This page provides information about the most common and recommended parametrization methods available for the DocumentQuery API. The methods allow you to parameterize queries, for example to limit which pages are retrieved or specify which page columns are loaded to improve performance.

Page filtering

Method	Description
CheckPermissions	Retrieves pages based on the context of a specific user (the pages are filtered according to the user's permissions). By default, the current user's context is used. *CMSActionContext* may be used to provide a different user's context. `// Retrieves all pages visible to the public user UserInfo user = UserInfoProvider.GetUserInfo("public"); using (new CMSActionContext(user)) { var pages = new MultiDocumentQuery() .CheckPermissions(); }`
ExcludePath	Filters the retrieved pages to exclude a website section. See the **Path** parametrization method for general filtering based on included or excluded website sections (page paths). `// Retrieves pages from the '/Products' section, including the parent '/Product' page // Excludes the entire '/Products/Software' sub-section var pages = new MultiDocumentQuery() .Path("/Products", PathTypeEnum.Section) .ExcludePath("/Products/Software", PathTypeEnum.Section);`
FilterDuplicates	Allows filtering of the retrieved data to remove any duplicates caused by linked pages. `// Retrieves all pages without duplicate (linked) pages var pages = new MultiDocumentQuery() .FilterDuplicates();`
InCategories	Selects pages assigned to specific categories. To limit the filtering only to pages in enabled categories, use the *InEnabledCategories* method. `// Retrieves all pages that are in the 'Reviews' category var pages = new MultiDocumentQuery() .InCategories("Reviews");`

InRelationWith	Retrieves pages that are connected to another page through a named relationship.
	`// Simulates a page GUID Guid nodeGuid = Guid.NewGuid(); // Retrieves pages that are in any relationship with a specific page var pages = new MultiDocumentQuery() .InRelationWith(nodeGuid);`
	`// Simulates a page GUID Guid nodeGuid = Guid.NewGuid(); // Retrieves pages that are in a specified relationship with a specific page var pages = new MultiDocumentQuery() .InRelationWith(nodeGuid, "IsRelatedTo");`
	`// Simulates a page GUID Guid nodeGuid = Guid.NewGuid(); // Retrieves pages that are on the left side of a specified relationship with a specific page var pages = new MultiDocumentQuery() .InRelationWith(nodeGuid, "IsRelatedTo", RelationshipSideEmun.Left);`

NestingLevel	Specifies the relative content tree depth from which pages are retrieved (starting from the root of the retrieved content tree section). `// Retrieves one level of child pages under the '/Products' section var pages = new MultiDocumentQuery() .Path("/Products", PathTypeEnum.Children) . NestingLevel(1);`
OnSite	Specifies the site from which the pages are retrieved. `// Retrieves all pages from the specified site var pages = new MultiDocumentQuery() .OnSite("CorporateSite");`

OrderBy*	Allows ordering of the results based on the value of a specified column.
	`// Retrieves smartphones ordered by their display size var pages = new DocumentQuery("Custom.Smartphone") .OrderBy("SmartphoneDisplaySize");`
	`// Retrieves smartphones based on their display size in descending order var pages = new DocumentQuery("Custom.Smartphone") .OrderByDescending ("SmartphoneDisplaySize");`

Page	Allows implementation of pagination – separating the retrieved data into "pages" of a certain size. Retrieves only the data items that belong within a specific page. `// Retrieves the second page of size 5 of all smartphones var pages = new DocumentQuery("Custom.Smartphone") .Page(1, 5);`

Path	Specifies the website section from which the pages are retrieved.
	`// Retrieves pages from the '/Products' section including the parent page var pages = new MultiDocumentQuery() .Path("/Products", PathTypeEnum.Section);`
	`// Retrieves child pages from the '/Products' section excluding the parent page var pages = new MultiDocumentQuery() .Path("/Products", PathTypeEnum.Children);`

TopN	Specifies the number of records, which are retrieved from the database. `// Retrieves top 5 smartphones with the largest display size var pages = new DocumentQuery("Custom.Smartphone") .OrderByDescending("SmartphoneDisplaySize") .TopN(5);`

Where*	Allows filtering of the pages based on their properties.
	`// Retrieves smartphones whose 'DocumentName' value starts with 'Apple' var pages = new DocumentQuery("Custom.Smartphone") .Where("DocumentName", QueryOperator.Like, "Apple%");`
	`// Retrieves smartphones whose 'DocumentName' starts with 'Apple' or 'BlackBerry' var pages = new DocumentQuery("Custom.Smartphone") .Where("DocumentName", QueryOperator.Like, "Apple%") .Or() .Where("DocumentName", QueryOperator.Like, "BlackBerry%");`
	`// Retrieves smartphones whose 'DocumentName' value starts with 'Apple' var pages = new DocumentQuery("Custom.Smartphone") .WhereLike("DocumentName", "Apple%");`
	`// Retrieves smartphones whose 'DocumentName' does NOT start with with 'Apple' var pages = new DocumentQuery("Custom.Smartphone") .WhereNotStartsWith("DocumentName", "Apple%");`
	There are many *Where** conditions available. Experiment to find the one that suits your needs.

