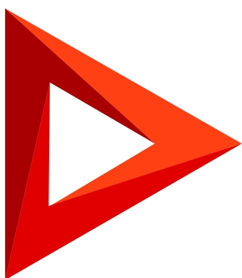
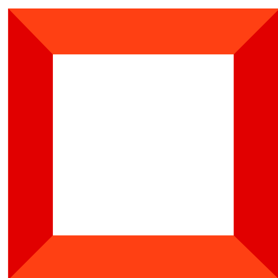
Creatio Academy

Data sources

Version 8.0

This documentation is provided under restrictions on use and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this documentation, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

© 2023 Creatio. All rights reserved.

Table of Contents

Data handling **4**

DataStorage layer 4

DataAccessObject layer 5

BusinessObject layer 6

CRUD operations with data sources **13**

Retrieve data 13

Retrieve the default value 14

Add or update data 15

Copy data 16

Delete data 16

Implement a custom request handler 17

Model class **18**

Methods 18

Process collection type data **20**

Set up pagination 21

Set up sorting 21

Set up filtering 23

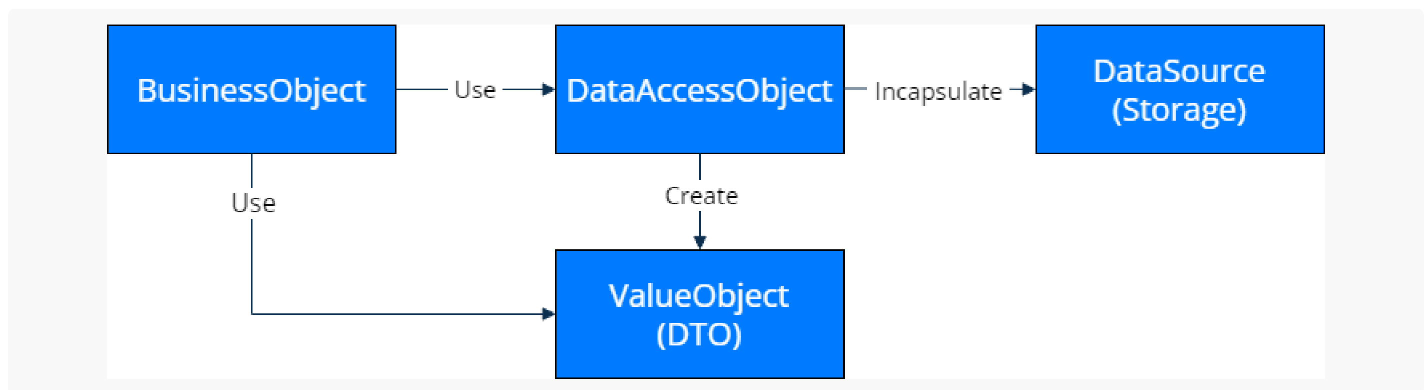
© 2023 Creatio. All rights reserved.

Data handling

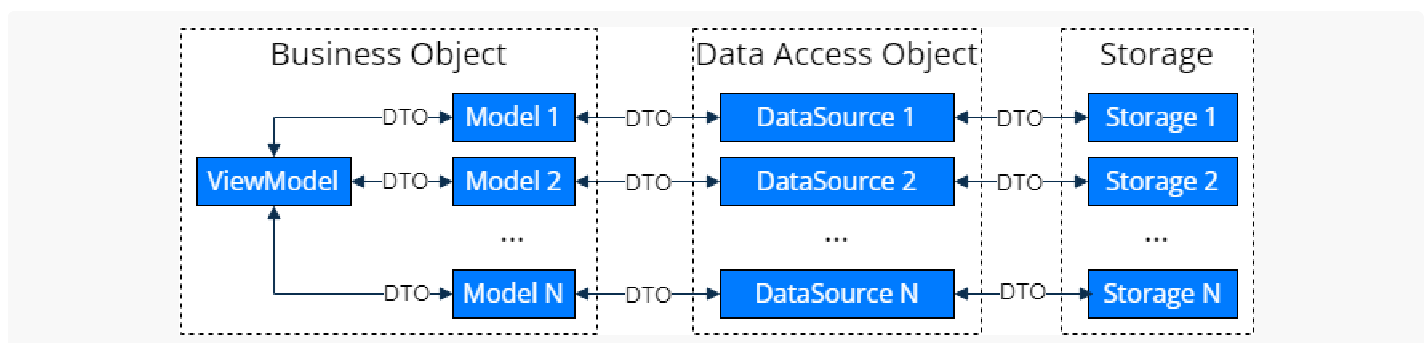
Data handling in Creatio 8.x is based on the **DAO (Data Access Object)** pattern. Learn more about the DAO pattern in Wikipedia.

