



A Guide to Data Centric Functions

2017-02-14

Contents

1	INTRODUCTION	3
1.1	Who Should Read This Guide	3
1.2	Getting Started	3
2	GETTING DATA AS XML	4
3	SELECTING FIELDS.....	6
4	SORTING DATA.....	8
5	FILTERING DATA	9
6	PAGING.....	11
7	USING REFERENCES	12
8	NAMING	13
9	CACHING	14
10	USING DATA REFERENCE FILTER.....	15
11	USING FIELD PREDICATES FILTER	17
12	USING COMPOUND FILTER	20
13	USING ACTIVE PAGE REFERENCE FILTER	21
14	OTHER DATA-CENTRIC FUNCTIONS.....	22
14.1	AddDataInstance	22
14.2	UpdateDataInstance	22
14.3	DeleteDataInstance	23
14.4	GetDataReference	23
14.5	GetNullableDataReference	23
15	TEST YOUR KNOWLEDGE	24
15.1	Task: Get Data as XML	24
15.2	Task: Sort Data	24
15.3	Task: Filter Data	24
15.4	Task: Use Paging	24
15.5	Task: Use Active Page Reference Filter	24

1 Introduction

Data-centric functions are XSLT-based CMS functions that allow you to programmatically manage data in data types - retrieve, create, update and delete data items.

In this guide you will learn how to use these functions, primarily, the Get<DataType>Xml-like functions generated by the system automatically.

1.1 Who Should Read This Guide

This guide is intended for web developers who want to learn how to use data-centric functions in C1 CMS to manage data.

As a web developer, you must be an expert in XML and XSLT and know how to work with C1 CMS and in its Administrative Console.

You need to have access to the Data and Functions perspectives with sufficient permissions to create, edit and delete data types and functions. To use the XSLT functions on pages and layout templates, you might also need to have access to the Content and Layout perspectives.

1.2 Getting Started

To get started with data-centric functions, you are supposed to take a number of steps.

Getting Started		
Step	Activity	Chapter or section
1	Get data as XML	<i>Getting Data As Xml</i>
2	Select fields	<i>Selecting Fields</i>
3	Sort data	<i>Sorting Data</i>
4	Filter data	<i>Filtering Data</i> <i>Using Data Reference Filter</i> <i>Using Field Predicates Filter</i> <i>Using Compound Filter</i> <i>Using Active Page Reference Filter</i>
5	Use paging	<i>Paging</i>
6	Use references	<i>Using References</i>
7	Set up naming	<i>Naming</i>
8	Set up caching	<i>Caching</i>
9	Add, update and delete data	<i>Other Data-Centric Functions</i>

In the following few chapters, you will learn more about these and other activities.

2 Getting Data as XML

When you create a data type in C1 CMS, the system automatically generates a number of functions for it - the so-called “data-centric” functions.

One of the frequently used data-centric functions is the function that allows you to get data items of the data type as an enumerable list of XML elements (XElements).

The function is placed within the namespace the data type belongs to, and its name follows this pattern:

Get<DataType>XML

where <DataType> stands for the name of the data type.

For example, if you create a data type called Demo.Customers, C1 CMS generates a function:

Demo.Customers.GetCustomersXml

You can call such a function from another function. When used as a function call in an XSLT function (with the local name of “GetCustomersXml”), it can be referred to in the markup as

```
/in:inputs/in:result[@name='GetCustomersXml']
```

Each data item retrieved with such a function is presented as an XML element:

```
<Customers Id="4ee4b37d-4388-4f15-a88b-bc8f1ada2b4c" Name="John Doe"
Email="john.doe@somecompany.com" xmlns="" />
```

Listing 1: A data item presented as an XML element

In this element:

- the name is that of the data type (without namespaces)
- the attributes are the names of the fields in the data type
- the attribute values are the values in the data type fields

You can thus iterate the data items by using XPath.

For example, having added a call to the function ‘GetCustomersXml’ as its local name to an XSLT function, you can iterate its data items as follows:

```
<xsl:for-each
select="/in:inputs/in:result[@name='GetCustomersXml']/Customers"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
User Name: <xsl:value-of select="@Name" /> <br/>
Email Address: <xsl:value-of select="@Email" /> <br/>
</xsl:for-each>
```

Listing 2: Iterating a data type via its data-centric function

The function itself has a number of parameters by which you can fine tune its output as you need in your markup or code.

- **Selected fields [PropertyNames]** (IEnumerable<String>)
- **Filter** (Expression<Func<Customers, Boolean>>)
- **Order by [OrderByField]** (String)
- **OrderAscending** (Boolean)
- **PageSize** (Int32)
- **PageNumber** (Int32)
- **Show reference data inline [ShowReferencesInline]** (Boolean)
- **IncludePagingInfo** (Boolean)
- **Randomized** (Boolean)

- **ElementName** (String)
- **ElementNamespace** (XNamespace)
- **CachePriority** (GetXmlCachePriority)

Each parameter or a group of parameters serve their own purpose in retrieving or presenting data as XML. By setting these parameters, you can:

- [select what fields to include in the output](#)
- [define how to sort data items in the output](#)
- [select what data items to include in the output](#)
- [define how to page data items in the output](#)
- [define how to present data retrieved from the referenced data types](#)
- [specify the name and the namespace for the elements that stand for data items](#)
- [define whether to use cache and, if so, how to use it](#)

3 Selecting Fields

You can select what fields to include in the XML output by setting the PropertyNames parameter:

- **Selected fields [PropertyNames]:** (IEnumerable<String>) The data fields to output in the XML.

