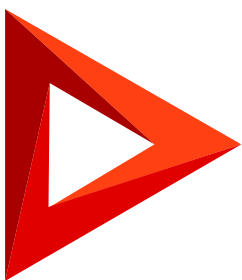


# Work with details

Version 7.17



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

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

# Table of Contents

<b>How to hide menu commands of the detail with list</b>	<b>5</b>
<b>Adding details</b>	<b>6</b>
Overview	6
<b>Deleting a detail</b>	<b>7</b>
Introduction	7
<b>Creating a detail in wizards</b>	<b>7</b>
Introduction	7
Case description	8
Case implementation algorithm	8
<b>Adding an edit page detail</b>	<b>12</b>
General provisions	12
General procedure for adding an edit page detail to an existing section	12
Case description	13
<b>Adding a detail with an editable list</b>	<b>22</b>
Introduction	22
Case description	23
Source code	23
Case implementation algorithm	23
Detail wizard, Section wizard and details with editable lists	30
<b>Creating a detail with selection from lookup</b>	<b>31</b>
Introduction	31
Case description	32
Source code	32
Case implementation algorithm	32
<b>Adding multiple records to a detail</b>	<b>41</b>
Introduction	41
Case description	42
Source code	42
Case implementation algorithm	42
<b>Creating a custom detail with fields</b>	<b>45</b>
Introduction	45
Case description	46
Case implementation algorithm	46
<b>Advanced settings of a custom detail with fields</b>	<b>51</b>
Introduction	51
Adding custom styles	51

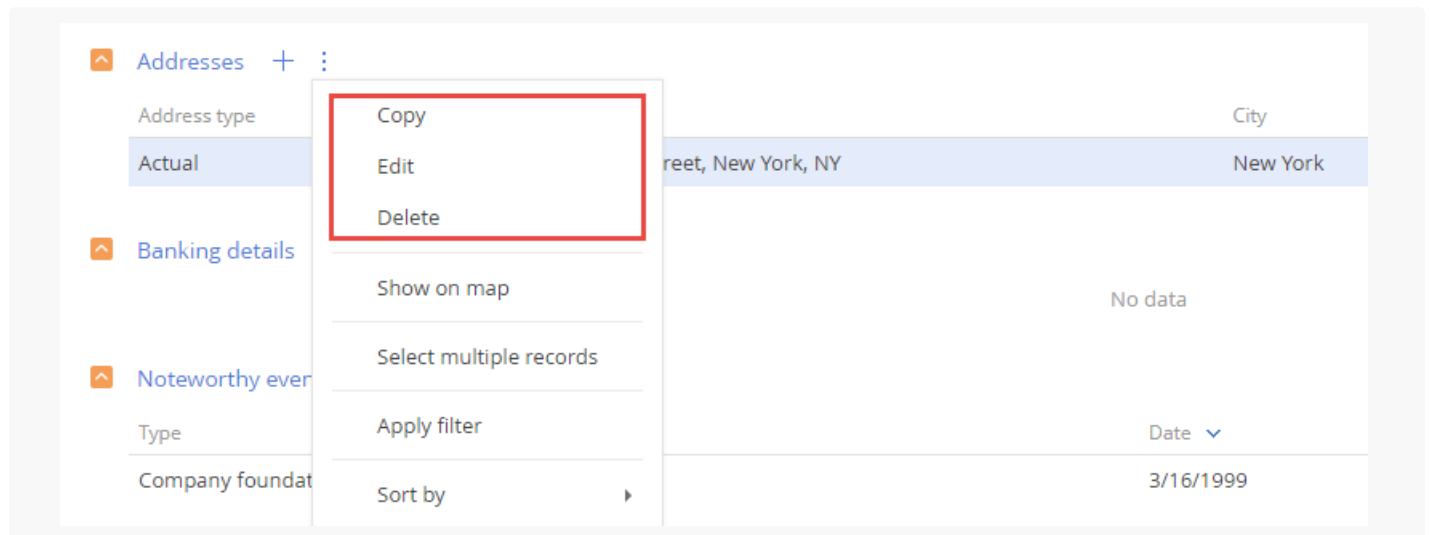
Adding additional custom logic for detail records	54
Adding a virtual record	56
<b>Adding the [Attachments] detail</b>	<b>57</b>
Introduction	57
Case description	57
Case implementation algorithm	58
<b>Displaying additional columns on the [Attachments] tab</b>	<b>68</b>
Introduction	68
Case description	69
Case implementation algorithm	69

# How to hide menu commands of the detail with list

 Medium

The [ *Copy* ], [ *Edit* ], and [ *Delete* ] commands in a detail menu are used to manage records in the detail list (Fig. 1).

Fig. 1. The [ *Addresses* ] detail menu



To hide menu detail commands:

1. Create a replacing schema of the detail list. For example, for the [ *Addresses* ] detail of the account edit page it will be the [ *Account addresses detail* ]. The procedure for creating a replacing client schema is covered in the ["Create a client schema"](#) article.
2. Add the following source code to the schema:

```
define("AccountAddressDetailV2", [], function() {
  return {
    entitySchemaName: "AccountAddress",
    methods: {
      // Disabling the [Copy] command
      getCopyRecordMenuItem: Terrasoft.emptyFn,
      // Disabling the [Edit] command
      getEditRecordMenuItem: Terrasoft.emptyFn,
      // Disabling the [Delete] command
      getDeleteRecordMenuItem: Terrasoft.emptyFn
    },
    diff: /**SCHEMA_DIFF*/[]/**SCHEMA_DIFF*/
  };
});
```

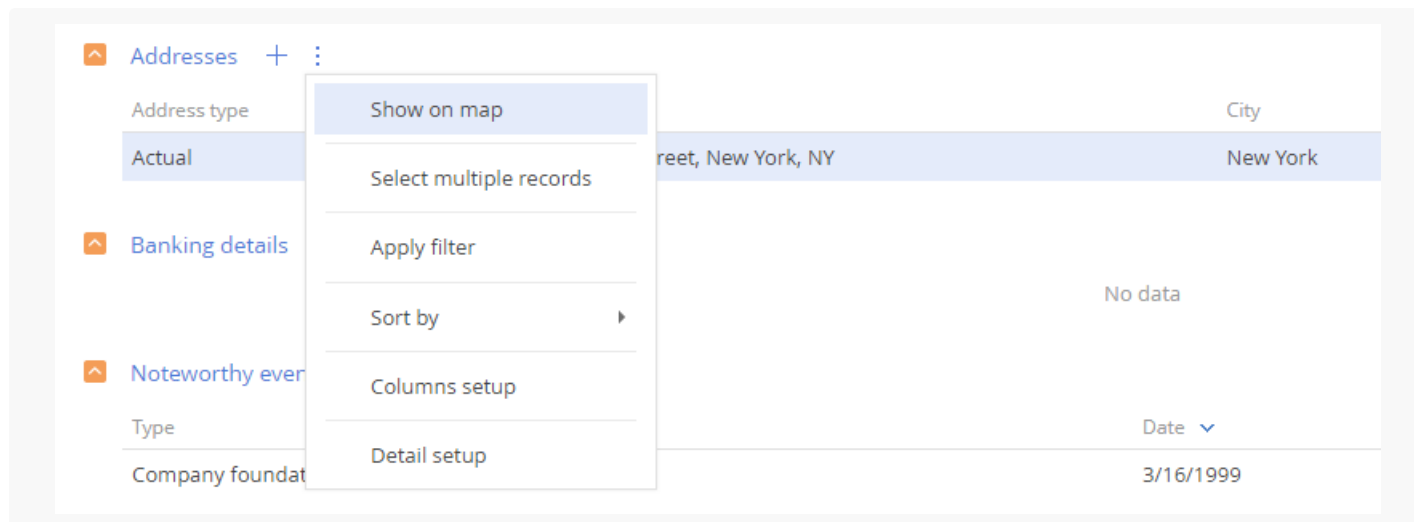
```
});
```

3. Save the changes.

4. Refresh the browser page.

As a result, commands will be disabled in the detail menu (Fig.2).

Fig. 2. [ Addresses ] detail menu without menu commands



## Adding details

 Easy

### Overview

Details are the elements of the section record edit page that display supplemental data for a primary section object. The details displayed on the tabs of section edit page in the tabs container.

There are four main types of details:

1. A detail with adding page is a standard Creatio detail. It can be created manually or added to the section via the [detail wizard](#). More information about detail creation can be found in the [“Adding an edit page detail”](#) and [“Creating a detail in wizards”](#) articles.
2. A detail with editable list different from the standard detail. The data can be added, deleted and modified directly in the detail. More information about creation of the detail with editable list can be found in the [“Adding a detail with an editable list”](#) article.
3. A detail with selection from lookup – data are selected from a lookup displayed in the modal window. More information about detail creation can be found in the [“Creating a detail with selection from lookup”](#) article.
4. A detail with edit fields – data are filled in and edited in the detail data fields. More information about detail creation can be found in the [“Creating a custom detail with fields”](#) and [“Advanced settings of a custom detail with fields”](#) article.

More information about the details of each type is described in the “[Detail](#)” article of the “[Application interface and structure](#)” section.

### Attention.

Main schemas, classes, methods and properties of detail functions are described in the “[The BaseDetailV2 schema](#)” article of the “[Controls](#)” section.

# Deleting a detail



Medium

## Introduction

To delete a custom detail in Creatio, you need access to the system configuration tools and its database.

### Attention.

Before you delete a custom detail, unlock the corresponding detail in the SVN storage.

First, delete the database records. Use the following script to delete a detail:

```
DECLARE @Caption nvarchar(max);
SET @Caption = 'ToDelete';
DECLARE @UID UNIQUEIDENTIFIER;
select @UID = EntitySchemaUID from SysDetail
where Caption = @Caption
delete from SysDetail where EntitySchemaUID = @UID
```

Replace the “ToDelete” value with the detail schema name, which you can view in the [ *Advanced settings* ] of the System Designer. After deleting data from the database, delete the custom detail schema in the [ *Configuration* ] section. The [ *Advanced settings* ] in the System Designer allow deleting the object of the deleted detail.

# Creating a detail in wizards



Medium

## Introduction

Detail is an element of the section record edit page, designed to display additional data related to the main section object. Details are displayed on edit page tabs. The difference between details and section records is that details are stored in a separate object, and the records in this database object are usually associated with the main section record entity with the “many-to-one” ratio. Please refer to the “[Detail](#)” article in the “[Application interface](#)

[and structure](#)” section for more information.

Details are standard Creatio elements and can be added to the section by using a [detail wizard](#) or a [section wizard](#).

The general outline for adding a detail with an “add” page to an existing system section:

1. Create a detail object schema
2. Create a schema of a client detail list module and the schema of a detail edit page
3. Implement the detail on the section record edit page using the section wizard.

## Case description

Create a custom [ *Contact's ID* ] detail in the [ *Contacts* ] section. The detail must display the contact's ID number and the document number. One contact may have several ID's.

## Case implementation algorithm

### 1. Creating a detail object schema

Learn more about adding object schema columns in the “[Create the entity schema](#)” article.

Create an object schema with the following parameters (Fig. 1):

- [Title] – “Contact Identity Card”.
- [Name] — “UsrContactIdentityCard”.
- [Package] — “Custom” (or a different custom package).
- [Parent object] – “Base object”, implemented in the `Base` package.

Add three columns in the object structure. Column properties are listed in Table 1.