The DAO pattern has the following **layers**:

- `BusinessObject`. Implements the business logic.
- `DataAccessObject`. Enables Creatio to access data. The business logic and data source are separated from each other. The DataAccessObject layer can create a **DTO (Data Transfer Object)**. The DTO can use the `BusinessObject` layer to save DTOs and return them to the DataAccessObject layer. Learn more about the DTO pattern in Wikipedia.
- `DataStorage`. Organizes various data repositories.



View the detailed data storage diagram in the figure below.



DataStorage layer

The `DataStorage` layer is linked to a particular data source.

The **types** of storage are as follows:

- Creatio database
- web service

© 2023 Creatio. All rights reserved.

- file system

In Creatio 8.0 Atlas, you can access only the Creatio database.

`DataAccessObject` layer

The `DataAccessObject` layer contains data sources (Data Source). Each data source is linked to the corresponding model within the business logic layer.

The `DataAccessObject` layer performs the following **functions**:

- Enable users to perform CRUD operations (`load`, `save`, `insert`, `delete`).
- Provide permissions for data operations (`canEdit`, `canCreate`, `canDelete`).
- Provide data structure (`getSchema`).

Data schema describes the structure of data managed by Creatio.

Data schema includes the following **components:**

- name
- title
- attributes

Important structural elements of attributes are validators. Learn more about validators in a separate article: Freedom UI page customization basics.

Learn more about the **classes** that describe the data schema and attribute structure below.

Data schema structure

```
export class DataSchema {
    public name: string;
    public caption: LocalizableString;
    public attributes: DataSchemaAttribute[];
    public primaryAttributeName?: string;
    public primaryDisplayAttributeName?: string;
}
```

Attribute schema structure

```
export class DataSchemaAttribute {
    public name: string;
    public caption: string;
    public path: string;
    public dataValueType: DataValueType;
    public validators: DataSchemaValidatorConfig;
    public defaultValue: JsonData;
```

© 2023 Creatio. All rights reserved.

```
    public isValueCloneable: boolean;
    public attributeType: DataSchemaAttributeType;
    public referenceSchemaName?: string;
}
```

Attention. Creatio version 8.0 Atlas implements `EntityDataSource` that lets you manage Creatio database.

`BusinessObject` layer

The `BusinessObject` layer contains the view model (`View model`), which, in turn, can contain several models (`Model`). Learn more about the `View model` and `Model` layers in a separate article: Creatio front-end architecture.

Model

The `modelConfig` section of the client schema describes data models. The model contains the data source description. Creatio version 8.0 Atlas implements `EntityDataSource` that lets you manage the Creatio database.

Use the data source to perform various **data actions**, for example:

- Add new and existing data source elements to the canvas of the Freedom UI Designer.
- Edit the data source elements.
- Bind the controls to the data source elements on the canvas.

View an example that registers a model in the client schema below.