To select fields:

1. Select the **Selected fields** parameter.
2. In the **Parameter Value** group box, click **Edit Selection**.

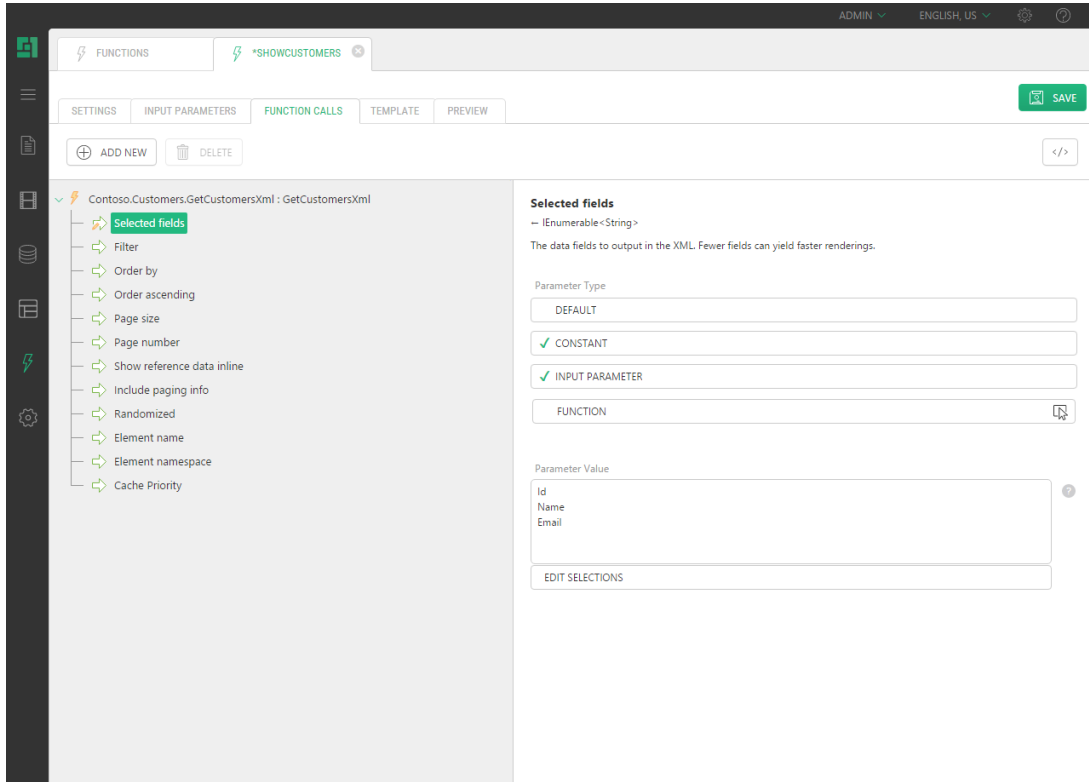


Figure 1: Selected fields

3. In the **Parameter Value** window, add the fields to be included in the output by using the proper buttons.

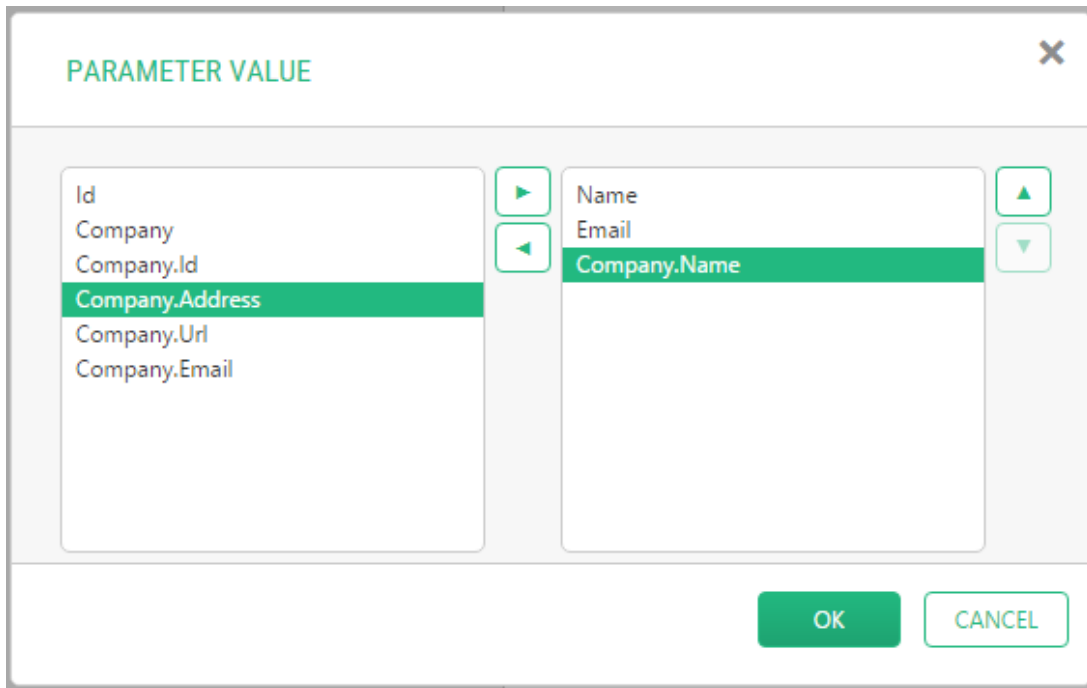


Figure 2: Editing selection

4. Remove the fields to be excluded from the output.
5. Click **OK**.
6. Save the function.

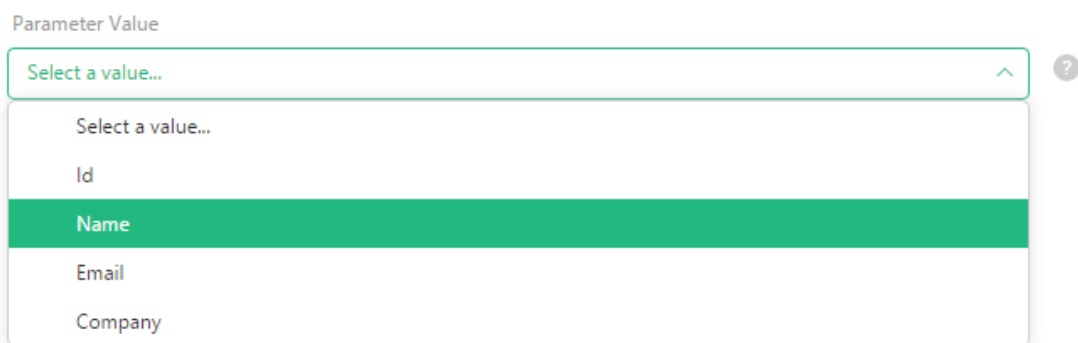
4 Sorting Data

You can select whether to sort data items, what field to sort them by and whether to sort them in ascending or descending order.

- **Order by [OrderByField]:** (String) The field to order data by
- **Order ascending [OrderAscending]:** (Boolean) When set to true results are delivered in ascending order, otherwise descending order is used. Default is ascending order.

To sort data:

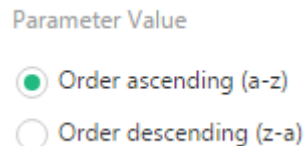
1. Select the **Order By** parameter.
2. Select **Constant** for the **Parameter Type**.
3. For the **Parameter Value**, select the field to sort data by in the dropdown list. (The field should not necessarily be one of the selected ones.)



The image shows a dropdown menu titled "Parameter Value". The menu is open, displaying a list of options. The first option is "Select a value...". Below it are "Id", "Name", "Email", and "Company". The "Name" option is highlighted with a green background. There is a small question mark icon to the right of the dropdown.

Figure 3: Selecting fields to sort data by

4. Select the **Order Ascending** parameter.
5. Select **Constant** for the **Parameter Type**.
6. Select one of the two options: **Order ascending (a-z)** or **Order descending (z-a)**.



The image shows two radio button options under the heading "Parameter Value". The first option is "Order ascending (a-z)" with a green radio button. The second option is "Order descending (z-a)" with an unselected radio button.

Figure 4: Selecting the sorting order