Table 1. – Column properties of the UsrRegDocument detail object schema

Title	Name	Data Type	Lookup
Series	UsrSeries	Text (50 characters) Text (50 characters)	–
Number	UsrNumber	Text (50 characters) Text (50 characters)	–
Contact	UsrContact	Lookup	Contact

Publish the schema to apply changes.

#### Note.



Add columns in the detail wizard.

## 2. Creating a schema of the detail list client module and a schema of the detail edit page.

### Attention.

If the development needs to be carried out in a custom package, it needs to be specified in the [ *Current package* ] system setting. Otherwise, the detail wizard will not be able to save the changes to the package used for development.

To create a new detail in a wizard, go to the [ *Detail wizard* ] section in the [ *System setup* ] group of the system designer.

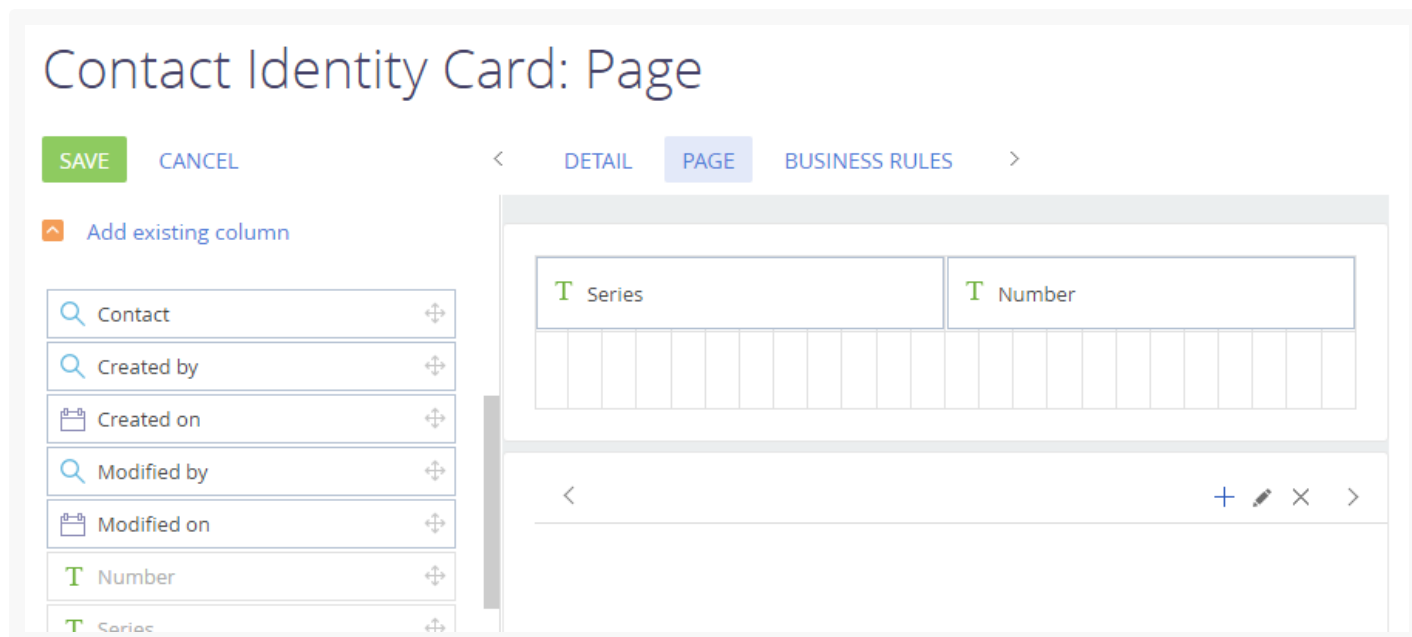
On the [ *DETAIL* ] step, specify the title of the detail and select the main detail object (Fig. 1).

Fig. 1. The [ *DETAIL* ] step in the detail wizard

Figure 1 shows the 'DETAIL' step of the 'Contact Identity Card: Detail' wizard. The title is 'Contact Identity Card: Detail'. The navigation bar includes 'SAVE', 'CANCEL', '<', 'DETAIL' (active), 'PAGE', 'BUSINESS RULES', and '>'. The 'Title' and 'Object' fields are both set to 'Contact Identity Card'.

Arrange the required detail columns on the [ *PAGE* ] step.

Fig. 2. The [ *PAGE* ] step in the detail wizard



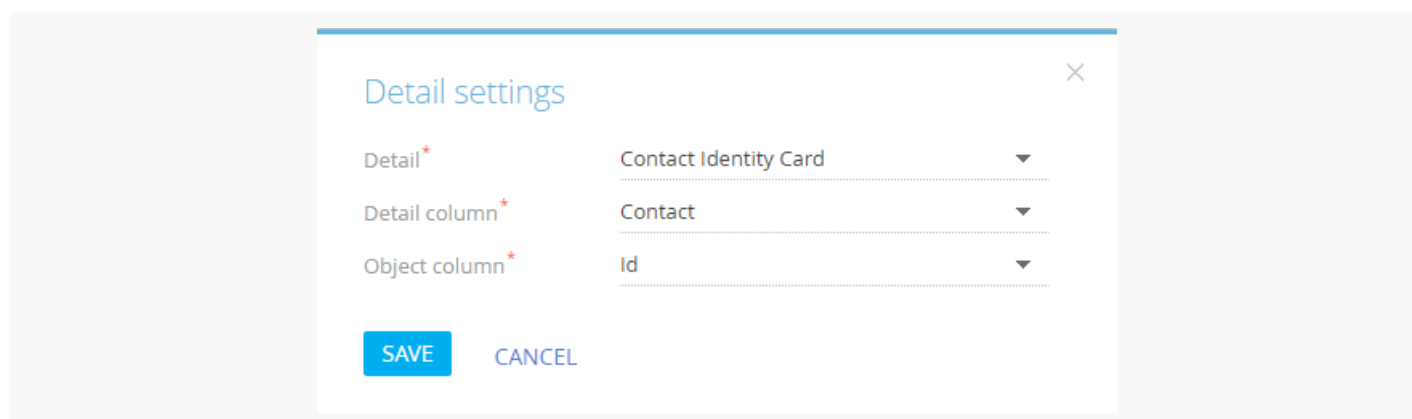
Save the detail when the setup is done.

As a result, the custom package will have a schema of the detail list client module and a schema of the detail edit page.

### 3. Implement the detail on the section record edit page using the detail wizard

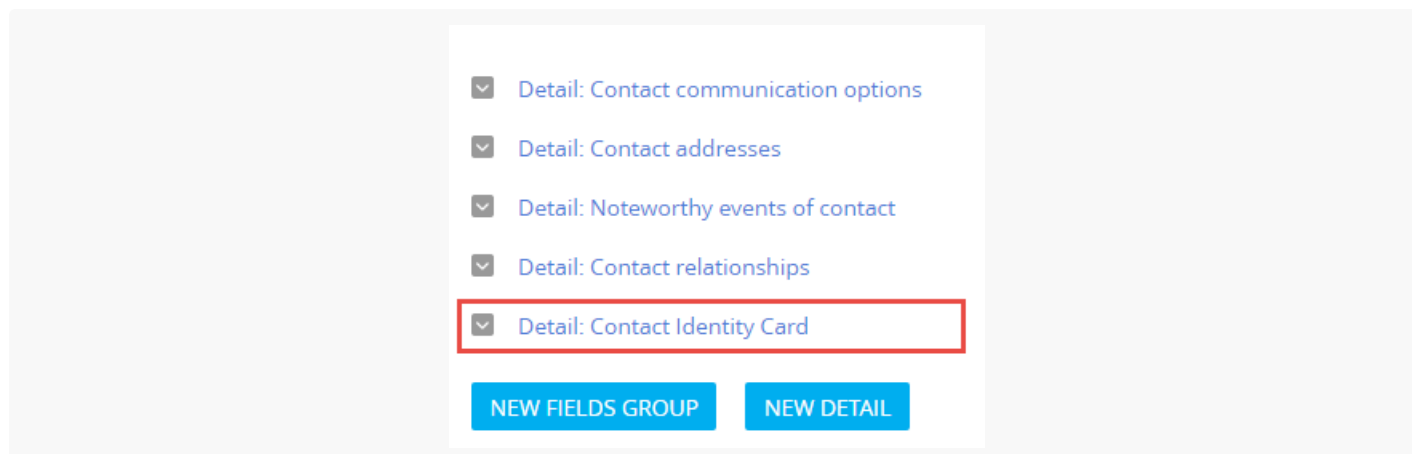
Open the detail wizard in the [ *Contacts* ] section, and select [ *NEW DETAIL* ] on the [ *PAGE* ] step. In the opened window, select the [ *Contact's ID* ] detail and configure the connection between detail object columns and the section object (Fig. 3).

Fig. 3. Detail properties setup



The detail will be displayed in the section record page constructor (Fig. 4).

Fig. 4. A detail in the section record page constructor

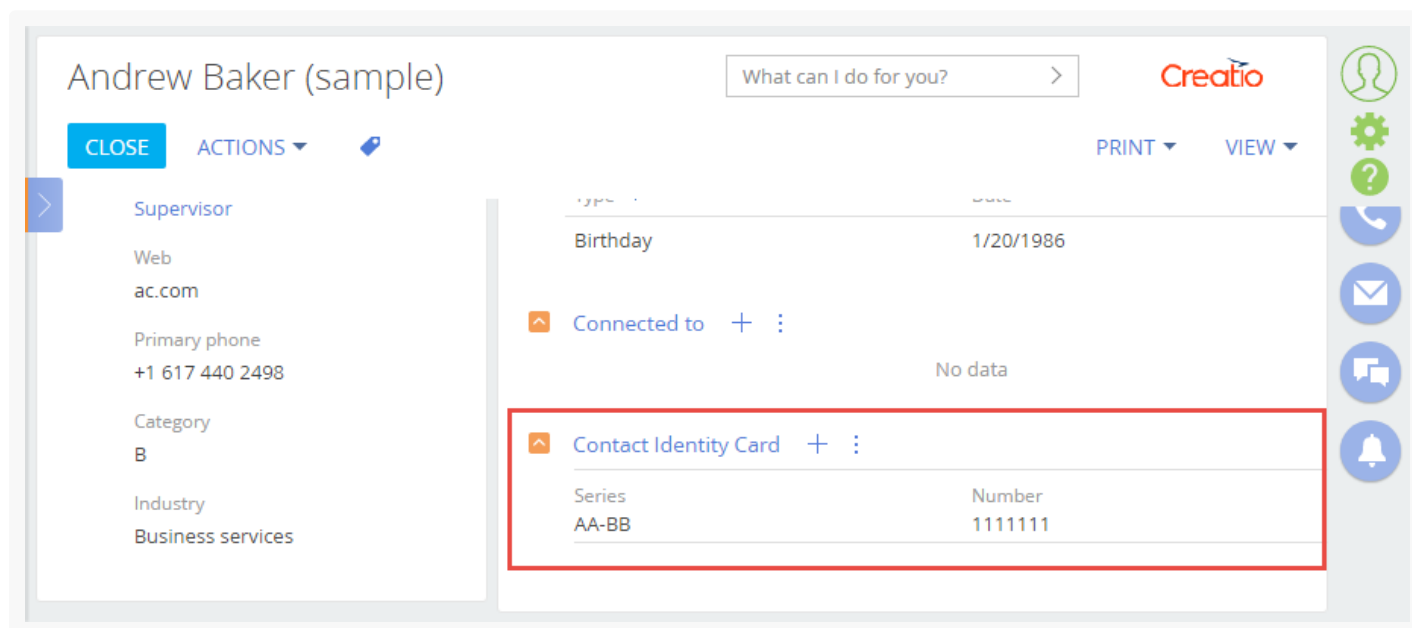


Save the changes when the section record page setup is done.