Example that registers a model in the client schema

```
modelConfig: /**SCHEMA_MODEL_CONFIG*/{
    "dataSources": {
        /* Unique names of data sources. */
        "ContactDS": {
            /* Data source type. */
            "type": "crt.EntityDataSource",
            "config": {
                /* Data schema code. */
                "entitySchemaName": "Contact"
            }
        }
    }
}/**SCHEMA_MODEL_CONFIG*/
```

Learn more about the client schema structure in a separate article: Client Schema.

© 2023 Creatio. All rights reserved.

View model

The **view model** handles the `ViewModel` layer that contains the business logic of the interaction between the `View` and `Model` layers.

Configure the view model in the `viewModelConfig` section of the client schema. For each view model attribute, specify the corresponding model attribute in the `modelConfig` schema section.

View an example that describes a view model in the client schema below.

Example that describes a view model in the client schema

```
viewModelConfig: /**SCHEMA_VIEW_MODEL_CONFIG*/{
    "attributes": {
        /* Bind the ContactName view model attribute to the Name attribute of the ContactDS mode
        "ContactName": {
            "modelConfig": {
                "path": "ContactDS.Name"
            }
        }
    }
}/**SCHEMA_VIEW_MODEL_CONFIG*/
```

The view model can have the following **attribute types**:

- simple type columns
- collection columns
- direct link columns

Attributes for columns that contain simple data types

Simple data types can include the following data source elements:

- string
- number
- boolean
- date/time

To create an **attribute for a column that contains a simple data type**:

1. Register the data source in the schema of the Freedom UI page.

   Register the data source

   ```
   modelConfig: /**SCHEMA_MODEL_CONFIG*/{
       "dataSources": {
           "ContactDS": {
               "type": "crt.EntityDataSource",
   ```

© 2023 Creatio. All rights reserved.

```
            "config": {
                "entitySchemaName": "Contact",
            }
        }
    }
}/**SCHEMA_MODEL_CONFIG*/
```

2. Create an attribute in the `attributes` property of the `viewModelConfig` schema section.

Create an attribute

```
viewModelConfig: /**SCHEMA_VIEW_MODEL_CONFIG*/{
    "attributes": {
        "StringAttribute_jghoo32": {
            "modelConfig": {
                "path": "ContactDS.Name"
            }
        }
    }
}/**SCHEMA_VIEW_MODEL_CONFIG*/,
```

Attention. Use a unique attribute name.

The base attribute property is `modelConfig`. This property describes the link to the data source element using the `path` property. The `path` property value must have the following format: `[Data Source Name].[Column Code]`. For this example, the `path` property value is `ContactDS.Name`.

Attributes for collections

The view model lets you bind attributes that are data collections. To bind a collection type attribute, set the `isCollection` attribute property to `true`.

Example that binds a collection type attribute to a data model

```
viewModelConfig: /**SCHEMA_VIEW_MODEL_CONFIG*/{
    "attributes": {
        /* Bind a collection type attribute of the AccountList view model to the AccountDS model
        "AccountList": {
            "isCollection": true,
            "modelConfig": {
                "path": "AccountDS",
                "pagingConfig": {},
                "sortingConfig": {},
                "filterAttributes": {}
```

© 2023 Creatio. All rights reserved.

```
            },
            "viewModelConfig": {
                "attributes": {
                    "value": {
                        "modelConfig": {
                            "path": "AccountDS.Id"
                        }
                    },
                    "displayValue": {
                        "modelConfig": {
                            "path": "AccountDS.Name"
                        }
                    }
                }
            },
            "embeddedModel": {
                "name": "AccountDS",
                "config": {
                    "type": "crt.EntityDataSource",
                    "config": {
                        "entitySchemaName": "Account"
                    }
                }
            }
        }
    }
}/**SCHEMA_VIEW_MODEL_CONFIG*/
```

The advanced settings of the collection type attribute let you configure data pagination and sorting.

