The Kentico API allows you to retrieve and manage page content using custom code.

Use the following classes from the **CMS.DocumentEngine** namespace to work with pages in the API:

- **TreeNode** – object that represents pages. The *TreeNode* class encapsulates data from the *CMS_Tree* and *CMS_Document* tables, and the coupled data tables of individual page types.
- **DocumentQuery** and **MultiDocumentQuery** – classes that represent a query for loading pages. You can further parametrize the query to only retrieve the pages you need. Both are similar in functionality:
  - **DocumentQuery** allows you to retrieve pages of a single page type (when an instance is created with a page type class name as the constructor parameter) or general pages of any type (when using a parameterless constructor). If used for a single specific page type, the resulting query automatically includes coupled data of the given page type (e.g., news, articles, blogs). *DocumentQuery* is also used internally by generated page type providers and any custom providers or repositories built around them.
  - **MultiDocumentQuery** allows you to retrieve pages of multiple page types in a single query at the cost of effectiveness. The coupled data of page types is not retrieved unless explicitly requested using the *WithCoupledData* parametrization method.
- **DocumentHelper** – provides static methods for retrieving and managing the latest edited versions of pages. Internally, the *DocumentHelper* class utilizes *DocumentQuery* or *MultiDocumentQuery* to retrieve pages and then further adjusts the query, for example by applying the *CombineWithDefaultCulture* parametrization method based on the value of the *Combine with default culture* setting.

Page query parametrization methods

For detailed information about the options that allow you to parameterize page queries, see the dedicated reference: Reference - DocumentQuery methods

Loading other objects

To learn how to retrieve other types of non-page data from the Kentico database, see Retrieving database data using ObjectQuery API.

Retrieving pages

For basic retrieval of pages in your code, use the **DocumentHelper.GetDocuments** method and parameterize it using parametrization methods to only include the pages you need. If you have generated page type providers available, you can use them with the same parametrization methods, and then work with strongly typed page objects.

Example

```
// Retrieves the published English version of articles that are in the '/Articles/May'
section of the 'MySite' site's content tree
// Only loads pages that are currently published on the live site according to the
value of their Published from/to setting
var pages = DocumentHelper.GetDocuments("MySite.Articles")
                                        .OnSite("MySite")
                                        .Culture("en-us")
                                        .Path("/Articles/May", PathTypeEnum.Children)
                                        .PublishedVersion()
                                        .Published();
```

Retrieving pages under workflow

When working with pages under workflow, you may want to retrieve the latest edited version in some cases and the published version in others.

- Use the **PublishedVersion** parametrization method to retrieve the published version of pages under workflow (when retrieving pages to be displayed on the live site).
- Use the **LatestVersion** method to retrieve the latest edited version of pages under workflow (when retrieving pages for further editing or previewing of unpublished changes).

> **Note**: The **Published** parametrization method does not specify the version of the pages you retrieve, instead it limits the retrieved pages to only those that are currently published according to the value of their **Published from** / **Published to** settings and have a published version.

Retrieving pages for custom scenarios

DocumentHelper automatically performs some parameterization of the query to simplify the code for the most common scenarios. However, if you require full control over the query, you can directly use the **DocumentQuery** or **MultiDocumentQuery** classes to ensure that there is not any unexpected parametrization done in the background.

```
// Retrieves all pages that match the query
var pages = new MultiDocumentQuery()
                    .OnCurrentSite()
                    .Path("/Articles/Reviews")
                    .Culture("en-us")
                    .PublishedVersion()
                    .WithCoupledColumns();
```

```
// Retrieves pages of the specified page type that match the query
var pages = new DocumentQuery("Custom.Smartphone")
                    .OnCurrentSite()
                    .Path("/Products")
                    .Culture("en-us")
                    .PublishedVersion();
```

Working with retrieved pages

You can iterate through the retrieved collection to access the properties of individual pages. The available columns depend on how you parametrized the query when retrieving the pages.

```
// Retrieves smartphone pages
DocumentQuery smartphones = DocumentHelper.GetDocuments("CMS.Smartphone")
                                .Path("/Products/", PathTypeEnum.Children)
                                .OnSite("CorporateSite");

// Writes the 'DocumentName' and 'SmartphoneOS' values of each of the retrieved
smartphones into an HTTP output response stream
foreach (TreeNode smartphone in smartphones)
{
    string smartphoneOS = smartphone.GetValue<string>("SmartphoneOS", "Default OS");
    Response.Write(HTMLHelper.HTMLEncode(smartphone.DocumentName) + " - " + HTMLHelper.
HTMLEncode(smartphoneOS) + "<br />");
}
```

Updating pages

To update a page (*TreeNode* object):

1. Retrieve the latest edited version of a page using the methods described above.

2. Modify the page's data:
    - For general page fields, directly set the corresponding *TreeNode* properties.
    - For the fields of specific page types, call the *TreeNode.SetValue("FieldName", value)* method.
3. Call the **TreeNode.Update()** method.

Updating pages under workflow

When using the API to update pages under workflow or versioning, always retrieve the page objects with all fields. Otherwise, the update may cause data loss. Use one of the following approaches:

- Call the *DocumentHelper.GetDocuments(string className)* method with a *className* parameter for a specific page type.
- Use the **Types** query method to identify the page types and then apply the **WithCoupledColumns** method.