As a result, the custom package will have a replacing client module of a section page and a schema of the section record edit page.

Upon refreshing, the detail will be displayed on a record edit page.

Fig. 5. Case result



### Attention.

It is necessary to configure the columns in a detail menu, and add a few records to see the result.

Fig. 6. Adding a record to a detail

# Adding an edit page detail

 Advanced

## General provisions

Details are special elements of section edit pages, which display additional data that is not stored in the fields of the section primary object. The details are displayed on the edit page tabs in the tab container.

There are four basic types of details:

- details with a record edit page;
- details with edit fields;
- details with editable list;
- details with lookup selection.

For more information on details, please see the "[Details](#)" article in the "[Application interface and structure](#)" section.

An edit page detail is a standard Creatio detail that can be added to sections using the [Detail Wizard](#). Below is an example of adding an edit page detail without the use of the Detail Wizard.

## General procedure for adding an edit page detail to an existing section

To add a custom edit page detail:

1. Create a detail object schema.
2. Create a detail schema.
3. Create a detail edit page schema.
4. Set up the detail in a replacing section edit page schema.
5. Register links between object, detail and detail page schemas, using special SQL queries to system tables.
6. Set up the detail fields.

**Note.**

To bind custom schemas, make changes to the SysModuleEdit, SysModulentity and SysDetail system tables in the Creatio database, using SQL queries.

Be extremely careful when composing and executing SQL queries to the database. Executing an invalid query may damage existing data and disrupt system operation.

## Case description

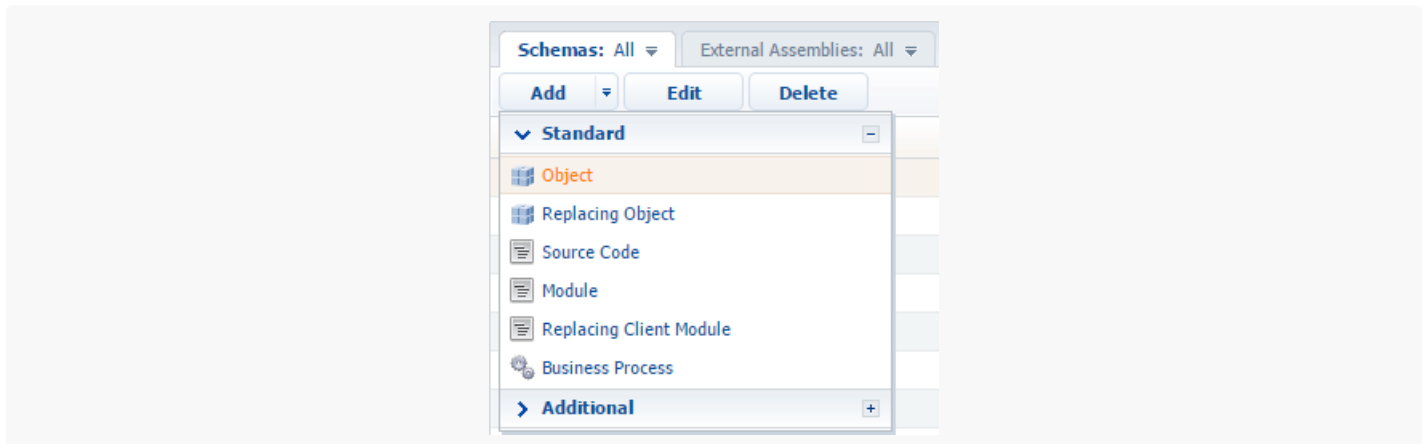
Create a custom [ *Couriers* ] detail in the [ *Orders* ] section. The detail must contain the list of couriers for the current order.

## Case implementation algorithm

### 1. Create detail object schema

In the settings mode, go to the [ *Configuration* ] section, open the [ *Schemas* ] tab and execute the [ *Add* ] > [ *Object* ] menu command (Fig. 1).

Fig. 1. Adding a detail object schema



In the opened [Object Designer](#), fill out the properties of the detail object schema, as shown on Fig. 2.

Fig. 2. Setup of detail object schema properties

The screenshot shows the 'Properties' dialog for a column. The 'General' section contains fields for Name (CourierInOrder), Title (Courier in order), and Package (CustomOrderDetails). The 'Inheritance' section is highlighted with a red box and contains a 'Parent object' dropdown set to 'Base object ( Base )'. Below it is a 'Replace parent' checkbox. The 'Access rights' section has checkboxes for 'Operations' and 'Records'.

Add a lookup column [ *Order* ], which will connect the detail object with the section object, and a lookup [ *Contact* ] column where the courier's contact will be stored. Both columns must be required. Column properties setup is shown on Fig. 3 and Fig. 4.

Fig. 3. Specifying the properties of the [ *Order* ] column

The screenshot shows the 'Properties' dialog for a column. The 'General' section has fields for Title (Order) and Name (Order). The 'String' section has a 'Multi-string text' checkbox. The 'Lookup' section is highlighted with a red box and contains a 'Lookup' dropdown set to 'Order', with 'Cascade connection' and 'List' checkboxes below it. The 'Behavior' section is also highlighted with a red box and contains a 'Required' dropdown set to 'Application Level', a 'Default value' field set to '(No)', and a 'Make copy' checkbox checked.

Fig. 4. Specifying the properties of the [ *Contact* ] column

The screenshot shows the 'Properties' window for a detail schema. The 'General' section contains 'Title' and 'Name' both set to 'Contact'. The 'String' section has 'Multi-string text' unchecked. The 'Lookup' section has 'Lookup' set to 'Contact' (highlighted with a red box) and 'Cascade connection' unchecked. The 'Behavior' section has 'Required' set to 'Application Level' (highlighted with a red box), 'Default value' set to '(No)', and 'Make copy' checked.

Object schema must be saved and published.

## 2. Create detail schema

In the settings mode, go to the [ *Configuration* ] section, open the [ *Schemas* ] tab and execute the [ *Add* ] > [ *Module* ] menu command (Fig. 1).

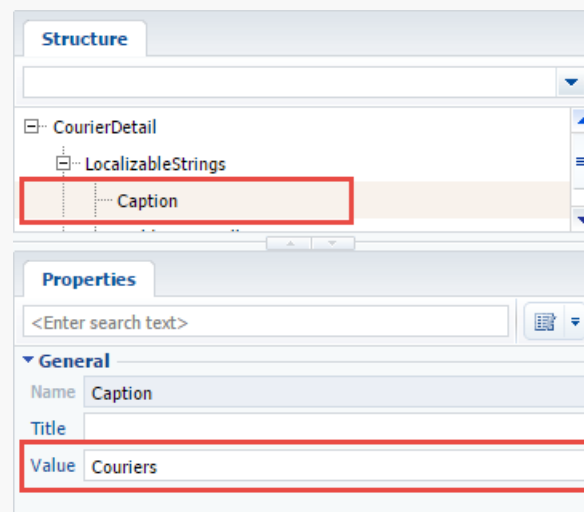
The new module must inherit the functionality of the base detail schema with list `BaseGridDetailV2`, which is available in the `NUI` package. To do this, specify this schema as the parent for the created detail schema (Fig. 5). The rest of the properties must be set up as shown on Fig. 5. In the [ *Package* ] property, the system will specify the current package name.

Fig. 5. Detail schema properties

The screenshot shows the 'Properties' window for a detail schema. The 'General' section contains 'Title' set to '"Courier in order" detail schema', 'Name' set to 'CourierDetail', and 'Package' set to 'CustomOrderDetails'. The 'Inheritance' section has 'Parent object' set to 'Base detail schema' (highlighted with a red box) and 'Replace parent' unchecked.

Set the `couriers` value for the [ *Caption* ] localizable string of the detail schema (Fig. 6). The [ *Caption* ] string contains detail title, shown on the edit page.

Fig. 6. Setting the value of the [ *Caption* ] string



Below is the detail schema source code, which must be added to the [ *Source code* ] section of the client module designer. The code defines the schema name and its connection to the object schema.

```
define("CourierDetail", [], function() {
    return {
        // Detail object schema name.
        entitySchemaName: "UsrCourierInOrder",
        details: /**SCHEMA_DETAILS*/{}/**SCHEMA_DETAILS*/,
        methods: {},
        diff: /**SCHEMA_DIFF*/[/**SCHEMA_DIFF*/
    ];
});
```

Save the detail schema to apply the changes.

### 3. Create detail edit page schema

In the settings mode, go to the [ *Configuration* ] section, open the [ *Schemas* ] tab and execute the [ *Add* ] > [ *Module* ] menu command (Fig. 1).

The created detail edit page schema must inherit the functionality of base page schema `BasePageV2`, which is available in the `NUI` package. To do this, specify this schema as parent (Fig. 7). The rest of the properties must be set up as shown on Fig. 7. In the [ *Package* ] property, the system will specify the current package name.

Fig. 7. Detail edit page schema properties



The screenshot shows a 'Properties' panel with a search bar at the top. Below it, the 'General' section contains fields for 'Title' (Detail edit page), 'Name' (CourierDetailPage), and 'Package' (CustomOrderDetails). The 'Inheritance' section below it has a 'Parent object' dropdown menu set to 'Base card schema', which is highlighted with a red rectangular border. A 'Replace parent' checkbox is visible at the bottom of the Inheritance section.

To set up fields displayed on the detail edit page, add the following code to the [ *Source code* ] section of the client module designer. In this code, in the diff array, configuration objects of metadata for adding, setting location and binding the order and courier fields are specified.

```
define("CourierDetailPage", [], function() {
  return {
    // Detail object schema name.
    entitySchemaName: "UsrCourierInOrder",
    details: /**SCHEMA_DETAILS*/{}/**SCHEMA_DETAILS*/,
    // Array with modifications.
    diff: /**SCHEMA_DIFF*/[
      // Metadata for adding the [Order] field.
      {
        "operation": "insert",
        //Field name.
        "name": "Order",
        "values": {
          // Field position setup on the edit page.
          "layout": {
            "colSpan": 12,
            "rowSpan": 1,
            "column": 0,
            "row": 0,
            "layoutName": "Header"
          },
          // Binding to the [Order] column of the object schema.
          "bindTo": "UsrOrder"
        },
        "parentName": "Header",
        "propertyName": "items",
        "index": 0
      },
      // Metadata for adding the [Contact] field.
      {
        "operation": "insert",
        //Field name.
        "name": "Contact",
```

```

    "values": {
      // Field position setup on the edit page.
      "layout": {
        "colSpan": 12,
        "rowSpan": 1,
        "column": 12,
        "row": 0,
        "layoutName": "Header"
      },
      // Binding to the [Contact] column of the object schema.
      "bindTo": "UsrContact"
    },
    "parentName": "Header",
    "propertyName": "items",
    "index": 1
  }
]/**SCHEMA_DIFF*/,
methods: {},
rules: {}
};
});

```

Save the detail edit page schema to apply the changes.