Example of advanced settings of a collection type attribute

```
viewModelConfig: /**SCHEMA_VIEW_MODEL_CONFIG*/{
    "attributes": {
        "AccountList": {
            "isCollection": true,
            "modelConfig": {
                "path": "AccountDS",
                /* Set up pagination.*/
                "pagingConfig": {
                    "rowsOffset"?: "<OffsetAttributeName>",
                    "rowCount": 5 | "<RowCountAttributeName>"
                },
                /* Set up collection sorting.*/
                "sortingConfig": {
                    "attributeName": "AccountsSorting",
                    "default": [{
```

© 2023 Creatio. All rights reserved.

```
                        "columnName": "Name",
                        "direction": "Asc"
                    }]
                }
            },
            "viewModelConfig": {...}
        }
    }
}/**SCHEMA_VIEW_MODEL_CONFIG*/
```

Attributes for direct link columns

Direct link columns are columns that contain objects linked to the current data source. For example, the `[Contact]` object contains the `[Account]` property, which is also an object. Register the attribute in the `[Name]` column of the linked `[Account]` object.

To create an attribute for a direct link column:

1. Register the data source in the page schema.

   Example that registers a data source

   ```
   modelConfig: /**SCHEMA_MODEL_CONFIG*/{
       "dataSources": {
           "ContactDS": {
               "type": "crt.EntityDataSource",
               "config": {
                   "entitySchemaName": "Contact",
               }
           }
       }
   }/**SCHEMA_MODEL_CONFIG*/
   ```

2. Create a page schema attribute in the `attributes` property of the `viewModelConfig` schema section.

   Example that creates an attribute

   ```
   viewModelConfig: /**SCHEMA_VIEW_MODEL_CONFIG*/{
       "attributes": {
           "AccountName": {
               "modelConfig": {
                   "path": "ContactDS.AccountName"
               }
           }
       }
   }/**SCHEMA_VIEW_MODEL_CONFIG*/,
   ```

© 2023 Creatio. All rights reserved.

The `path` property stores not the path to the column, but the path to the data source attribute that will be created on the next step. The `path` property value must have the following format: `[Data Source Name].[Data Source Attribute Name]`, for example, `ContactDS.AccountName`.

3. Create an attribute in the `modelConfig` schema section.

Example that creates a data source attribute

```
modelConfig: /**SCHEMA_MODEL_CONFIG*/{
    "dataSources": {
        "ContactDS": {
            "type": "crt.EntityDataSource",
            "config": {
                "entitySchemaName": "Contact",
                "attributes": {
                    "AccountName": {
                        "path": "Account.Name",
                        "type": "ForwardReference"
                    }
                }
            }
        }
    }
}/**SCHEMA_MODEL_CONFIG*/,
```

Attention. The name of the data source attribute must match the attribute name specified in the `path` property of the `viewModelConfig` schema section on step 2.

You can create a direct link attribute that has multiple nesting levels. For example, you can display the city name of the account linked to the current contact. In this case, the data source attribute can look like this:

Example that creates an attribute that has multiple nesting levels

```
"dataSources": {
    "ContactDS": {
        "type": "crt.EntityDataSource",
        "config": {
            "entitySchemaName": "Contact",
            "attributes": {
                "AccountCityName": {
                    "path": "Account.City.Name",
                    "type": "ForwardReference"
                },
            }
        }
    }
```

© 2023 Creatio. All rights reserved.

```
}
```

Embedded model

The **Embedded Model** is a data model embedded into the view model without being connected to the data source.

Creatio lets you use the embedded model for various **collection type elements**, for example:

- index of list values
- drop-down list generated from a lookup

Creatio does not display embedded data models in the Freedom UI Designer, nor does it let you bind embedded models to controls on the canvas.

