture}/Articles/{pageAlias}",
                         defaults: new { culture = defaultCulture.Name,
controller = "Articles", action = "ShowByPageAlias" }
                        );
                ...
        }
};
```

2. Add a controller action for retrieving articles based on a page alias.

```
public class ArticlesController : Controller
{
...
        // GET: Articles/{pageAlias}
        public ActionResult ShowByPageAlias(string pageAlias)
        {
                var article = mArticleRepository.GetArticle(pageAlias);
                if (article == null)
                {
                        return HttpNotFound();
                }
                return View("Show", article);
        }
...
}
```

3. Create a method for generating article URLs based on a page alias.

```
public static class UrlHelperExtensions
{
...
        /// <summary>
        /// Generates a fully qualified URL to the action method handling the
detail of given article.
        /// </summary>
        /// <param name="urlHelper">Url helper</param>
        /// <param name="article">Article object to generate URL for.</param>
        public static string ForArticleByPageAlias(this UrlHelper urlHelper,
Article article)
        {
                return urlHelper.Action("ShowByPageAlias", "Articles", new
                {
                        pageAlias = article.NodeAlias
                });
        }
...
}
```

4. Retrieve articles based on a page alias. For example in repositories.

```
public interface IArticleRepository
{
...
        /// <summary>
        /// Returns the article with the specified page alias.
        /// </summary>
        /// <param name="pageAlias">The article page alias.</param>
        /// <returns>The article with the specified page alias, if found;
otherwise, null.</returns>
        Article GetArticle(string pageAlias);
...
}
```

```
public class KenticoArticleRepository : IArticleRepository
{
...
        /// <summary>
        /// Returns the article with the specified page alias.
        /// </summary>
        /// <param name="pageAlias">The article page alias.</param>
        /// <returns>The article with the specified page alias, if found;
otherwise, null.</returns>
        public Article GetArticle(string pageAlias)
        {
                return ArticleProvider.GetArticles()
                        .OnSite(mSiteName)
                        .Culture(mCultureName)
                        .WhereEquals("NodeAlias", pageAlias)
                        .LatestVersion(mLatestVersionEnabled)
                        .Published(!mLatestVersionEnabled);
        }
...
}
```

5. Work with article URLs in Views based on a page alias.

```
<a href="@Url.ForArticleByPageAlias(article)">@article.Fields.Title </a>
```