Creatio Academy

Detail

Version 8.0

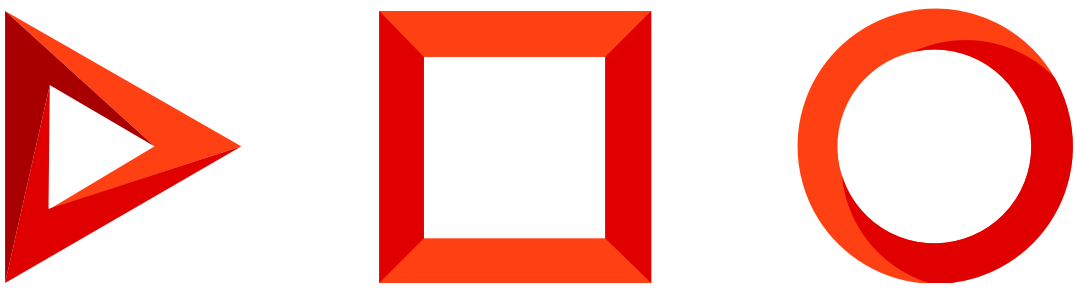


Table of Contents

Detail

Beginner

A **detail** is a UI element on the record page that displays records of an object bound to the current record. For example, the contact page stores data about contact activities, addresses, documents, etc. in details. Most details have a dedicated list. Some details, for example, [*Communication options*], are not displayed as a list. Visually, a detail and a field group are different in that the former has a data management toolbar. The available actions include adding and editing records, sorting, filtering, setting up the detail, and more.

The **purpose** of a detail is to display additional data relevant to the main section object. Creatio displays section details on tabs of the section record page in the tab container.

Detail structure and types

A detail includes the following **components**:

- **Detail object schema**. Connected to the section object. For example, the `ContactAddress` object schema of the `Base` package corresponds to the [*Addresses*] detail of the contact page. The detail is connected to the section object via the required [*Contact*] column of the detail object.
- Set up the structure, position, and behavior of detail UI elements in the **detail view model schema**. For example, set up the [*Addresses*] detail on the contact page in the `ContactAddressDetailV2` view model schema of the detail that inherits the `Uiv2` package's `BaseAddressDetailV2` schema.
- Set up the detail page in the **view model schema of the detail record page**. For example, set up the properties of the [*Addresses*] detail page on the contact page in the `ContactAddressPageV2` view model schema of the detail record that inherits the `Uiv2` package's `BaseAddressPageV2` schema.

Creatio provides the following detail **types**:

- Editable list detail
- Add page detail
- Lookup detail
- Field detail
- [*Attachments*] type detail

The detail type depends on the input and display method.

Implement a detail

The `BaseDetailV2` schema of the `NUI` package implements the functionality of a base detail.

Implement a detail using the following **tools**:

- Detail Wizard
- Creatio IDE

The Detail Wizard is insufficient to implement certain detail types. In those cases, use both the Detail Wizard and Creatio IDE. Cases of implementing various detail types are covered below.

The general procedure to implement a detail in the **Detail Wizard** is as follows:

1. **Create a custom detail**. To do this, follow the guide in the user documentation: Create a detail.
2. **Add the custom detail to a record page**. To do this, follow the guide in the user documentation: Add an existing detail on a record page.
3. **Set up the appearance of the custom detail** (optional). To do this, follow the guide in the user documentation: Add an existing detail on a record page.

The general procedure to implement a detail in **Creatio IDE** is as follows:

1. **Create a custom detail**.
   a. Create a detail object schema. To do this, follow the guide in a separate article: Develop configuration elements.
   b. Create a view model schema of the detail. To do this, follow the guide in a separate article: Develop configuration elements.
   c. Add custom detail styles (optional).
   d. Register the detail in the database (optional).
2. **Add the custom detail to a record page**.
   Create a replacing view model schema of the record page to place the detail. To do this, follow the guide in a separate article: Develop configuration elements.
3. **Set up the appearance of the custom detail** (optional). To do this, follow the guide in the user documentation: Add an existing detail on a record page.

Implement an editable list detail

An **editable list detail** lets you enter and edit data in the detail list. The data is displayed as a list. An editable list detail is a subtype of a list detail. The `BaseGridDetailV2` schema of the `NUI` package implements the functionality of a base list detail. For example, the [*Products*] detail on the order page is an editable list detail. You can edit the data of each product on the order page.

Implement an editable list detail in the Detail Wizard

1. **Create a custom detail**.

   - Set up the editable list. To do this, select the [*Make the list editable*] checkbox. Otherwise, the Wizard will create an add page detail.
   - Set up multi-line text (optional). If you want to display data in multiple lines, select the [*Multi-line text*] checkbox. The multi-line option is available only for [*String*] type columns.

2. Take step 2 of the general procedure to implement a detail in the Detail Wizard.

Implement an editable list detail in Creatio IDE

Attention. Use the Detail Wizard to implement an editable list detail. If the [*Make the list editable*] checkbox in the Detail Wizard is cleared, use Creatio IDE to implement an editable list detail.

To implement an editable list detail in **Creatio IDE**:

1. **Create a custom detail**.

   a. Create a detail object schema.

      - Select `BaseEntity` as the parent object.
      - Add a [*String*] type column and other required columns to the object schema.

   d. Create a view model schema of the editable list detail.

      - Select `BaseGridDetailV2` as the parent object.
      - Select the [*Caption*] localizable string in the context menu of the [*Localizable strings*] node on the toolbar and specify the detail name in the [*Value*] property.
      - Implement an editable list.
        - Add the `ConfigurationGrid`, `ConfigurationGridGenerator`, `ConfigurationGridUtilities` module schemas as dependencies.
        - Add the `ConfigurationGridUtilities` mixin to the `mixins` property.
        - Add the `IsEditable` attribute to the `attributes` property. Set the `value` property of the attribute to `true`.
      - Implement multi-line text (optional). To do this, add a [*String*] type column to the `attributes` property. Set the `contentType` column property to `Terrasoft.ContentType.LONG_TEXT`.

      View the example of the `ContactPageV2` view model schema of the `UsrCourierServiceDetail` editable list detail that has multi-line text in the [*UsrDescription*] column below.

      Example of the replacing view model schema

      ```
      /* Define the schema and set its dependencies on other modules. */
      ```

```
define("UsrCourierServiceDetail", ["ConfigurationGrid", "ConfigurationGridGenerator",
    "ConfigurationGridUtilities"], function() {
    return {
        /* The name of the detail object schema. */
        entitySchemaName: "UsrCourierService",
        /* The list of schema attributes. */
        attributes: {
            /* The flag that enables editability. */
            "IsEditable": {
                /* Set the data type to boolean. */
                dataValueType: Terrasoft.DataValueType.BOOLEAN,
                /* Set the attribute type to a virtual column of the view model. */
                type: Terrasoft.ViewModelColumnType.VIRTUAL_COLUMN,
                /* The value to set. */
                value: true
            }
        },
        /* The mixins used. */
        mixins: {
            ConfigurationGridUtilities: "Terrasoft.ConfigurationGridUtilities"
        },
        /* The array of view model modifications. */
        diff: /**SCHEMA_DIFF*/[
            {
                /* Set the operation type to merge. */
                "operation": "merge",
                /* The name of the schema element on which to execute the operation. */
                "name": "DataGrid",
                /* The object whose properties to merge with the schema element properties
                "values": {
                    /* The class name. */
                    "className": "Terrasoft.ConfigurationGrid",
                    /* The view generator must generate only a part of the view. */
                    "generator": "ConfigurationGridGenerator.generatePartial",
                    /* Bind the active string edit element's configuration retrieval event
                    "generateControlsConfig": {"bindTo": "generateActiveRowControlsConfig"
                    /* Bind the active record change event to the handler method. */
                    "changeRow": {"bindTo": "changeRow"},
                    /* Bind the record’s selection cancel event to the handler method. */
                    "unSelectRow": {"bindTo": "unSelectRow"},
                    /* Bind the list click event to the handler method. */
                    "onGridClick": {"bindTo": "onGridClick"},
                    /* The operations to execute on the active record. */
                    "activeRowActions": [
                        /* Set up the Save action. */
                        {
                            /* The name of the control class to which the action is connect
                            "className": "Terrasoft.Button",
                            /* Set the button style to transparent. */
```

```
                            "style": this.Terrasoft.controls.ButtonEnums.style.TRANSPARENT,
                            /* The tag. */
                            "tag": "save",
                            /* The marker value. */
                            "markerValue": "save",
                            /* Bind to the button image. */
                            "imageConfig": {"bindTo": "Resources.Images.SaveIcon"}
                        },
                        /* Set up the Cancel action. */
                        {
                            "className": "Terrasoft.Button",
                            "style": this.Terrasoft.controls.ButtonEnums.style.TRANSPARENT,
                            "tag": "cancel",
                            "markerValue": "cancel",
                            "imageConfig": {"bindTo": "Resources.Images.CancelIcon"}
                        },
                        /* Set up the Delete action. */
                        {
                            "className": "Terrasoft.Button",
                            "style": this.Terrasoft.controls.ButtonEnums.style.TRANSPARENT,
                            "tag": "remove",
                            "markerValue": "remove",
                            "imageConfig": {"bindTo": "Resources.Images.RemoveIcon"}
                        }
                    ],
                    /* Bind to the method that initializes listening to button click events
                    "initActiveRowKeyMap": {"bindTo": "initActiveRowKeyMap"},
                    /* Bind the active record's action execution event to the handler metho
                    "activeRowAction": {"bindTo": "onActiveRowAction"},
                    /* Flag that enables selection of multiple records. */
                    "multiSelect": {"bindTo": "MultiSelect"},
                    /* The description column. */
                    "UsrDescription": {
                        /* Set the content type to long text. */
                        "contentType": Terrasoft.ContentType.LONG_TEXT
                    }
                }
            }
        ]/**SCHEMA_DIFF*/
    };
});
```

i. Register the detail in the database. To do this, execute the SQL query to the `[SysDetails]` database table.

SQL query

```
DECLARE
-- The name of the detail schema.
```

```
@ClientUnitSchemaName NVARCHAR(100) = 'UsrDetailSchemaName',
-- The name of the detail object schema.
@EntitySchemaName NVARCHAR(100) = 'UsrDetailObjectSchemaName',
-- The name of the detail.
@DetailCaption NVARCHAR(100) = 'DetailName'

INSERT INTO SysDetail(
    Caption,
    DetailSchemaUId,
    EntitySchemaUId
)
VALUES(
    @DetailCaption,
    (SELECT TOP 1 UId
    from SysSchema
    WHERE Name = @ClientUnitSchemaName),
    (SELECT TOP 1 UId
    from SysSchema
    WHERE Name = @EntitySchemaName)
)
```

Register the detail to make it visible in the Section Wizard and Detail Wizard.

2. **Add the custom detail to a record page**.

   Create a replacing view model schema of the record page to place the editable list detail.

   - Select the view model schema to replace as the parent object.
   - Add the detail to the `details` property.
   - Add the configuration object of the detail view model to the `diff` modification array.

   View the example of the `ContactPageV2` view model schema of the record page, on which the `UsrRegDocumentFieldsDetail` editable list detail, below.

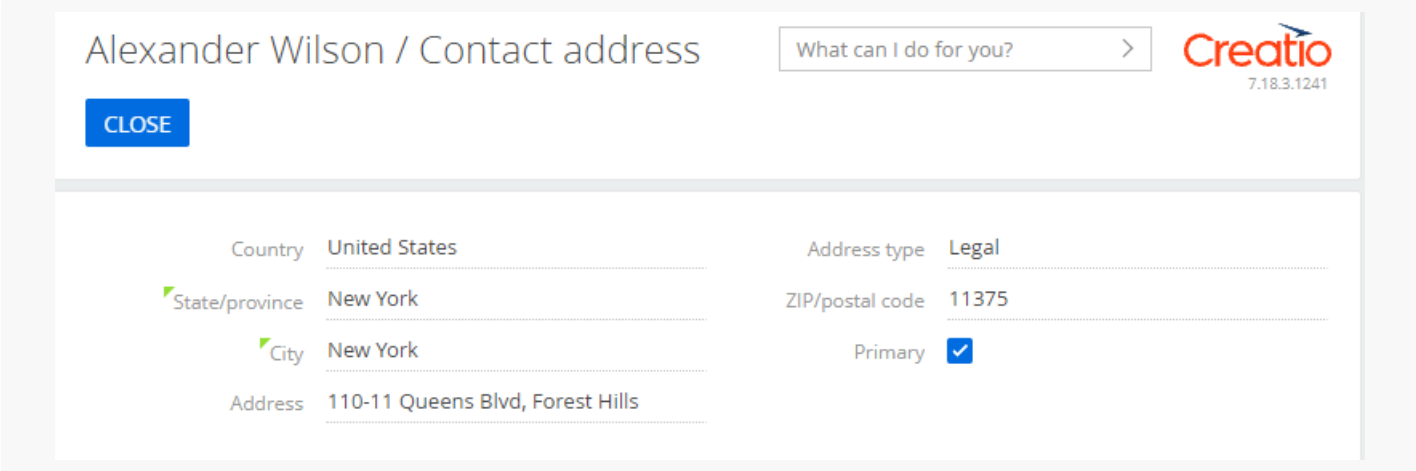
   Example of the replacing view model schema

```
define("ContactPageV2", [], function() {
    return {
        entitySchemaName: "Contact",
        details: /**SCHEMA_DETAILS*/ {
            /* Add a field detail. */
            "UsrRegDocumentFieldsDetail": {
                /* The name of the detail's client schema name. */
                "schemaName": "UsrRegDocumentFieldsDetail",
                /* Filter records of the current contact detail. */
                "filter": {
                    /* The detail object column. */
```

```
                    "detailColumn": "UsrContact",
                    /* The contact ID column. */
                    "masterColumn": "Id"
                }
            }
        } /**SCHEMA_DETAILS*/ ,
        diff: /**SCHEMA_DIFF*/ [{
            /* Add a new element. */
            "operation": "insert",
            /* The element name. */
            "name": "UsrRegDocumentFieldsDetail",
            /* The configuration object of values. */
            "values": {
                /* The element type. */
                "itemType": Terrasoft.ViewItemType.DETAIL
            },
            /* The container element name. */
            "parentName": "HistoryTab",
            /* The property name of the container element that includes the nested element co
            "propertyName": "items",
            /* The index of the element to add to the collection. */
            "index": 0
        }] /**SCHEMA_DIFF*/
    };
});
```

Implement an add page detail

An **add page detail** lets you enter and edit data on the detail page. An add page detail is a subtype of a list detail. The `BaseGridDetailV2` schema of the `NUI` package implements the functionality of a base list detail. For example, the [*Addresses*] detail of the contact page is an add page detail. You can add and edit the data of each address on the [*Contact address*] page.



To implement an add page detail in the **Detail Wizard**, follow the general procedure to implement a detal in the

Detail Wizard.

To implement an add page detail in **Creatio IDE**:

1. **Create a custom detail**.

   a. Create a detail object schema.

   - Select `BaseEntity` as the parent object.
   - Add the needed columns to the object schema.

   d. Create a view model schema of the add page detail.

   - Select `BaseGridDetailV2` as the parent object.
   - Select the [*Caption*] localizable string in the context menu of the [*Localizable strings*] node on the toolbar and specify the detail name in the [*Value*] property.

   g. Create a view model schema of the detail record page. To do this, follow the guide in a separate article: Develop configuration elements.

   - Select `BasePageV2` as the parent object.
   - Add the configuration object of the detail view model to the `diff` modification array.

   View the example of the `UsrCourierDetailPage` view model schema of the `UsrCourierInOrder` detail record page below.

Example of the detail record page's view model schema

```
define("UsrCourierDetailPage", [], function() {
    return {
        /* The name of the detail object schema. */
        entitySchemaName: "UsrCourierInOrder",
        details: /**SCHEMA_DETAILS*/{}/**SCHEMA_DETAILS*/,
        /* The array of modifications. */
        diff: /**SCHEMA_DIFF*/[
            /* The metadata for adding the [Order] field. */
            {
                "operation": "insert",
                /* The field name. */
                "name": "Order",
                "values": {
                    /* Set up the field location on the record page. */
                    "layout": {
                        "colSpan": 12,
                        "rowSpan": 1,
                        "column": 0,
                        "row": 0,
                        "layoutName": "Header"
                    },
                    /* Bind to the [Order] column of the object schema. */
```

```
                    "bindTo": "UsrOrder"
                },
                "parentName": "Header",
                "propertyName": "items",
                "index": 0
            },
            /* The metadata for adding the [Contact] field. */
            {
                "operation": "insert",
                /* The field name. */
                "name": "Contact",
                "values": {
                    /* Set up the field location on the record page. */
                    "layout": {
                        "colSpan": 12,
                        "rowSpan": 1,
                        "column": 12,
                        "row": 0,
                        "layoutName": "Header"
                    },
                    /* Bind to the [Contact] column of the object schema. */
                    "bindTo": "UsrContact"
                },
                "parentName": "Header",
                "propertyName": "items",
                "index": 1
            }
        ]/**SCHEMA_DIFF*/,
        methods: {},
        rules: {}
    };
});
```

j. Register the detail in the database.

a. Register the connection between the detail object schema and the detail list schema. To do this, execute the SQL query to the `[SysDetails]` database table.

SQL query

```
DECLARE
-- The name of the detail schema.
@DetailSchemaName NCHAR(100) = 'UsrDetailSchemaName',
-- The name of the detail object schema.
@EntitySchemaName NVARCHAR(100) = 'UsrDetailObjectSchemaName',
-- The name of the detail.
@DetailCaption NVARCHAR(100) = 'DetailName'
```

```
INSERT INTO SysDetail(
    ProcessListeners,
    Caption,
    DetailSchemaUId,
    EntitySchemaUId
)
VALUES (
    0,
    @DetailCaption,
    (SELECT TOP 1 UId
    FROM SysSchema
    WHERE name = @DetailSchemaName),
    (SELECT TOP 1 UId
    FROM SysSchema
    WHERE name = @EntitySchemaName)
)
```

b. Register the connection between the detail object schema and the detail record page schema. To do this, execute the SQL query to the `[SysModuleEntity]` and `[SysModuleEdit]` database tables.

SQL query

```
DECLARE
-- The name of the detail page schema.
@CardSchemaName NCHAR(100) = 'UsrDetailPageSchemaName',
-- The name of the detail object schema.
@EntitySchemaName NVARCHAR(100) = 'UsrDetailObjectSchemaName',
-- The name of the detail page.
@PageCaption NVARCHAR(100) = 'DetailPageName',
-- An empty string.
@Blank NCHAR(100) = ''

INSERT INTO SysModuleEntity(
    ProcessListeners,
    SysEntitySchemaUId
)
VALUES (
    0,
    (SELECT TOP 1 UId
    FROM SysSchema
    WHERE Name = @EntitySchemaName
    )
)

INSERT INTO SysModuleEdit(
    SysModuleEntityId,
    UseModuleDetails,
    Position,
```

```
    HelpContextId,
    ProcessListeners,
    CardSchemaUId,
    ActionKindCaption,
    ActionKindName,
    PageCaption
)
VALUES (
    (SELECT TOP 1 Id
    FROM SysModuleEntity
    WHERE SysEntitySchemaUId = (
        SELECT TOP 1 UId
        FROM SysSchema
        WHERE Name = @EntitySchemaName
        )
    ),
    1,
    0,
    @Blank,
    0,
    (SELECT TOP 1 UId
     FROM SysSchema
     WHERE name = @CardSchemaName
    ),
    @Blank,
    @Blank,
    @PageCaption
)
```

Register the detail to make it visible in the Section Wizard and Detail Wizard.

k. Restart Creatio in IIS to apply the changes.

2. **Add the custom detail to a record page**.

   Create a replacing view model schema of the record page to place the add page detail.

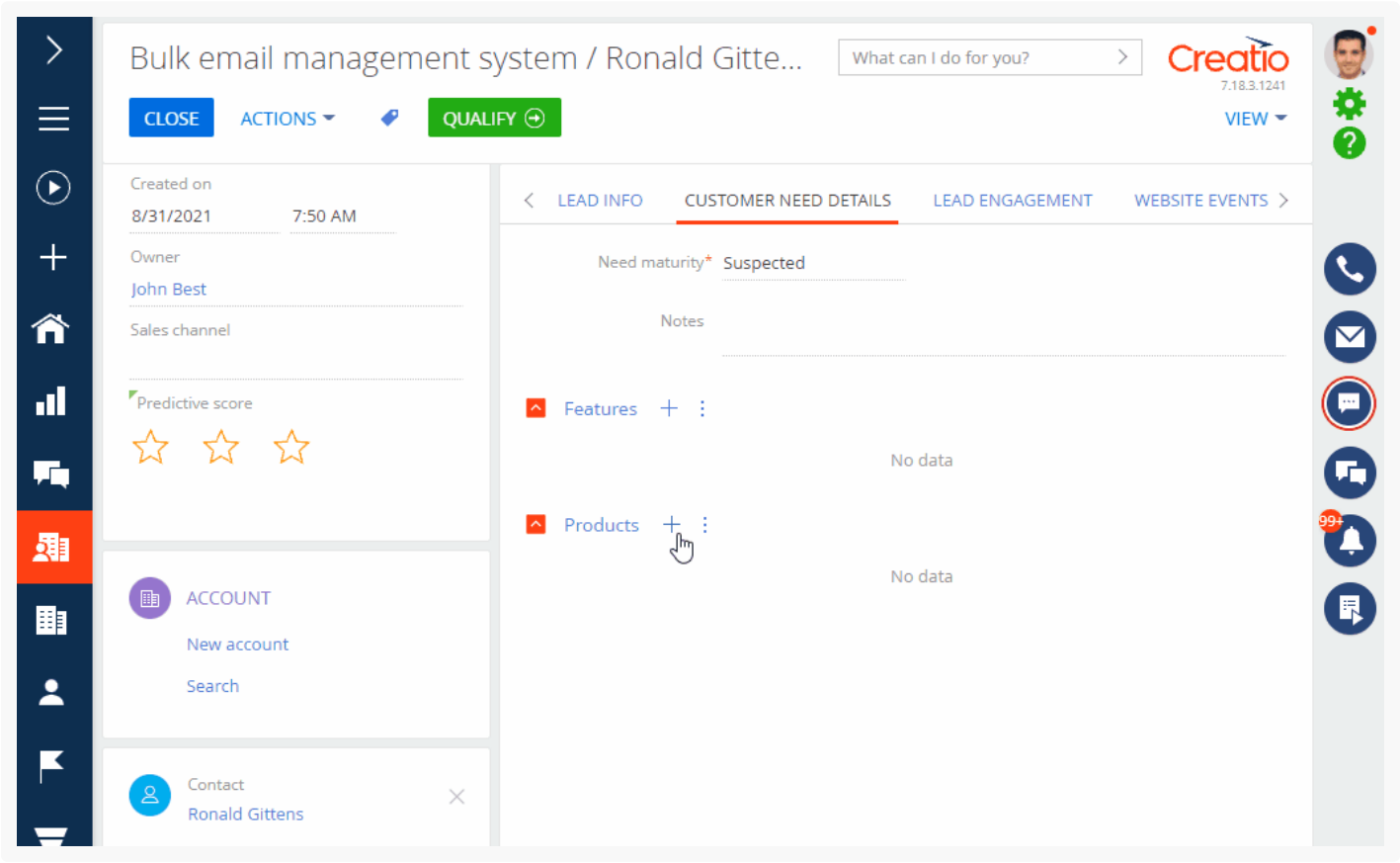
   - Select the view model schema to replace as the parent object.
   - Add the detail to the `details` property.
   - Add the configuration object of the detail view model to the `diff` array of modifications.

3. Take step 4 of the general procedure to implement a detail in Creatio IDE. If you skip this step, the detail will appear on the record page but will not contain records since the columns for display will not be specified.

Implement a lookup detail

A **lookup detail** lets you select data from a lookup displayed in a pop-up box. A lookup detail is a subtype of a list detail. The `BaseGridDetailV2` schema of the `NUI` package implements the functionality of a base list detail. For example, the [*Products*] detail on the lead page is a lookup detail. You can select the products in the [*Select:*

Products] pop-up box.



Implement the lookup detail in the Detail Wizard

1. **Create a custom detail**.

   - Add a [*Lookup*] type column to the detail.
   - Set up the lookup view. To do this, select "`Selection window`" in the [*Lookup view*] property.

2. Take step 2 of the general procedure to implement a detail in the Detail Wizard.

Implement the lookup detail in Creatio IDE

1. **Create a custom detail**.

   a. Create a detail object schema.

      - Select `BaseEntity` as the parent object.
      - Add the [*Lookup*] type column and other needed columns to the object schema.

   d. Implement the view model schema of the lookup detail.

      - Select `BaseGridDetailV2` as the parent object.
      - Select the [*Caption*] localizable string in the context menu of the [*Localizable strings*] node on the toolbar and specify the detail name in the [*Value*] property.

2. **Add the custom detail to a record page**.

   Create a replacing view model schema of the record page to place the lookup detail.

   - Select the view model schema to replace as the parent object.
   - Add the `ConfigurationEnums` module schema as a dependency of the record page's view model schema.
   - Add the following methods to the `methods` property:
     - `onDocumentInsert()`. Handles the record add event of the detail list.
     - `onCardSaved()`. Handles the save event of the detail record page.
     - `openDocumentLookup()`. Opens the lookup pop-up box.
     - Auxiliary data management methods.
   - Add the configuration object of the detail view model to the `diff` modification array.

   View the example of the `UsrCourierCertDetail` view model schema of the record page with the lookup detail for `UsrCourierCertInOrder` below.