#### 4. Set up detail in replacing section edit page schema

To display the detail on the order edit page, first create a replacing client module, and specify the order edit page `OrderPageV2` (located in the Order package) as the parent (Fig. 8). For more information on creating replacing schemas, please see the ["Create a client schema"](#) article.

Fig. 8. Order edit page replacing schema properties

The screenshot shows a 'Properties' window with a search bar at the top. Below it, the 'General' section contains fields for 'Title' (Order edit page), 'Name' (OrderPageV2), and 'Package' (CustomerOrderDetails). The 'Inheritance' section is expanded, and the 'Parent object' dropdown is highlighted with a red rectangle, showing 'Order edit page ( Order )'. The 'Replace parent' checkbox is checked.

To display the [ *Couriers* ] detail on the [ *Delivery* ] tab of the order edit page, add the following source code. In the `details` section, a new `CourierDetail` model will be defined, and its location on the edit page will be specified in the modification section of the `diff` array.

```

define("OrderPageV2", [], function() {

```

```

return {
    // Name of the edit page object schema.
    entitySchemaName: "Order",
    // List of edit page details being added.
    details: /**SCHEMA_DETAILS*/{
        // [Couriers] detail setup.
        "CourierDetail": {
            // Detail schema name.
            "schemaName": "UsrCourierDetail",
            // Detail object schema name.
            "entitySchemaName": "UsrCourierInOrder",
            // Filtering contacts for current order only.
            "filter": {
                // Detail object schema column.
                "detailColumn": "UsrOrder",
                // Section object schema column.
                "masterColumn": "Id"
            }
        }
    }/**SCHEMA_DETAILS*/,
    // Array of modifications.
    diff: /**SCHEMA_DIFF*/[
        // Metadata for adding the [Couriers] detail.
        {
            "operation": "insert",
            // Detail name.
            "name": "CourierDetail",
            "values": {
                "itemType": 2,
                "markerValue": "added-detail"
            },
            // Parent container ([Delivery] tab).
            "parentName": "OrderDeliveryTab",
            // Container where detail is located.
            "propertyName": "items",
            // Index in the list of added elements.
            "index": 3
        }
    ]/**SCHEMA_DIFF*/,
    methods: {},
    rules: {}
};
});

```

To apply the changes, the replacing page schema must be saved.

After this, the detail will be displayed on the edit page of the [ *Orders* ] section. New records cannot be added to the detail until the connections between the detail schemas are registered.

## 5. Register connections between the schemas using SQL queries to system tables

To register connection between a detail object schema and detail schema, execute the following SQL query.

```
DECLARE
-- Schema name of the created detail.
@DetailSchemaName NCHAR(100) = 'CourierDetail',
-- Detail title.
@DetailCaption NCHAR(100) = 'Couriers',
--Schema name of the object, to which the detail is bound.
@EntitySchemaName NCHAR(100) = 'CourierInOrder'

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)
)
```

To register connection between the detail object schema and edit page schema, execute the following SQL query.

```
DECLARE
-- Schema name of the detail page
@CardSchemaName NCHAR(100) = 'CourierDetailPage',
-- Object schema.
@EntitySchemaName NCHAR(100) = 'CourierInOrder',
-- Detail page caption.
@PageCaption NCHAR(100) = 'Page of detail "Courier In Order"',
-- Empty string.
@Blank NCHAR(100) = ''

INSERT INTO SysModuleEntity(
    ProcessListeners,
    SysEntitySchemaUid
)
VALUES(
    0,
```

```

        (SELECT TOP 1 UIId
        FROM SysSchema
        WHERE Name = @EntitySchemaName
        )
    )

    INSERT INTO SysModuleEdit(
        SysModuleEntityId,
        UseModuleDetails,
        Position,
        HelpContextId,
        ProcessListeners,
        CardSchemaUIId,
        ActionKindCaption,
        ActionKindName,
        PageCaption
    )
    VALUES (
        (SELECT TOP 1 Id
        FROM SysModuleEntity
        WHERE SysEntitySchemaUIId = (
            SELECT TOP 1 UIId
            FROM SysSchema
            WHERE Name = @EntitySchemaName
        )
        ),
        1,
        0,
        @Blank,
        0,
        (SELECT TOP 1 UIId
        FROM SysSchema
        WHERE name = @CardSchemaName
        ),
        @Blank,
        @Blank,
        @PageCaption
    )

```

### Attention.

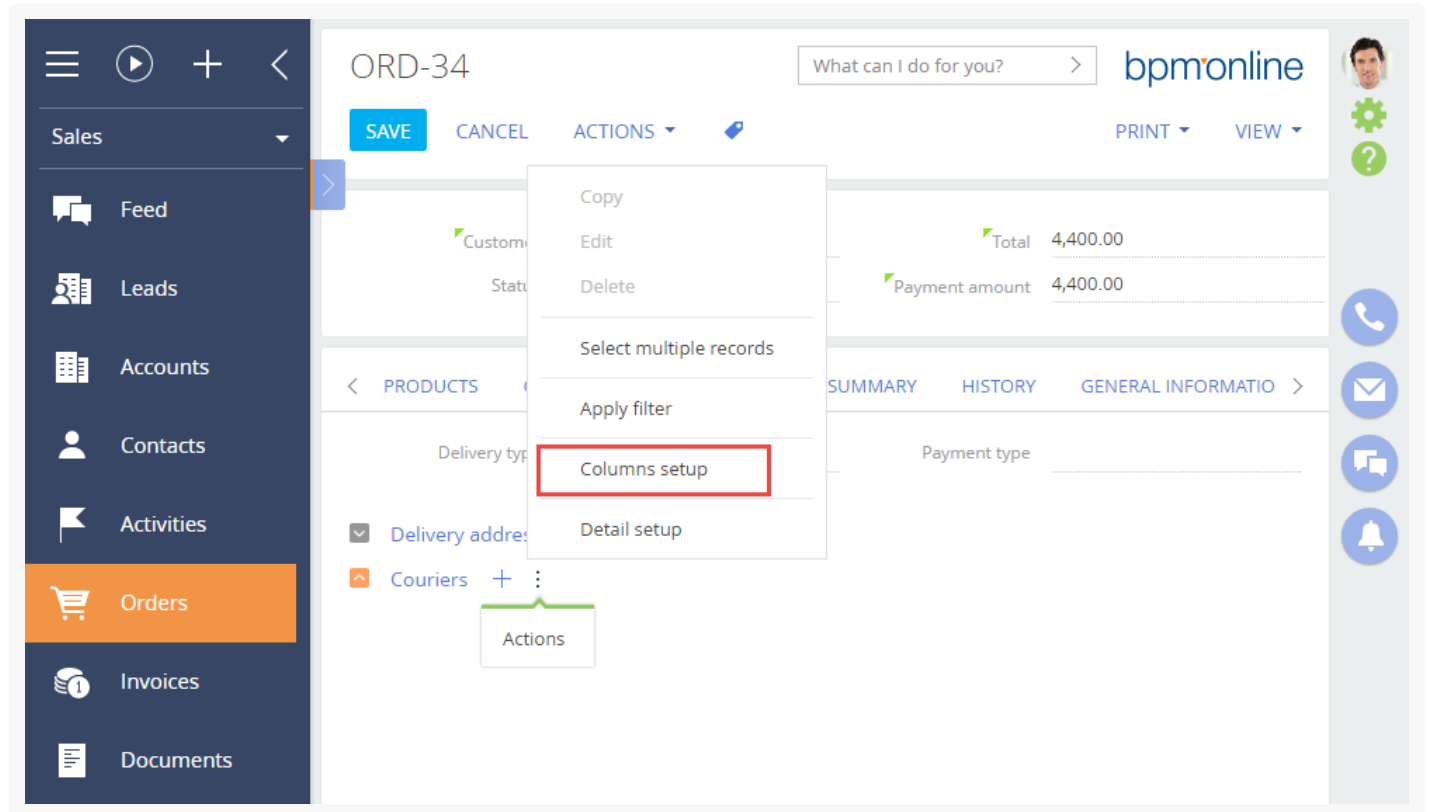
To apply these changes on the application level, restart the application site in IIS, or compile the application in the [ *Configuration* ] section.

## 6. Set up detail list columns

At this stage, the detail is completely operational, however contacts are not displayed on the detail, because the

detail list does not have displayed columns specified for it. Go to the detail menu and set up columns to display (Fig. 9).

Fig. 9. Detail actions menu



## Adding a detail with an editable list

 **Advanced**

### Introduction

Details are elements of section edit pages. Details display records bound to the current section record by a lookup field. These can be, for example, records from other sections, whose primary objects contain lookup columns linked to the primary object of the current section. Details are displayed on tabs of section edit pages.

More information about details is available in the “[Details](#)” article of the “[Application interface and structure](#)” section.

#### **Attention.**

A detail with an editable list is not a standard Creatio detail and can not be added to sections using the [Detail Wizard](#).

To add a custom detail with editable list to an existing section:

1. Create a detail object schema.
2. Create and configure the detail schema.
3. Set up the detail in the section edit page replacing schema.
4. Set up detail fields.

## Case description

Create a custom [ *Courier services* ] detail in the [ *Orders* ] section. The detail should display a list of courier services for the current order.

## Source code

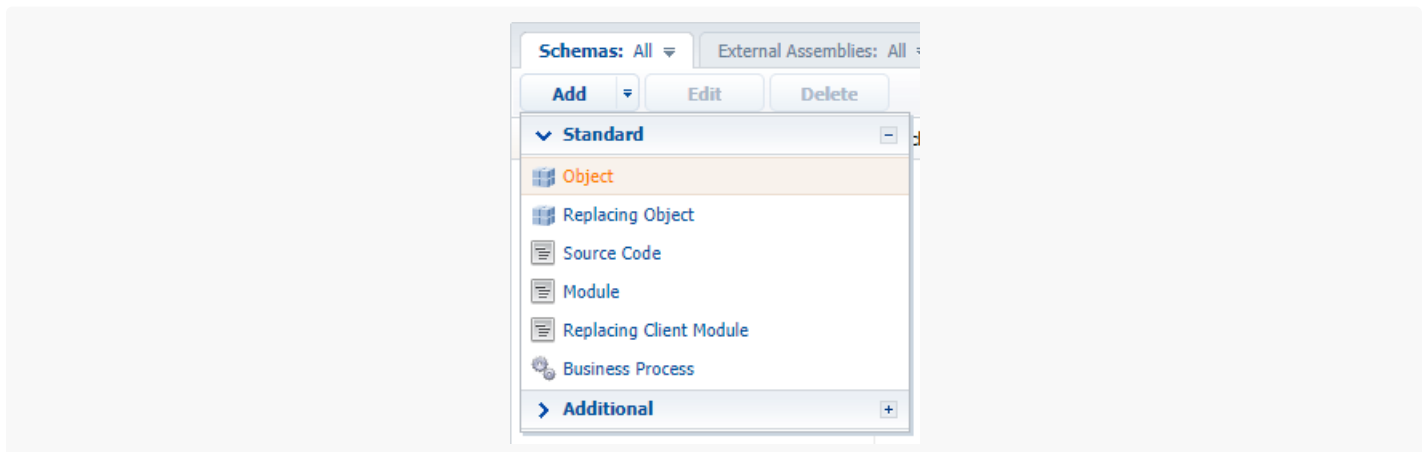
You can download the package with case implementation using the following [link](#).

## Case implementation algorithm

### 1. Creating a detail object schema.

Perform the [ *Add* ] - [ *Object* ] action on the [ *Schemas* ] tab of the [ *Configuration* ] section.