7. Save the function.

You can also order data randomly by setting the **Randomized** parameter:

- **Randomized:** (Boolean) When true, data can be ordered randomly.

To specify the number of random results you need, use the **Page size** parameter.

Please note that if a filter is specified, it is applied before the random selection.

For hands-on experience on sorting, please refer to "[Sorting and Paging](#)".

5 Filtering Data

By default, the `Get<DataType>Xml` function retrieves all the data items of a specific data type. In practice, you might only need specific data items that match certain conditions. And this is where filtering comes in handy.

You specify your filters in the **Filter** parameter of the `Get<DataType>Xml` function:

- **Filter**: (`Expression<Func<[DataType], Boolean>>`): An expression function that takes the datatype as its input parameter, iterates the data items and returns 'true' if a data item matches a specific condition and 'false', if it doesn't.

To filter data, you should use one of the filter functions available for the data type in question. You can use two types of filters:

- [DataReferenceFilter](#), which selects a single data item
- [FieldPredicatesFilter](#), which selects multiple data items that meet the criteria you can set with predicates on each of the data type's field.

And the [CompoundFilter](#) function allows you to combine two filters defining relations between them as "And" or "Or".

To filter data:

1. Select the **Filter** parameter.
2. Click **Function** in the **Parameter type** group box.
3. In the **Value for Parameter 'Filter'** window, expand **All functions**, then namespaces the data type belongs to (e.g. "Contoso.Customers").

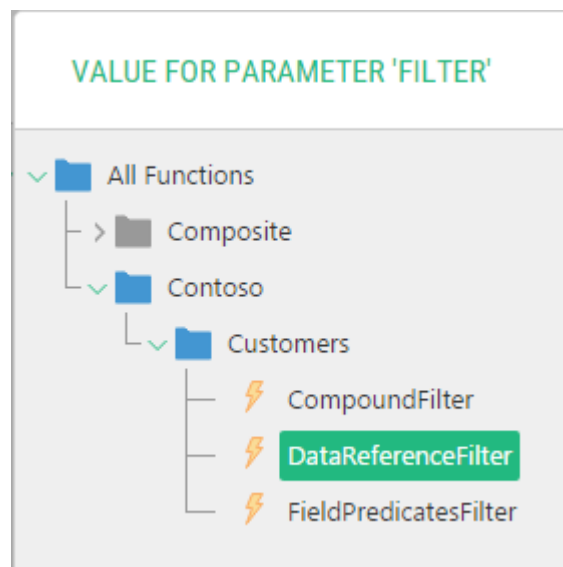


Figure 5: Selecting one of the filter functions

4. Select one of the filter functions and click **OK**. The function will appear under the **Filter** parameter of the `Get<DataType>Xml` function in the **Function Calls** tree.
5. Set its parameters.