Example that sets up an embedded model

```
viewModelConfig: /**SCHEMA_VIEW_MODEL_CONFIG*/{
    "attributes": {
        "LookupAttributeCityList": {
            "isCollection": true,
            "modelConfig": {
                "path": "EmbeddedDS"
            },
            "embeddedModel" {
                "name": "EmbeddedDS",
                "config": {
                    "type": "crt.EntityDataSource",
                    "config": {
                        "entitySchemaName": "City"
                    }
                }
            }
            "viewModelConfig": {
                "attributes": {
                    "value": {
                        "modelConfig": {
                            "path": "EmbeddedDS.Id"
                        }
                    },
                    "displayValue": {
                        "modelConfig": {
                            "path": "EmbeddedDS.Name"
                        }
                    }
                }
            }
        }
```

© 2023 Creatio. All rights reserved.

```
    }
}/**SCHEMA_VIEW_MODEL_CONFIG*/,
```

CRUD operations with data sources

In Creatio, you can perform CRUD operations with data sources using handlers that can be bound to buttons or other system actions, for example, changing an attribute value.

View a general example that binds a handler to a button below:

Example that binds a handler to a button

```
{
    "operation": "insert",
    "name": "Button",
    "values": {
        "type": "crt.Button",
        "clicked": {
            "request": "SOME_HANDLER_NAME",
            "params": {
                ...
            }
        },
    },
},
```

The `clicked` property binds handlers to buttons and manages the action executed on button click.

The `clicked` property includes the following **internal properties**:

- `request`. The handler name.
- `params`. The index of parameters needed to manage the handler.

This article describes CRUD operations with data sources using the example that binds a handler to a button.

Retrieve data

Use the `crt.LoadDataRequest` handler to **retrieve data from a data source**.

Example that binds the handler that retrieves data

```
{
    "operation": "insert",
    "name": "Button",
    "values": {
        "type": "crt.Button",
```

© 2023 Creatio. All rights reserved.

```
            "clicked": {
                "request": "crt.LoadDataRequest",
                "params": {
                    "dataSourceName": "ContactDS",
                    "parameters": [{type: "primaryColumnValue", value: "SOME_CONSTANT_OR_ATTRIBUTE"}
                    "config": {
                        "loadType": "load",
                        "payload": "SOME_CUSTOM_OBJECT"
                    }
                },
            },
        },
    },
```

The `crt.LoadDataRequest` handler includes the following **parameters**:

- `dataSourceName`. The data source name. Required.
- `parameters`. An array of objects that contains parameters (filters) to apply when retrieving data. Required.
- `config`. The configuration object that manages data upload to the page.

`parameters` array of objects includes the following **properties**:

- `type`. The filter type. Available values: `primaryColumnValue`, `filter`, `primaryDisplayValueFilter`.
- `value`. The filter value. Can be a constant (any custom value: string, number, etc.) or attribute. You can specify an attribute with or without the “$” character, for example, `$ContactFilter`. Create a filter in the `attributes` property of the `viewModelConfig` schema section. If the filter finds more than 1 record, only the first record is loaded.

The `config` property includes the following **internal properties**:

- `loadType`. How to load the data collection. Available **values**:
  - `load`. Load every record that meets the filter conditions without pagination.
  - `loadNext`. Perform an offset (i. e., the `rowsOffset` property value) as part of the first query and load the number of records specified in the `rowCount` property. Load the next portion of data as part of the following queries. The offset is the number of already loaded collection records, and the `rowCount` value remains unchanged for every query.
  - `reload`. Set the offset to the value of the attribute specified as the `rowsOffset` property. If the property is not specified, set the offset to 0 and load the number of records specified in the `rowCount` property. The collection of the loaded data is cleared.
- `payload`. Pass additional query parameters (filters) in the internal `parameters` property

Retrieve the default value

Use the `crt.LoadDefaultValuesRequest` handler to **retrieve the default value from a data source**.

© 2023 Creatio. All rights reserved.

Example that retrieves the default value

```
{
    "operation": "insert",
    "name": "Button_nptvn7b",
    "values": {
        "type": "crt.Button",
        "clicked": {
            "request": "crt.LoadDefaultValuesRequest",
            "params": {
                "dataSourceName": "ContactDS",
            },
        },
    },
},
```

The `dataSourceName` parameter of the `crt.LoadDefaultValuesRequests` handler is required. The parameter contains the data source name.