Fig. 1. Adding a detail object schema



For the created object schema (Fig. 2) specify following:

- [Name] - "UsrCourierService"
- [Title] - "CourierService"
- [Parent object] - [Base object].

Fig. 2. Setup of detail object schema properties

The screenshot shows the 'Properties' window for a column. The 'General' section is expanded, showing the following fields:

- Name:** UsrCourierService
- Title:** CourierService
- Package:** sdkCreateDetailWithEditableGrid

The 'Inheritance' section is also expanded, showing:

- Parent object:** Base object ( Base )

The 'Behavior' section is expanded, showing:

- Allow records deactivation:** ☐

The 'Access rights' section is expanded, showing:

- Operations:** ☐
- Records:** ☐

Add the [ *Order* ] lookup column to the object schema establishing connection with the [ *Orders* ] section and the [ *Account* ] lookup column containing the account carrying out delivery for this order. Mark both columns as “required” to avoid adding empty records. Column setup is shown in fig. 3 and fig. 4.

Fig. 3. The [ *Order* ] column properties setup

The screenshot shows the 'Properties' window for a column. The 'General' section is expanded, showing the following fields:

- Title:** Order
- Name:** UsrOrder

The 'String' section is expanded, showing:

- Multi-line text:** ☐

The 'Lookup' section is expanded, showing:

- Lookup:** Order
- Cascade connection:** ☐
- List:** ☐

The 'Behavior' section is expanded, showing:

- Required:** Application Level
- Default value:** (No)
- Make copy:** ☒

Fig. 4. The [ *Account* ] column properties setup



**Properties**

<Enter search text>

**General**

Title: Account

Name: UsrAccount

**String**

Multi-line text: ☐

**Lookup**

Lookup: Account

Cascade connection: ☐

List: ☐

**Behavior**

Required: Application Level

Default value: (No)

Make copy: ☒

Save and publish the object schema.

## 2. Creating a detail schema.

Run the [ *Add* ] - [ *Module* ] menu command on the [ *Schemas* ] tab of the [ *Configuration* ] section (fig. 1).

The created module should inherit the `BaseGridDetailV2` base detail list schema functions (specify it as the parent schema) defined in the `NUI` package.

Specify other properties (Fig. 5):

- [Name] - "UsrCourierServiceDetail"
- [Title] - "Courier Service in Order detail schema".

Fig. 5. Detail schema properties

**Properties**

<Enter search text>

**General**

Title: Courier Service in Order detail schema

Name: UsrCourierServiceDetail

Package: sdkCreateDetailWithEditableGrid

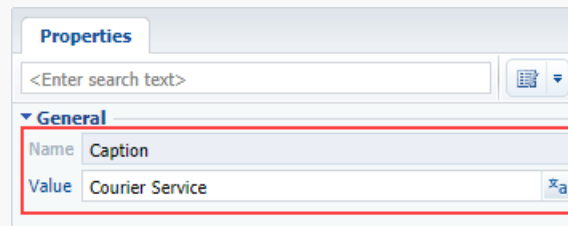
**Inheritance**

Parent object: Base schema - Detail with list ( NUI )

Replace parent: ☐

Set the [ *Courier Service* ] value for the [ *Caption* ] localizable string of the detail list schema (fig. 6). The [ *Caption* ] localizable string stores the detail caption, displayed on the edit page.

Fig. 6. Setting the [ *Caption* ] localizable string value



To make the list of the detail editable, make the following changes to the detail schema:

1. Add dependencies from the `ConfigurationGrid`, `ConfigurationGridGenerator`, `ConfigurationGridUtilities` modules.
2. Connect the `ConfigurationGridUtilities` mixin.
3. Set the `IsEditable` attribute value to `true`.
4. In the [diff array of modifications](#), add the configuration object in which the properties are set and handler methods for detail list events are bound.

Below is the detail schema source code with comments:

```
// Defining schema and setting its dependencies from other modules.
define("UsrCourierServiceDetail", ["ConfigurationGrid", "ConfigurationGridGenerator",
  "ConfigurationGridUtilities"], function() {
  return {
    // Detail object schema name.
    entitySchemaName: "UsrCourierService",
    // Schema attribute list.
    attributes: {
      // Determines whether the editing is enabled.
      "IsEditable": {
        // Data type – logic.
        dataValueType: Terrasoft.DataValueType.BOOLEAN,
        // Attribute type – virtual column of the view model.
        type: Terrasoft.ViewModelColumnType.VIRTUAL_COLUMN,
        // Set value.
        value: true
      }
    },
    // Used mixins.
    mixins: {
      ConfigurationGridUtilities: "Terrasoft.ConfigurationGridUtilities"
    },
    // Array with view model modifications.
    diff: /**SCHEMA_DIFF*/[
      {
        // Operation type – merging.
        "operation": "merge",
        // Name of the schema element, with which the action is performed.
        "name": "DataGrid",
```

```

// Object, whose properties will be joined with the schema element properties.
"values": {
    // Class name
    "className": "Terrasoft.ConfigurationGrid",
    // View generator must generate only part of view.
    "generator": "ConfigurationGridGenerator.generatePartial",
    // Binding the edit elements configuration obtaining event
    // of the active page to handler method.
    "generateControlsConfig": {"bindTo": "generateActiveRowControlsConfig"},
    // Binding the active record changing event to handler method.
    "changeRow": {"bindTo": "changeRow"},
    // Binding the record selection cancellation event to handler method.
    "unSelectRow": {"bindTo": "unSelectRow"},
    // Binding of the list click event to handler method.
    "onGridClick": {"bindTo": "onGridClick"},
    // Actions performed with active record.
    "activeRowActions": [
        // [Save] action setup.
        {
            // Class name of the control element, with which the action is connected
            "className": "Terrasoft.Button",
            // Display style – transparent button.
            "style": this.Terrasoft.controls.ButtonEnums.style.TRANSPARENT,
            // Tag.
            "tag": "save",
            // Marker value.
            "markerValue": "save",
            // Binding button image.
            "imageConfig": {"bindTo": "Resources.Images.SaveIcon"}
        },
        // [Cancel] action setup.
        {
            "className": "Terrasoft.Button",
            "style": this.Terrasoft.controls.ButtonEnums.style.TRANSPARENT,
            "tag": "cancel",
            "markerValue": "cancel",
            "imageConfig": {"bindTo": "Resources.Images.CancelIcon"}
        },
        // [Delete] action setup.
        {
            "className": "Terrasoft.Button",
            "style": this.Terrasoft.controls.ButtonEnums.style.TRANSPARENT,
            "tag": "remove",
            "markerValue": "remove",
            "imageConfig": {"bindTo": "Resources.Images.RemoveIcon"}
        }
    ],
    // Binding to method that initializes subscription to events
    // of clicking buttons in the active row.

```

```

        "initActiveRowKeyMap": {"bindTo": "initActiveRowKeyMap"},
        // Binding the active record action completion event to handler method.
        "activeRowAction": {"bindTo": "onActiveRowAction"},
        // Identifies whether multiple records can be selected.
        "multiSelect": {"bindTo": "MultiSelect"}
    }
}
]/**SCHEMA_DIFF*/
};
});

```

Save the created detail list schema to apply the changes.

### 3. Setting up detail in the section edit page replacing schema.

To display the detail on the order edit page, create a client replacing module and indicate the [ *Order edit page* ] as the parent object, `OrderPageV2` ) (fig. 7). Creating a replacing page is covered in the [“Create a client schema”](#) article.

Fig. 7. Properties of the order edit page replacing schema

The screenshot shows a 'Properties' window for a schema named 'OrderPageV2'. The 'General' section shows the title 'Order edit page', name 'OrderPageV2', and package 'sdkCreateDetailWithEditableGrid'. The 'Inheritance' section is expanded, showing 'Parent object' set to 'Order edit page ( Order )' and 'Replace parent' checked. A red box highlights the 'Parent object' dropdown menu.

To display the [ *Courier Service* ] detail on the [ *Delivery* ] tab of the order edit page, add the following source code. It defines a new `UsrCourierServiceDetail` detail in the `details` section and its location on the order edit page is specified in the `diff` modification array section.

```

define("OrderPageV2", [], function() {
    return {
        // Name of the edit page object schema.
        entitySchemaName: "Order",
        // List of edit page details being added.
        details: /**SCHEMA_DETAILS*/{
            // [Courier services] detail setup.
            "UsrCourierServiceDetail": {
                // Detail schema name.

```

```

        "schemaName": "UsrCourierServiceDetail",
        // Detail object schema name.
        "entitySchemaName": "UsrCourierService",
        // Filtering displayed contacts for current order only.
        "filter": {
            // Detail object schema column.
            "detailColumn": "UsrOrder",
            // Section object schema column.
            "masterColumn": "Id"
        }
    }
}
}/**SCHEMA_DETAILS*/,
// Array with modifications.
diff: /**SCHEMA_DIFF*/[
    // Metadata for adding the [Courier services] detail.
    {
        "operation": "insert",
        // Detail name.
        "name": "UsrCourierServiceDetail",
        "values": {
            "itemType": Terrasoft.core.enums.ViewItemType.DETAIL,
            "markerValue": "added-detail"
        },
        // Containers, where the detail is located.
        // Detail is placed on the [Delivery] tab.
        "parentName": "OrderDeliveryTab",
        "propertyName": "items",
        // Index in the list of added elements.
        "index": 3
    }
]/**SCHEMA_DIFF*/
};
});

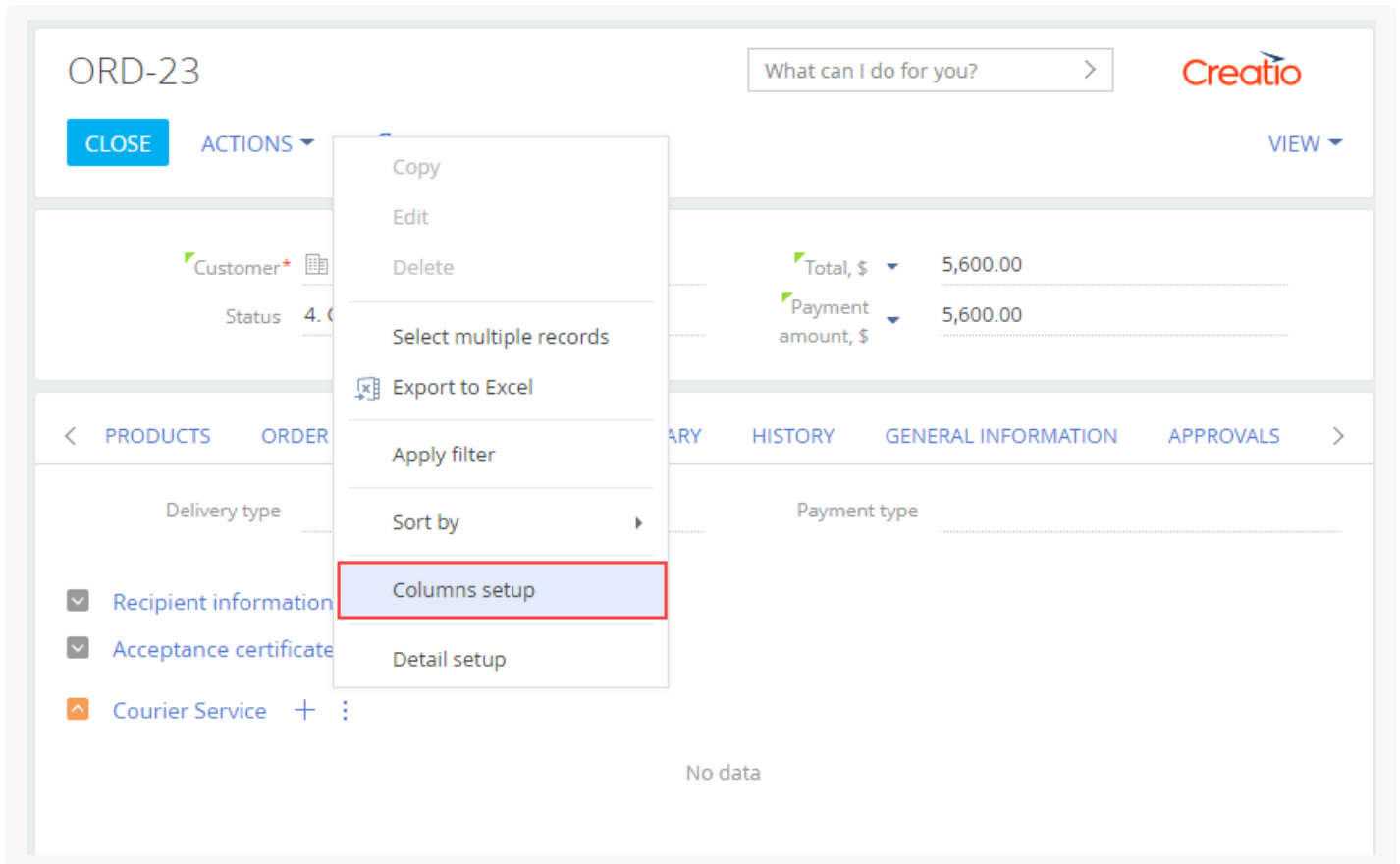
```