Example of the replacing view model schema

```
/* Define the schema and set its dependencies on other modules. */
define("UsrCourierCertDetail", ["ConfigurationEnums"],
    function(configurationEnums) {
        return {
            /* The name of the detail object schema. */
            entitySchemaName: "UsrCourierCertInOrder",
            /* The methods of the detail schema. */
            methods: {
                /* Return the columns the query selects. */
                getGridDataColumns: function() {
                    return {
                        "Id": {path: "Id"},
                        "Document": {path: "UsrDocument"},
                        "Document.Number": {path: "UsrDocument.Number"}
                    };
                },

                /* Configure and display the lookup pop-up box. */
                openDocumentLookup: function() {
                    /* The configuration object. */
                    var config = {
                        /* The schema name for the object whose records to display on the loo
                        entitySchemaName: "Document",
                        /* The flag that enables multiple selection. */
                        multiSelect: true,
                        /* Columns to use in the lookup. For example, for sorting. */
                        columns: ["Number", "Date", "Type"]
                    };
```

```
            var OrderId = this.get("MasterRecordId");
            if (this.Ext.isEmpty(OrderId)) {
                return;
            }
            /* The EntitySchemaQuery class instance. */
            var esq = this.Ext.create("Terrasoft.EntitySchemaQuery", {
                /* Set the root schema. */
                rootSchemaName: this.entitySchemaName
            });
            /* Add the Id column. */
            esq.addColumn("Id");
            /* Add the Id column from the Document schema. */
            esq.addColumn("Document.Id", "DocumentId");
            /* Create filters and add them to the query collection. */
            esq.filters.add("filterOrder", this.Terrasoft.createColumnFilterWithParam
                this.Terrasoft.ComparisonType.EQUAL, "UsrOrder", OrderId));
            /* Retrieve the entire record collection and display it in the lookup pop
            esq.getEntityCollection(function(result) {
                var existsDocumentsCollection = [];
                if (result.success) {
                    result.collection.each(function(item) {
                        existsDocumentsCollection.push(item.get("DocumentId"));
                    });
                }
                /* Add the filter to the configuration object. */
                if (existsDocumentsCollection.length > 0) {
                    var existsFilter = this.Terrasoft.createColumnInFilterWithParamet
                        existsDocumentsCollection);
                    existsFilter.comparisonType = this.Terrasoft.ComparisonType.NOT_E
                    existsFilter.Name = "existsFilter";
                    config.filters = existsFilter;
                }
                /* Call the lookup pop-up box. */
                this.openLookup(config, this.addCallBack, this);
            }, this);
        },

        /* Record page's save event handler. */
        onCardSaved: function() {
            this.openDocumentLookup();
        },

        /* Open the document lookup if the order page has been saved earlier. */
        addRecord: function() {
            var masterCardState = this.sandbox.publish("GetCardState", null, [this.sa
            var isNewRecord = (masterCardState.state === configurationEnums.CardState
            masterCardState.state === configurationEnums.CardStateV2.COPY);
            if (isNewRecord === true) {
```

```
            var args = {
                isSilent: true,
                messageTags: [this.sandbox.id]
            };
            this.sandbox.publish("SaveRecord", args, [this.sandbox.id]);
            return;
        }
        this.openDocumentLookup();
    },

    /* Add the selected products. */
    addCallBack: function(args) {
        /* The instance of the BatchQuery batch query class. */
        var bq = this.Ext.create("Terrasoft.BatchQuery");
        var OrderId = this.get("MasterRecordId");
        /* The collection of documents selected in the lookup. */
        this.selectedRows = args.selectedRows.getItems();
        /* The collection to pass to the query. */
        this.selectedItems = [];
        /* Copy the needed data. */
        this.selectedRows.forEach(function(item) {
            item.OrderId = OrderId;
            item.DocumentId = item.value;
            bq.add(this.getDocumentInsertQuery(item));
            this.selectedItems.push(item.value);
        }, this);
        /* Execute the batch query if it is not empty. */
        if (bq.queries.length) {
            this.showBodyMask.call(this);
            bq.execute(this.onDocumentInsert, this);
        }
    },

    /* Return the query to add the current object. */
    getDocumentInsertQuery: function(item) {
        var insert = Ext.create("Terrasoft.InsertQuery", {
            rootSchemaName: this.entitySchemaName
        });
        insert.setParameterValue("UsrOrder", item.OrderId, this.Terrasoft.DataVal
        insert.setParameterValue("UsrDocument", item.DocumentId, this.Terrasoft.D
        return insert;
    },

    /* The method to call when adding records to the detail list. */
    onDocumentInsert: function(response) {
        this.hideBodyMask.call(this);
        this.beforeLoadGridData();
        var filterCollection = [];
        response.queryResults.forEach(function(item) {
```

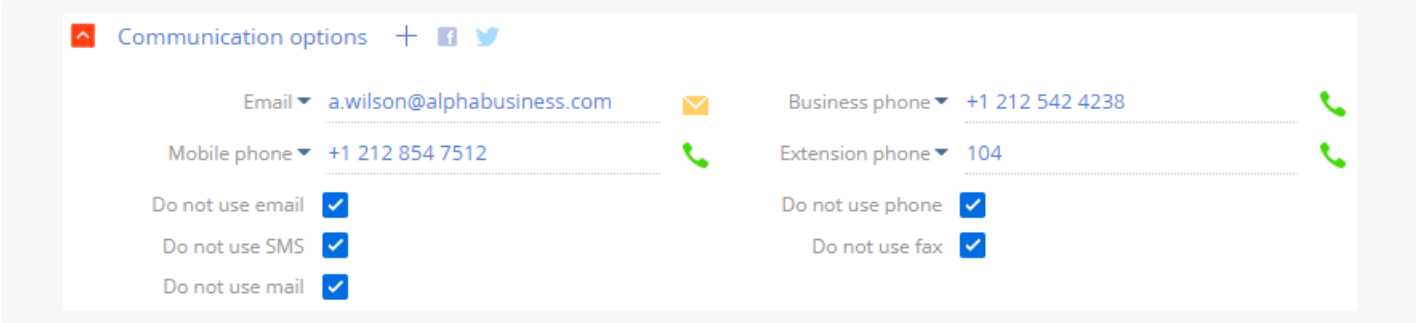
```
                    filterCollection.push(item.id);
                });
                var esq = Ext.create("Terrasoft.EntitySchemaQuery", {
                    rootSchemaName: this.entitySchemaName
                });
                this.initQueryColumns(esq);
                esq.filters.add("recordId", Terrasoft.createColumnInFilterWithParameters(
                /* Create a view model. */
                esq.on("createviewmodel", this.createViewModel, this);
                esq.getEntityCollection(function(response) {
                    this.afterLoadGridData();
                    if (response.success) {
                        var responseCollection = response.collection;
                        this.prepareResponseCollection(responseCollection);
                        this.getGridData().loadAll(responseCollection);
                    }
                }, this);
            },

            /* The method to call when deleting the selected detail records. */
            deleteRecords: function() {
                var selectedRows = this.getSelectedItems();
                if (selectedRows.length > 0) {
                    this.set("SelectedRows", selectedRows);
                    this.callParent(arguments);
                }
            },

            /* Hide the Copy menu item. */
            getCopyRecordMenuItem: Terrasoft.emptyFn,
            /* Hide the Edit menu item. */
            getEditRecordMenuItem: Terrasoft.emptyFn,
            /* Return the name of the default column for the filter. */
            getFilterDefaultColumnName: function() {
                return "UsrDocument";
            }
        },
        /* The array of modifications. */
        diff: /**SCHEMA_DIFF*/[
            {
                /* Set the operation type to merge. */
                "operation": "merge",
                /* The name of the schema element on which to execute the operation. */
                "name": "DataGrid",
                /* The object whose properties to merge with the schema element propertie
                "values": {
                    "rowDataItemMarkerColumnName": "UsrDocument"
                }
```

```
                },
                {
                    /* Set the operation type to merge. */
                    "operation": "merge",
                    /* The name of the schema element on which to execute the operation. */
                    "name": "AddRecordButton",
                    /* The object whose properties to merge with the schema element propertie
                    "values": {
                        "visible": {"bindTo": "getToolsVisible"}
                    }
                }
            ]/**SCHEMA_DIFF*/
        };
    }
);
```

3. Take step 4 of the general procedure to implement a detail in Creatio IDE. If you skip this step, the detail will appear on the record page without records since the columns to display will not be specified.

Implement a field detail

A **field detail** lets you enter and edit data directly in the detail fields. The detail can contain several field groups. For example, the [*Communication options*] detail of the contact page is a field detail.



A field detail lets you perform the following **actions**:

- Add records to the detail without saving the page that contains the current detail.
- Manage the detail as a record page.
- Use the base field validation.
- Implement custom field validation.
- Add a virtual record.
- Expand the record behavior logic.

You cannot implement a field detail using the Detail Wizard only since the Detail Wizard creates a list detail by default. To implement a detail, use both the Detail Wizard and Creatio IDE.

Implementation of a field detail for the Financial Services Creatio products is somewhat specific. The

`BaseFieldsDetail` schema of the `BaseFinance` package implements the functionality of a base field detail for the Financial Services Creatio products. The `BaseFieldRowViewModel` schema of the `BaseFinance` package implements the record view model of a field detail.

Implement a field detail in Detail Wizard and Creatio IDE

1. Download the `sdkFieldsDetailPackage` package to implement a field detail in Creatio CRM products.
2. Import the package into your Creatio application to implement a field detail in Creatio CRM products. To do this, follow the guide in a separate article: Transfer packages.
3. Add the `sdkFieldsDetailPackage` package as a dependency of the custom package to implement a field detail in Creatio CRM products. To do this, follow the guide in a separate article: Create a custom package.
4. Take step 1 of the general procedure to implement a detail in the Detail Wizard. If needed, set up the detail column in Creatio IDE.
5. Replace the parent object of the detail object with `BaseFieldsDetail` in Creatio IDE.
6. Take step 2 of the general procedure to implement a detail in the Detail Wizard.

Implement a field detail in Creatio IDE

1. Take steps 1-3 of the procedure to implement a field detail in the Detail Wizard and Creatio IDE to implement a field detail in Creatio CRM products.
2. **Create a custom detail**.

   a. Create a detail object schema.

   - Select `BaseEntity` as the parent object.
   - Add the needed columns to the object schema.

   d. Create a view model schema of the field detail.

   - Select `BaseFieldsDetail` as the parent object.
   - Select the [*Caption*] localizable string in the context menu of the [*Localizable strings*] node on the toolbar and specify the detail name in the [*Value*] property.
   - Add the `getDisplayColumns` method to the `methods` property. The method returns the array of column names displayed as fields in the detail.

   h. Add custom detail styles (optional).

   a. Create a module schema. Define the styles there. To do this, follow the guide in a separate article: Develop configuration elements.

   b. Specify the parent class.

   - Specify `UsrBaseFieldRowViewModel` to implement a field detail in Creatio CRM products.
   - Specify `BaseFieldRowViewModel` to implement a field detail in Financial Services Creatio product lineup.

   e. Add the schema of the module that contains the style implementation as a dependency of the detail list's view model schema.

f. Add the methods that override base CSS style classes to the `methods` property:

- `getRowViewModelClassName()`. Returns the class name of the detail record's view model.
- `getLeftRowContainerWrapClass()`. Returns the string array of CSS class names required to generate the view of containers that have record field signatures.

i. Register the detail in the database. To do this, execute the SQL query to the `[SysDetails]` database table.

SQL query

```
DECLARE
-- The name of the detail schema.
@ClientUnitSchemaName NVARCHAR(100) = 'UsrDetailSchemaName',
-- The name of the detail object schema.
@EntitySchemaName NVARCHAR(100) = 'UsrDetailObjectSchemaName',
-- The name of the detail.
@DetailCaption NVARCHAR(100) = 'DetailName'

INSERT INTO SysDetail(
    Caption,
    DetailSchemaUId,
    EntitySchemaUId
)
VALUES(
    @DetailCaption,
    (SELECT TOP 1 UId
    from SysSchema
    WHERE Name = @ClientUnitSchemaName),
    (SELECT TOP 1 UId
    from SysSchema
    WHERE Name = @EntitySchemaName)
)
```

Register the detail to make it visible in the Section Wizard and Detail Wizard.

3. **Add the custom detail to a record page**.

Create a replacing view model schema of the record page to place the field detail.

- Select the view model schema to replace as the parent object.
- Add the detail to the `details` property.
- Add the configuration object of the detail view model to the `diff` array of modifications.

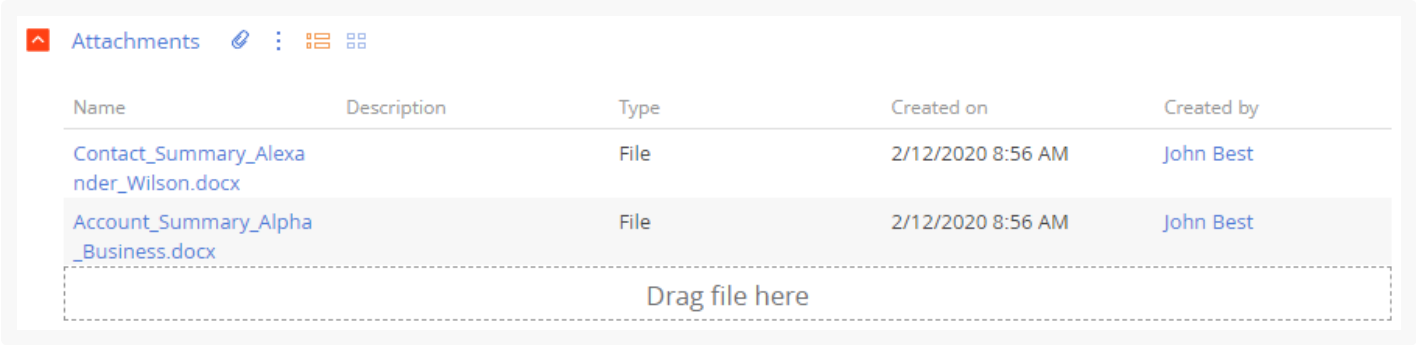
View an example of the `ContactPageV2` replacing view model schema of the record page that contains the `UsrRegDocumentFieldsDetail` detail that has fields below.

Example of the replacing view model schema

```
define("ContactPageV2", [], function() {
    return {
        entitySchemaName: "Contact",
        details: /**SCHEMA_DETAILS*/ {
            /* Add a field detail. */
            "UsrRegDocumentFieldsDetail": {
                /* The name of the detail's client schema name. */
                "schemaName": "UsrRegDocumentFieldsDetail",
                /* Filter records of the current contact detail. */
                "filter": {
                    /* The detail object column. */
                    "detailColumn": "UsrContact",
                    /* The contact ID column. */
                    "masterColumn": "Id"
                }
            }
        } /**SCHEMA_DETAILS*/ ,
        diff: /**SCHEMA_DIFF*/ [{
            /* Add a new element. */
            "operation": "insert",
            /* The element name. */
            "name": "UsrRegDocumentFieldsDetail",
            /* The configuration object of values. */
            "values": {
                /* The element type. */
                "itemType": Terrasoft.ViewItemType.DETAIL
            },
            /* The container element name. */
            "parentName": "HistoryTab",
            /* The property name of the container element for the nested element collection.
            "propertyName": "items",
            /* The index of the element to add to the collection. */
            "index": 0
        }] /**SCHEMA_DIFF*/
    };
});
```

Implement an [*Attachments*] type detail

An [*Attachments*] **type detail** lets you store external files, links to web resources, and knowledge base articles. Available for all Creatio sections. The `FileDetailV2` schema of the `Uiv2` package implements the functionality of a base [*Attachments*] type detail. Learn more about the [*Attachments*] type detail in user documentation: Work with attachments. For example, the [*Attachments*] detail of the contact page is an [*Attachments*] type detail.



The [*Attachments*] type detail cannot be implemented only in the Detail Wizard since the Detail Wizard creates a list detail by default. To implement a detail, use both the Detail Wizard and Creatio IDE.

To implement an [*Attachments*] type detail in the **Detail Wizard and Creatio IDE**:

1. **Create a custom detail**.

   a. Set up the object of the [*Attachments*] type detail.

      - Select "`Based on existing object`" in the [*How to create detail?*] property to do this.
      - Select "`[CustomSectionName] attachment`" in the [*Object*] property to do this.

   d. Replace the parent object of the detail record page with `FileDetailV2` in Creatio IDE.

   e. Add custom detail styles (optional).

      a. Create a module schema. Define the styles there. To do this, follow the guide in a separate article: Develop configuration elements.

      b. Add the module schema that contains the style implementation as a dependency of the detail's view model schema.

2. Take step 2 of the general procedure to implement a detail in the Detail Wizard.

Implement bulk addition of records to a detail

By default, a detail lets you add only one record. The **purpose** of the `LookupMultiAddMixin` is to expand the action that adds a record to the detail. Use the mixin to enable users to select multiple lookup records at once.

To **implement the bulk addition of records to a detail**:

1. Create a replacing view model schema of the detail. To do this, follow the guide in a separate article: Develop configuration elements.
2. Add the `LookupMultiAddMixin` mixin to the `mixins` property.
3. Make the following changes in the `methods` property:

   - Overload the following methods:

     - `init()`. Implements the logic Creatio executes when loading the module. Initialize the `LookupMultiAddMixin` mixin in the method. Learn more about the `init()` method in a separate article: Module types and their specificities.
     - `getAddRecordButtonVisible()`. Displays the add button.

- `onCardSaved()`. Saves the detail page. Use the `openLookupWithMultiSelect()` method to call the multiple selection dialog box in the overridden method.
- `addRecord()`. Adds a record to the detail. Use the `openLookupWithMultiSelect()` method in the overloaded method, similar to the `onCardSaved()` method. Set to `true` to check if the record is new.

- Implement the `getMultiSelectLookupConfig()` method connected to the `openLookupWithMultiSelect()` method. The `getMultiSelectLookupConfig()` window configures the dialog box and returns the configuration object of the box.

Delete a detail

Attention. You must have access to the Creatio configuration and database to delete a detail.

To **delete a detail**:

1. Unlock the files of the detail to delete in the SVN repository.
2. Delete the records from the database. To do this, execute the following SQL database query:

   SQL query

   ```
   DECLARE @Caption nvarchar(max);
   SET @Caption = 'UsrDetailSchemaName';
   DECLARE @UId UNIQUEIDENTIFIER;
   select @UId = EntitySchemaUId from SysDetail
   where Caption = @Caption
   delete from SysDetail where EntitySchemaUId = @UId
   ```

3. Go to the [*Configuration*] section and delete the view model schema of the detail and schema of the detail object.

Set up a detail with fields

Advanced

Attention. For Financial Services Creatio product lineup, use the `BaseFieldsDetail` schema of the `BaseFinance` package to set up details with fields. This package is available only in Financial Services Creatio product lineup.

To **use details with fields in CRM Creatio products**:

1. Download the `sdkFieldsDetailPackage` package: download package.
2. Import the package into Creatio. To do this, follow the guide in a separate article: Transfer packages.

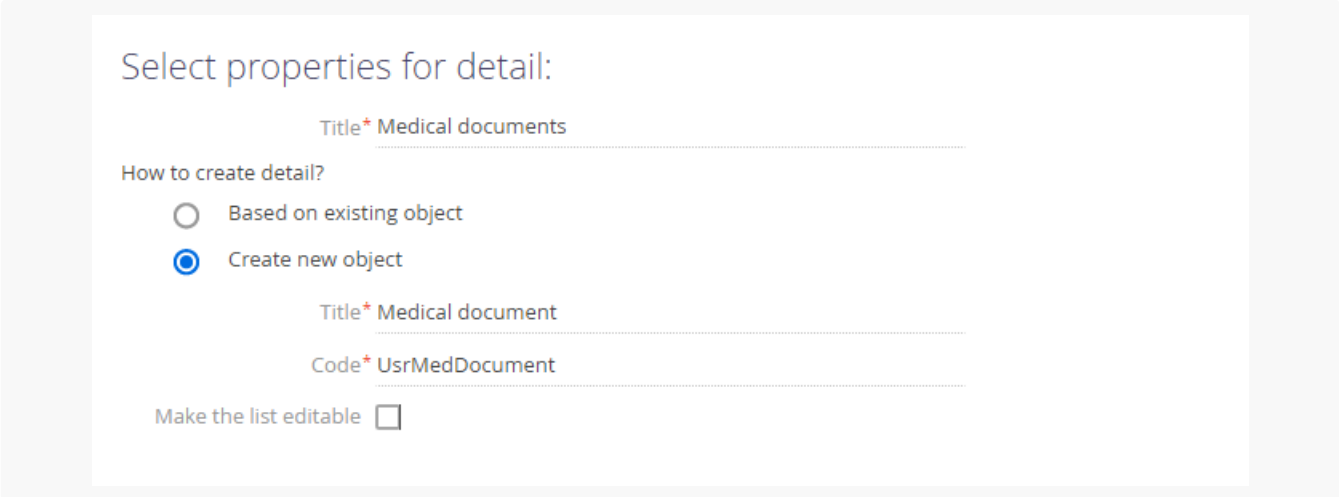
3. Add the `sdkFieldsDetailPackage` package to the dependencies of a custom package.

Example. Implement the [*Medical documents*] custom detail with the [*Number*] and [*Series*] virtual fields. Add the detail to the [*History*] tab of the contact page. The value of the [*Number*] field cannot be negative. Render the detail field names blue.

1. Create a custom detail

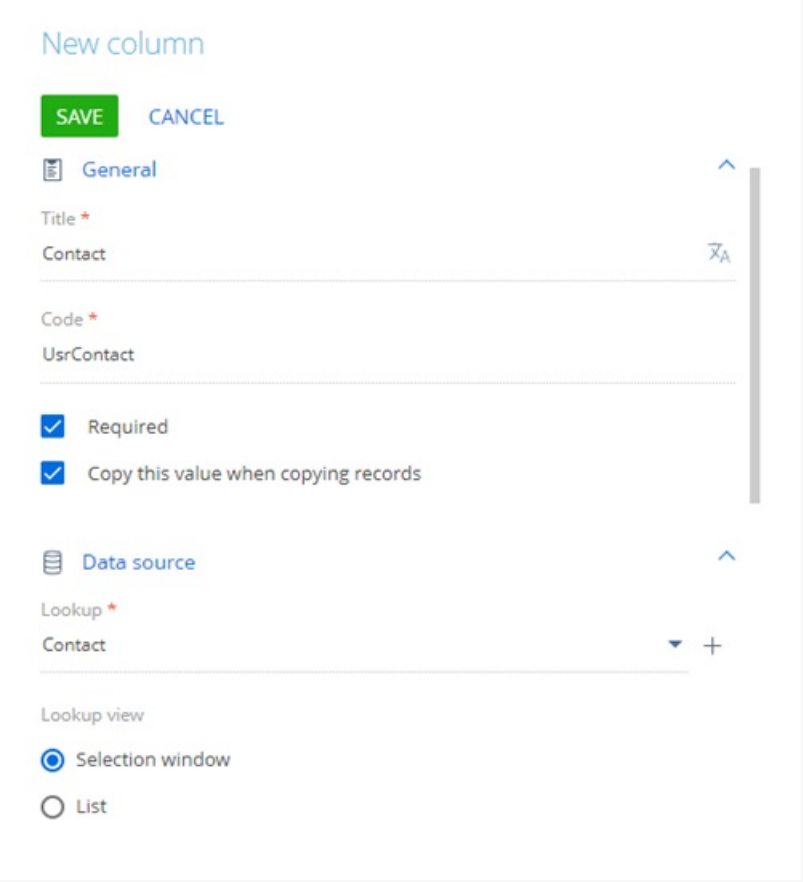
1. Create a custom package and set it as current. Learn more in a separate article: Packages basics.
2. Click to open the System Designer.
3. Go to the [*System setup*] block → [*Detail wizard*].
4. Fill out the detail properties:
   - Set [*Title*] to "`Medical documents`."
   - Set [*How to create detail?*] to "`Create new object`."

     Fill out the object properties:
     - Set [*Title*] to "`Medical document`."
     - Set [*Code*] to "`UsrMedDocument`."

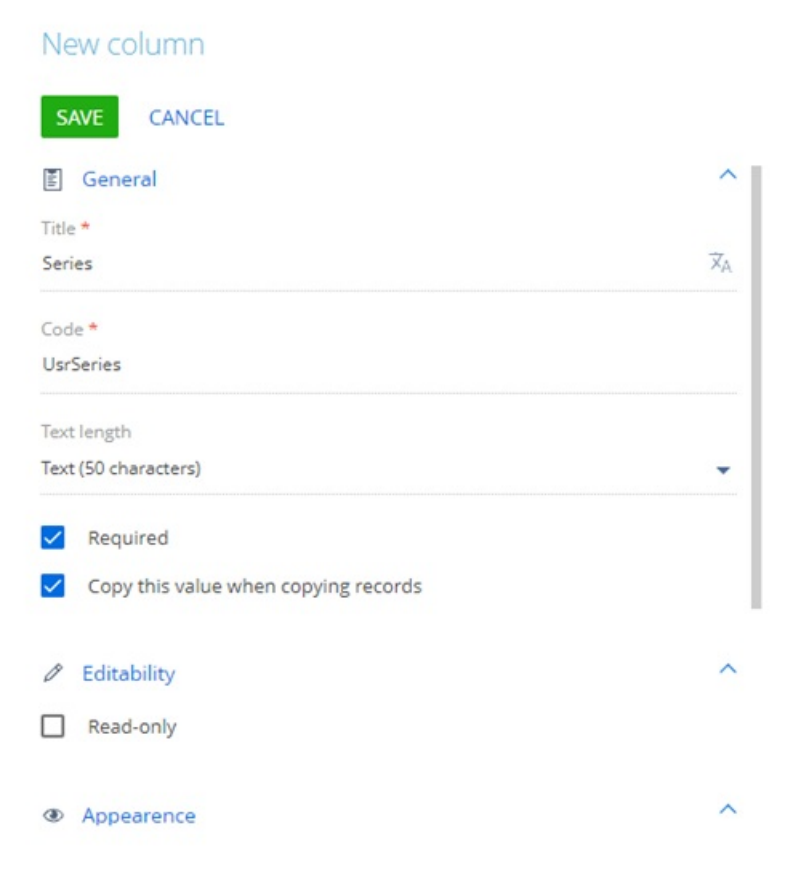


   As a result, Creatio will add the following schemas to the configuration:
   - The `UsrMedDocument` schema of the detail object.
   - The `UsrSchemac6fd3fd0Detail` schema of the [*Medical documents*] detail list's view model.
   - The `UsrUsrMedDocument4988cee4Page` schema of the [*Medical documents*] detail record page's view model.
5. Go to the [*Page*] tab and set up the detail record page.
6. Set up the detail fields.
   a. Fill out the [*Contact*] field of the [*Lookup*] type.

a. Set [*Title*] to "`Contact`."
b. Set [*Code*] to "`UsrContact`."
c. Select the [*Required*] checkbox.
d. Set [*Lookup*] to "`Contact`."



b. Fill out the [*Series*] field of the [*String*] type.
   a. Set [*Title*] to "`Series`."
   b. Set [*Code*] to "`UsrSeries`."
   c. Set [*Text length*] to "`Text (50 characters)`."
   d. Select the [*Required*] checkbox.