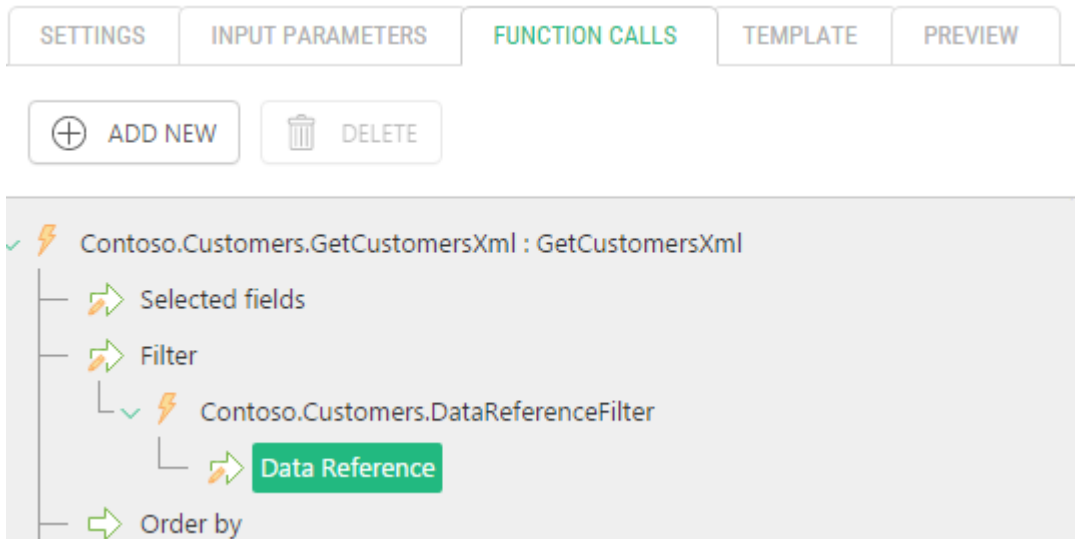


Figure 6: Setting a filter function's parameters

6. Save the function.

6 Paging

By default, the `Get<DataType>Xml` function retrieves all the data items of a specific data type. If the number of items is great, it makes sense to retrieve and render the items in portions.

Say, you have a data type that contains your contacts and the list exceeds 1000 entries. You can retrieve 20 entries at a time and allow the user the user to select which group of 20 entries to show, i.e. select a page.

The following parameters allows you to set the number of items to retrieve at a time and select the page as well as provide information about paging such as how many pages it splits into, what page number is currently in use etc.

- **Page size [PageSize]:** (Int32) The number of items to display on one page – the maximum number of elements to return.
- **Page number [PageNumber]:** (Int32) If the number of data elements exceed the page size you can use paging to move to the other pages.
- **Include paging info [IncludePagingInfo]:** (Boolean) When selected, the data XML will be preceded by a `<PagingInfo />` element detailing number of pages, items and more.

The last parameter returns XML similar to:

```
<PagingInfo CurrentPageNumber="1" TotalPageCount="1" TotalItemCount="3"
ShownItemsCount="3" MaximumItemsPerPage="1000" CurrentItemNumberStart="1"
CurrentItemNumberEnd="3" xmlns="" />
```

Listing 3: Paging Info XML

Its attributes gives detailed information on paging:

- **CurrentPageNumber:** What page number is currently in use (set in the `PageNumber` parameter)
- **TotalPageCount:** How many pages data is split into
- **TotalItemCount:** How many items the data type contains
- **ShownItemsCount:** How many items is actually shown on a page (which can be fewer than the maximum, e.g. on the last page)
- **MaximumItemsPerPage:** How many items are supposed to be shown on one page at maximum(set in the `PageSize` parameter)
- **CurrentItemNumberStart:** The item number the current page starts with
- **CurrentItemNumberEnd:** The item number the current page ends with

For hands-on experience on paging, please refer to [“Sorting and Paging”](#).

7 Using References

If a field in a data type is of the data reference type, you can choose how to present the referenced data in XML returned by `Get<DataType>Xml` - inline or nested - by setting this parameter:

- **Show reference data inline [ShowReferencesInline]** (Boolean)

The use of this parameter is only valid if you select a field of the data reference type in the Selected fields (PropertyNames) parameter.