Save a replacing schema of the edit page

## 4. Setup of detail list columns.

At this stage, the detail is completely operational, however accounts are not displayed on the detail, because the detail list does not have displayed columns specified for it. Go to the detail menu and set up the columns to be displayed (Fig. 8)

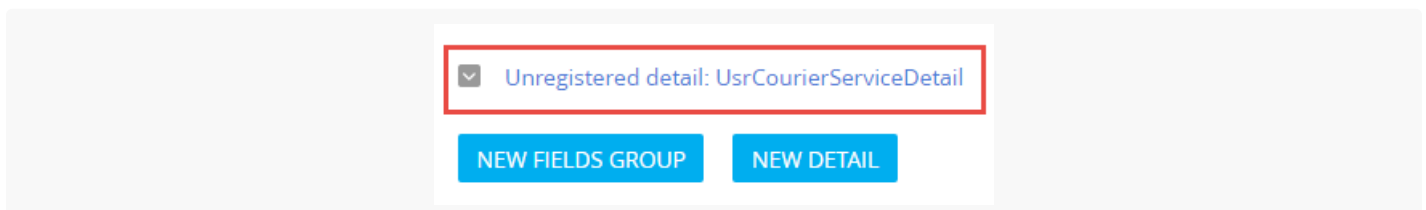
Fig. 8. Detail action menu



## Detail wizard, Section wizard and details with editable lists

A detail with an editable list is not a standard Creatio detail and can not be added to sections using the [Detail Wizard](#). If you try perform the [ *Detail setup* ] action (Fig. 8), Creatio will display the “detail is not registered” message. The [Section wizard](#) displays a similar message (Fig. 9).

Fig. 9. Unregistered detail in the Section wizard



Usually details with editable lists do not require registration. However, if you still need to register such a detail for some reason, use the following SQL script:

```
DECLARE
-- Name of the created pop-up summary view schema.
@ClientUnitSchemaName NVARCHAR(100) = 'UsrCourierServiceDetail',
-- Name of the object schema, to which the pop-up summary is bound.
@EntitySchemaName NVARCHAR(100) = 'UsrCourierService',
-- Detail name.
@DetailCaption NVARCHAR(100) = 'Courier service'
```

```

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))

```

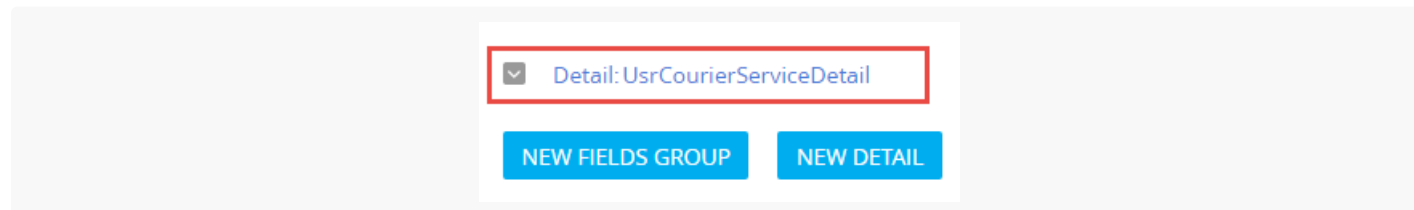
### Attention.

To register a custom detail, make changes to the `SysDetails` table in the Creatio database using the SQL query.

Pay high attention to creating and executing the SQL query. Executing an incorrect SQL query can damage the existing data and disrupt the system.

The detail becomes available in the corresponding lookup and displayed as registered in the section wizard after you update the page. After this you can use the Section wizard to add this detail to other tabs of the [ *Orders* ] edit page.

Fig. 9. Registered detail is displayed in the Section wizard



# Creating a detail with selection from lookup



## Introduction

Details are elements of section edit pages. Details display records bound to the current section record by a lookup field. These can be, for example, records from other sections, whose primary objects contain lookup columns linked to the primary object of the current section. Details are displayed on tabs of section edit pages.

More information about details is available in the “[Details](#)” article of the “[Application interface and structure](#)” section.

Since a detail with selection from lookup is not a standard Creatio detail, it is not enough to use [section wizard](#) to add it to the section. Below we will describe a way of adding such a detail to a section edit page.

To add a custom detail with selection from lookup to an existing section:

1. Create a detail object schema.
2. Create a detail via detail wizard and add it to the section.
3. Configure the detail schema.
4. Set up detail fields.

## Case description

Create the [ *Acceptance certificates* ] custom detail on the [ *Delivery* ] tab of the [ *Orders* ] section. The detail should display the list of documents – acceptance certificates for the current order.

## Source code

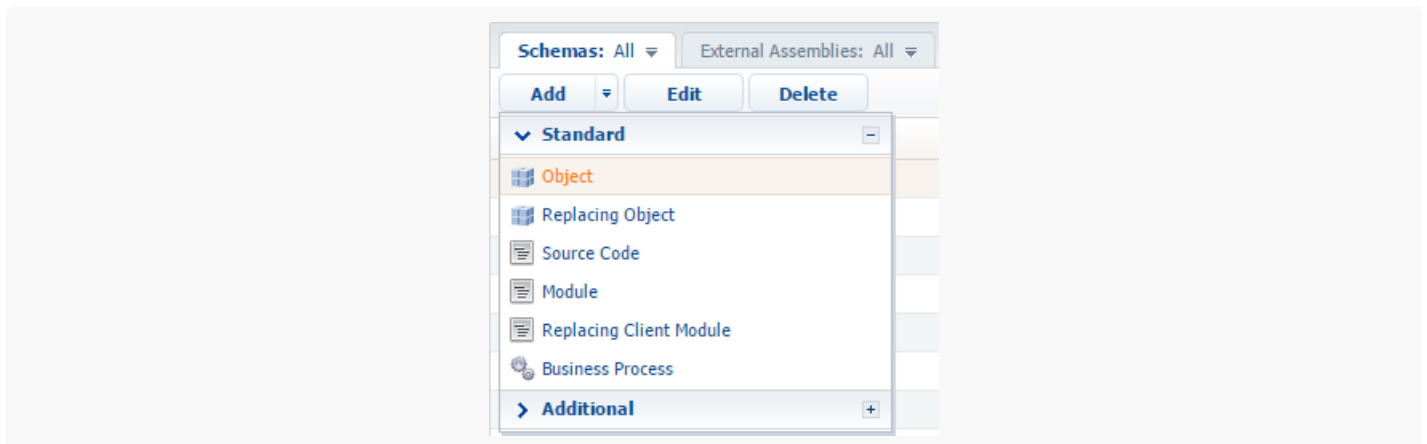
You can download the package with case implementation using the following [link](#).

## Case implementation algorithm

### 1. Creating a detail object schema

Run the [ *Add* ] – [ *Object* ] menu command on the [ *Schemas* ] tab of the [ *Configuration* ] section (Fig. 1).

Fig. 1. Adding a detail object schema



Select the “Base object” as the parent object (Fig. 2).

Object properties:

- [Name] – “UsrCourierCertInOrder”
- [Title] – “Acceptance certificates for deliveries of orders”

Find more information about object schema property setup in object designer in the “[Workspace of the Object Designer](#)” article.

Fig. 2. Setup of detail object schema properties



**Properties**

<Enter search text>

**General**

Name: CourierCertInOrder

Title: Acceptance certificates for deliveries of orders

Package: CustomOrderDetails

**Inheritance**

Parent object: Base object ( Base )

Replace parent: ☐

**Access rights**

Operations: ☐

Records: ☐

Add the [ *Order* ] lookup column to the object schema establishing connection with the [ *Orders* ] section and the [ *Document* ] lookup column containing the acceptance certificate. Mark both columns as “required” to avoid adding empty records. Column setup is shown in fig. 3 and fig. 4.

Fig. 3. The [ *Order* ] column property setup

. The 'Lookup' section shows Lookup: Order, which is highlighted with a red box. The 'Behavior' section shows Required: Application Level, which is highlighted with a red box. Other options include Cascade connection, List, Default value, and Make copy."/>

**Properties**

<Enter search text>

**General**

Title: Order

Name: Order

**String**

Multi-string text: ☐

**Lookup**

Lookup: Order

Cascade connection: ☐

List: ☐

**Behavior**

Required: Application Level

Default value: (No)

Make copy: ☒

Fig. 4. The [ *Document* ] column property setup

**Properties**

<Enter search text>

**General**

Title Document

Name Document

**String**

Multi-string text ☐

**Lookup**

Lookup Document

Cascade connection ☐

List ☐

**Behavior**

Required Application Level

Default value (No)

Make copy ☒

Save and publish the object schema.

## 2. Creating a detail and adding it to the section

Via [detail wizard](#) create the [ *Acceptance certificates* ] detail (Fig. 5). Add the detail to the [ *Delivery* ] tab of the [ *Orders* ] section page via section wizard (Fig. 6).