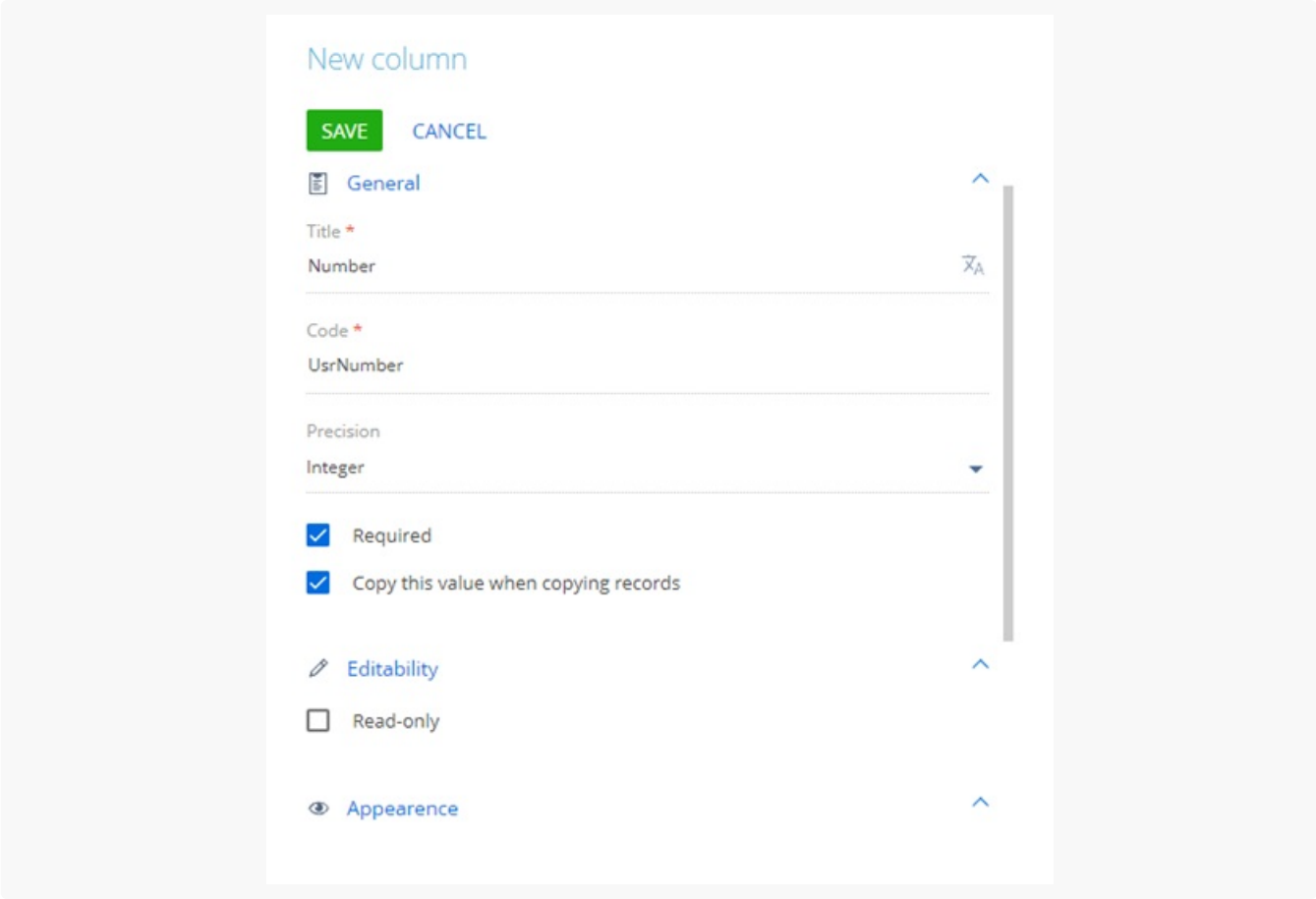


c. Fill out [*Number*] field of the [*Integer*] type.

   a. Set [*Title*] to "`Number`."

   b. Set [*Code*] to "`UsrNumber`."

   c. Select the [*Required*] checkbox.

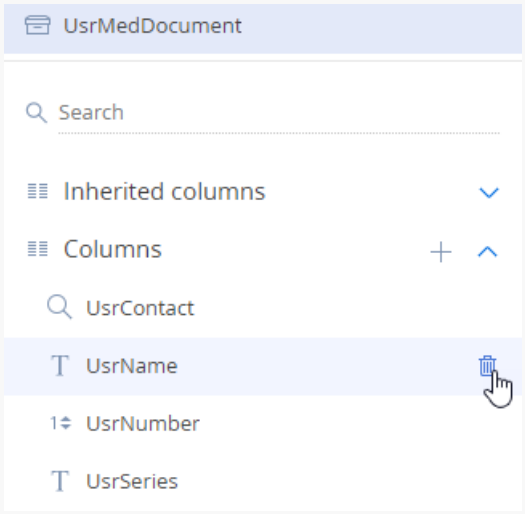


7. If necessary, modify the position of detail fields.
8. Click [*Save*] on the Detail Wizard's toolbar.

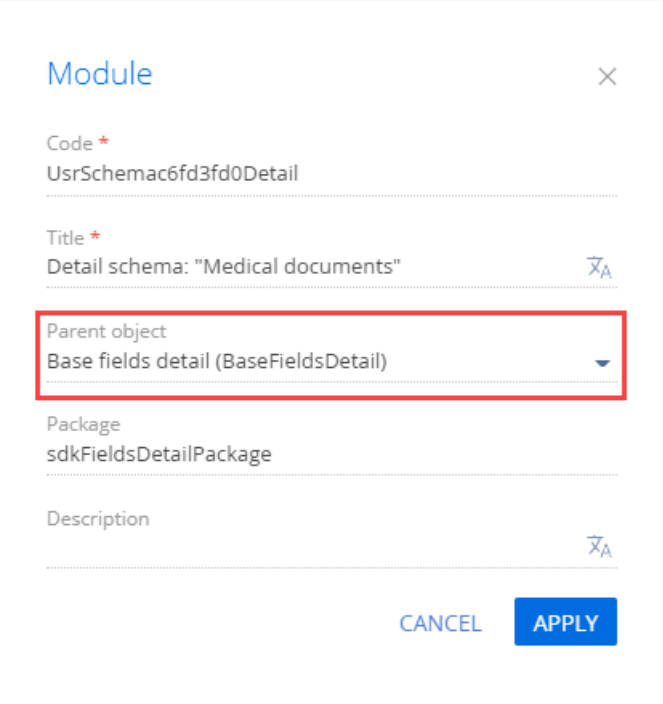
As a result, Creatio will modify the `UsrUsrMedDocument4988cee4Page` configuration schema of the [*Medical documents*] detail record page's view model.

2. Set up the custom detail

1. Go to the [*Configuration*] section and select a custom package to set as the current package.
2. Open the `UsrMedDocument` schema of the detail object.
3. Delete the `[UsrName]` required column in the context menu of the object structure's [*Columns*] node.



4. Click [*Publish*] on the Object Designer's toolbar.
5. Open the `UsrSchemac6fd3fd0Detail` schema of the [*Medical documents*] detail list's view model.
6. Click the button on the properties panel and specify `BaseFieldsDetail` in the [*Parent object*] field. The `BaseFieldsDetail` schema implements the detail that has fields. By default, the parent object in the Detail Wizard is the schema of a base detail that has a list.



7. Add the `getDisplayColumns` method to the `methods` property of the detail's view model schema. The method returns the array of column names displayed as fields in the detail.

   View the source code of the `UsrSchemac6fd3fd0Detail` schema below.

UsrSchemac6fd3fd0Detail

```
define("UsrSchemac6fd3fd0Detail", [], function() {
    return {
        entitySchemaName: "UsrMedDocument",
        details: /**SCHEMA_DETAILS*/{}/**SCHEMA_DETAILS*/,
        diff: /**SCHEMA_DIFF*/[]/**SCHEMA_DIFF*/,
        methods: {
            getDisplayColumns: function() {
                return ["UsrSeries", "UsrNumber"];
            }
        }
    };
});
```

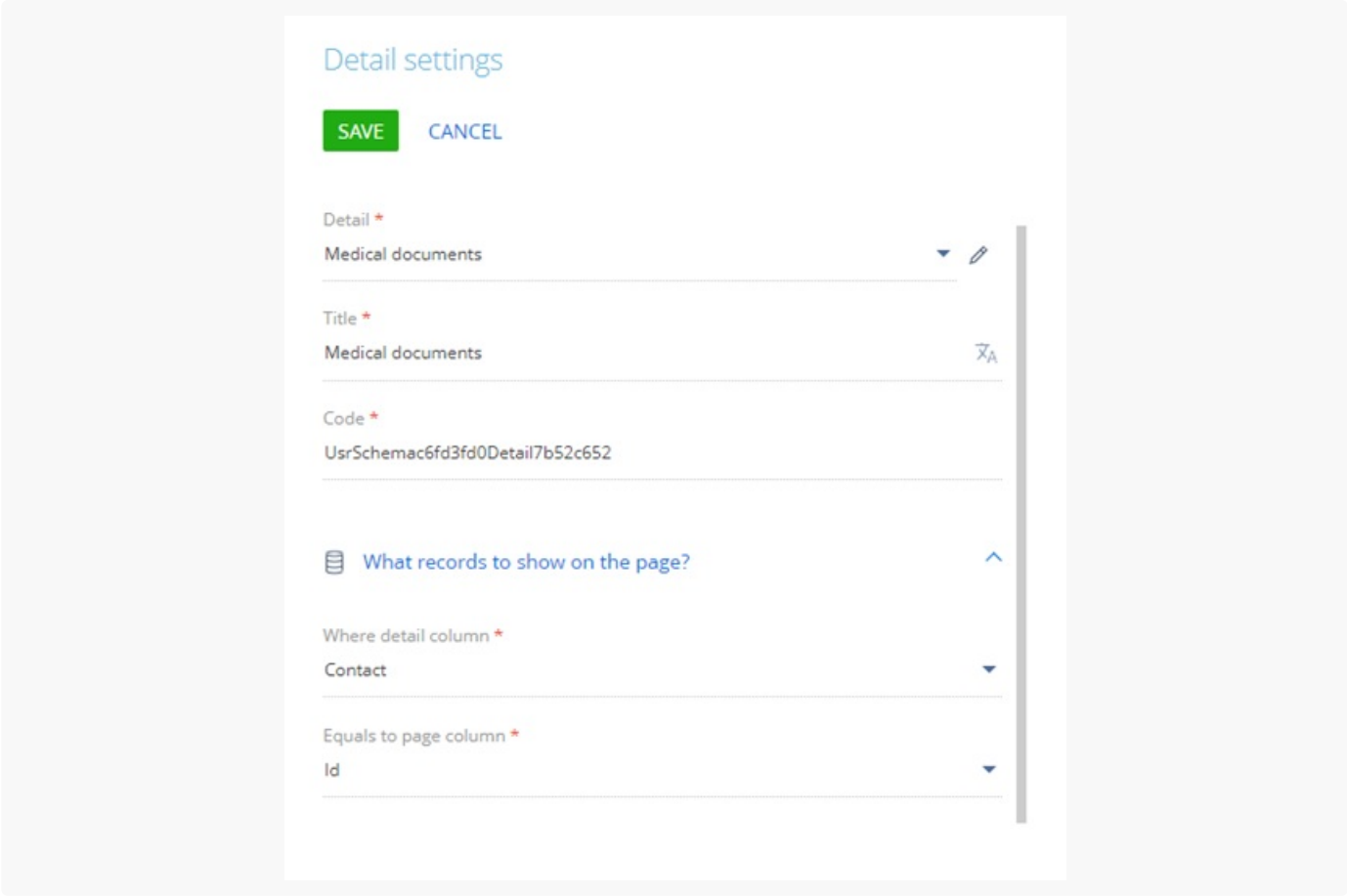
8. Click [*Save*] on the Module Designer's toolbar.

3. Add the detail to the section record page

1. Go to the [*Contacts*] section and open the contact page.
2. Click [*View*] → [*Open section wizard*] on the toolbar.
3. Go to the [*History*] tab of the Section Wizard workspace and click the [*New detail*] button.
4. Fill out the detail parameters.

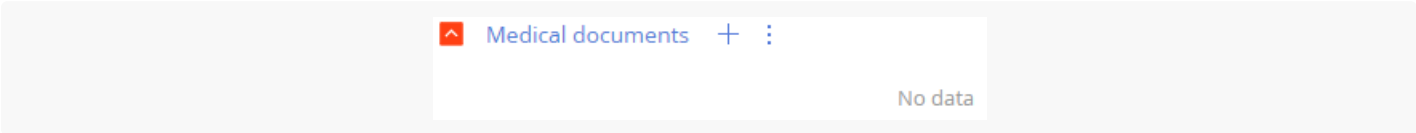
   - Set [*Detail*] to "`Medical documents`." Creatio will populate the [*Title*] and [*Code*] fields.
   - Set [*Where detail column*] to "`Contact`."

     Leave other columns as they are.



5. Click [*Save*] → [*Section wizard*] → [*Save*].

As a result, Creatio will add the [*Medical documents*] detail to the [*History*] tab of the contact page.



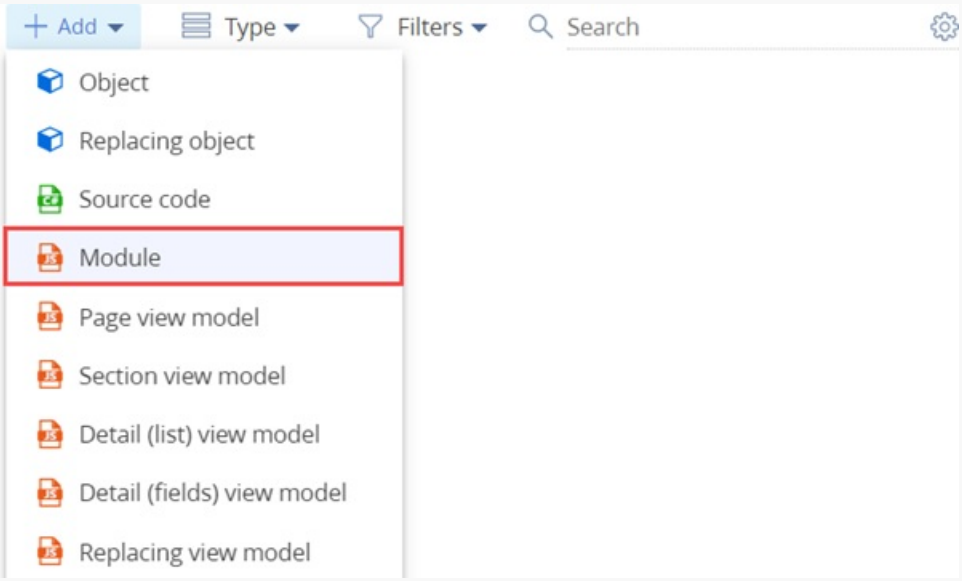
4. Add custom detail styles

The view model schema of a detail page does not support visual styles. As such, take the following steps to add the styles:

1. Create a module schema. Define the styles there.
2. Add the style module to the dependencies of the detail module.

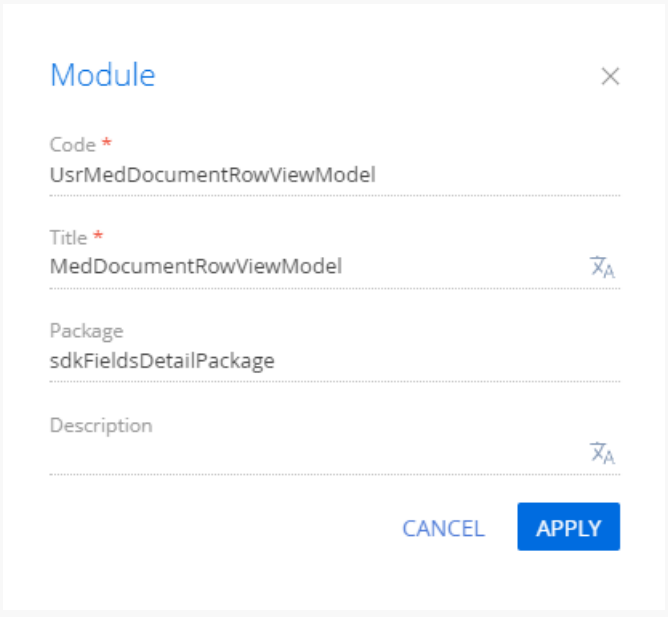
1. Create a module schema

1. Go to the [*Configuration*] section and select a custom package to set as current.
2. Click [*Add*] → [*Module*] on the section list toolbar.



3. Fill out the following schema properties:

   - Set [*Code*] to "`UsrMedDocumentRowViewModel`."
   - Set [*Title*] to "`MedDocumentRowViewModel`."



   Click [*Apply*] to apply the properties.

4. Add the source code in the Schema Designer. Create a module description in the source code of the schema. Define the `Terrasoft.configuration.UsrMedDocumentRowViewModel` class inherited from the `Terrasoft.BaseFieldRowViewModel` class in the module description.

```
UsrMedDocumentRowViewModel
```

```
define("UsrMedDocumentRowViewModel", ["BaseFieldRowViewModel"], function() {
```

```
    Ext.define("Terrasoft.configuration.UsrMedDocumentRowViewModel", {
        extend: "Terrasoft.BaseFieldRowViewModel",
        alternateClassName: "Terrasoft.UsrMedDocumentRowViewModel"
    });
    return Terrasoft.UsrMedDocumentRowViewModel;
});
```

5. Go to the [*LESS*] node of the object structure and set up the needed visual styles of the detail.

Set up the visual styles of the detail

```
.med-document-left-row-container {
    .t-label {
        color: blue;
    }
}
.field-detail-row {
    width: 100;
    display: inline-flex;
    margin-bottom: 10px;

    .field-detail-row-left {
        display: flex;
        flex-wrap: wrap;

        .control-width-15 {
            min-width: 300px;
            width: 50;
            margin-bottom: 5px;
        }
    }
    .field-detail-row-left.singlecolumn {
        width: 50%;
    }
}
```

6. Click [*Save*] on the Designer's toolbar.

2. Modify the view model schema of the detail

To **use the module and its styles** in the detail schema:

1. Open the `UsrSchemac6fd3fd0Detail` schema of the [*Medical documents*] detail list's view model.
2. Add the `UsrMedDocumentRowViewModel` module to the dependencies of the `UsrSchemac6fd3fd0Detail` schema.
3. Add the following methods that override the base CSS style classes to the definition of the detail schema module:

- The `getRowViewModelClassName()` method. Returns the class name for the view model of the record on the detail.
- The `getLeftRowContainerWrapClass()` method. Returns the string array of CSS class names. The names are required to generate the view of containers that include the record field signatures.

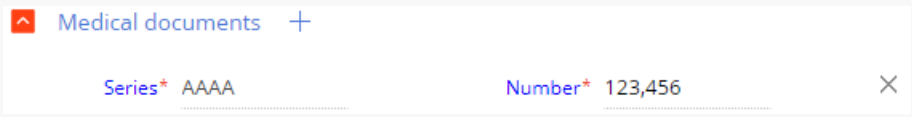
View the source code of the modified schema below.

UsrSchemac6fd3fd0Detail

```
define("UsrSchemac6fd3fd0Detail", ["UsrMedDocumentRowViewModel", "css!UsrMedDocumentRowViewMo
    return {
        entitySchemaName: "UsrMedDocument",
        details: /**SCHEMA_DETAILS*/{}/**SCHEMA_DETAILS*/,
        diff: /**SCHEMA_DIFF*/ [], /**SCHEMA_DIFF*/
        methods: {
            getDisplayColumns: function() {
                return ["UsrSeries", "UsrNumber"];
            },
            getRowViewModelClassName: function() {
                return "Terrasoft.UsrMedDocumentRowViewModel";
            },
            getLeftRowContainerWrapClass: function() {
                return ["med-document-left-row-container", "field-detail-row"];
            }
        }
    };
});
```

4. Click [*Save*] on the Designer's toolbar.

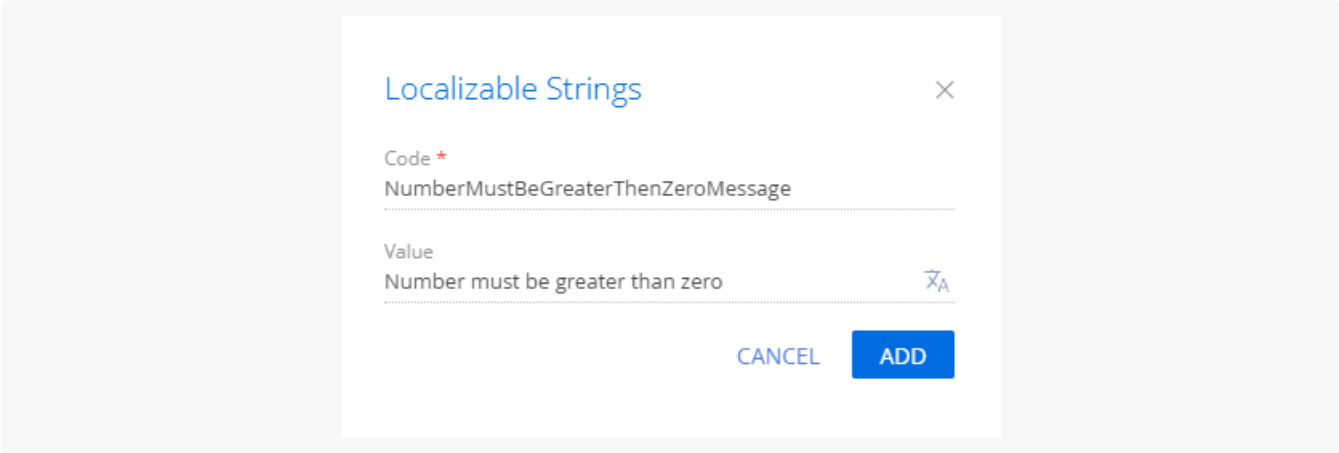
As a result, Creatio will render the field names of the [*Medical documents*] detail added to the [*History*] tab of the contact page blue.



5. Add validation to the detail field

1. Open the `UsrMedDocumentRowViewModel` schema of the module.
2. Add a localizable string with the warning about an incorrect [*Number*] field value.
   a. Click the + button in the context menu of the [*Localizable strings*] node.
   b. Fill out the localizable string properties:
      - Set [*Code*] to "`NumberMustBeGreaterThenZeroMessage`."

- Set [*Value*] to "Number must be greater than zero."



e. Click [*Add*] to add a localizable string.

f. Add the `UsrMedDocumentRowViewModelResources` resource module to the dependencies of the `UsrMedDocumentRowViewModel` module. This is required for Creatio to display the localizable string value in the front-end.

3. Add the validation logic of the [*Number*] field value. To do this, implement the following methods:

- the `validateNumberMoreThenZero()` method that contains the validation logic of the field value
- the `setValidationConfig()` method that binds the `[Number]` column to the `validateNumberMoreThenZero()` validator method.
- the `init()` overridden base method that calls the base logic and the `setValidationConfig()` method

View the source code of the modified schema below.

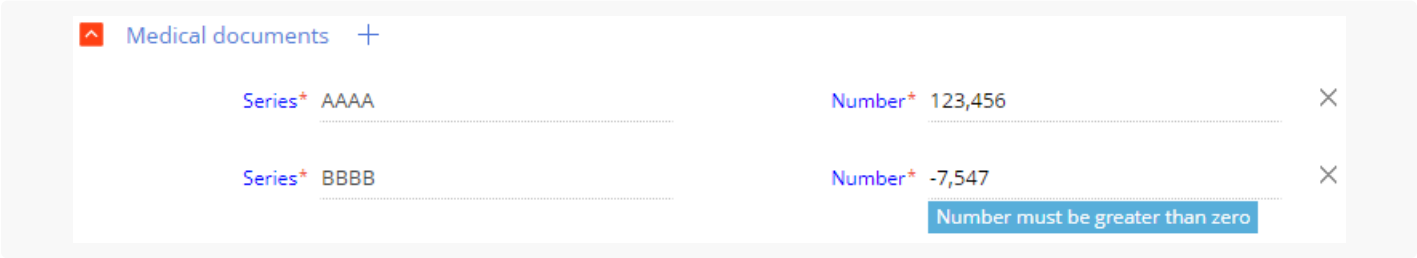
`UsrMedDocumentFieldsDetail`

```
define("UsrMedDocumentRowViewModel", ["BaseFieldRowViewModel", "UsrMedDocumentRowViewModelRes
function(resources) {
    Ext.define("Terrasoft.configuration.UsrMedDocumentRowViewModel", {
        extend: "Terrasoft.BaseFieldRowViewModel",
        alternateClassName: "Terrasoft.UsrMedDocumentRowViewModel",
        validateNumberMoreThenZero: function(columnValue) {
            var invalidMessage ="";
            if (columnValue < 0) {
                invalidMessage = resources.localizableStrings.NumberMustBeGreaterThenZeroMess
            }
            return {
                fullInvalidMessage: invalidMessage,
                invalidMessage: invalidMessage
            };
        },
        setValidationConfig: function() {
            this.addColumnValidator("UsrNumber", this.validateNumberMoreThenZero);
```

```
        },
        init: function() {
            this.callParent(arguments);
            this.setValidationConfig();
        }
    });
    return Terrasoft.UsrMedDocumentRowViewModel;
});
```

4. Click [*Save*] on the Designer's toolbar.

As a result, if you enter a negative value in the [*Number*] field, Creatio will display the corresponding warning.