When inline (true), the referenced field will be presented as an attribute within the data element (see the `GalleryId.Name` attribute)

```
<GalleryItem Id="20f59d12-c30a-420a-afec-ec23f107d009" Title="Jellyfish"
GalleryId.Name="Images" xmlns="" />
<GalleryItem Id="906ae622-ec7e-4f62-a77e-717e972acd3c" Title="Koala"
GalleryId.Name="Images" xmlns="" />
<GalleryItem Id="e34fcd17-4a4e-4ad9-bfa5-6234ffc29e58" Title="Penguins"
GalleryId.Name="Images" xmlns="" />
```

Listing 4: Inline referenced fields

The inline reference is easy to use in XSLT while bloating the XML

When nested, the referenced field will be presented as an individual element (see the `GalleryId` element):

```
<GalleryItem Id="20f59d12-c30a-420a-afec-ec23f107d009" Title="Jellyfish"
xmlns="" />
<GalleryItem Id="906ae622-ec7e-4f62-a77e-717e972acd3c" Title="Koala"
xmlns="" />
<GalleryItem Id="e34fcd17-4a4e-4ad9-bfa5-6234ffc29e58" Title="Penguins"
xmlns="" />
<GalleryId Name="Images" xmlns="" />
```

Listing 5: Nested referenced field

8 Naming

By default, each data element has the same name as that of the data type. For example, if the data type is called “Customers”, the data element in XML will also read “Customers”. However, you can override the name of the element in the Element Name parameter.

Besides, you can specify the namespace for the element in the Element Namespace parameter. By default, no namespace is used.

- **Element name [ElementName]** (String): The name of the XML element. The default is the name of the data type (e.g. “Customers”)
- **Element namespace [ElementNamespace]** (XNamespace): The namespace the XML element belongs to. No namespace is used by default.

9 Caching

To determine whether the resulting XML should be cached and what priority the cache records should have, use the following parameter:

- **Cache Priority** (GetXmlCachePriority)

This parameter can take 3 constant values:

- **Disabled**: Caching is disabled
- **<NONE>**: Caching is enabled and no priority is set
- **Default**: Caching is enabled and the default priority is set

The **Default** value is used by default. To override this setting, change the parameter type to **Constant**, an **Input Parameter** (if available) or **Function** and change it accordingly.

10 Using Data Reference Filter

To select a single data item, use the DataReferenceFilter function:

1. Select the **DataReferenceFilter** for the **Filter** parameter.
2. Set its **Data Reference** parameter.

You can add the Data Reference parameter by:

- Selecting a *constant* value
- Binding it to one of the XSLT function's *input parameters* (if any)
- Having a specific *function* provide the value for this parameter

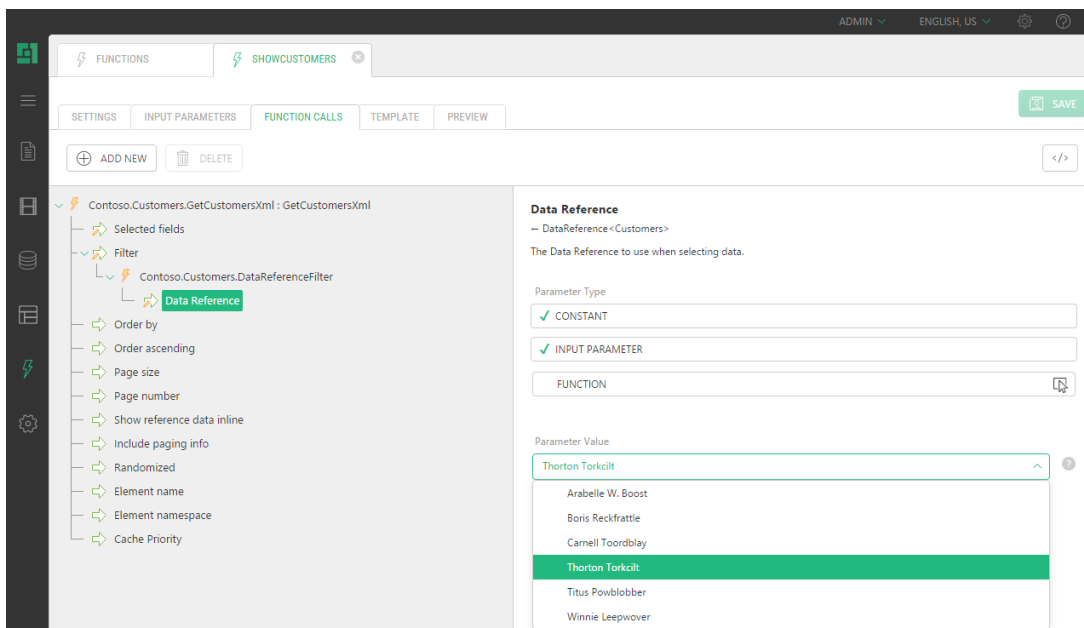


Figure 7: Using a constant value in the Data Reference Filter

The Data Reference parameter expects a value of the DataReference<DataType>, where DataType stands for the name of the data type (e.g. “Customers”).

To bind the value to that of the input parameter, you can change the parameter type of Data Reference to:

- **Input Parameter** and select the XSLT function's input parameter
- **Function** and select the **Composite.Util.GetInputParameter** function

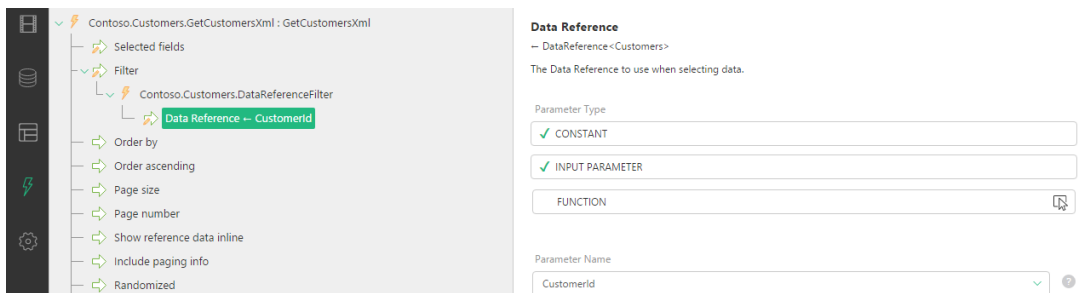


Figure 8: Using the value of the XSLT function's input parameter

If you use a query string in the URL to pass a GUID of the item, you can get this GUID from the URL with one of the Composite.Web.Request functions: QueryStringGuidValue or FormPostGuidValue and get the reference to the corresponding data item by using the

GetDataReference function generated for the data type in question (e.g. Demo.Customers.GetDataReference.)