Add or update data

Use the `crt.SaveDataRequest` handler to **add or update data source data**. The handler updates data if a data source is loaded, otherwise, data is added.

Example that binds the handler that adds or updates data

```
{
    "operation": "insert",
    "name": "Button",
    "values": {
        "type": "crt.Button",
        "clicked": {
            "request": "crt.SaveDataRequest",
            "params": {
                "dataSourceName": "ContactDS",
                "parameters": [{type: "primaryColumnValue", value: "SOME_CONSTANT_OR_ATTRIBUTE"}
                "config": {
                    "silent": "true",
                    "payload": "SOME_CUSTOM_OBJECT"
                }
            },
        },
    },
},
```

© 2023 Creatio. All rights reserved.

The `crt.SaveDataRequest` handler includes the following **parameters**:

- `dataSourceName`. The data source name. Required.
- `parameters.` An array of objects that contains parameters (filters) that specify the record to update. Required.
- `config`. The configuration object that manages page data addition or update.

The `config` property includes the **internal** `silent` **property**. The internal property specifies whether to display a server error that might occur when adding or updating data (`true` – yes, `false` – no).

Copy data

Use the `crt.CopyDataRequest` handler to **copy data from a data source**.

Example that binds the handler that copies data

```
{
    "operation": "insert",
    "name": "Button",
    "values": {
        "type": "crt.Button",
        "clicked": {
            "request": "crt.CopyDataRequest",
            "params": {
                "dataSourceName": "ContactDS",
                "recordId": "SOME_CONSTANT"
            },
        },
    },
},
```

The `crt.CopyDataRequest` handler includes the following **required parameters**:

- `dataSourceName`. The data source name.
- `recordId`. The ID of the record to copy.

The `crt.CopyDataRequest` handler does not send a direct query to the server. The data of the copied record is stored in memory. The next time records are added, the copied data is merged with the data the user enters using the Creatio UI.

Delete data

Use the `crt.DeleteDataRequest` handler to **delete data from a data source**.

Example that binds the handler that deletes data

© 2023 Creatio. All rights reserved.

```
{
    "operation": "insert",
    "name": "Button",
    "values": {
        "type": "crt.Button",
        "clicked": {
            "request": "crt.DeleteDataRequest",
            "params": {
                "dataSourceName": "ContactDS",
                "parameters": [{type: "primaryColumnValue", value: "SOME_CONSTANT_OR_ATTRIBUTE"}
                "config": {
                    "payload": "SOME_CUSTOM_OBJECT"
                }
            },
        },
    },
},
```

The `crt.DeleteDataRequest` handler includes the following **parameters**:

- `dataSourceName`. The data source name. Required.
- `parameters`. An array of objects that contains parameters (filters) that specify the record to delete. Required.
- `config`. The configuration object that manages page data deletion. Includes an internal payload property that lets you pass additional parameters (filters) to the query in the internal `parameters` property.

Implement a custom request handler

Creatio lets you manage data sources using both out-of-the-box and custom handlers.

Implement a custom handler in the `handlers` schema section. Use the `sdk.Model` class to access the data source.

Example of a custom handler

```
define("StudioHomePage", /**SCHEMA_DEPS*/[]/**SCHEMA_DEPS*/, function/**SCHEMA_ARGS*/()/**SCHEMA
    return {
        ...
        handlers: /**SCHEMA_HANDLERS*/[
            {
                "request": "crt.HandleViewModelInitRequest",
                "handler": async (request, next) => {
                    const accountModel = await sdk.Model.create('Account');
                    const data = await accountModel.load(['Id', 'Name'], [], {});
                    console.log(data);
                    next.handle(request);
```

© 2023 Creatio. All rights reserved.

```
                }
            }
        ]/**SCHEMA_HANDLERS*/,
        ...
    };
});
```

This example creates the `crt.HandleViewModelInitRequest` handler in the `handlers` schema section. Then, Creatio passes an object that has the `entitySchemaName: 'Account'` data source configuration to the `create()` method of the `sdk.Model` class and creates the `accountModel` model. The model calls the `load()` method, which, in turn, loads the data of the columns to pass in method parameters (e. g., `[Id]` and `[Name]`).