6. Make the detail fields virtual

1. Open the `UsrSchemac6fd3fd0Detail` schema of the [*Medical documents*] detail list's view model.
2. Add the `useVirtualRecord()` method implementation.

   View the source code of the modified schema below.

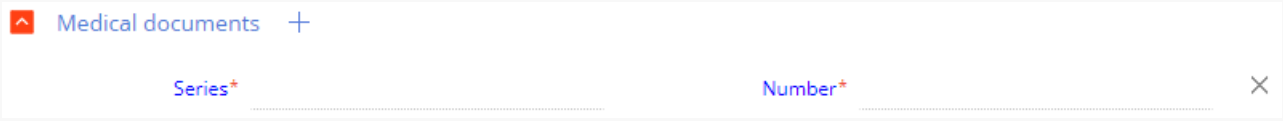
UsrSchemac6fd3fd0Detail

```
define("UsrSchemac6fd3fd0Detail", ["UsrMedDocumentRowViewModel", "css!UsrMedDocumentRowViewMo
    return {
        entitySchemaName: "UsrMedDocument",
        details: /**SCHEMA_DETAILS*/{}/**SCHEMA_DETAILS*/,
        diff: /**SCHEMA_DIFF*/ [], /**SCHEMA_DIFF*/
        methods: {
            getDisplayColumns: function() {
                return ["UsrSeries", "UsrNumber"];
            },
            getRowViewModelClassName: function() {
                return "Terrasoft.UsrMedDocumentRowViewModel";
            },
            getLeftRowContainerWrapClass: function() {
                return ["med-document-left-row-container", "field-detail-row"];
            },
            useVirtualRecord: function() {
                return true;
```

```
            }
        }
    };
});
```

3. Click [*Save*] on the Designer's toolbar.

As a result, if you open the [*History*] tab with the [*Medical documents*] detail, Creatio will display a virtual record.



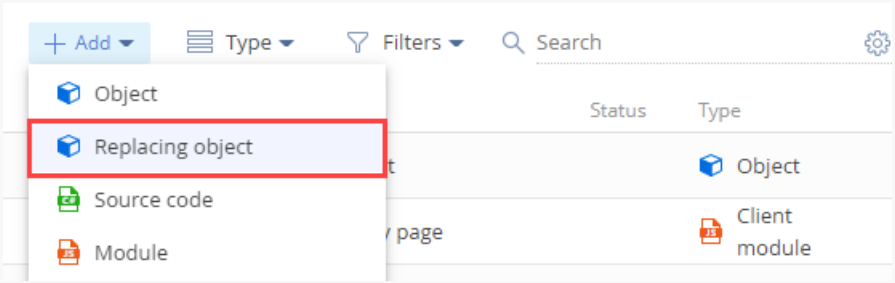
Add an editable list to the detail

Medium

Example. Add an editable list to the [*Discount, %*] column on the add product page (the [*Products*] detail) of the [*Orders*] section. The column already exists in the product object schema. Also, add an editable list to the [*Custom price*] custom column.

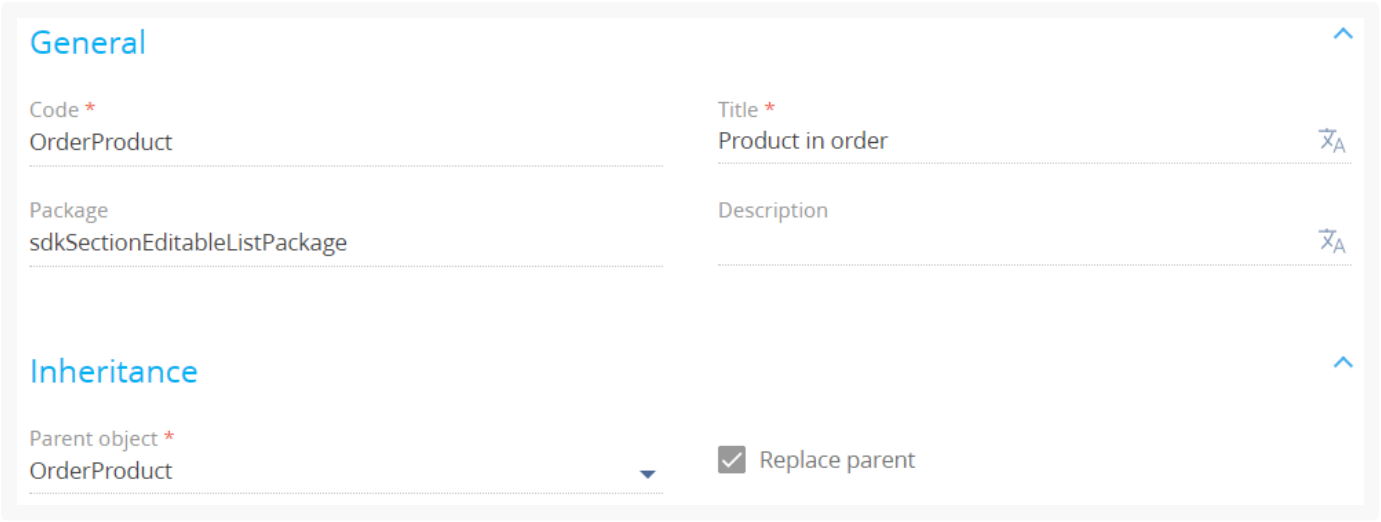
1. Create a replacement object schema

1. Go to the [*Configuration*] section and select a custom package to add the schema.
2. Click [*Add*] → [*Replacing object*] on the section list toolbar.



3. Fill out the **schema properties**.

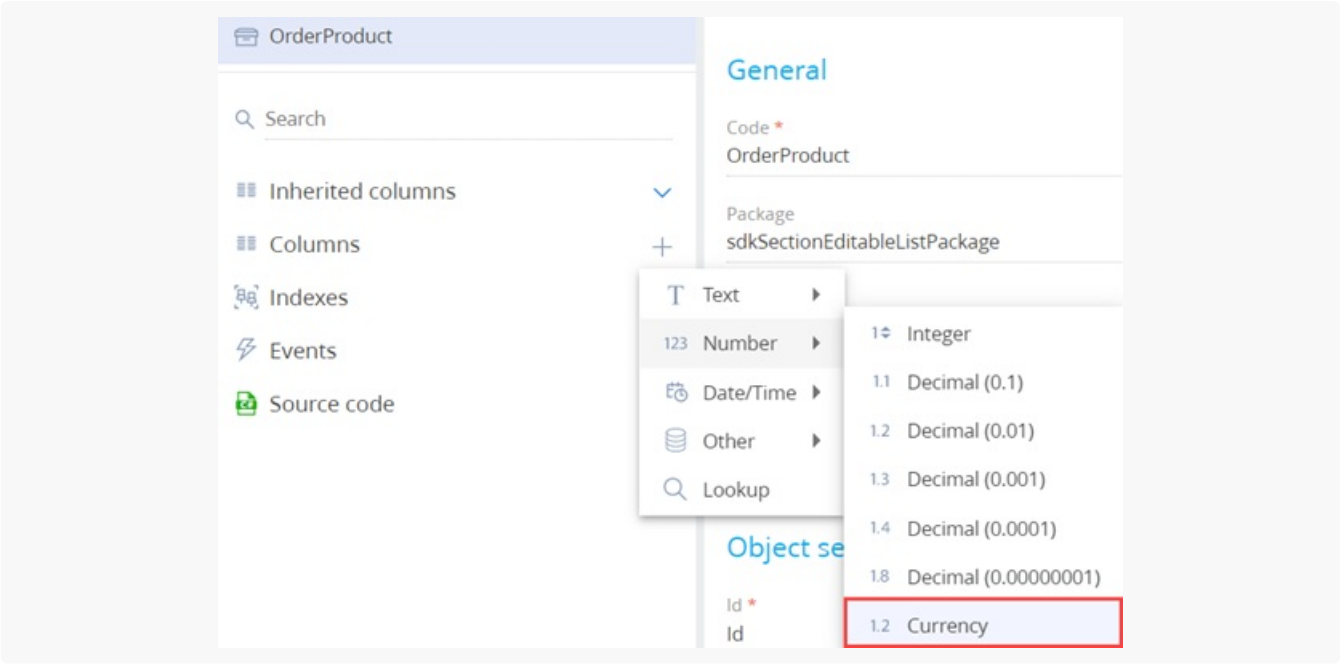
- Set [*Code*] to "`OrderProduct`."
- Set [*Title*] to "`Product in order`."
- Set [*Parent object*] to "`OrderProduct`."



4. Add a **column** to the schema.

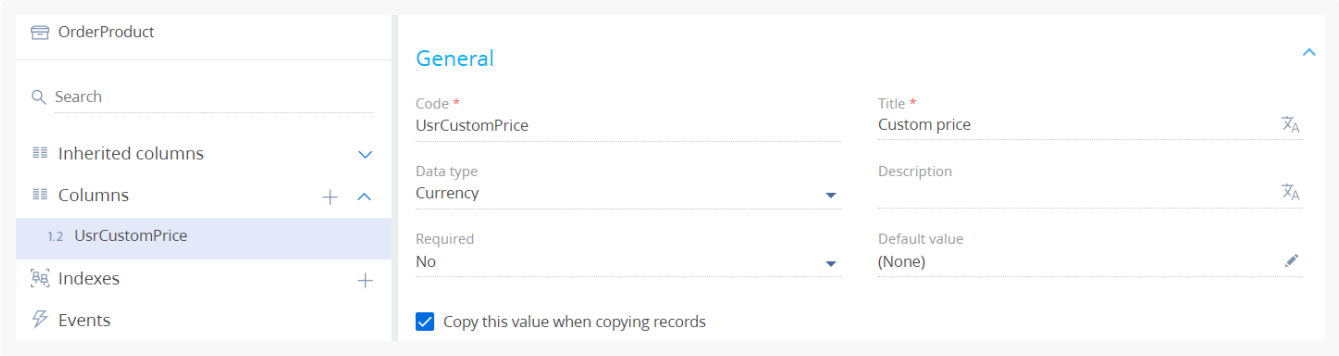
   a. Click + in the context menu of the object structure's [*Columns*] node.

   b. Click [*Number*] → [*Currency*] in the drop-down list.



   c. Fill out the **properties of the added column**.

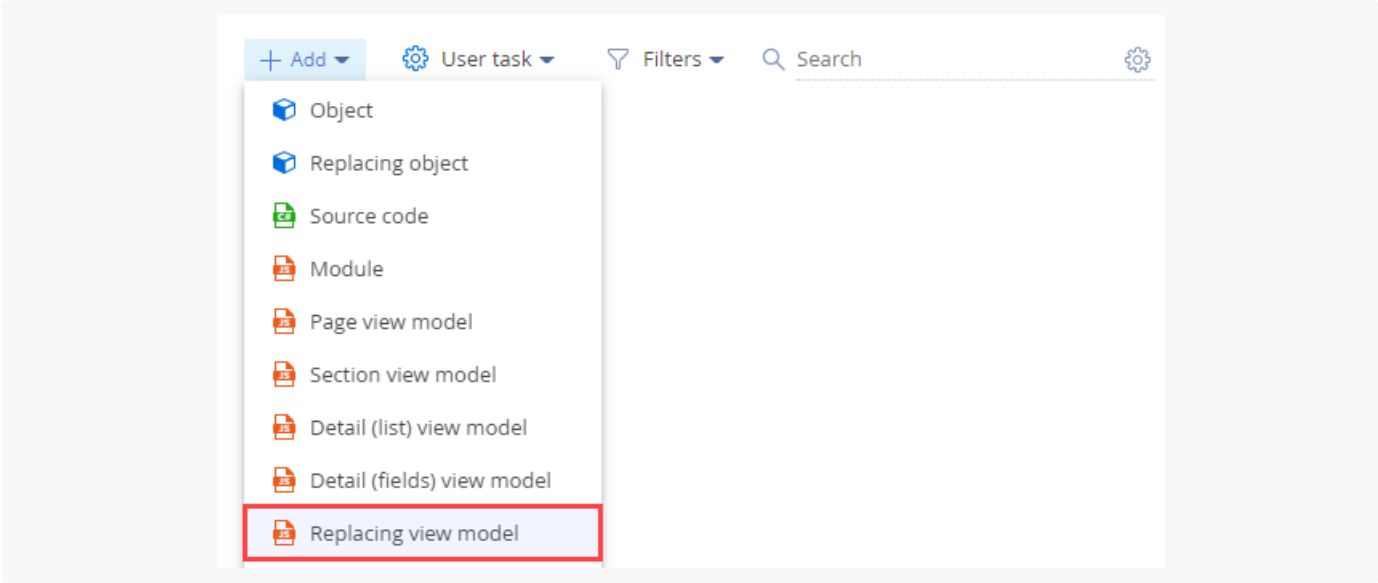
   - Set [*Code*] to "`UsrCustomPrice`."
   - Set [*Title*] to "`Custom price`."



5. Click [*Save*] then [*Publish*] on the Object Designer's toolbar.

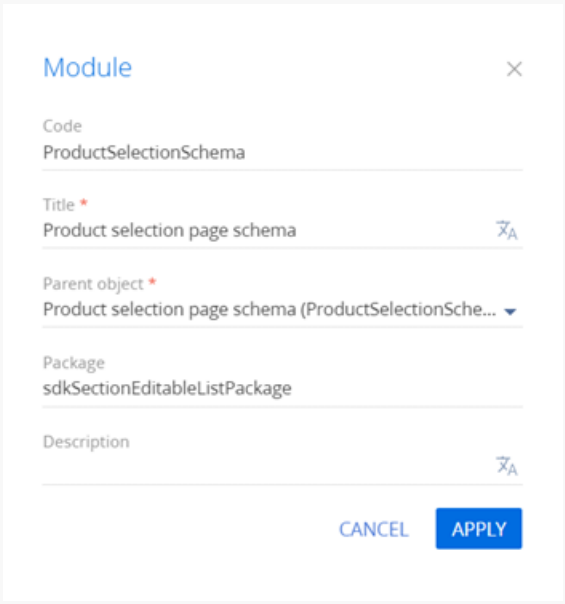
2. Create a schema of the replacing section view model

1. Go to the [*Configuration*] section and select a custom package to add the schema.
2. Click [*Add*] → [*Replacing view model*] on the section list toolbar.



3. Fill out the **schema properties**.

   - Set [*Code*] to "`ProductSelectionSchema`."
   - Set [*Title*] to "`Product selection page schema`."
   - Set [*Parent object*] to "`ProductSelectionSchema`."



4. Implement an **editable list**. To do this, implement the following **methods** in the `methods` property:

   - `getEditableColumns()`. Retrieves the array of columns to edit and adds a custom column to the array.
   - `setColumnHandlers()`. Binds the custom column's update event handler.
   - `onCustomPriceChanged()`. Handler method called upon changes to the field value.

   View the source code of the replacing view model schema of the section below.

ProductSelectionSchema

```
define("ProductSelectionSchema", [], function() {
    return {
        methods: {
            getEditableColumns: function() {
                /* Retrieve the array of columns to edit. */
                var columns = this.callParent(arguments);
                /* Add the [Discount, %] column to the array of columns to edit. */
                columns.push("DiscountPercent");
                /* Add the custom column. */
                columns.push("UsrCustomPrice");
                return columns;
            },
            setColumnHandlers: function(item) {
                this.callParent(arguments);
                /* Bind the custom column's update event handler. */
                item.on("change:UsrCustomPrice", this.onCustomPriceChanged, this);
            },
            /* Handler method called upon changes to the field value. */
            onCustomPriceChanged: function(item, value) {
                window.console.log("Changed: ", item, value);
            }
```

```
            }
        };
    });
```

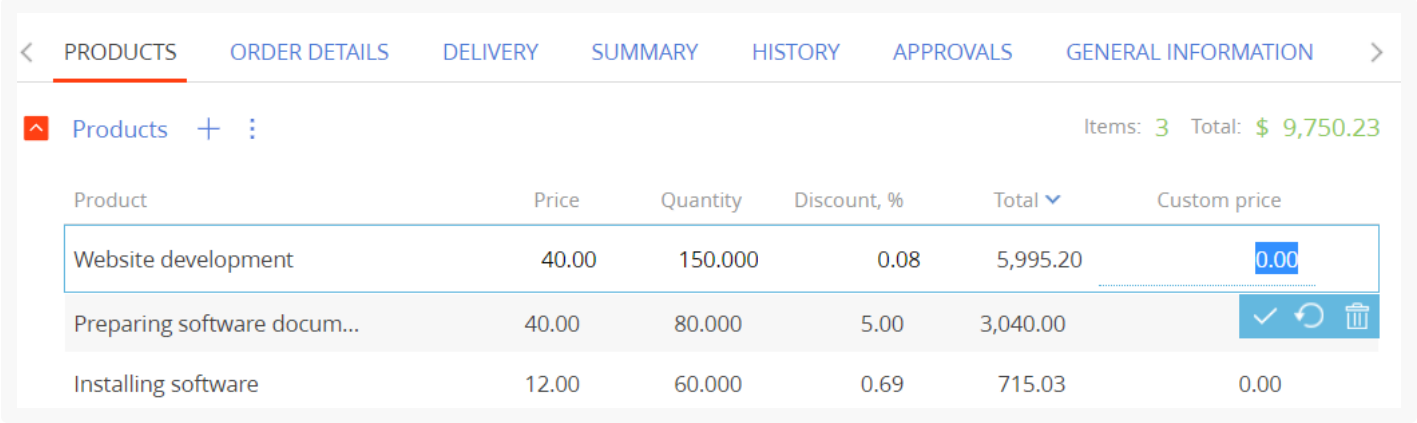
5. Click [*Save*] on the Designer's toolbar.

Outcome of the example

To **view the outcome of the example**:

1. Refresh the [*Orders*] section page.
2. Set up the **columns of the add product page**.
   a. Click [*View*] → [*Select fields to display*] on the section toolbar and open the setup mode of the section list's tile view ([*Tile view*]) on the column setup page.
   b. Add the column to the section list. To do this, click . Then, click + and select the [*Product in order*] object in the [*Select object*] field.
   c. Select the [*Discount, %*] column in the [*Column*] field.
   d. Add the [*Custom price*] column similarly.

As a result of the example, the list on the add product page (the [*Products*] detail) of the [*Orders*] section now includes the [*Discount, %*] and [*Custom price*] editable columns.



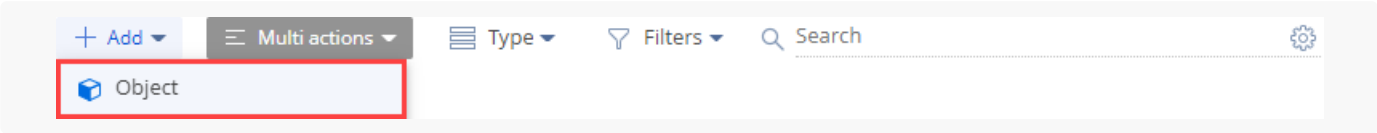
Create an editable list detail using Creatio IDE

Medium

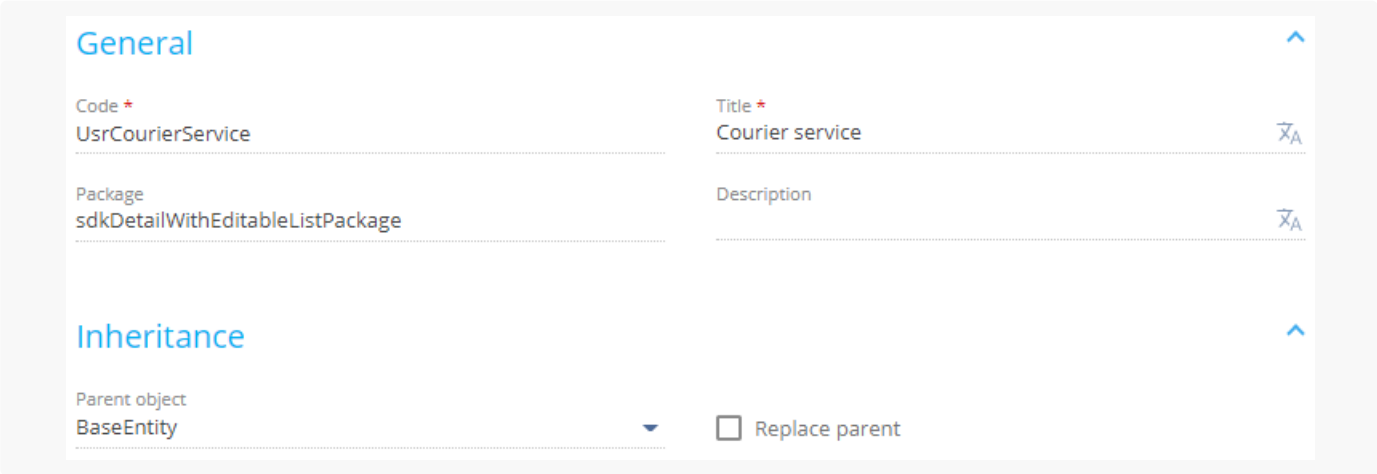
Example. Add a custom [*Courier service*] detail to the [*Delivery*] tab of the order page. Display the index of courier services (i. e., the values of the [*Account*] lookup) available for the current order on the detail.

1. Create a detail object schema

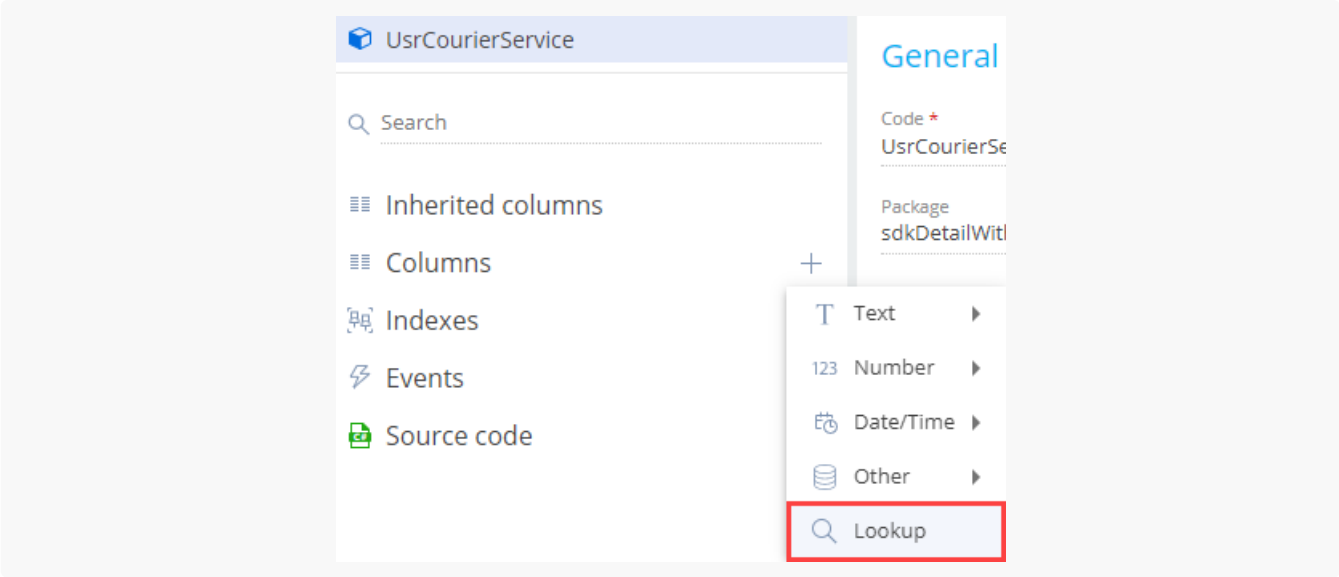
1. Go to the [*Configuration*] section and select a custom package to add the schema.
2. Click [*Add*] → [*Object*] in the section list toolbar.



3. Fill out the **schema properties**.
   - Set [*Code*] to "`UsrCourierService`."
   - Set [*Title*] to "`Courier service`."
   - Select "`BaseEntity`" in the [*Parent object*] property.

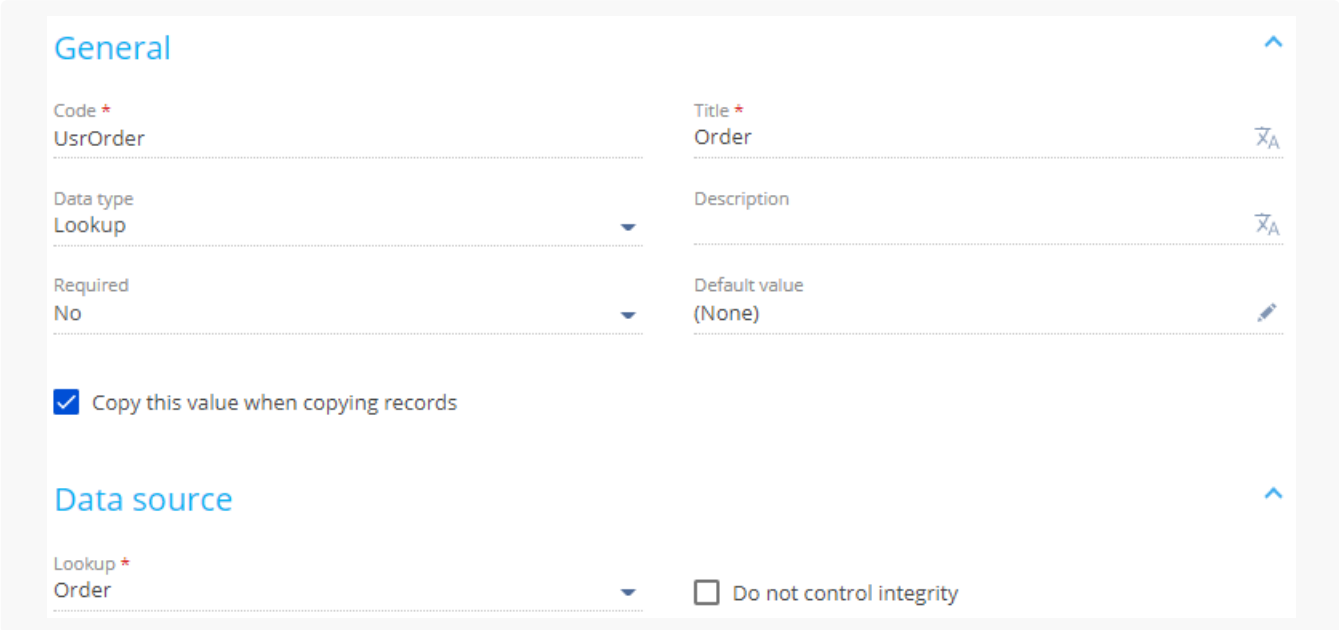


4. Add a **column that stores the order** to the schema.
   a. Click + in the context menu of the object structure's [*Columns*] node.
   b. Click [*Lookup*] in the drop-down menu.



c. Fill out the **properties of the added column**.