The screenshot displays a configuration interface for a data function. On the left, a tree view shows the function hierarchy: Contoso.Customers.GetCustomersXml : GetCustomersXml, Selected fields, Filter, Contoso.Customers.DataReferenceFilter, Data Reference, Contoso.Customers.GetDataReference, Key value, Composite.Web.Request.QueryStringGuidValue, Parameter name (highlighted in green), and Fallback value. Below the tree are options for 'Order by' and 'Order ascending'. On the right, a configuration panel for the 'Parameter name' (type: String) is shown. It includes a 'Parameter Type' section with three radio buttons: 'CONSTANT' (checked), 'INPUT PARAMETER' (checked), and 'FUNCTION'. Below this is a 'Parameter Value' field containing the text 'Id'.

Figure 9: Using a value passed in the parameterized URL

11 Using Field Predicates Filter

To select multiple data items that meet some criteria you, use the FieldPredicatesFilter function:

1. Select the **FieldPredicatesFilter** function for the **Filter** parameter.
2. Use the **Composite.Utils.Predicates** functions to filter on one or more fields.

When you add the FieldPredicatesFilter function, it lists a number of parameters, each of which is a filter parameter for one field of the data type.

For example, if you have such fields in your data type as Id, Name, Email, Company, the FieldPredicatesFilter function will have such parameters as Id filter, Name filter, Email filter, Company filter.

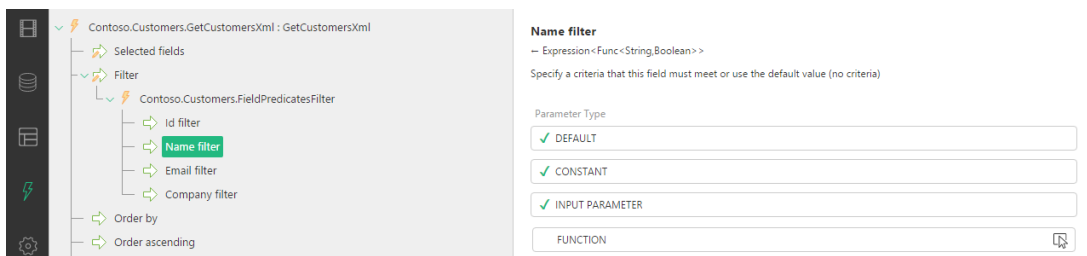


Figure 10: Setting one of the field filters

Setting criteria on multiple fields has them enforced.

To set a field filter parameter, you should use one of the predicate functions (Composite.Utils.Predicates). Each predicate function expects a function expression ([expression tree](#)) as its value and always returns a Boolean.

Each type of the field has its own set of predicates:

Boolean

- BooleanEquals
- NullableBooleanEquals
- BooleanNoValue

DateTime

- DateTimeEquals
- DateTimeGreaterThan
- DateTimeLessThan
- NullableDateTimeEquals
- NullableDateTimeGreaterThan
- NullableDateTimeLessThan
- NullableDateTimeNoValue

Decimal

- DecimalEquals
- DecimalGreaterThan
- DecimalLessThan
- NullableDecimalEquals
- NullableDecimalNoValue

Guid

- GuidEquals
- NullableGuidEquals

- NullableGuidNoValue

Integer

- IntegerEquals
- IntegerGreaterThan
- IntegerLessThan
- NullableIntegerEquals
- NullableIntegerNoValue

String

- StringContains
- StringEndsWith
- StringEquals
- StringInCommaSeparatedList
- StringInList
- StringNoValue
- StringStartsWith

Basically, each type has an “Equals” predicate, and if they are nullable, they also include a nullable version of the “Equals” predicate and a “NoValue” predicate.

Number-related types such as Decimal, Integer and DateTime have “GreaterThan” and “LessThan” predicates.

Strings and Guids also have the “InCommaSeparatedList” predicate.

Besides, the String type has “StartsWith” and “EndsWith” predicates.

For the data reference types, the Guid-related predicate functions are used.

The functions are self-explanatory but you can [generate documentation for these predicates functions in the CMS Console](#).

To set a field filter’s value:

1. Select the field filter parameter of the **FieldPredicatesFilter**, e.g. “Email filter”
2. Click **Function** in the **Parameter type** box.
3. In the **Value for Parameter** window, select the predicate function, e.g. **Composite.Utils.Predicates.StringEndsWith**.