Fig. 5. Creating the detail

Acceptance certificates: Detail

SAVE CANCEL < DETAIL PAGE BUSINESS RULES BUSINESS PROC

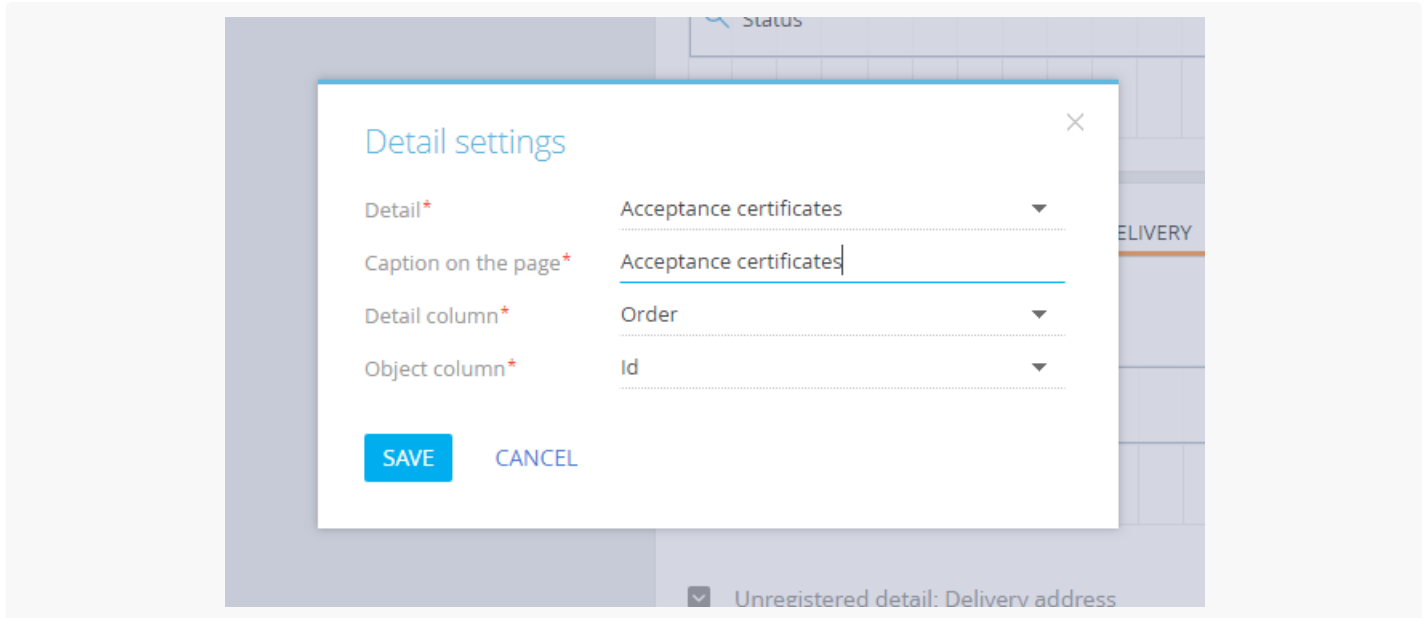
Select properties for detail:

Title\* Acceptance certificates

Object\* Acceptance certificates for deliveries of orders

Acceptance certificates for deliveries of orders

Fig. 6. Adding the detail to the section



As a result, the `UsrSchema1Detail` detail schema and the `OrderPageV2` section replacing schema will be created in the development package.

#### Attention.

We recommend changing the name of the detail schema to a more unique name, since the auto generated name can be duplicated in a different development package. We replaced the name for `UsrCourierCertDetail` in the current case.

Change the detail schema name for a new one in the `OrderPageV2` section replacing schema.

## 3. Configuring the detail

Change the source code of the created detail schema:

1. Add dependency from the `ConfigurationEnums` module.
2. Add the `onDocumentInsert()`, `onCardSaved()` event handler-methods, the `openDocumentLookup()` method of calling the modal lookup window and additional data control methods.
3. Add the necessary configuration objects to the `diff` modification array.

The `Terrasoft.EntitySchemaQuery` class is used for executing database queries in the current case. You can learn more about using `EntitySchemaQuery` in the [“The use of EntitySchemaQuery for creation of queries in database”](#) article. To simplify the case, we blocked the detail menu options that enable editing and copying detail list records.

Source code of the detail schema:

```
// Defining the shcema and setting up its dependencies from other modules.
```

```

define("UsrCourierCertDetail", ["ConfigurationEnums"],
function(configurationEnums) {
    return {
        // Name of the detail object schema.
        entitySchemaName: "UsrCourierCertInOrder",
        // Detail schema methods.
        methods: {
            //Returns columns selected by query.
            getGridDataColumns: function() {
                return {
                    "Id": {path: "Id"},
                    "Document": {path: "UsrDocument"},
                    "Document.Number": {path: "UsrDocument.Number"}
                };
            },

            //Configures and displays modal lookup window.
            openDocumentLookup: function() {
                //Configuration object
                var config = {
                    // Name of the object schema whose records will be displayed in the look
                    entitySchemaName: "Document",
                    // Multiple selection option.
                    multiSelect: true,
                    // Columns used in the lookup, e.g., 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", {
                    // Setting up the root schema.
                    rootSchemaName: this.entitySchemaName
                });
                // Adding the [Id] column.
                esq.addColumn("Id");
                // Adding the [Id] column for the [Document] schema.
                esq.addColumn("Document.Id", "DocumentId");
                // Creating and adding filters to query collection.
                esq.filters.add("filterOrder", this.Terrasoft.createColumnFilterWithParameters(
                    this.Terrasoft.ComparisonType.EQUAL, "UsrOrder", OrderId));
                // Receiving the whole record collection and its display in the modal lookup
                esq.getEntityCollection(function(result) {
                    var existsDocumentsCollection = [];
                    if (result.success) {
                        result.collection.each(function(item) {

```

```

        existsDocumentsCollection.push(item.get("DocumentId"));
    });
}
// Adding filter to the configuration object.
if (existsDocumentsCollection.length > 0) {
    var existsFilter = this.Terrasoft.createColumnInFilterWithParameters(
        existsDocumentsCollection);
    existsFilter.comparisonType = this.Terrasoft.ComparisonType.NOT_EQUAL;
    existsFilter.Name = "existsFilter";
    config.filters = existsFilter;
}
// Call of the modal lookup window
this.openLookup(config, this.addCallBack, this);
}, this);
},

// Event handler of saving the edit page.
onCardSaved: function() {
    this.openDocumentLookup();
},

//Opens the document lookup if the order edit page has been saved.
addRecord: function() {
    var masterCardState = this.sandbox.publish("GetCardState", null, [this.sandbox.id]);
    var isNewRecord = (masterCardState.state === configurationEnums.CardStateV2.NEW);
    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();
},

// Adding the selected products.
addCallBack: function(args) {
    // Class instance of the BatchQuery package query.
    var bq = this.Ext.create("Terrasoft.BatchQuery");
    var OrderId = this.get("MasterRecordId");
    // Collection of the selected documents from the lookup.
    this.selectedRows = args.selectedRows.getItems();
    // Collection passed over to query.
    this.selectedItems = [];
    // Copying the necessary data.
    this.selectedRows.forEach(function(item) {
        item.OrderId = OrderId;
    });
}

```

```

        item.DocumentId = item.value;
        bq.add(this.getDocumentInsertQuery(item));
        this.selectedItems.push(item.value);
    }, this);
    // Executing the package query if it is not empty.
    if (bq.queries.length) {
        this.showBodyMask.call(this);
        bq.execute(this.onDocumentInsert, this);
    }
},

//Returns query for adding the current object.
getDocumentInsertQuery: function(item) {
    var insert = Ext.create("Terrasoft.InsertQuery", {
        rootSchemaName: this.entitySchemaName
    });
    insert.setParameterValue("UsrOrder", item.OrderId, this.Terrasoft.DataValueT
    insert.setParameterValue("UsrDocument", item.DocumentId, this.Terrasoft.Data
    return insert;
},

//Method called when adding records to the detail record 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("Id
    // Create viewmodel for new items.
    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);
},

// Method called when deleting records from the detail record list.
deleteRecords: function() {

```

```

        var selectedRows = this.getSelectedItems();
        if (selectedRows.length > 0) {
            this.set("SelectedRows", selectedRows);
            this.callParent(arguments);
        }
    },

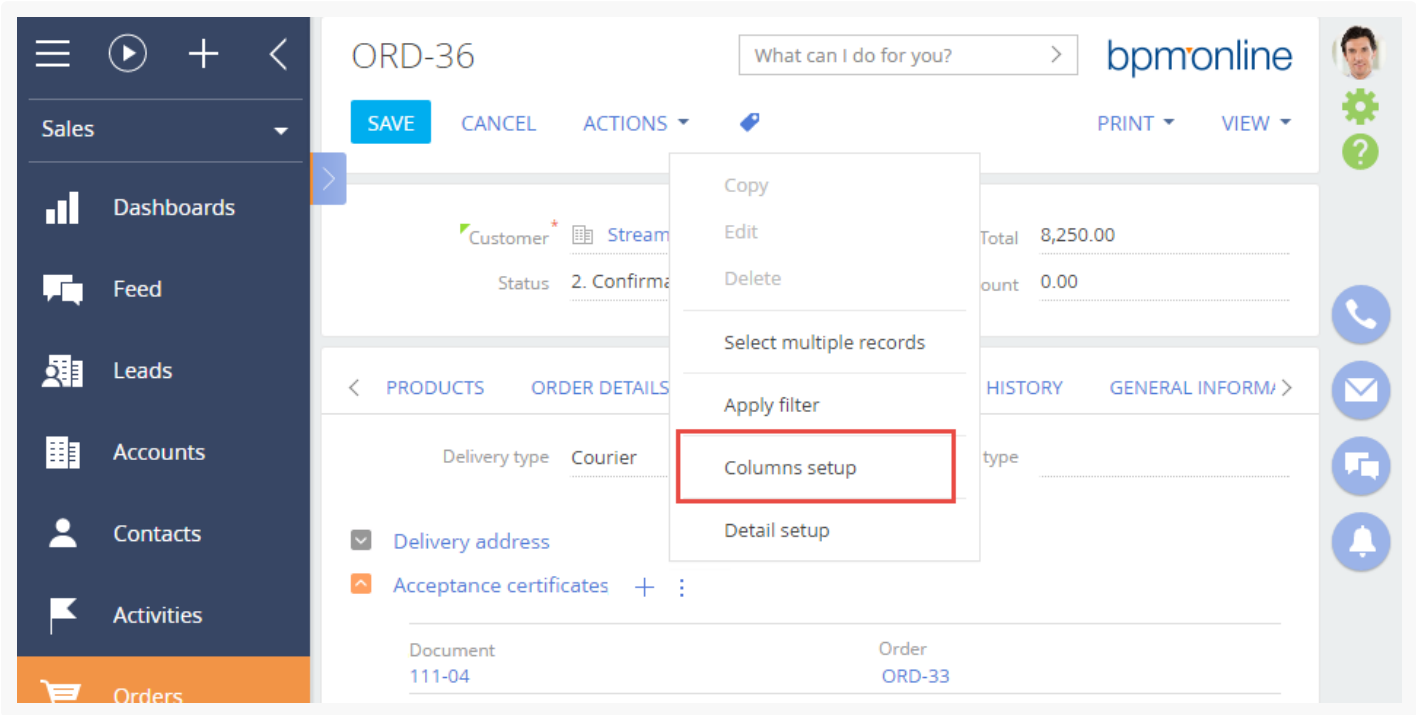
    // Hide the [Copy] menu option.
    getCopyRecordMenuItem: Terrasoft.emptyFn,
    // Hide the [Edit] menu option.
    getEditRecordMenuItem: Terrasoft.emptyFn,
    // Returns the default filter column name.
    getFilterDefaultColumnName: function() {
        return "UsrDocument";
    }
},
// Modification array.
diff: /**SCHEMA_DIFF*/[
    {
        // Operation type - merging.
        "operation": "merge",
        // Name of the schema element under operation.
        "name": "DataGrid",
        // The object, whose properties will be combined with the schema element properties.
        "values": {
            "rowDataItemMarkerColumnName": "UsrDocument"
        }
    },
    {
        // Operation type - merging.
        "operation": "merge",
        // Name of the schema element under operation.
        "name": "AddRecordButton",
        // The object, whose properties will be combined with the schema element properties.
        "values": {
            "visible": {"bindTo": "getToolsVisible"}
        }
    }
]/**SCHEMA_DIFF*/
};
}
);

```

## 4. Setting up detail list columns

At this stage the detail is completely operable but contact records are not displayed on the detail as the display columns are not specified. Call the detail action menu and set up the column display (fig. 7).