You can bind a custom handler to UI events similarly to the out-of-the-box Creatio handlers.

Model class JS

Medium

The `sdk.Model` class lets you access the data source.

Pass the name of the data source entity to the public `create()` method to create a `Model` class entity.

Example that creates a `Model` class entity

```
const someModel = await sdk.Model.create('SomeEntity');
```

Methods

```
load(loadConfig?: JsonObject)
```

Retrieves the data.

Parameters

```
{JsonObject} loadConfig
```

Configuration object.

A configuration object is a key-value collection. View the item collection structure in the example below:

Item configuration object

```
loadConfig?: {
    attributes?: string[];
    parameters?: DataSourceParameters;
```

© 2023 Creatio. All rights reserved.

```
    options?: DataSourceLoadOptions;
}
```

Item properties

`{string[]} attributes`	The index of attributes to retrieve.
`{DataSourceParameters} parameters`	An array of query filters.
`{DataSourceLoadOptions} options`	Additional properties required to set up pagination and sorting.

`insert(dto: JsonObject)`

Adds the data.

Parameters

`{JsonObject} dto`	JSON object that contains column names as keys and column values as key values.

`update(dto: JsonObject, parameters: DataSourceParameters)`

Updates the data.

Parameters

`{JsonObject} dto`	JSON object that contains column names as keys and column values as key values.
`{DataSourceParameters} parameters`	An array of query filters.

`copy(primaryColumnValue: string | JsonObject, data: JsonObject = {})`

Retrieves the record in copy mode. Clonable attributes are loaded. The next time the insert() method is called, the new record is saved and the clonable attribute data is copied from the original record.

Parameters

© 2023 Creatio. All rights reserved.

`{string | JsonObject} primaryColumnValue`	The primary record key.
`{JsonObject} data`	JSON object that contains column names as keys and column values as key values. The default value is an empty object.

`delete(parameters: DataSourceParameters)`

Deletes the data.

Parameters

`{DataSourceParameters} parameters`	An array of query filters.

`canSave(params: DataSourceCanExecuteOperationPayload)`

Checks for user permissions to save the model.

Parameters

`{DataSourceCanExecuteOperationPayload} params`	An array of query filters.

`canDelete(params: DataSourceCanExecuteOperationPayload)`

Checks for user permissions to delete the model.

Parameters

`{DataSourceCanExecuteOperationPayload} params`	An array of query filters.

Process collection type data

Additional data processing can be required when displaying collection type attributes.

Creatio lets you manage:

- pagination

© 2023 Creatio. All rights reserved.

- sorting
- filtering

Set up data processing in the `viewModelConfig` schema section.

Set up pagination

Use the internal `pagingConfig` property of the collection type attribute's modelConfig property to **set up pagination**.

The `pagingConfig` property includes the following **internal properties**:

- `rowCount`. The number of records to upload to the page. The `rowCount` property value can be both a constant and the name of the attribute that contains this number.
- `rowsOffset`. An initial position (offset) to load the first portion of data. Can only be the name of the attribute that contains the offset number, not a constant. If you omit the property, Creatio sets the offset to 0.

Example of pagination

```
viewModelConfig: /**SCHEMA_VIEW_MODEL_CONFIG*/{
    "attributes": {
        "LookupAttribute": {
            "isCollection": true,
            "modelConfig": {
                ...
                "pagingConfig": {
                    "rowCount": SOME_QUANTITY_OF_DATA,
                    "rowsOffset": "Some_Offset_Of_Data",
                },
                ...
             },
             "viewModelConfig": {
                ...
             },
             "embeddedModel": {
                 ...
            }
        }
    }
}/**SCHEMA_VIEW_MODEL_CONFIG*/,
```

Set up sorting

Use the internal `sortingConfig` property of the collection type attribute's `modelConfig` property to **set up sorting**.