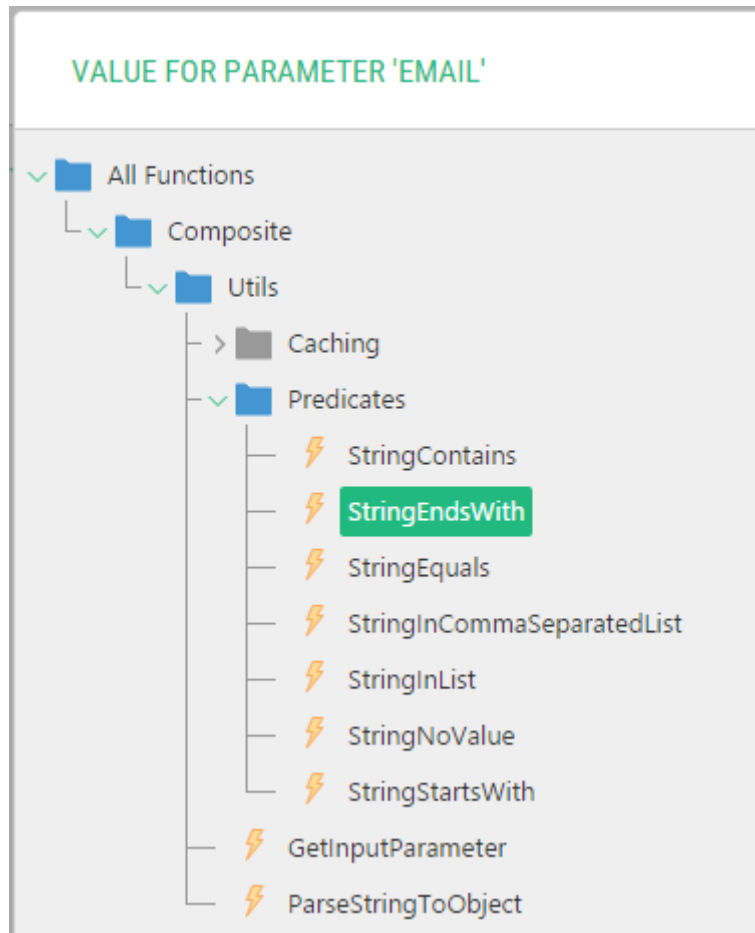


Figure 11: Selecting a predicate function for the string field filter

4. If the predicate function has parameters, set them, too. (For example, for the StringEndsWith, you should set the **Value to Compare with** parameter, e.g. “gmail.com”).

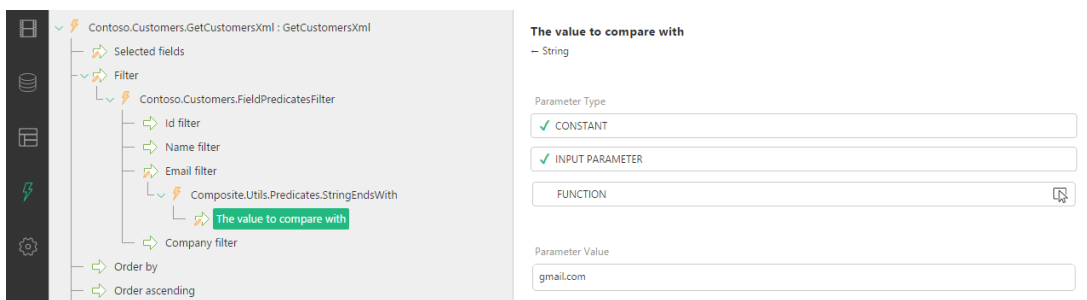


Figure 12: Setting a parameter of a predicate function

(In the example used with the steps above, those customers will be only retrieved whose email address is that of Google Mail.)

12 Using Compound Filter

To combine two or more filters specifying relations between them either logical conjunction (“and”) or disjunction (“or”), you should use the CompoundFilter.

It has three parameters, one of which sets the relation (“and” or “or”) and the other two expect one of the filter functions.

Apart from Data Reference, Fields Predicate and Active Page Reference filters, you can also specify another Compound Filter as its parameters and thus make complex combination of filters.

These are the parameters of the compound filter.

- **IsAndQuery** (Boolean): If you select “And” both filters are applied to the data. Selecting “Or” will give you the data that matches just one of the filters.
- **Left** (Expression<Func<Customers, Boolean>>): One of the two filters (the one to evaluate first)
- **Right** (Expression<Func<Customers, Boolean>>): One of the two filters (the one to evaluate last)

13 Using Active Page Reference Filter

Another type of filter is `ActivePageReferenceFilter`, which is used solely with page data folders and page metatypes.

When you use the `Get<DataType>Xml` function for these two page-specific data types, it retrieves all the data items regardless the page they are attached or added to.

The `ActivePageReferenceFilter` allows you to retrieve only those items that are specific to a certain page. And not only so.

By using its `Page Scope` parameter, you can retrieve items the current page has relation with. For example, you can retrieve data items specific to the child pages instead of, or in addition to, those specific to the current page.

The following are the page scopes you can use to control what data items to retrieve for page-specific data types.

- `<NONE>`
- Current page
- All pages (no filter)
- Ancestors and current (breadcrumbs)
- Ancestor pages
- Parent pages
- Current and descendant pages
- Child pages
- Sibling pages
- Level 1 page (homepage)
- Level 1 and descendant pages (current site)
- Level 1 and sibling pages (all homepages)
- Level 2 page
- Level 2 and descendant pages
- Level 2 and sibling pages (site main areas)
- Level 3 page
- Level 3 and descendant pages
- Level 3 and sibling pages
- Level 4 page
- Level 4 and descendant pages
- Level 4 and sibling pages

Their names are self-explanatory.