- Set [*Code*] to "`UsrOrder`."
- Set [*Title*] to "`Order`."
- Select "`Order`" in the [*Lookup*] property.



5. Add a **column that stores the account** to the schema in a similar way.

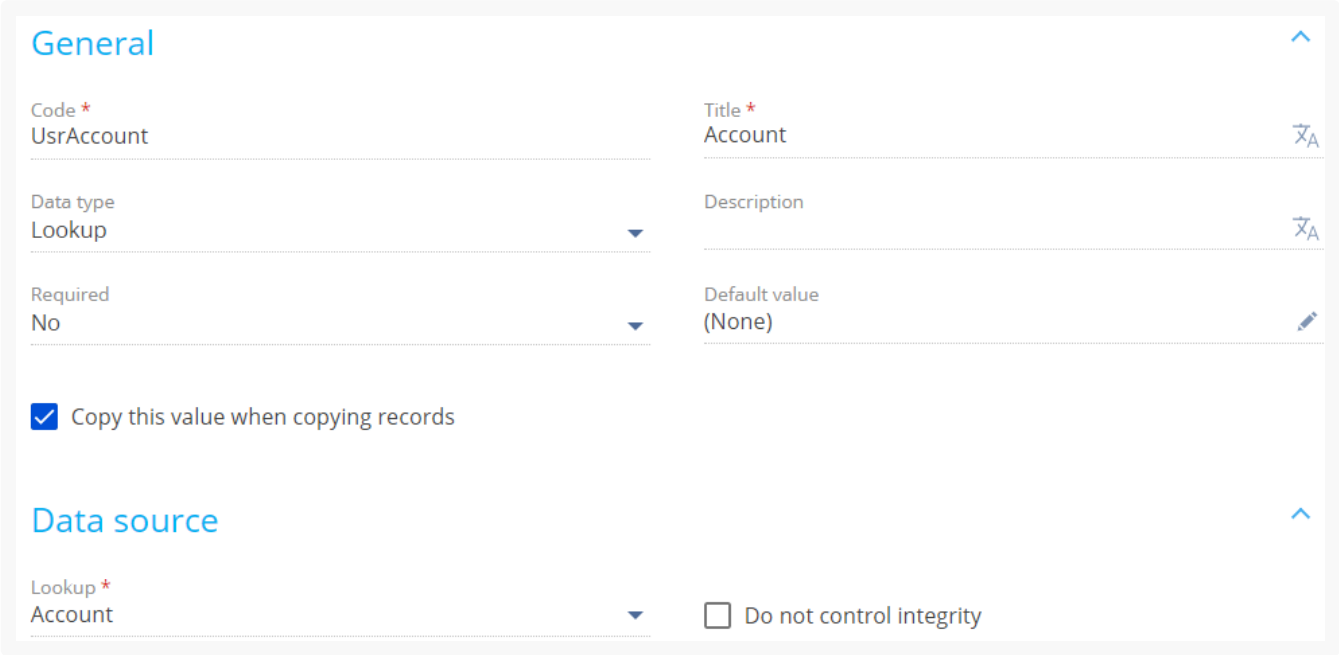
a. Click + in the context menu of the object structure's [*Columns*] node.

b. Click [*Lookup*] in the drop-down menu.

c. Fill out the **properties of the added column**.

- Set [*Code*] to "`UsrAccount`."
- Set [*Title*] to "`Account`."

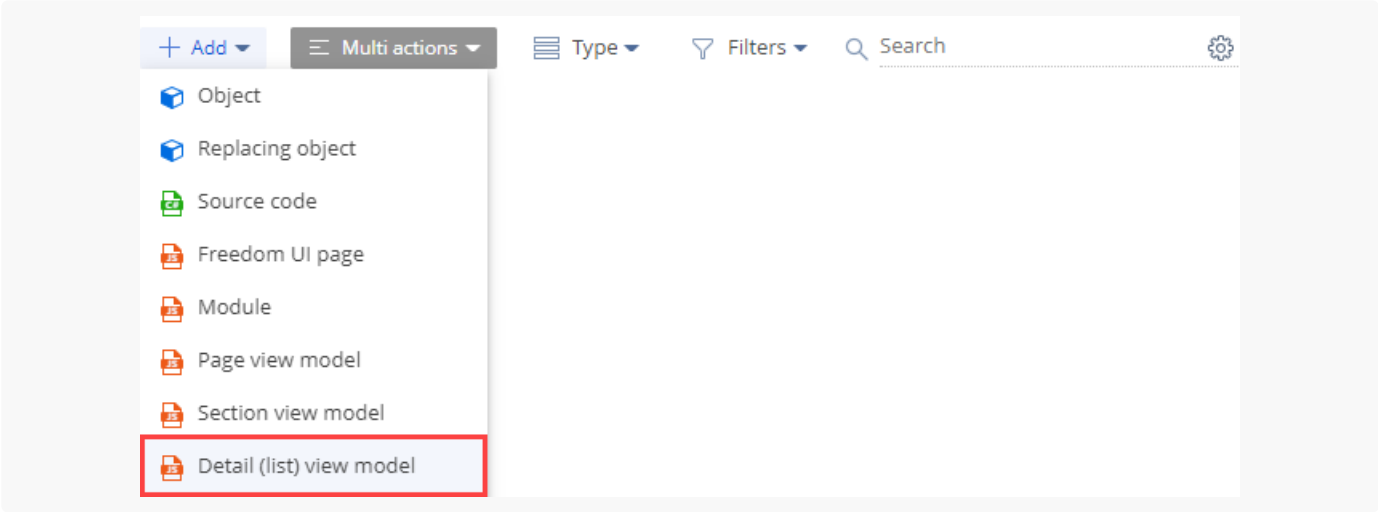
- Select "`Account`" in the [*Lookup*] property.



6. Click [*Save*] then [*Publish*] on the Object Designer's toolbar.

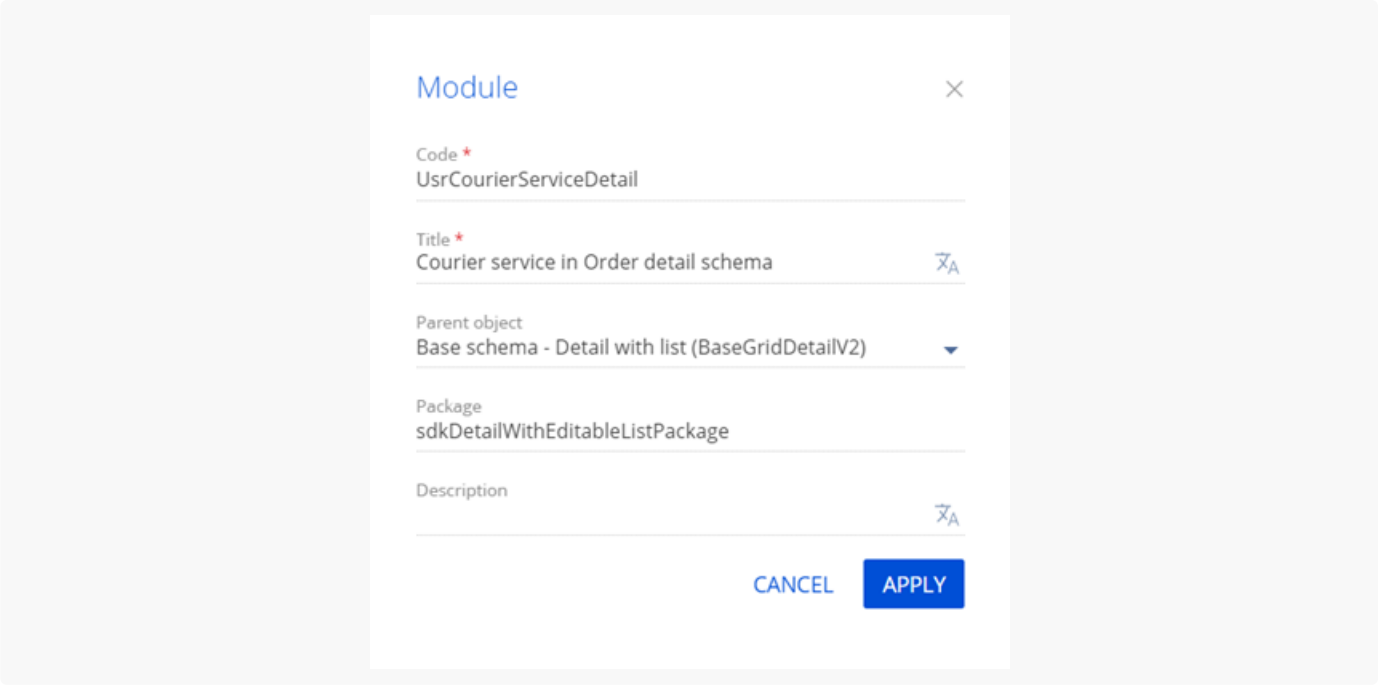
2. Create a view model schema of the detail

1. Go to the [*Configuration*] section and select a custom package to add the schema.
2. Click [*Add*] → [*Detail (list) view model*] on the section list toolbar.



3. Fill out the **schema properties**.

- Set [*Code*] to "`UsrCourierServiceDetail`."
- Set [*Title*] to "`Courier service in Order detail schema`."
- Select "`BaseGridDetailV2`" in the [*Parent object*] property.

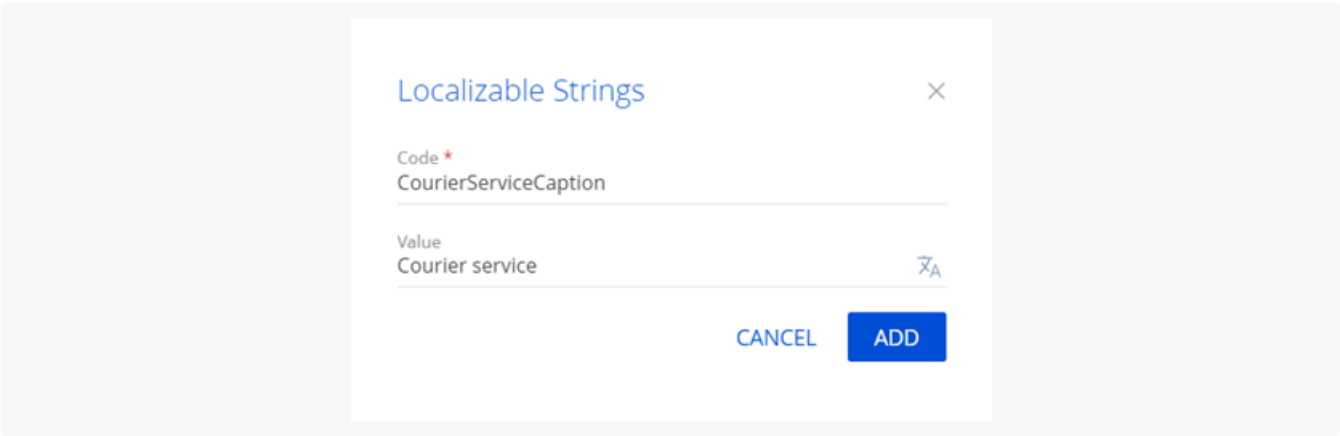


4. Add a **localizable string**.

   a. Click the + button in the context menu of the [*Localizable strings*] node.

   b. Fill out the **localizable string properties**.

   - Set [*Code*] to "`CourierServiceCaption`."
   - Set [*Value*] to "`Courier service`."



   e. Click [*Add*] to add a localizable string.

5. Implement an **editable detail list**.

   a. Add the `ConfigurationGrid`, `ConfigurationGridGenerator`, `ConfigurationGridUtilitiesV2` module schemas as dependencies.

   b. Add the `ConfigurationGridUtilitiesV2` mixin to the `mixins` property.

   c. Add the `IsEditable` attribute to the `attributes` property. Set the `value` property of the attribute to `true`.

   d. Add the configuration object to the `diff array of modifications`. This object must have the property settings and binding of the detail list events' handler methods.

View the source code of the `UsrCourierServiceDetail` view model schema of the detail below.

UsrCourierServiceDetail

```
/* Define the schema and set its dependencies on other modules. */
define("UsrCourierServiceDetail", ["ConfigurationGrid", "ConfigurationGridGenerator", "Config
    return {
        /* The name of the detail object schema. */
        entitySchemaName: "UsrCourierService",
        /* The index of schema attributes. */
        attributes: {
            /* The flag that enables editability. */
            "IsEditable": {
                /* Set the data type to boolean. */
                dataValueType: Terrasoft.DataValueType.BOOLEAN,
                /* Set the attribute type to a virtual column of the view model. */
                type: Terrasoft.ViewModelColumnType.VIRTUAL_COLUMN,
                /* The value to set. */
                value: true
            }
        },
        /* The mixins used. */
        mixins: {
            ConfigurationGridUtilities: "Terrasoft.ConfigurationGridUtilitiesV2"
        },
        /* The array of view model modifications. */
        diff: /**SCHEMA_DIFF*/[
            {
                /* Set the operation type to merge. */
                "operation": "merge",
                /* The name of the schema element on which to execute the operation. */
                "name": "DataGrid",
                /* The object whose properties to merge with the schema element properties. *
                "values": {
                    /* The class name. */
                    "className": "Terrasoft.ConfigurationGrid",
                    /* The view generator must generate only a part of the view. */
                    "generator": "ConfigurationGridGenerator.generatePartial",
                    /* Bind the active string edit element’s configuration retrieval event to
                    "generateControlsConfig": {"bindTo": "generateActiveRowControlsConfig"},
                    /* Bind the active record change event to the handler method. */
                    "changeRow": {"bindTo": "changeRow"},
                    /* Bind the record’s selection cancel event to the handler method. */
                    "unSelectRow": {"bindTo": "unSelectRow"},
                    /* Bind the list click event to the handler method. */
                    "onGridClick": {"bindTo": "onGridClick"},
                    /* Bind the field caption to the localizable schema string. */
```

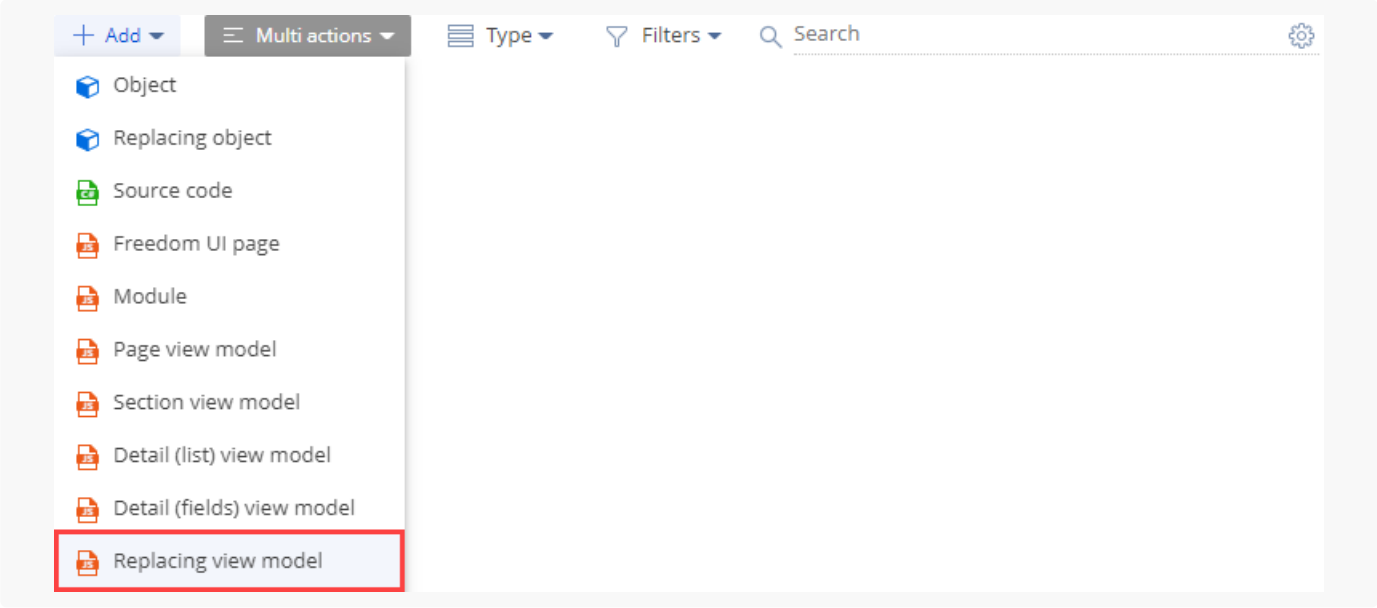
```
                    "caption": {"bindTo": "Resources.Strings.CourierServiceCaption"},
                    /* The operations to execute on the active record. */
                    "activeRowActions": /* Set up the Save] action. */
                        {
                            /* The name of the control class to which the action is connected
                            "className": "Terrasoft.Button",
                            /* Set the button style to transparent. */
                            "style": this.Terrasoft.controls.ButtonEnums.style.TRANSPARENT,
                            /* The tag. */
                            "tag": "save",
                            /* The marker value. */
                            "markerValue": "save",
                            /* Bind to the button image. */
                            "imageConfig": {"bindTo": "Resources.Images.SaveIcon"}
                        },
                        /* Set up the [Edit on page] action. */
                        {
                            "className": "Terrasoft.Button",
                            "style": this.Terrasoft.controls.ButtonEnums.style.TRANSPARENT,
                            "tag": "card",
                            "markerValue": "card",
                            "imageConfig": {"bindTo": "Resources.Images.CardIcon"}
                        },
                        /* Set up the [Copy] action. */
                        {
                            "className": "Terrasoft.Button",
                            "style": Terrasoft.controls.ButtonEnums.style.TRANSPARENT,
                            "tag": "copy",
                            "markerValue": "copy",
                            "imageConfig": {"bindTo": "Resources.Images.CopyIcon"}
                        },
                        /* Set up the [Cancel] action. */
                        {
                            "className": "Terrasoft.Button",
                            "style": this.Terrasoft.controls.ButtonEnums.style.TRANSPARENT,
                            "tag": "cancel",
                            "markerValue": "cancel",
                            "imageConfig": {"bindTo": "Resources.Images.CancelIcon"}
                        },
                        /* Set up the [Delete] action. */
                        {
                            "className": "Terrasoft.Button",
                            "style": this.Terrasoft.controls.ButtonEnums.style.TRANSPARENT,
                            "tag": "remove",
                            "markerValue": "remove",
                            "imageConfig": {"bindTo": "Resources.Images.RemoveIcon"}
                        }
                    ],
                    /* Bind the method that initializes listening to button click events in t
```

```
                    "initActiveRowKeyMap": {"bindTo": "initActiveRowKeyMap"},
                    /* Bind the active record's action execution event to the handler method.
                    "activeRowAction": {"bindTo": "onActiveRowAction"},
                    /* Flag that enables selection of multiple records. */
                    "multiSelect": {"bindTo": "MultiSelect"}
                }
            }
        ]/**SCHEMA_DIFF*/
    };
});
```

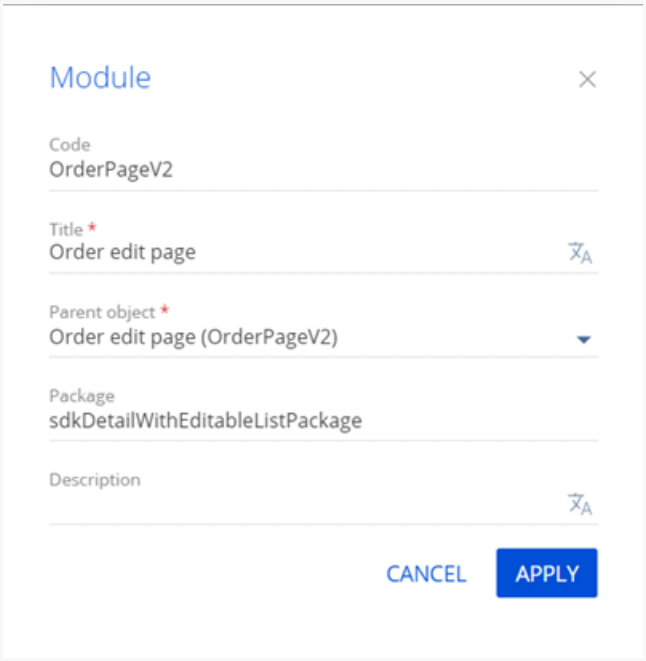
6. Click [*Save*] on the Module Designer's toolbar.

3. Create a replacing view model schema of the order page

1. Go to the [*Configuration*] section and select a custom package to add the schema.
2. Click [*Add*] → [*Replacing view model*] on the section list toolbar.



3. Fill out the **schema properties**.
   - Set [*Code*] to "`OrderPageV2`."
   - Set [*Title*] to "`Order edit page`."
   - Select "`OrderPageV2`" in the [*Parent object*] property.



4. Add the **custom** `UsrCourierServiceDetail` **detail to the order page**.

   - Add the `UsrCourierServiceDetail` detail to the `details` property.
   - Add a configuration object with the settings that determine the detail layout to the `diff` array of modifications.

   View the source code of the order page's replacing view model below.

OrderPageV2

```
define("OrderPageV2", [], function() {
    return {
        /* The name of the record page object's schema. */
        entitySchemaName: "Order",
        /* The details of the record page. */
        details: /**SCHEMA_DETAILS*/{
            /* Set up the [Courier service] detail. */
            "UsrCourierServiceDetail": {
                /* The name of the detail schema. */
                "schemaName": "UsrCourierServiceDetail",
                /* The name of the detail object schema. */
                "entitySchemaName": "UsrCourierService",
                /* Display only the contacts relevant to the current order. */
                "filter": {
                    /* The column of the detail object schema. */
                    "detailColumn": "UsrOrder",
                    /* The column of the section object schema. */
                    "masterColumn": "Id"
                }
            }
```

```
        }/**SCHEMA_DETAILS*/,
        /* Display the detail on the record page. */
        diff: /**SCHEMA_DIFF*/[
            /* The metadata to add the custom detail to the page. */
            {
                /* Add the element to the page. */
                "operation": "insert",
                /* The meta name of the detail to add. */
                "name": "UsrCourierServiceDetail",
                /* The meta name of the parent container to add the detail. */
                "parentName": "OrderDeliveryTab",
                /* Add the detail to the parent element's collection of elements. */
                "propertyName": "items",
                /* The index in the list of elements to add. */
                "index": 3,
                /* The properties to pass to the element's constructor. */
                "values": {
                    /* Set the type of the added element to detail. */
                    "itemType": Terrasoft.core.enums.ViewItemType.DETAIL,
                    "markerValue": "added-detail"
                }
            }
        ]/**SCHEMA_DIFF*/
    };
});
```

5. Click [*Save*] on the Module Designer's toolbar.

4. Register the custom detail in the database

To make the detail visible in the Section Wizard and Detail Wizard, register it in the database. To do this, execute the SQL query to the `[SysDetails]` database table.

SQL query

```
DECLARE
-- The name of the detail schema.
@ClientUnitSchemaName NVARCHAR(100) = 'UsrCourierServiceDetail',
-- The name of the detail object schema.
@EntitySchemaName NVARCHAR(100) = 'UsrCourierService',
-- The name of the detail.
@DetailCaption NVARCHAR(100) = 'CourierService'

INSERT INTO SysDetail(
    Caption,
    DetailSchemaUId,
    EntitySchemaUId
```

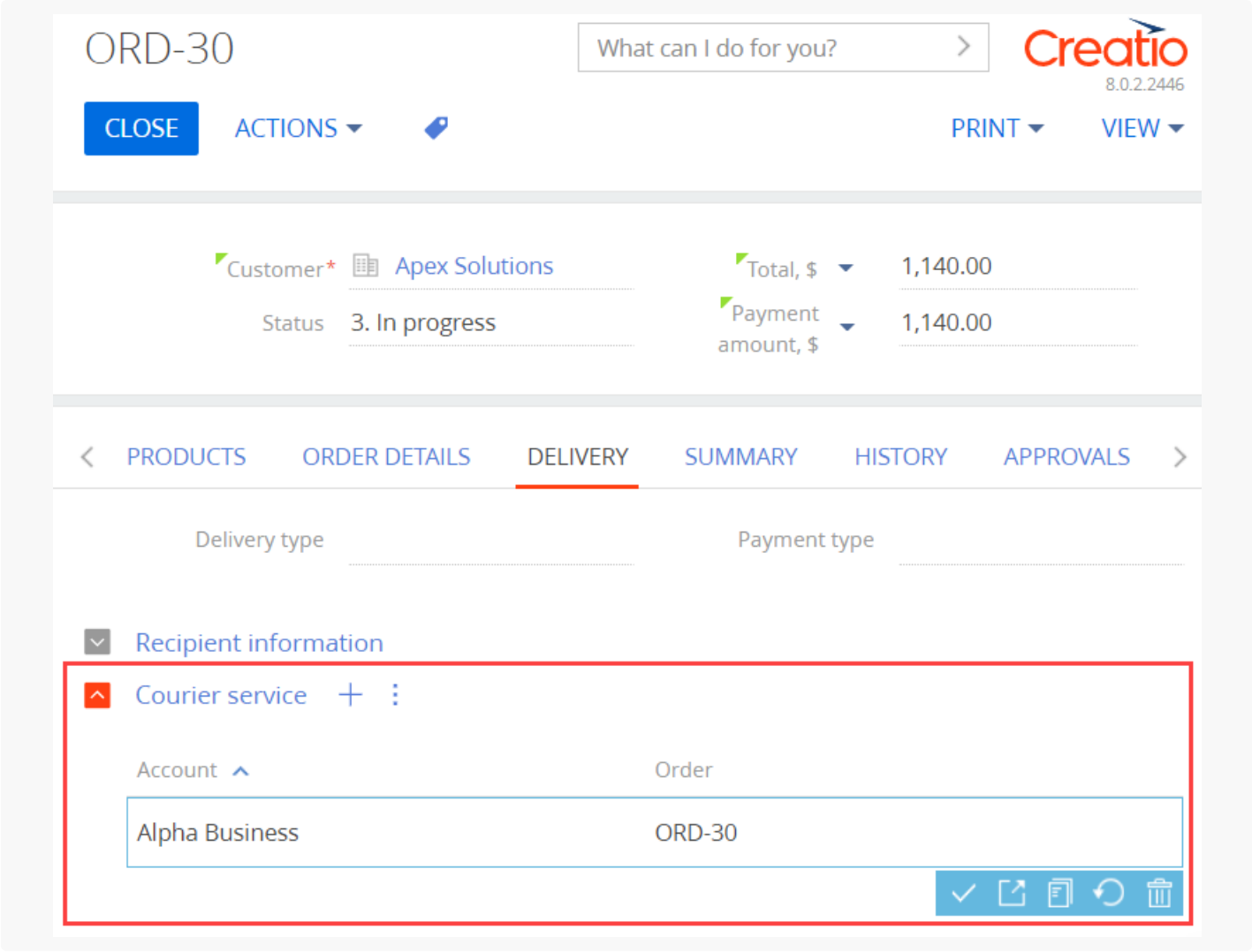
```
)
VALUES(
    @DetailCaption,
    (SELECT TOP 1 UId
    from SysSchema
    WHERE Name = @ClientUnitSchemaName),
    (SELECT TOP 1 UId
    from SysSchema
    WHERE Name = @EntitySchemaName)
)
```

Outcome of the example

To **view the outcome of the example**:

1. Refresh the [*Orders*] section page.
2. Open an order page and go to the [*Delivery*] tab.

As a result, Creatio will add the custom [*Courier service*] detail to the [*Delivery*] tab of the order page. The detail displays the index of available courier services (the values of the [*Account*] lookup) for the current order (the [*Order*] column). The order is populated automatically.



