Creatio Academy

CRUD operations

CRUD operations with data sources

Version 8.0

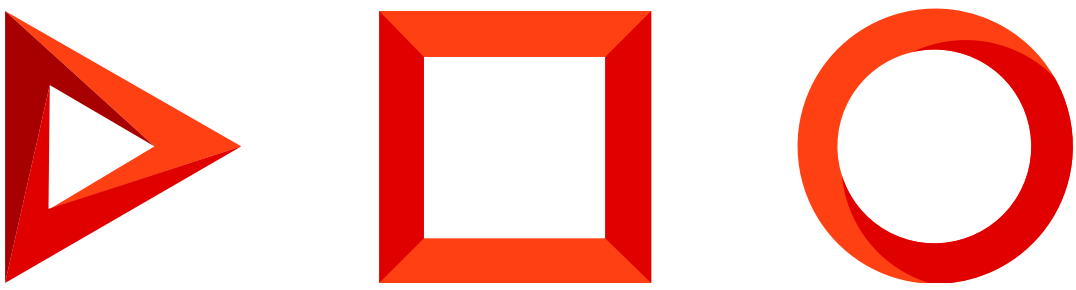

Table of Contents

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",
        "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**.

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"
                }
            },
        },
    },
},
```

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

```
{
    "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);
                }
            }
        ]/**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;
    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

{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.