WithTag	Retrieves pages based on the assigned tags. Note that multiple tags assigned to different tag groups can have identical names. To search for a tag across all tag groups, omit the second method parameter. `// Retrieves smartphones that have the 'Android' tag from the 'Smartphones' tag group var pages = new DocumentQuery("Custom.Smartphone") .WithTag("Android", "Smartphones");`

Published status

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

Method	Description
LatestVersion	Retrieves the latest edited versions of pages. This parametrization method does not limit the set of retrieved pages, instead, it specifies what version of data is retrieved for pages. `// Retrieves the latest edited version of pages var pages = new MultiDocumentQuery() .LatestVersion(true);`
PublishedVersion	Retrieves the published versions of pages. This parametrization method does not limit the set of retrieved pages, instead, it specifies what version of page data is retrieved for pages. `// Retrieves the published version of pages var pages = new MultiDocumentQuery() .PublishedVersion(true);`
Published	Retrieves pages that are in the published state on the live site (currently set to be published on the live site according to the value of their *Published from*/*Published to* settings and have a published version). This parametrization method limits the set of retrieved pages but does not ensure that the published version of page data is retrieved. To retrieve the latest published version of pages as related to workflow and versioning, use the **PublishedVersion** parametrization method. `// Retrieves pages that are currently set as published var pages = new MultiDocumentQuery() .Published(true);`

Data columns

Method	Description

Columns	Specifies the columns of the page which are retrieved. `// Retrieves smartphone pages, with only the basic required page columns and a specific column holding operating system data var pages = new DocumentQuery ("Custom.Smartphone") .Columns("SmartphoneOS");` **Performance best practice** Use the *Columns* method to restrict the amount of data loaded from the database and speed up the querying process. When retrieving pages with the *Columns* query method, the system automatically adds several system columns to the list of retrieved columns. The extra columns are not added to queries where it is not desirable, for example, in queries including DISTINCT or GROUP BY clauses. You can disable the automatic addition of the extra system columns by setting the *Properties.EnsureExtraColumns* method to false. **Updating pages under workflow** When working with pages under workflow or versioning, use the *Columns* method only when performing read-only operations. Do not use this method with pages you want to update. The update would likely cause a data loss.
WithCoupledColumns	Allows you to access coupled data (i.e., fields of individual page types). This parametrization only needs to be used with *MultiDocumentQuery*, as *DocumentQuery* loads coupled data by default. `// Retrieves pages from the specified section and allows you to access page type fields var pages = new MultiDocumentQuery() .Path("/Products /Smartphones") .WithCoupledColumns(); // Gets the value of the "SmartphoneManufacturer" text field string manufacturer = pages. FirstOrDefault().GetValue<string>("SmartphoneManufacturer", "Default value");`

Culture

Method	Description
CombineWithDefaultCulture	Determines whether the default culture version is retrieved for pages that are not available in the selected culture (when used together with the **Culture** parametrization method). `// Retrieves the 'sw-se' culture version of pages, or the default culture version if not available var pages = new MultiDocumentQuery() .Culture("sw-se"); .CombineWithDefaultCulture();`

Combine WithAny Culture	Intended for use together with the **Culture** method and multiple culture values. If a page is not available in the first specified culture, the query then attempts to load the next culture in the list. Retrieves the default culture version for pages that are not available for any of the specified cultures. `// Retrieves pages in the 'sw-se' culture. Pages unavailable in the 'sw-se' are returned in the 'cs-cz' culture // Pages unavailable in any of the specified cultures are returned in the site's default culture var pages = new MultiDocumentQuery() .Culture("sw-se", "cs-cz") .CombineWithAnyCulture();`
Culture	Specifies the culture of pages to be retrieved on multilingual websites. `// Retrieves all pages that are in the specified culture var pages = new MultiDocumentQuery() .Culture("sw-se");`