

Querying MongoDB

You can use the MongoDB adapter to interact with a MongoDB and to insert, get and manipulate documents.

Using one of the **find** operations from the MongoDB adapter, you can retrieve data. MongoDB stores data in form of documents that are depicted in a JSON-like format. Queries always return one or more complete documents.

For all actions that refer to existing documents, you need to provide a query string (**queryString**) to identify them. A query string contains all properties of the document you want to use for selection.

Assume we have the following sample **Customer** document structure:

```
{
  "id": "ebd7c78b-44e0-4cbd-8164-d28431716942"

  "name": "John Snow",
  "company": "Winter & Partners",
  "address": {
    "street": "99, Malamute Street",
    "city": "Anchorage, AK 99506",
    "country": "USA"
  },
  orderValue: "16323.00"
}
```

The simplest way to create a query string is the following:

1. Create an object having the structure of the document (**Customer** in the example).

```
create queryData;
```

2. Set all query values to this object (the **customerID** in the example).

```
set queryData.id = customerID;
```

3. Provide this object as **queryString** by converting it to JSON using [classToExtendedJSON\(\)](#).


```
set queryString = queryData.classToExtendedJSON();
```

To build a query string, we recommend to **not** use [concat\(\)](#) operations but to create a data structure that represents the update string and can be converted to JSON with [classToExtendedJSON\(\)](#).



Building a query string manually (e.g. using [concat\(\)](#)) is susceptible to code injection.

The MongoDB adapter comes with three **find** operations: two returning the result set in different formats, one returning a handle to the result set.

Name	Type	Description
result	Array of String	An array of all resulting documents in JSON format.
result	Array of <document class>	An array of objects of an xUML class representing the document structure. <div> This only makes sense if you know the structure of the documents you are accessing.</div>

On this Page:

- [Selecting Output Data](#)
- [Sorting](#)
- [Using Regular Expressions](#)
 - [Security Considerations](#)

MongoDBAdapter_CustomerData_Example



Click the icon to download a simple example model that shows the usage of the MongoDB adapter in **Scheer PAS Designer**.

Related Pages:

- [Querying MongoDB](#)
- [Updating MongoDB Documents](#)
- [Aggregating Data](#)
- [Inserting and Deleting Documents](#)
- [MongoDB Adapter Reference](#)
- [classToExtendedJSON\(\) Operation](#)

handle	MongoDBHandle	<p>A handle to a result set.</p> <p>This is helpful if</p> <ul style="list-style-type: none"> • you expect a huge amount of documents being returned, and do not want to load the complete result set to the memory • you want to iterate over the result set one by one anyway, and e.g. only regard a subset of the result for further processing. 	You need to process the result set one by one using fetch .
---------------	-------------------------------	--	---

Refer to the reference of [find operations](#) and [fetch operations](#) for a detailed description of all parameters and options.

Selecting Output Data

MongoDB uses the concept of "projection" to define which properties should be selected from a document. The projection is supplied to the adapter call via the **projection** attribute of the [MongoDBFindOptions](#).

The following rules apply to projections:

Rule	Example
You can select dedicated properties.	{ name: 1 }
You can select all properties and omit dedicated properties.	{ name: 0 }
You cannot mix both above mentioned rules. This will lead to an exception.	{ name: 1, company: 0 }
You can select properties from within a structure.	{ address: { street: 1 } }
You cannot select all properties and omit dedicated properties from within a structure. This will be ignored.	{ address: { street: 0 } }

Sorting

You can sort the document list you get back from an adapter call by providing the **sort** attribute of [MongoDBFindOptions](#). Parameter **sort** contains the document properties to sort by. Value **1** is ascending sorting, value **-1** is descending sorting. The order of JSON properties reflects the sort hierarchy.

```
create options;
set options.`sort` = '{"company":1,"orderVolume":-1}';
```



You need to escape the attribute name of **sort** because there is an operation having the same name.



When using single and double quotes, you do not need to escape the inner quotes.

You can also create a class defining the sort options, and use [classToExtendedJSON\(\)](#) to create the sort string.

Using Regular Expressions

You can use regular expressions to find MongoDB documents in a **LIKE** fashion. This can be done using the MongoDB operator `$regex`.

Generally, you can do this using the same three steps as described above with querying documents:

1. Create an object having the structure of the document you want to query.
In this case, you want to use the regex operator and apply it to a document property. In the example, this is the **name** property of the **Customer** object. Create a structure like the following:

RegExCustomer
+name: NameWithRegEx

NameWithRegEx
<<XMLElement>> +regexOperator: String {externalName = "\$regex"}

2. Set the query values to this object, e.g. as in the example using Action Script, where **queryData** is an object of **RegExCustomer**:

```
create queryData;  
set queryData.name.regexOperator = concat(".*\Q", customerName, "\E.  
*") if customerName.exists() and customerName != "";
```

3. Provide this object as **queryString** by converting it to JSON using **classToExtendedJSON()**.

```
set queryString = queryData.classToExtendedJSON();
```

Security Considerations



Building a query string manually (e.g. using **concat()**) is susceptible to code injection.

This also applies to building the regex operator in the example above. A maleficent user could use the variable part of the regex **customerName** to inject regex syntax.



To escape a variable part of a regular expression, you can wrap it into **\Q . \E**, like is done in the example:

```
set queryData.name.regexOperator = concat(".*\Q", customerName, "\E.  
*") if customerName.exists() and customerName != "";
```

Regular expression syntax within that enclosed part will not be considered when evaluating the expression.