14 Other Data-Centric Functions

When you create a data type in C1 CMS, the system automatically generates the data-centric functions. They are not limited to `Get<DataType>Xml` and the filter functions (`DataReferenceFilter`, `FieldPredicatesFilter`, `CompoundFilter`, `ActivePageReferenceFilter`).

Along with `Get<DataType>Xml`, it also generates three more functions to do CRUD operations on a data type.

- `AddDataInstance`
- `UpdateDataInstance`
- `DeleteDataInstance`

These two functions get references to data items:

- `GetDataReference`
- `GetNullableDataReference`

14.1 `AddDataInstance`

The `AddDataInstance` function creates a new instance of a given type. It is a programmatic counterpart of adding a data item to a data type manually.

Most of its parameters match the names and types of the data type's fields.

It also has or may have a number of system-defined parameters.

- **Id** (Guid): The unique ID of the data item to add.
- **Pageld** (Guid): [Required] The ID of the page a page data folder or metatype are related to. Not applicable to global data types.
- **PublicationStatus** (String): The publication status of a data item to add. Only applicable to data types with enabled publication.
- **CultureName** (String): The culture name to use when adding a data item. Only applicable to data types with enabled localization.
- **SourceCultureName** (String): The source culture name to use when adding a data item. Only applicable to data types with enabled localization.
- **FieldName** (String): [Required] The unique name of the meta field to add to the page. Only applicable to page metatypes.

Please note that every time you add a new item with this function, you should generate a new ID. You can use the `Composite.Utils.Guid.NewGuid` function for that.

The function returns void.

14.2 `UpdateDataInstance`

The `UpdateDataInstance` function updates one or more instances of a given type with provided values. It is a programmatic counterpart of editing one or more data items manually.

Most of its parameters match the names and types of the data type's fields.

It also may have a number of system-defined parameters such as **Pageld** (optional here), **PublicationStatus**, **CultureName**, **SourceCultureName** and **FieldName** (optional here). Please see [AddDataInstance](#) for their description.

Besides, it has another system-defined parameter:

- **Filter** (`Expression<Func<(DataType), Boolean>>`): The filter expression to select one or more data items that match the criteria specified.

The function returns void.

14.3 DeleteDataInstance

The DeleteDataInstance function deletes one or more instances of a given type based on the filter's criteria. It is a programmatic counterpart of deleting one or more data items manually.

It has one system-defined parameter:

- **Filter** (Expression<Func<(DataType),Boolean>>): The filter expression to select one or more data items that match the criteria specified.

The function returns void.

14.4 GetDataReference

The GetDataReference function creates a reference of a specific data type based on a key value of the GUID type passed to it as a parameter.

It has one required parameter:

- **KeyValue** (Guid): The key value of the data to reference.

It returns a reference to the item of the data type it is used with.

14.5 GetNullableDataReference

The GetNullableDataReference function creates a reference of a specific data type based on a key value of the GUID type passed to it as a parameter; but, unlike GetDataReference, it can return a null reference.

It has one optional parameter:

- **KeyValue** (Guid) The key value of the data to reference.

It returns a reference to the item of the data type it is used with, or null, if none can be created.

15 Test Your Knowledge

15.1 Task: Get Data as XML

1. Create a global data type "Test.Users" with fields "Name" (string (64)), "Age" (Int32), "Email" (string (64)).
2. Add 10 data items to the data type. Use the email addresses of two email servers (e.g. "gmail.com" and "yahoo.com").
3. Create an XSLT function "Test.ShowUsers".
4. Call the Test.Users.GetUsersXml function in the Test.ShowUsers function.
5. Select the Name, Age and Email fields.
6. Edit the XSLT to display user data (Name, Age, Email) in a table.
7. Insert the Test.ShowUsers function on a page and preview the page.

15.2 Task: Sort Data

1. Edit the Test.ShowUsers function.
2. Sort the data retrieved by the Test.Users.GetUsersXml function by the Name field in ascending order.
3. Preview the Test.ShowUsers function on the page it has been inserted on.

15.3 Task: Filter Data

1. Edit the Test.ShowUsers function.
2. Use the FieldPredicatesFilter to set the filter on the Test.Users.GetUsersXml function.
3. Use the Composite.Utils.Predicates.StringEndsWith to filter data by the Email field so users with an email address of one server (e.g. "gmail.com") should be retrieved.
4. Preview the Test.ShowUsers function on the page it has been inserted on.

15.4 Task: Use Paging

1. Edit the Test.ShowUsers function.
2. Use the Page Size parameter on the Test.Users.GetUsersXml function to limit the number of retrieved data items to 2.
3. Use Include Paging Info parameter and edit the XSLT to display the total number of users.
4. Preview the Test.ShowUsers function on the page it has been inserted on.

15.5 Task: Use Active Page Reference Filter

1. Create a metatype "Test.Summary" with one field "Text" (String (1024)) changing its widget from TextBox to TextArea.
2. Add the metafield of this type to the root page of the website making sure the sub pages inherit it.
3. Edit the root and sub pages to write different summaries for each page.
4. Create an XSLT function "Test.ShowSummary" and call the Test.Summary.GetSummaryXml function in it.
5. Use the Active Page Reference Filter to retrieve the summary text related to the current page.
6. Select a page with the Summary meta field in the Page field (Debug) on the Settings tab of the Test.ShowSummary function.
7. Preview the Test.ShowSummary function and check for the summary text in the Input pane.