Hide menu items from the detail that contains a list

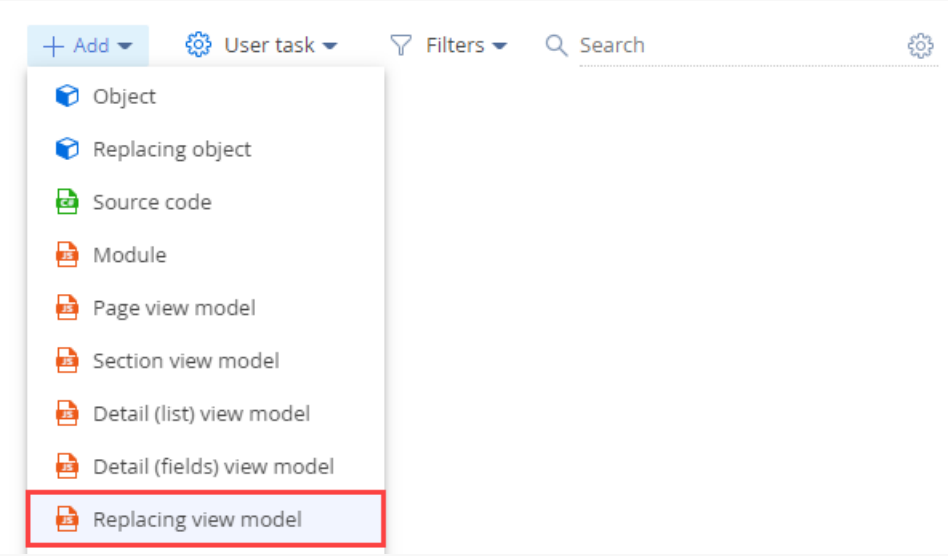
Medium

Example. Hide the [*Copy*], [*Edit*], [*Delete*] menu items located in the [*Addresses*] detail on the [*Contact info*] tab of the contact page.

The **purpose** of the [*Copy*], [*Edit*], [*Delete*] menu items is the management of detail list records.

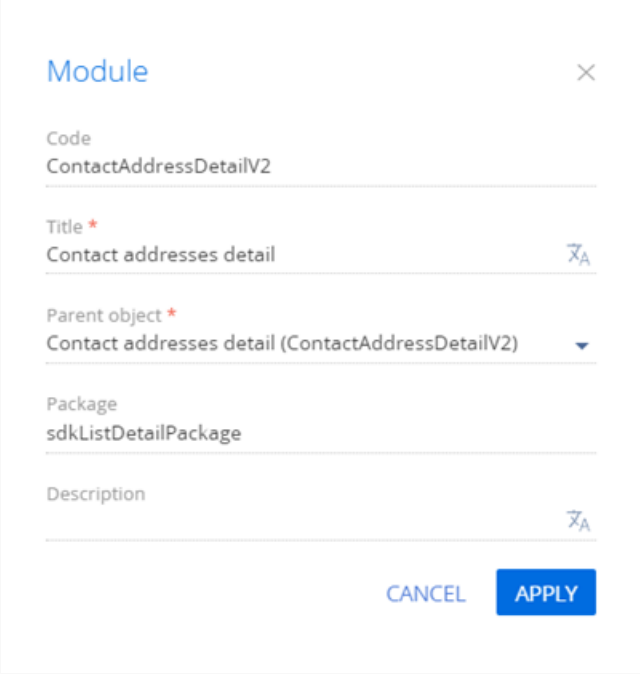
Create the view model schema of the detail list

1. Go to the [*Configuration*] section and select a custom package to add the schema.
2. Click [*Add*] → [*Replacing view model*] on the section list toolbar.



3. Fill out the following schema properties:

   - Set [*Code*] to "`ContactAddressDetailV2`."
   - Set [*Title*] to "`Contact addresses detail`."
   - Set [*Parent object*] to "`ContactAddressDetailV2`."



4. Add the source code in the Module Designer.

`ContactAddressDetailV2`

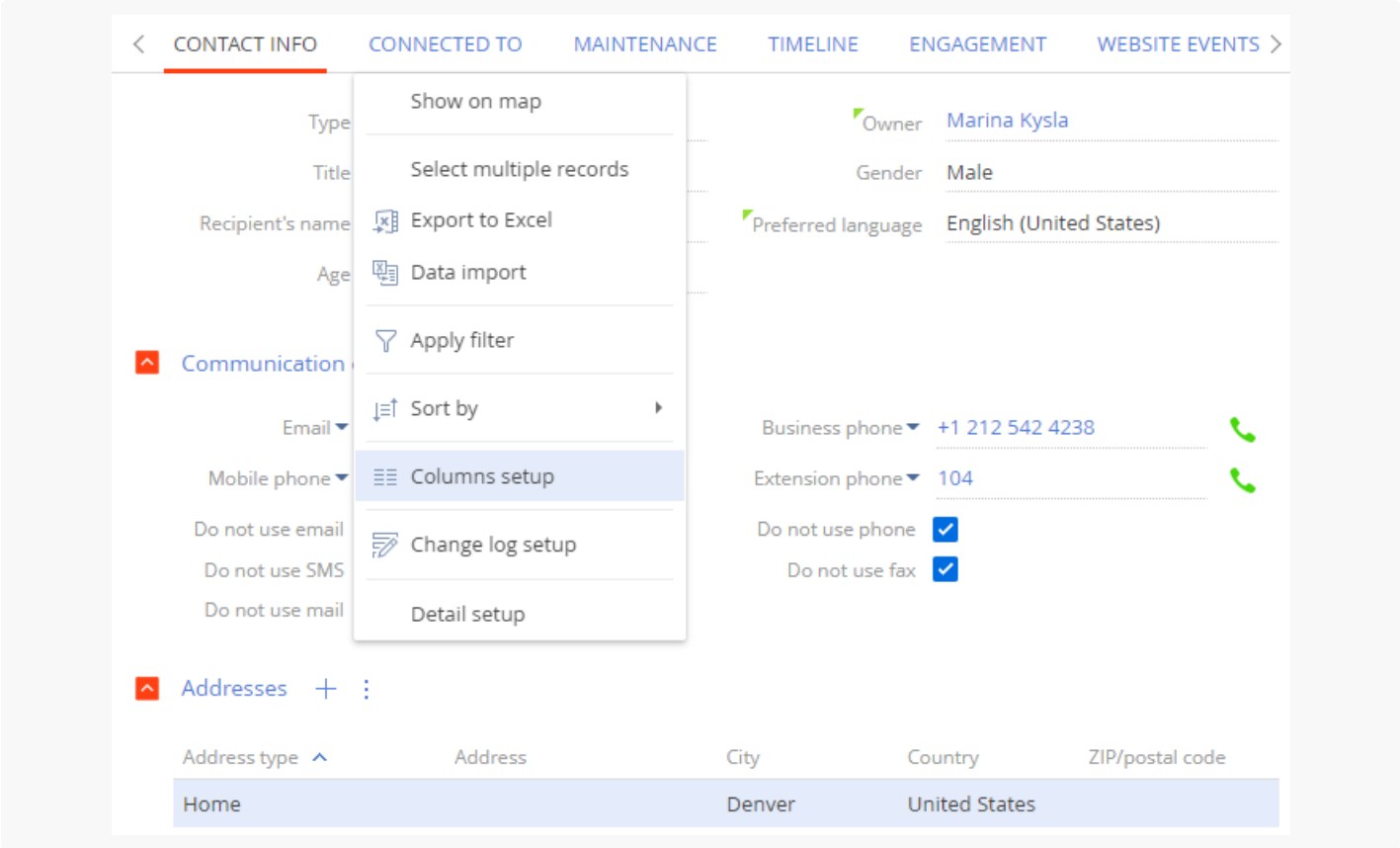
```
define("ContactAddressDetailV2", [], function() {
    return {
        entitySchemaName: "AccountAddress",
```

```
        methods: {
            /* Delete the [Copy] menu item. */
            getCopyRecordMenuItem: Terrasoft.emptyFn,
            /* Delete the [Edit] menu item. */
            getEditRecordMenuItem: Terrasoft.emptyFn,
            /* Delete the [Delete] menu item. */
            getDeleteRecordMenuItem: Terrasoft.emptyFn
        },
        diff: /**SCHEMA_DIFF*/[]/**SCHEMA_DIFF*/
    };
});
```

5. Click [*Save*] on the Designer's toolbar.

Outcome of the example

As a result, Creatio will hide the [*Copy*], [*Edit*], [*Delete*] menu items from the [*Addresses*] detail.



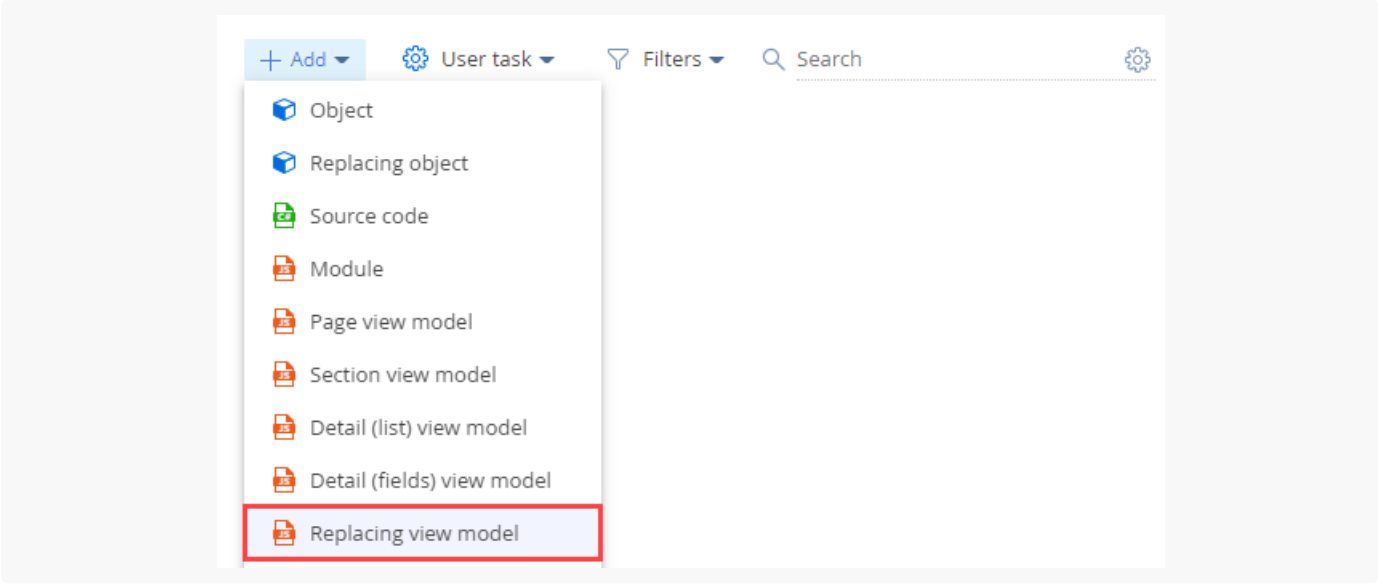
Implement the bulk addition of records to the detail

Advanced

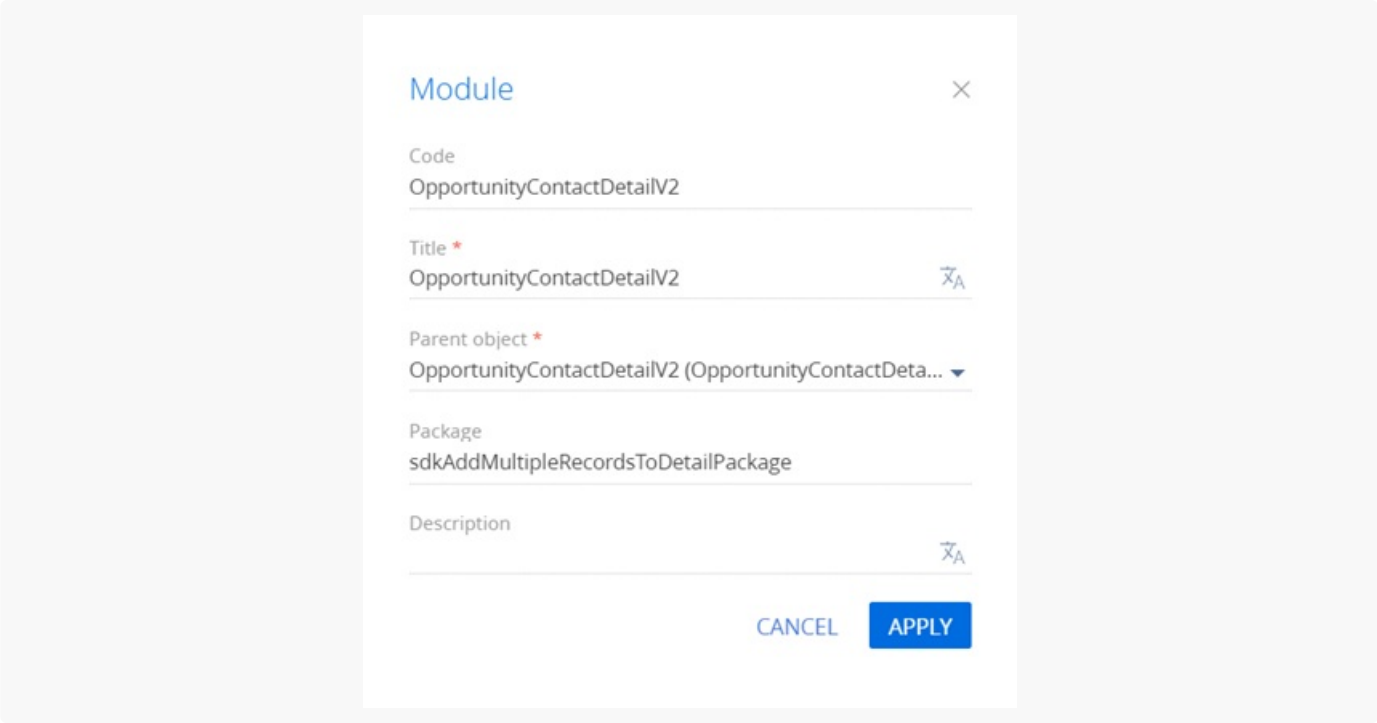
Example. Implement the bulk addition of records to the [*Contacts*] detail on the record page of the [*Opportunities*] section.

1. Create the view model schema of the detail

1. Go to the [*Configuration*] section and select a custom package to add the schema.
2. Click [*Add*] → [*Replacing view model*] on the section list toolbar.



3. Fill out the schema properties in the Module Designer:
   - Set [*Code*] to "`OpportunityContactDetailV2`."
   - Set [*Title*] to "`OpportunityContactDetailV2`."
   - Set [*Parent object*] to "`OpportunityContactDetailV2`."



2. Implement the business logic of the detail

1. Add the `LookupMultiAddMixin` mixin to the `mixins` property of the detail schema.
2. Initalize the `LookupMultiAddMixin` mixin in the overridden `init()` method of the detail schema. Learn more about the `init()` module in a separate article: Module types and their specificities.
3. Override the `getAddRecordButtonVisible()` method that displays the addition button.
4. Override the `onCardSaved()` method that saves the detail page.

   Use the `openLookupWithMultiSelect()` method that calls the multiple selection dialog box.
5. Implement the `getMultiSelectLookupConfig()` method that configures the dialog box. The method is connected to the `openLookupWithMultiSelect()` method and returns the configuration object for the dialog box.

   The object **properties** are as follows:

   - `rootEntitySchemaName` is the root object schema.
   - `rootColumnName` is the connected column that indicates the root schema record.
   - `relatedEntitySchemaName` is the connected schema.
   - `relatedColumnName` is the column that indicates the connected schema record.
6. Override the `addRecord()` method that adds records to the detail. Use the `openLookupWithMultiSelect()`, similar to `onCardSaved()` method. Set the value to `true` to run a check if the record is new.

In this example, the dialog box uses the data of the `[OpportunityContact]` table connected to the `[Opportunity]` column of the `[Opportunity]` root schema and the `[Contact]` column of the `[Contact]` connected schema.

View the source code of the detail schema below.

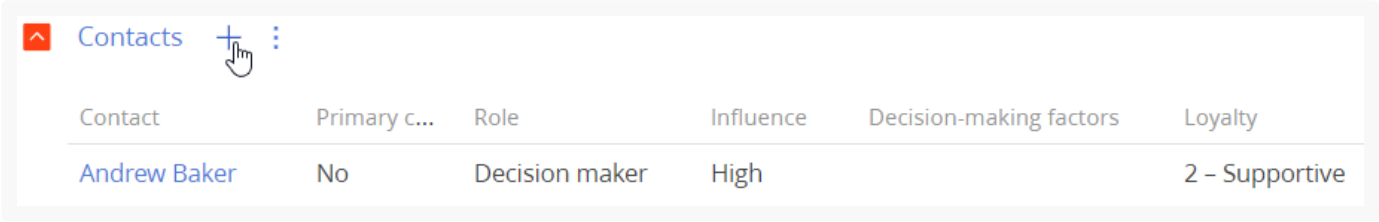
OpportunityContactDetailV2

```
define("OpportunityContactDetailV2", ["LookupMultiAddMixin"], function() {
    return {
        mixins: {
            /* Enable the mixin in the schema. */
            LookupMultiAddMixin: "Terrasoft.LookupMultiAddMixin"
        },
        methods: {
            /* Override the base method that initializes the schema. */
            init: function() {
                this.callParent(arguments);
                /* Initialize the mixin. */
                this.mixins.LookupMultiAddMixin.init.call(this);
            },
            /* Override the base method that displays the addition button. */
            getAddRecordButtonVisible: function() {
                /* Display the addition button if the detail is expanded regardless of whether y
                return this.getToolsVisible();
            },
            /* Override the base method.
            Handles the save event of the detail record page. */
            onCardSaved: function() {
                /* Open the multiple selection dialog box. */
                this.openLookupWithMultiSelect();
            },
            /* Override the base method that adds records to the detail. */
            addRecord: function() {
                /* Open the multiple selection dialog box. */
                this.openLookupWithMultiSelect(true);
            },
            /* The method that returns the configuration object for the dialog box. */
            getMultiSelectLookupConfig: function() {
                return {
                    /* Set the root schema to [Opportunity]. */
                    rootEntitySchemaName: "Opportunity",
                    /* The root schema column. */
                    rootColumnName: "Opportunity",
                    /* Set the connected schema to [Contact]. */
                    relatedEntitySchemaName: "Contact",
                    /* The connected schema column. */
                    relatedColumnName: "Contact"
                };
            }
        }
    };
});
```

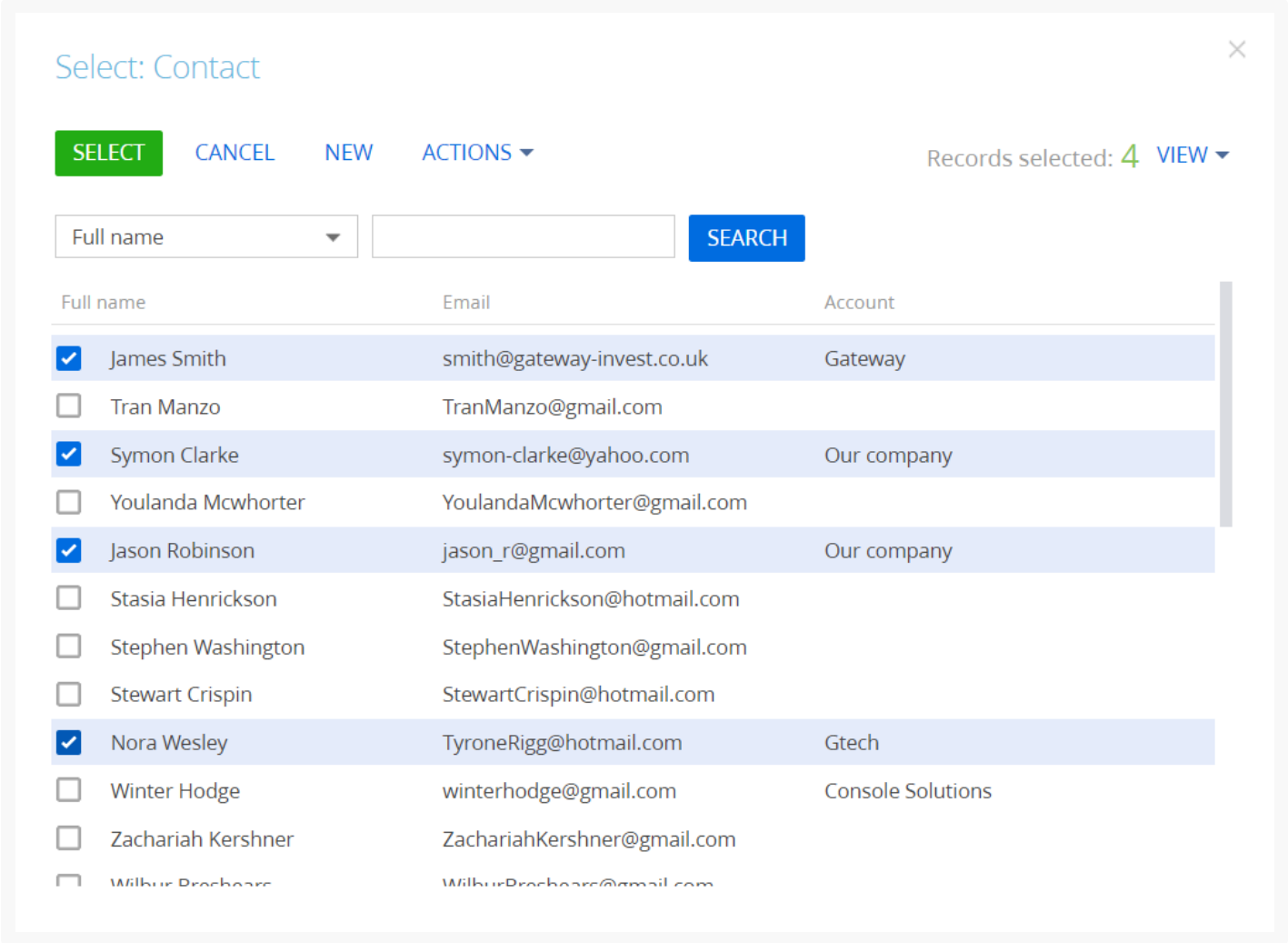
Click [*Save*] on the Module Designer's toolbar.

Outcome of the example

1. Refresh the Creatio page.
2. Click the + button on the [*Contacts*] detail on the record page of the [*Opportunities*] section.



As a result, you will be able to select multiple records from the lookup.



Once you confirm the selection, Creatio will add the records to the [*Contacts*] detail on the record page of the [*Opportunities*] section.

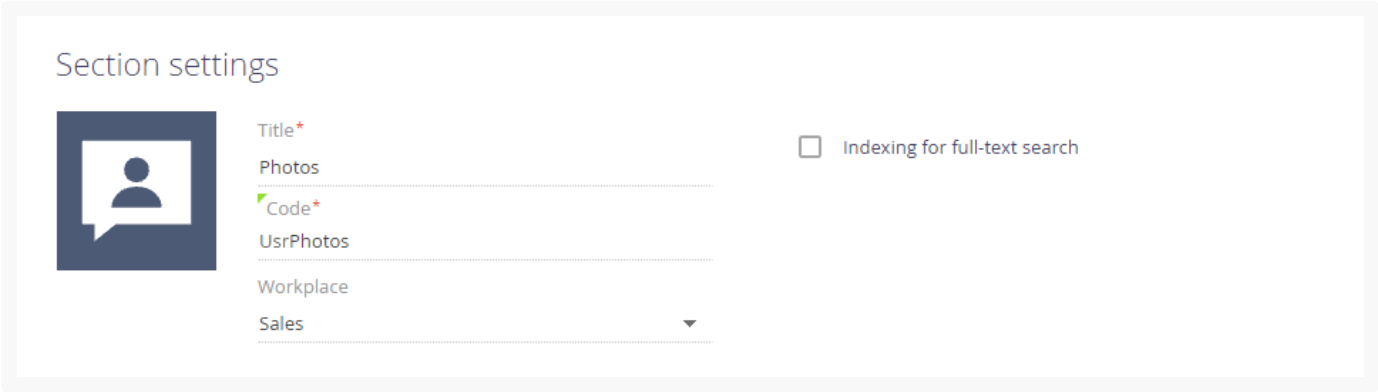
Implement an [Attachments] detail

Advanced

Example. Add the [*Photos attachment*] detail of the [*Attachments*] type to the record page of the [*Photos*] custom section.