The `sortingConfig` property includes the following **internal properties**:

© 2023 Creatio. All rights reserved.

- `attributeName`. Sorting attribute that manages sorting in Creatio UI, for example, in the section list. Required to load new data.

> **Attention.** You do not need to set the `columnName` and `direction` of the sorting attribute in the source code of the Freedom UI page schema. Creatio manages the values of this attribute automatically.

- `default`. Specifies the initial data sorting settings. An array of objects that have the following **properties**:
  - `columnName`. Name of the column by which to sort data.
  - `direction`. Sorting order. Available values: `asc` (ascending), `desc` (descending).

Example of sorting

```
viewModelConfig: /**SCHEMA_VIEW_MODEL_CONFIG*/{
    "attributes": {
        "LookupAttribute": {
            "isCollection": true,
            "modelConfig": {
                ...
                "sortingConfig": {
                    "attributeName": "Some_Attribute_Name",
                    "default": [
                        {
                            "columnName": SomeColumnName,
                            "direction": 'asc',
                         },
                     ],
                 },
                 ...
            },
            "viewModelConfig": {
                ...
            },
            "embeddedModel": {
        ...
            }
        },
        "Some_Attribute_Name": {
            "isCollection": true,
            "viewModelConfig": {
                "attributes": {
                    "columnName": {},
                    "direction": {},
                },
            },
         }
    }
```

© 2023 Creatio. All rights reserved.

```
}/**SCHEMA_VIEW_MODEL_CONFIG*/,
```

Set up filtering

Use the internal `filterAttributes` property of the collection type attribute's `modelConfig` property to **set up filtering**.

The `filterAttributes` property is an array of objects that have the following **properties**:

- `name`. Name of the attribute to filter data. For example, "`FolderTree_items_DS_filter`." Declare the attribute in the `viewModelConfig` schema section. Set the value property of this attribute to the object that configures the filter based on `EntitySchemaQuery`.
- `loadOnChange`. Specifies whether to reload the collection on filter change.

Example of filtering

```
viewModelConfig: /**SCHEMA_VIEW_MODEL_CONFIG*/{
    "attributes": {
        /* Collection type attribute to filter. */
        "FolderTree_items": {
            "isCollection": true,
            "viewModelConfig": {
                ...
            },
            "modelConfig": {
                "path": "FolderTree_items_DS",
                /* Set up filtering. */
                "filterAttributes": [
                    {
                        /* Name of the attribute to filter data. */
                        "name": "FolderTree_items_DS_filter",
                        "loadOnChange": true
                    }
                ]
            },
            "embeddedModel": {
                ...
            }
        },
        /* The attribute to filter data.*/
        "FolderTree_items_DS_filter": {
            "value": {
                "isEnabled": true,
                "trimDateTimeParameterToDate": false,
                "filterType": 6,
                "logicalOperation": 0,
                /* Load all folders whose EntitySchemaName = Account.*/
```

© 2023 Creatio. All rights reserved.

```
                    "items": {
                        "6f20f5b6-42e6-d6b8-caee-5e54636b76d1": {
                            "isEnabled": true,
                            "trimDateTimeParameterToDate": false,
                            "filterType": 1,
                            "comparisonType": 3,
                            "leftExpression": {
                                "expressionType": 0,
                                "columnPath": "EntitySchemaName"
                            },
                            "rightExpression": {
                                "expressionType": 2,
                                "parameter": {
                                    "dataValueType": 1,
                                    "value": "Account"
                                }
                            }
                        }
                    }
                }
            }
        }
}/**SCHEMA_VIEW_MODEL_CONFIG*/,
```

© 2023 Creatio. All rights reserved.