1. Create a custom section

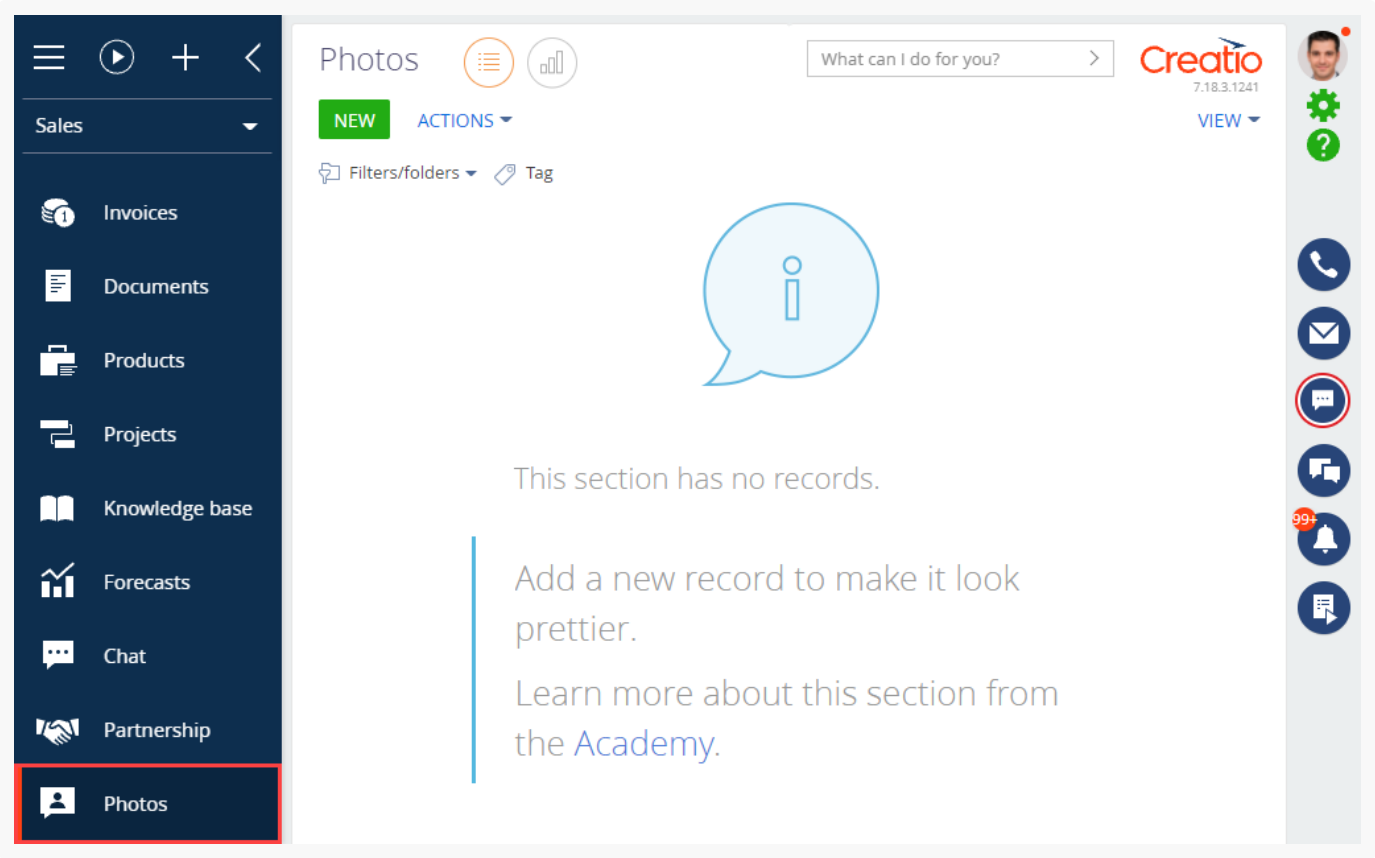
1. Create a custom package and set it as the current package. Learn more in a separate article: Packages basics.
2. Click to open the System Designer.
3. Go to the [*System setup*] block → [*Section wizard*].
4. Fill out the section properties:
   - Set [*Title*] to "`Photos`."
   - Set [*Code*] to "`UsrPhotos`."
   - Set [*Workplace*] to "`Sales`."



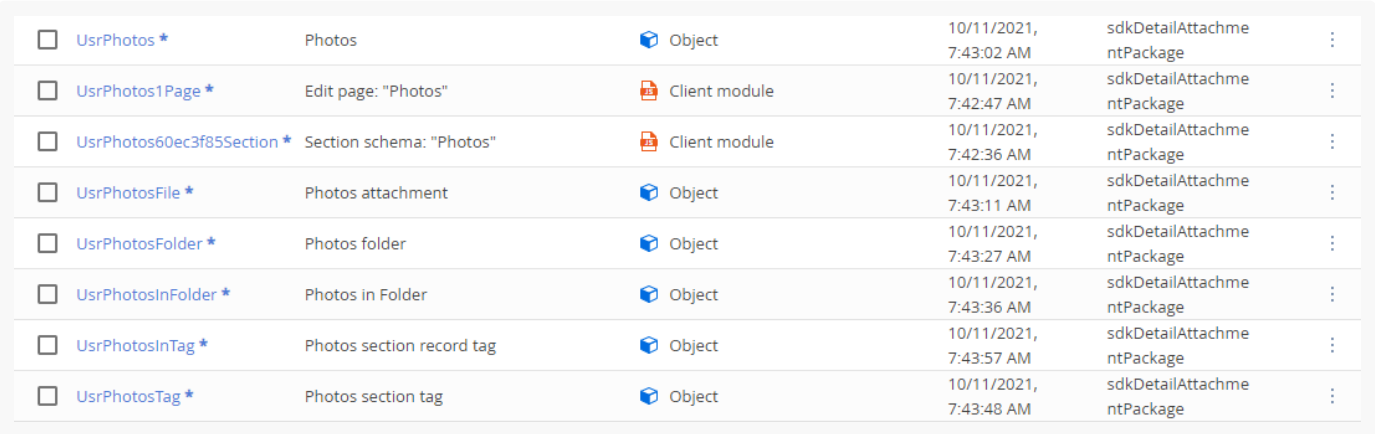
5. Click [*Save*] on the Section Wizard's toolbar.

As a result:

- Creatio will display the [*Photos*] custom section in the [*Sales*] workplace.



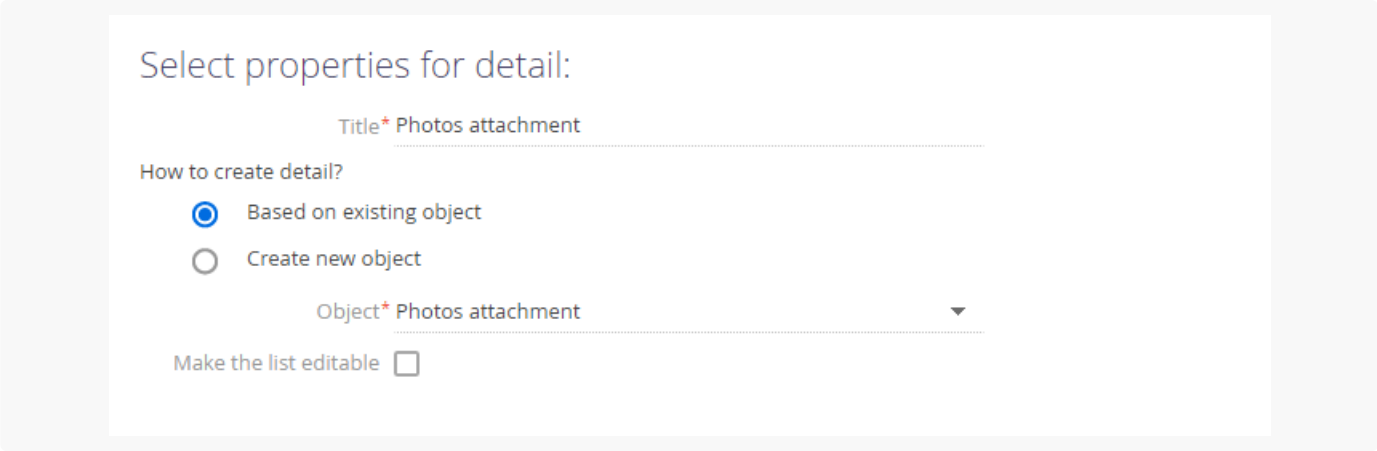
- Creatio will add the schemas of the [*Photos*] section to the configuration.



UsrPhotos *	Photos	Object	10/11/2021, 7:43:02 AM	sdkDetailAttachmentPackage
UsrPhotos1Page *	Edit page: "Photos"	Client module	10/11/2021, 7:42:47 AM	sdkDetailAttachmentPackage
UsrPhotos60ec3f85Section *	Section schema: "Photos"	Client module	10/11/2021, 7:42:36 AM	sdkDetailAttachmentPackage
UsrPhotosFile *	Photos attachment	Object	10/11/2021, 7:43:11 AM	sdkDetailAttachmentPackage
UsrPhotosFolder *	Photos folder	Object	10/11/2021, 7:43:27 AM	sdkDetailAttachmentPackage
UsrPhotosInFolder *	Photos in Folder	Object	10/11/2021, 7:43:36 AM	sdkDetailAttachmentPackage
UsrPhotosInTag *	Photos section record tag	Object	10/11/2021, 7:43:57 AM	sdkDetailAttachmentPackage
UsrPhotosTag *	Photos section tag	Object	10/11/2021, 7:43:48 AM	sdkDetailAttachmentPackage

2. Create a custom detail

1. Click to open the System Designer.
2. Go to the [*System setup*] block → [*Detail wizard*].
3. Fill out the detail properties:
   - Set [*Title*] to "`Photos attachment`."
   - Set [*How to create detail?*] to "`Based on existing object`."
   - Set [*Object*] to "`Photos attachment`."



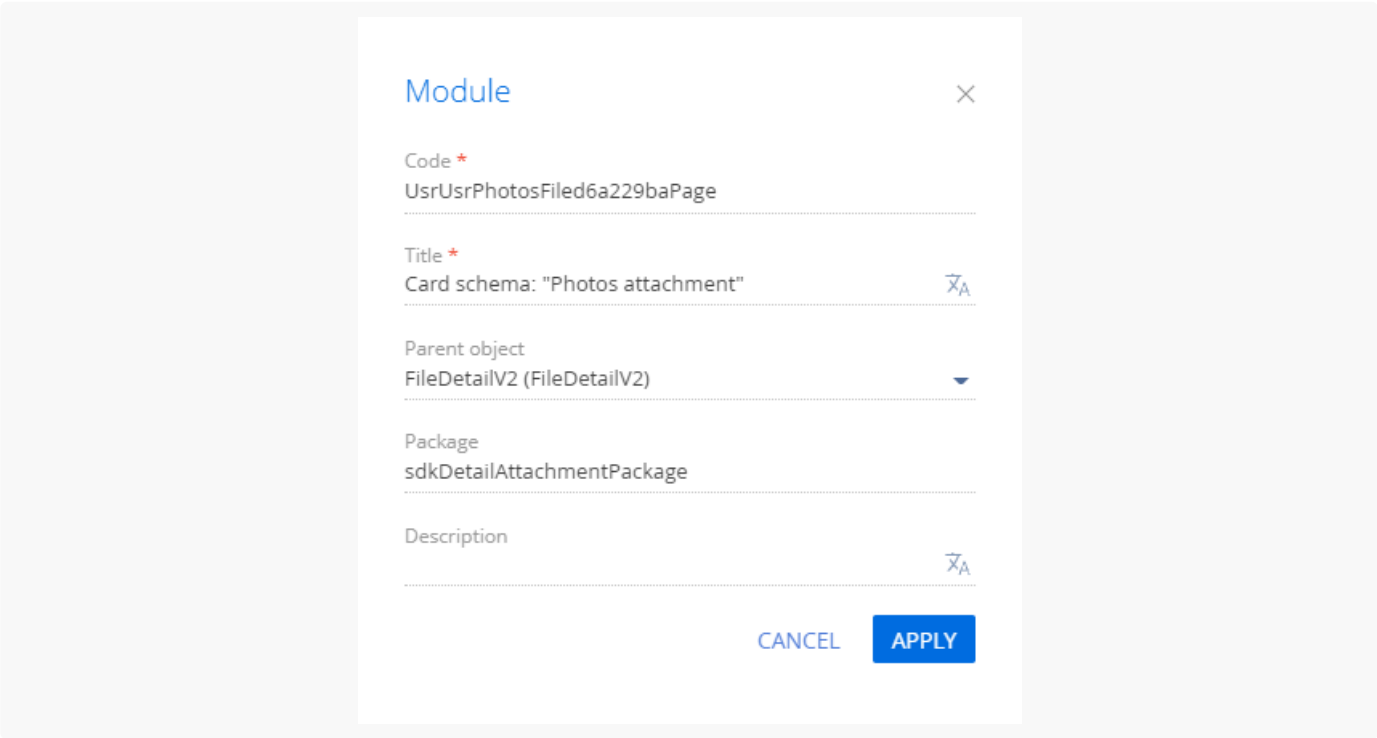
4. Click [*Save*] on the Detail Wizard's toolbar.

As a result, Creatio will add the following schemas to the configuration:

- The `UsrSchemae9733d1bDetail` schema of the [*Photos attachments*] detail's view model.
- The `UsrUsrPhotosFiled6a229baPage` schema of the [*Photos attachments*] detail's record page.

3. Set up the custom detail

1. Go to the [*Configuration*] section and select a custom package to set as current.
2. Open the `UsrUsrPhotosFiled6a229baPage` schema of the [*Photos attachments*] detail's record page.
3. Click the button on the properties panel and specify `FileDetailV2` in the [*Parent object*] field. The `FileDetailV2` schema of the `UIv2` package implements the [*Attachments*] detail. By default, the parent object in the Detail Wizard is the base detail schema that contains the list.

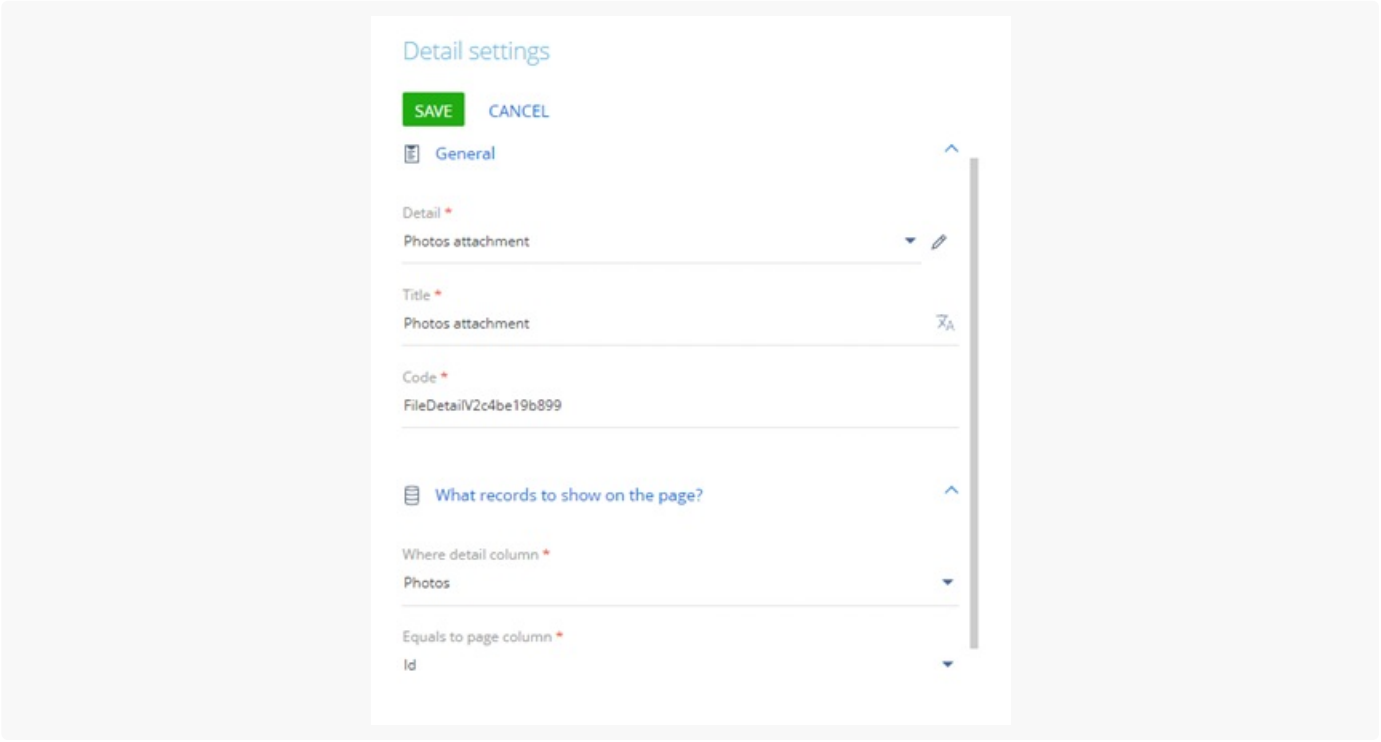


4. Click [*Apply*] to apply the properties.
5. Click [*Save*] on the Designer's toolbar.

4. Add the detail to the section

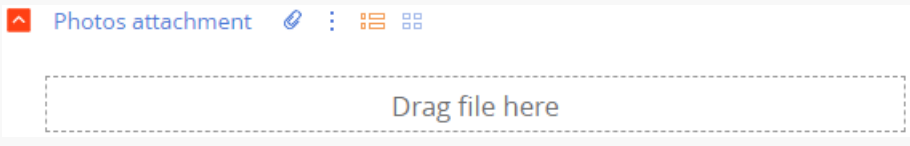
1. Go to the [*Photos*] section.
2. Click [*View*] → [*Open section wizard*] on the toolbar.
3. Go to the [*Section pages*] block → the [*Edit page*] button.
4. Click [*New detail*] in the Section Wizard workspace.
5. Fill out the detail parameters.

   - Set [*Detail*] to "`Photos attachment`." Creatio will populate the [*Title*] and [*Code*] fields.
   - Set [*Title*] to "`Photos attachment`."



6. Click [*Save*] → [*Section wizard*] → [*Save*].

As a result, Creatio will add the [*Photos attachment*] detail to the record page of the [*Photos*] section.



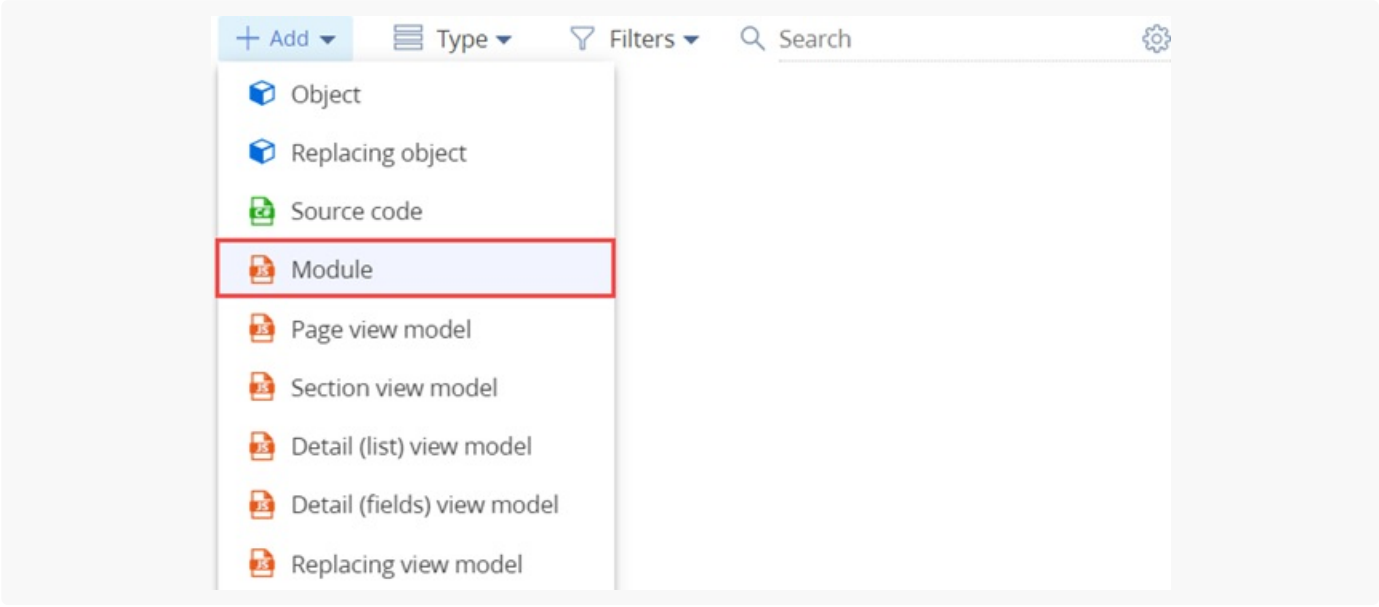
5. Add custom detail styles

The view model schema of the detail page does not support visual styles. As such, take the following steps to add styles:

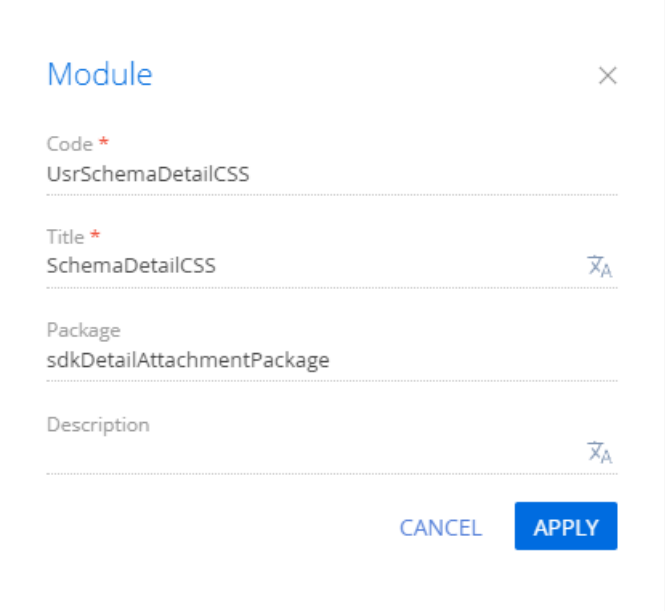
1. Create a module schema. Define the styles in it.
2. Add the style module to the dependencies of the detail module.

1. Create a module schema

1. Go to the [*Configuration*] section and select a custom package to set as the current package.
2. Click [*Add*] → [*Module*] on the section list toolbar.



3. Fill out the following schema properties:
   - Set [*Code*] to "`UsrSchemaDetailCSS`."
   - Set [*Title*] to "`SchemaDetailCSS`."



Click [*Apply*] to apply the properties.

4. Go to the [*LESS*] node of the object structure and set up the needed visual styles of the detail.

Set up the visual styles of the detail

```
div[id*="UsrSchemae9733d1bDetail"] {
    .grid-status-message-empty {
        display: none;
    }
    .grid-empty > .grid-bottom-spinner-space {
        height: 5px;
    }
    .dropzone {
        height: 35px;
        width: 100%;
        border: 1px dashed #999999;
        text-align: center;
        line-height: 35px;
    }
    .dropzone-hover {
        border: 1px dashed #4b7fc7;
    }
    .DragAndDropLabel {
        font-size: 1.8em;
        color: rgb(110, 110, 112);
    }
}

div[data-item-marker*="added-detail"] {
    div[data-item-marker*="tiled"], div[data-item-marker*="listed"] {
```

```
        .entity-image-class {
            width: 165px;
        }
        .entity-image-container-class {
            float: right;
            width: 128px;
            height: 128px;
            text-align: center;
            line-height: 128px;
        }
        .entity-image-view-class {
            max-width: 128px;
            max-height: 128px;
            vertical-align: middle;
        }
        .images-list-class {
            min-height: 0.5em;
        }
        .images-list-class > .selectable {
            margin-right: 10px;
            display: inline-block;
        }
        .entity-label {
            display: block;
            max-width: 128px;
            margin-bottom: 10px;
            text-align: center;
        }
        .entity-link-container-class > a {
            font-size: 1.4em;
            line-height: 1.5em;
            display: block;
            max-width: 128px;
            margin-bottom: 10px;
            color: #444;
            text-decoration: none;
            text-overflow: ellipsis;
            overflow: hidden;
            white-space: nowrap;
        }
        .entity-link-container-class > a:hover {
            color: #0e84cf;
        }
        .entity-link-container-class {
            float: right;
            width: 128px;
            text-align: center;
        }
        .select-entity-container-class {
```

```
            float: left;
            width: 2em;
        }
        .listed-mode-button {
            border-top-right-radius: 1px;
            border-bottom-right-radius: 1px;
        }
        .tiled-mode-button {
            border-top-left-radius: 1px;
            border-bottom-left-radius: 1px;
        }
        .tiled-mode-button, .listed-mode-button {
            padding-left: 0.308em;
            padding-right: 0.462em;
        }
    }
    .button-pressed {
        background: #fff;

        .t-btn-image {
            background-position: 0 16px !important;
        }
    }
    div[data-item-marker*="tiled"] {
        .tiled-mode-button {
            .button-pressed;
        }
    }
    div[data-item-marker*="listed"] {
        .listed-mode-button {
            .button-pressed;
        }
    }
}
```

5. Click [*Save*] on the Designer's toolbar.

2. Modify the view model schema of a detail

To **use the module and its styles** in the detail schema:

1. Open the `UsrSchemae9733d1bDetail` schema of the [*Photos attachment*] detail's view model.
2. Add the `UsrSchemaDetailCSS` module to the dependencies of the `UsrSchemae9733d1bDetail` schema.

   View the source code of the modified schema below.

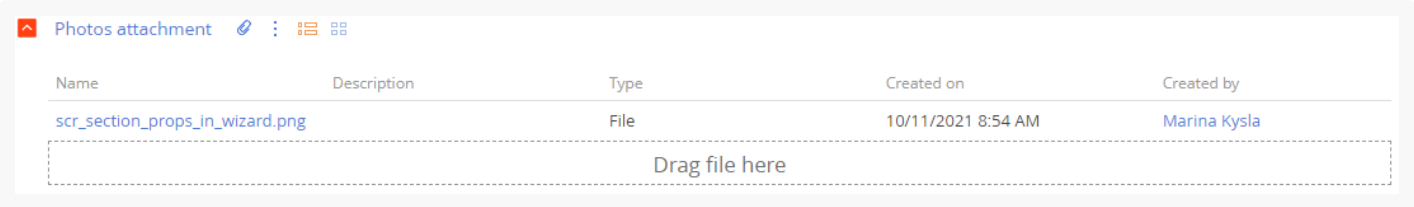
```
UsrSchemae9733d1bDetail
```

```
define("UsrSchemae9733d1bDetail", ["css!UsrSchemaDetailCSS"], function() {
    return {
        entitySchemaName: "UsrPhotosFile",
        details: /**SCHEMA_DETAILS*/{}/**SCHEMA_DETAILS*/,
        methods: {},
        diff: /**SCHEMA_DIFF*/[]/**SCHEMA_DIFF*/
    };
});
```

3. Click [*Save*] on the Designer's toolbar.

Outcome of the example

As a result, Creatio will add the [*Photos attachment*] detail to the record page of the [*Photos*] custom section.



BaseDetailV2 schema

Advanced

`BaseDetailV2` is the base detail schema. Provides base logic for detail data initialization and detail interaction with the record page Implemented in the `NUI` package. This is a view model schema. Learn more about the schema properties in a separate article: Client schema. Detail schemas must inherit the `BaseDetailV2` schema.

Messages

Base detail messages

Name	Mode	Direction	Description
`GetCardState`	Address	Publishing	Returns the record page status.
`IsCardChanged`	Address	Publishing	Informs of record page changes.
`SaveRecord`	Address	Publishing	Informs the record page of the requirement to save the data.
`DetailChanged`	Address	Publishing	Informs the record page of detail data changes.
`UpdateDetail`	Address	Subscription	Subscribes to record page updates.
`OpenCard`	Address	Publishing	Opens the record page.
`GetColumnsValues`	Address	Publishing	Returns the requested column values.
`UpdateCardProperty`	Address	Publishing	Changes the value of the record page model.
`GetEntityInfo`	Address	Publishing	Requests the main record entity data.

The `Terrasoft.core.enums.MessageMode` enumeration represents the message modes, and the `Terrasoft.core.enums.MessageDirectionType` enumeration represents the message directions. Learn more about the `MessageMode` enumeration in the JS class library. Learn more about the `MessageDirectionType` enumeration in the JS class library.

Attributes

`CanAdd` `BOOLEAN`

The flag that enables data addition.

`CanEdit` `BOOLEAN`

The flag that enables data editing.

`CanDelete` `BOOLEAN`

The flag that enables data deletion.

`Collection` `COLLECTION`

The data collection of the detail.

`Filter` `CUSTOM_OBJECT`

The detail filter. Required to filter the detail data.

`DetailColumnName` `STRING`

The name of the column by which to filter the data.

`MasterRecordId` `GUID`

The parent record key value.

`IsDetailCollapsed` `BOOLEAN`

The flag that collapses the detail.

`DefaultValues` `CUSTOM_OBJECT`

The default model column values.

`Caption` `STRING`

The detail title.

The `Terrasoft.core.enums.DataValueType` enumeration represents the attribute data types. Learn more about the `DataValueType` enumeration in the JS class library.

Methods

`init(callback, scope)`

Initializes the detail page.

Parameters

`{Function} callback`	The callback function.
`{Object} scope`	The method execution context.

`initProfile`

Initializes the schema profile. Set to `Terrasoft.emptyFn` by default.

`initDefaultCaption()`

Sets the default detail title.

`initDetailOptions()`

Initializes the collection of list view data.

`subscribeSandboxEvents()`

Subscribes to messages required for the detail to operate as intended.

`getUpdateDetailSandboxTags()`

Generates a tag array for the `UpdateDetail` message.

`updateDetail`

Updates the detail based on the passed parameters. Set to `Terrasoft.emptyFn` by default.

Parameters

`{Object} config`	Configuration object that contains the detail properties.

`initData(callback, scope)`

Initializes the collection of list view data.

Parameters

`{Function} callback`	The callback function.
`{Object} scope`	The method execution context.

`getEditPageName()`

Returns the record page name based on the type of selected record (when editing) or the selected type of record to add (when adding).

`onDetailCollapsedChanged(isCollapsed)`

Detail expansion or collapse handler.

Parameters

`{Boolean} isCollapsed`	Flag that collapses the detail.

`getToolsVisible()`

Returns the detail collapse value.

`getDetailInfo()`

Publishes the message for retrieving data about the detail.

Array of modifications

In base detail, the `diff` array of modifications defines only the base container for the detail view.

`diff` array of modifications

```
diff: /**SCHEMA_DIFF*/[
    /* Base container for the detail view. */
    {
        "operation": "insert",
        "name": "Detail",
        "values": {
            ...
        }
    }
]/**SCHEMA_DIFF*/
```

BaseGridDetailV2 schema JS

Advanced

`BaseGridDetailV2` is a base schema of a list detail. Provides the base logic for list management (loading, filtering), as well as deleting, adding, and editing detail records. Implemented in the `NUI` package. This is a view model schema. Learn more about the schema properties in a separate article: Client schema. Inherits the `BaseDetailV2` schema. Schemas of list details must inherit the `BaseGridDetailV2` schema.

Messages

Messages of a base list detail

Name	Mode	Direction	Description
`getCardInfo`	Address	Subscription	Returns the record page data: default values, typing column name, typing column value.
`CardSaved`	Broadcasting	Subscription	Processes the saving message of the record page.
`RerenderQuickFilterModule`	Address	Publishing	Publishes the message that has the filter applied.
`GetExtendedFilterConfig`	Address	Subscription	Publishes the configuration of a custom filter.
`GetModuleSchema`	Address	Subscription	Returns data about the entity that manages the filter.
`UpdateFilter`	Broadcasting	Subscription	Updated the filters in the detail.
`LoadedFiltersFromStorage`	Broadcasting	Publishing	The filters loaded from the repository.
`InitFilterFromStorage`	Broadcasting	Subscription	Initializes the filters loaded from the repository.
`GetColumnsValues`	Address	Publishing	Retrieves the column values of the record page model.
`IsCardChanged`	Address	Publishing	Notifies of record page changes.
`ValidateCard`	Address	Publishing	Requests to validate the record page.

The `Terrasoft.core.enums.MessageMode` enumeration represents the message modes, and the `Terrasoft.core.enums.MessageDirectionType` enumeration represents the message directions. Learn more about the `MessageMode` enumeration in the JS class library. Learn more about the `MessageDirectionType` enumeration in the JS class library.

Mixins

`GridUtilities` `Terrasoft.GridUtilities`

Mixin for list management.

`WizardUtilities` `Terrasoft.WizardUtilities`

Mixing for Detail Wizard management.

Attributes

`ActiveRow` `GUID`

The primary column value of the active list record.

`IsGridEmpty` `BOOLEAN`

The flag that marks the list as empty.

`MultiSelect` `BOOLEAN`

The flag that enables multiple selection.

`SelectedRows` `COLLECTION`

The array of selected records.

`RowCount` `INTEGER`

The number of rows in the list.

`IsPageable` `BOOLEAN`

The flag that enables list pagination.

`SortColumnIndex` `INTEGER`

Index of the sorting column.

`CardState` `TEXT`

Record page opening mode.

`EditPageUId` `GUID`

Unique ID of the record page.

`DetailFilters` `COLLECTION`

Collection of detail filters.

`IsDetailWizardAvailable` BOOLEAN

The flag that marks the Detail Wizard as available.

The `Terrasoft.core.enums.DataValueType` enumeration represents the attribute data types. Learn more about the `DataValueType` enumeration in the JS class library.

Methods

`init(callback, scope)`

Replaces the `BaseDetailV2` class method. Calls the logic of the parent schema's `init` method, registers messages, initializes filters.

Parameters

`{Function} callback`	The callback function.
`{Object} scope`	The method execution context.

`initData(callback, scope)`

Replaces the `BaseDetailV2` class method. Calls the logic of the parent's `initData` method, initializes the data collection of the list view model.

Parameters

`{Function} callback`	The callback function.
`{Object} scope`	The method execution context.

`loadGridData()`

Loads the list data.

`initGridData()`

Initializes the default values for list management.

`getGridData()`

Returns the list collection.

`getFilters()`

Returns the detail filters collection.

`getActiveRow()`

Returns the ID of the selected list record.

`addRecord(editPageUId)`

Adds a new record to the list. Saves the record page if needed.

Parameters

`{String} editPageUId`	The ID of the typed record page.

`copyRecord(editPageUId)`

Copies the record and opens the record page.

Parameters

`{String} editPageUId`	The ID of the typed record page.

`editRecord(record)`

Opens the page of the selected record.

Parameters

`{Object} record`	The model of the page to edit.

`subscribeSandboxEvents()`

Subscribes to messages required for detail to operate as intended.

`updateDetail(config)`

Replaces the `BaseDetailV2` class method. Calls the logic of the parent schema's `updateDetail` method, updates the detail.

Parameters

`{Object} config`	Configuration object that contains the detail properties.

`openCard(operation, typeColumnValue, recordId)`

Opens the record page.

Parameters

`{String} operation`	Operation type (add/edit).
`{String} typeColumnValue`	The value of the record typing column.
`{String} recordId`	Record ID.

`onCardSaved()`

Processes the detail record page's save event.

`addToolsButtonMenuItems(toolsButtonMenu)`

Adds elements to the collection of the functional button's drop-down list.

Parameters

`{Terrasoft. BaseViewModelCollection} toolsButtonMenu`	The collection of the functional button's drop-down list

---

`initDetailFilterCollection()`

Initializes the detail filter.

`setFilter(key, value)`

Sets the value of detail filters.

Parameters

`{String} key`	Type of the filters.
`{Object} value`	Value of the filters.

`loadQuickFilter(config)`

Loads the quick filter.

Parameters

`{Object} config`	The load parameters of the filter module.

`destroy()`

Clears data, unloads the detail.

Array of modifications

In base list detail, the `diff` array of modifications defines only the base container for the detail view.

`diff` array of modifications

```
diff: /**SCHEMA_DIFF*/ [
    {
        /* The element that displays the list. */
        "operation": "insert",
        "name": "DataGrid",
        "parentName": "Detail",
        "propertyName": "items",
        "values": {
        "itemType": Terrasoft.ViewItemType.GRID,
        …
        }
    },
    {
        /* The button that loads more list data. */
        "operation": "insert",
        "parentName": "Detail",
        "propertyName": "items",
        "name": "loadMore",
        "values": {
        "itemType": Terrasoft.ViewItemType.BUTTON,
        …
        }
    },
```

```
    {
        /* The button that adds records. */
        "operation": "insert",
        "name": "AddRecordButton",
        "parentName": "Detail",
        "propertyName": "tools",
        "values": {
        "itemType": Terrasoft.ViewItemType.BUTTON,
        …
        }
    },
    {
        /* The button that adds a typed record. */
        "operation": "insert",
        "name": "AddTypedRecordButton",
        "parentName": "Detail",
        "propertyName": "tools",
        "values": {
        "itemType": Terrasoft.ViewItemType.BUTTON,
        …
        }
    },
    {
        /* The detail menu. */
        "operation": "insert",
        "name": "ToolsButton",
        "parentName": "Detail",
        "propertyName": "tools",
        "values": {
        "itemType": Terrasoft.ViewItemType.BUTTON,
        …
        }
    }
] /**SCHEMA_DIFF*/
```

GridUtilitiesV2 mixin JS

Advanced

`GridUtilitiesV2` is a mixin that provides the management logic for the "List" control. Implemented in the `Terrasoft.configuration.mixins.GridUtilities` class of the `NUI` package. Learn more about the “List” control in the user documentation: Work with record lists.

The mixin **lets you**:

- Subscribe to messages.
- Load data.

- Manage the list:
  - Select records (search for active records).
  - Add, delete, edit records.
  - Set the filters.
  - Sort the records.
  - Export records to a file.
  - Check the access permissions to the list records.

Methods

`init()`

Subscribes to events.

`destroy()`

Clears event subscriptions.

`loadGridData()`

Loads the list data.

`beforeLoadGridData()`

Prepares the view model before the data is loaded.

`afterLoadGridData()`

Prepares the view model after the data is loaded.

`onGridDataLoaded(response)`

Handles data load events. Executed when the server returns data.

Parameters

`{Object} response`	The result of database data selection.

`addItemsToGridData(dataCollection, options)`

Adds the collection of new elements to the collection of the list.

Parameters

`{Object} dataCollection`	The collection of new elements.
`{Object} options`	Addition parameters.

`initQueryOptions(esq)`

Initializes the query instance settings (pagination, hierarchy).

Parameters

`{Terrasoft.data.queries. EntitySchemaQuery} esq`	The query in which to initialize the relevant settings.

`initQuerySorting(esq)`

Initializes the sorting columns.

Parameters

`{Terrasoft.data.queries. EntitySchemaQuery} esq`	The query in which to initialize the relevant settings.

`prepareResponseCollection(collection)`

Modifies the data collection before loading it to the list.

Parameters

`{Object} collection`	Collection of list elements.

`getFilters()`

Returns the filters applied to the schema. Overload the method in the inheritors.

`exportToFile()`

Exports the list contents to a file.

`sortGrid(tag)`

Sorts the list.

Parameters

{String} tag	Specifies how to sort the list.

`deleteRecords()`

Initiates deletion of the selected records.

`checkCanDelete(items, callback, scope)`

Checks whether the record can be deleted.

Parameters

{Array} items	IDs of selected records.
{Function} callback	The callback function.
{Object} scope	The method execution context.

`onDeleteAccept()`

Deletes the record after the user confirms the deletion.

`getSelectedItems()`

Returns the selected list records.

`removeGridRecords(records)`

Removes the deleted records from the list.

Parameters

{Array} records	The deleted records.

`reloadGridData()`

Reloads the list.

ConfigurationGrid module

ConfigurationGrid module JS

Advanced

`ConfigurationGrid` is a module that provides the management logic for the "Configuration grid" control. Implemented in the `Terrasoft.controls.ConfigurationGrid` class of the `UIv2` package. The `Terrasoft.controls.ConfigurationGrid` class inherits from the `Terrasoft.Grid` class.

Methods

`init()`

Initializes the component. Subscribes to events.

`activateRow(id)`

Selects a row and adds edit elements.

Parameters

`{String|Number} id`	The list row ID.

`deactivateRow(id)`

Deselects a row and deletes edit elements.

Parameters

`{String|Number} id`	The list row ID.

`formatCellContent(cell, data, column, link)`

Formats the row cell data.

Parameters

`{Object} cell`	Cell.
`{Object} data`	Data.
`{Object} column`	Cell configuration.
`{Object} link`	Link.

`onUpdateItem(item)`

Record’s update event handler.

Parameters

`{Terrasoft.BaseViewModel} item`	Collection item.

`onDestroy(clear)`

Destroys the list and its components.

Parameters

`{Boolean} clear`	Clears the list and its components.

ConfigurationGridGenerator module JS

Advanced

`ConfigurationGridGenerator` is a module that generates the list configuration. Implemented in the `Terrasoft.controls.ConfigurationGridGenerator` class of the `UIv2` package. The `Terrasoft.controls.ConfigurationGridGenerator` class inherits from the `Terrasoft.ViewGenerator` class.

Methods

`addLinks`

Overloaded method of the `Terrasoft.ViewGenerator` class. Set to `Terrasoft.emptyFn` by default. Does not add links to the edited list.

`generateGridCellValue(config)`

Overloaded method of the `Terrasoft.ViewGenerator` class. Generates configuration of the value in the cell.

Parameters

`{Object} config`	Column configuration.

ConfigurationGridUtilities module JS

Advanced

`ConfigurationGridUtilities` is a module that contains methods that initialize the list row view model, handle the active record actions, and handle hotkeys. Implemented in the `Terrasoft.configuration.mixins.ConfigurationGridUtilities` class of the `Uiv2` package.

Properties

`currentActiveColumnName` `String`

The name of the currently selected column.

`columnsConfig` `String`

Column configuration.

`systemColumns` `Array`

Collection of system column names.

Methods

`onActiveRowAction(buttonTag, primaryColumnValue)`

Handles the active record's action click event.

Parameters

`{String} buttonTag`	Tag of the selected action.
`{String} primaryColumnValue`	ID of the active record.

`activeRowSaved(row, callback, scope)`

Handles the record save event.

Parameters

`{Object} row`	List row.
`{Function} [callback]`	The callback function.
`{Object} scope`	The scope of the callback function call.

`initActiveRowKeyMap(keyMap)`

Subscribes to active row's button click events.

Parameters

`{Array} keyMap`	Description of events.

`getCellControlsConfig(entitySchemaColumn)`

Returns the configuration of the list cell's edit elements.

Parameters

`{Terrasoft.EntitySchemaColumn} entitySchemaColumn`	List cell column.

`copyRow(recordId)`

Copies and adds a record to the list.

Parameters

`{String} recordId`	ID of the record to copy.

`initEditableGridRowViewModel(callback, scope)`

Initializes classes of the collection items from the list to edit.

Parameters

`{Function} callback`	The callback function.
`{Object} scope`	The scope of the callback function call.

`saveRowChanges(row, callback, scope)`

Saves the record.

Parameters

`{Object} row`	List row.
`{Function} [callback]`	The callback function.
`{Object} [scope]`	The scope of the callback function call.

BasePageV2 schema

Medium

`BasePageV2` is a base schema of a mini page. Implemented in the `NUI` package. This is a view model schema. Learn more about the schema properties in a separate article: Client schema.

Messages

Messages of a base mini page

Name	Mode	Direction	Description
`UpdatePageHeaderCaption`	Address	Publishing	Updates the page header.
`GridRowChanged`	Address	Subscription	Retrieves the ID of the selected list record when the record changes.
`UpdateCardProperty`	Address	Subscription	Changes the model parameter value.
`UpdateCardHeader`	Address	Subscription	Updates the add page header.
`CloseCard`	Address	Publishing	Closes the add page.
`OpenCard`	Address	Subscription	Opens the add page.
`OpenCardInChain`	Address	Publishing	Opens an add page chain.
`GetCardState`	Address	Subscription	Returns the add page status.
`IsCardChanged`	Address	Publishing	Informs of the add page change.
`GetActiveViewName`	Address	Publishing	Retrieves the name of the active view.
`GetMiniPageMasterEntityInfo`	Address	Subscription	Retrieves data about the master entity of the mini add page.
`GetPageTips`	Address	Subscription	Retrieves page tooltips.

`GetColumnInfo`	Address	Subscription	Retrieves column data.
`GetEntityColumn Changes`	Broadcasting	Publishing	Sends entity column data when the column changes.
`ReloadSectionRow`	Address	Publishing	Reloads the section row depending on the primary column value.
`ValidateCard`	Address	Subscription	Launches the add page validation.
`ReInitializeActions Dashboard`	Address	Publishing	Relaunches the action dashboard validation.
`ReInitializeActions Dashboard`	Address	Subscription	Updates the action dashboard configuration.
`ReloadDashboard Items`	Broadcasting	Publishing	Reloads the dashboard elements.
`ReloadDashboard ItemsPTP`	Address	Publishing	Reloads the dashboard elements on the current page.
`CanChangeHistory State`	Broadcasting	Subscription	Allows or prohibits changing the current history state.
`IsEntityChanged`	Address	Subscription	Returns the changed entity.
`IsDcmFilterColumn Changed`	Address	Subscription	Returns `true` if the filtered columns changed.
`UpdateParentLookup DisplayValue`	Broadcasting	Bidirectional	Updates the parent lookup record in accordance with the configuration.
`UpdateParentLookup DisplayValue`	Broadcasting	Bidirectional	States that data must be reloaded on the following launch.

The `Terrasoft.core.enums.MessageMode` enumeration represents the message modes, and the `Terrasoft.core.enums.MessageDirectionType` enumeration represents the message directions. Learn more about the `MessageMode` enumeration in the JS class library. Learn more about the `MessageDirectionType` enumeration in the JS class library.

Attributes

`IsLeftModulesContainerVisible` `BOOLEAN`

The visibility flag of `LeftModulesContainer`.

`IsActionDashboardContainerVisible` `BOOLEAN`

The visibility flag of `ActionDashboardContainer`.

`HasActiveDcm` `BOOLEAN`

The visibility flag of `DcmActionsDashboardContainer`.

`ActionsDashboardAttributes` `CUSTOM_OBJECT`

Custom attributes of `DcmActionsDashboardContainer`.

`IsPageHeaderVisible` `BOOLEAN`

The visibility flag of the page title.

`ActiveTabName` `TEXT`

Saves the name of the active tab.

`GridDataViewName` `TEXT`

The name of the `GridData` view.

`AnalyticsDataViewName` `TEXT`

The name of the `AnalyticsData` view.

`IsCardOpenedAttribute` `STRING`

Attribute of the add page body when the add page is displayed or hidden.

`IsMainHeaderVisibleAttribute` `STRING`

Attribute of the add page body when the main header is displayed or hidden.

`PageHeaderColumnNames` `CUSTOM_OBJECT`

The array of page header column names.

`IsNotAvailable` `BOOLEAN`

Flag that marks the page as not available.

`CanCustomize` `BOOLEAN`

Flag that marks the page as customizable.

`Operation` `ENUM`

Operations of the add page.

`EntityReloadScheduled` `BOOLEAN`

Flag that forces entity reload on the following launch.

The `Terrasoft.core.enums.DataValueType` enumeration represents the attribute data types. Learn more about the `DataValueType` enumeration in the JS class library.

Methods

`onDiscardChangesClick(callback, scope)`

Handles the [*Discard*] button click event.

Parameters

`{String} [callback]`	The callback function.
`{Terrasoft.BaseViewModel} [scope]`	The method execution scope.

`addChangeDataViewOptions(viewOptions)`

Adds switchpoint views to the drop-down list of the [*View*] button.

Parameters

`{Terrasoft.BaseViewModelCollection} viewOptions`	Items in the drop-down list of the [*View*] button.

`addSectionDesignerViewOptions(viewOptions)`

Adds the [*Open section wizard*] item to the drop-down list of the [*View*] button.

Parameters

`{Terrasoft.BaseViewModelCollection} viewOptions`	Items in the drop-down list of the [*View*] button.

`getReportFilters()`

Returns the query filter collection.

`initPageHeaderColumnNames()`

Initializes the column names of the page header.

`getParameters(parameters)`

Returns the values of `ViewModel` parameters.

Parameters

`{Array} parameters`	Parameter names.

`setParameters(parameters)`

Sets the `ViewModel` parameters.

Parameters

`{Object} parameters`	Parameter values.

`getLookupModuleId()`

Returns the ID of the lookup page module.

`onReloadCard(defaultValues)`

`ReloadCard` message handler. Reloads the data of the add page entity.

Parameters

`{Object[]} defaultValues`	Array of default values.

`onGetColumnInfo(columnName)`

Returns the column information.

Parameters

`{String} columnName`	The name of the column.

`getTabsContainerVisible()`

Returns the visibility status of container tabs.

`getPrintMenuItemVisible(reportId)`

Returns the visibility status of the [*Print*] button's drop-down list items (i. e., reports).

Parameters

`{String} reportId`	Report ID.

`getDataViews()`

Retrieves the section view.

`runProcess(tag)`

Runs a business process using the start button for global business processes.

Parameters

`{Object} tag`	ID of the business process schema.

`runProcessWithParameters(config)`

Runs a business process with parameters.

Parameters

`{Object} config`	The configuration object.

`onCanBeDestroyed(cacheKey)`

Checks for unsaved data. Edit the cached `config.result` if the data was changed but not saved.

Parameters

`{String} cacheKey`	The key of the cached configuration object.

`onValidateCard()`

Displays an error message if the add page is invalid.

BaseFieldsDetail schema JS

Medium

`BaseGridDetailV2` is a base schema of a field detail. Implemented in the `BaseFinance` package of Financial Services Creatio product lineup. This is a view model schema. Learn more about the schema properties in a separate article: Client schema.

Messages

Messages of a base field detail

Name	Mode	Direction	Description
`LookupInfo`	Address	Subscription	Returns data about the lookup.
`UpdateCardProperty`	Address	Publishing	Changes the value of the record page model.
`CardSaved`	Broadcasting	Subscription	Listens to the parent page's save event.
`IsCardChanged`	Address	Publishing	Notifies of the add page changes.

FileDetailV2 schema JS

Medium

`FileDetailV2` is a base schema of an [*Attachments*] type detail. Implemented in the `UIv2` package. This is a view model schema. Learn more about the schema properties in a separate article: Client schema.

Attributes

`SchemaCardName` TEXT

Saves the record page name.

`parentEntity` `CUSTOM_OBJECT`

The parent entity.

The `Terrasoft.core.enums.DataValueType` enumeration represents the attribute data types. Learn more about the `DataValueType` enumeration in the JS class library.

Methods

`getShowPreviewSettingsValue()`

Retrieves the value of `ShowPreview` system settings.

`initParentEntity()`

Initializes the parent entity.

`itemsRendered()`

Handles the `itemsRendered` event raised in the `ContainerList` component.