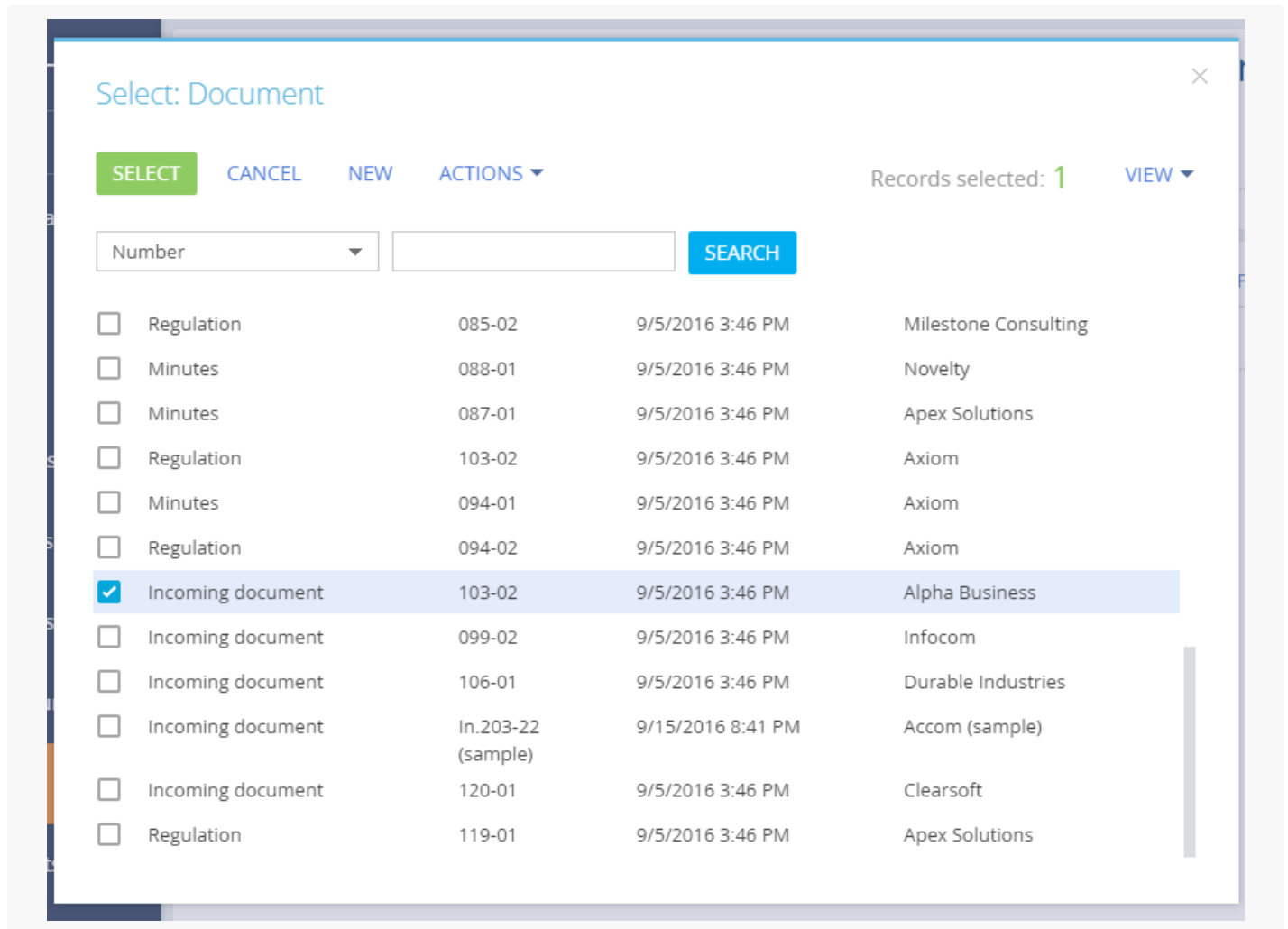
Fig. 7. Detail action menu



As a result, the new detail will enable adding records from the document lookup via the modal window (Fig. 8).

Fig. 8. Case result





# Adding multiple records to a detail

 Advanced

## Introduction

Normally, a detail allows you to only add one record. Adding multiple entries to a detail is done through the `LookupMultiAddMixin` mixin.

A mixin is a class designed to extend the functions of other classes. Learn more about mixins in the [“Mixins. The “mixins” property”](#).

The `LookupMultiAddMixin` is designed to extend the detail schemas. It provides an opportunity to add multiple lookup records to the detail at the same time.

The sequence of adding the multiple records selection functionality to a detail:

1. Create a replacing schema of the detail.
2. Use mixin methods instead of base detail methods.

## Case description

Implement the possibility of multiple records selection in the [ *Contacts* ] detail on the [ *Sales* ] section records edit page.

## Source code

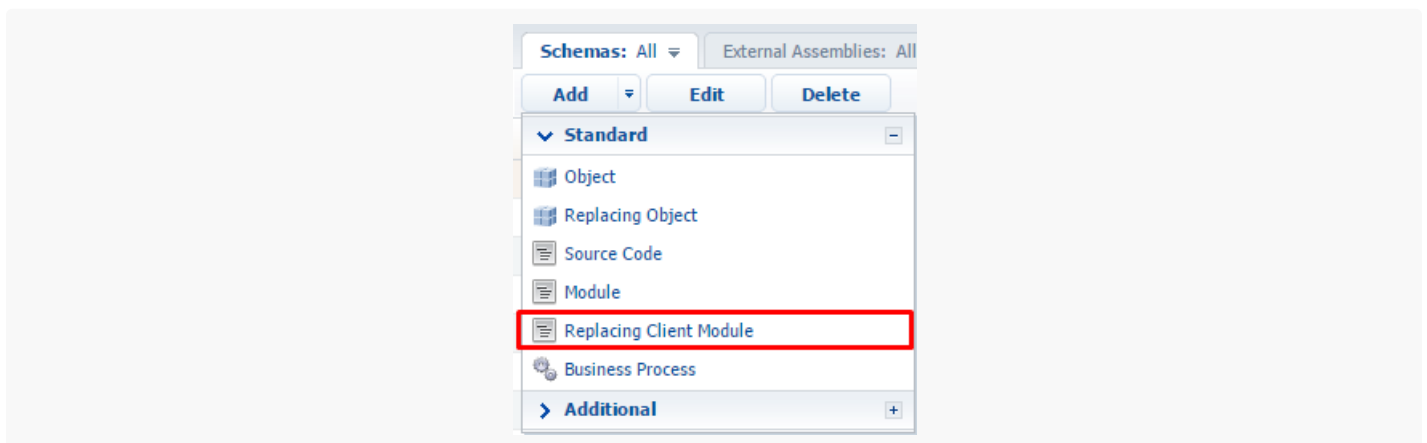
You can download the package with case implementation using the following [link](#).

## Case implementation algorithm

### 1. Create a replacing view model schema of a detail

Create a detail replacing schema in the development package (Fig. 1). The procedure for creating a replacing schema is covered in the “[Create a client schema](#)”.

Fig. 1. Adding replacing client module



Select the `OpportunityContactDetailV2` schema as the parent object (Fig. 2).

Fig. 2. Properties of the replacing client module

**Properties**

<Enter search text>

**General**

Title: OpportunityContactDetailV2

Name: OpportunityContactDetailV2

Package: Custom

**Inheritance**

Parent object: OpportunityContactDetailV2 ( Opportunity )

Replace parent: ☒

## 2. Use mixin methods instead of base detail methods.

To use the `LookupMultiAddMixin` mixin in a schema, add it to the `mixins` property and initialize it in the pre-defined `init()` schema method. Learn more about pre-defining the `init()` method in the “[Modular development principles in Creatio](#)” article.

To implement the required functionality, you need to pre-define the “add” button displaying methods ( `getAddRecordButtonVisible()` ), saving detail page methods ( `onCardSaved()` ), and adding a detail record methods ( `addRecord()` ).

Saving a detail page and adding record methods include the method of calling the help window for multiple selection `openLookupWithMultiSelect()` and the associated method `getMultiSelectLookupConfig()` which configures the help window, is used. The description and parameters of these methods are given in Table 1.

Table 1. Methods for calling and configuring the help window

Methods.	Description
<code>openLookupWithMultiSelect(isNeedCheckOfNew)</code>	Opens a lookup window with a multiple selection option. The <code>isNeedCheckOfNew {bool}</code> parameter indicates the need to check if the record is new.
<code>getMultiSelectLookupConfig()</code>	Returns the configuration object for the help window. Object properties: <code>rootEntitySchemaName</code> – root object schema; <code>rootColumnName</code> – a connected column that indicates the root schema record; <code>relatedEntitySchemaName</code> – connected schema; <code>relatedColumnName</code> – a column that indicates the connected schema record.

In this case the help window will pull data from the `OpportunityContact` table using the `Opportunity` and `Contact` columns.

Source code of the detail schema:

```
define("OpportunityContactDetailV2", ["LookupMultiAddMixin"], function() {
```

```

return {
  mixins: {
    // Connecting the mixin to the schema.
    LookupMultiAddMixin: "Terrasoft.LookupMultiAddMixin"
  },
  methods: {
    // Overriding the base method for initializing the schema.
    init: function() {
      this.callParent(arguments);
      //Initializing the mixin.
      this.mixins.LookupMultiAddMixin.init.call(this);
    },
    // Overriding the base method for displaying the "Add" button.
    getAddRecordButtonVisible: function() {
      //Displaying the "add" button if the detail is maximized, even if the detail edit
      return this.getToolsVisible();
    },
    // Overriding the base method.
    // The save event handler for the detail edit page.
    onCardSaved: function() {
      // Opens the window for multiple record selection.
      this.openLookupWithMultiSelect();
    },
    // Overriding the base method of adding a detail record.
    addRecord: function() {
      // Opens the window for multiple records selection.
      this.openLookupWithMultiSelect(true);
    },
    // A method that returns a window configuration object.
    getMultiSelectLookupConfig: function() {
      return {
        // Root schema – [Opportunities].
        rootEntitySchemaName: "Opportunity",
        // Root schema column.
        rootColumnName: "Opportunity",
        // Connected schema – [Contact].
        relatedEntitySchemaName: "Contact",
        // Root schema column.
        relatedColumnName: "Contact"
      };
    }
  }
};
});

```

After saving the schema and reloading the application page, the user will be able to select multiple records from the lookup (Fig. 4) by clicking the “add” detail record button (Fig. 3). All selected records will be added to the

[ *Contacts* ] detail of the [ *Opportunities* ] section record edit page (Fig. 5).

Fig. 3. Adding multiple records to a detail



Fig. 4. Selecting necessary records from a lookup

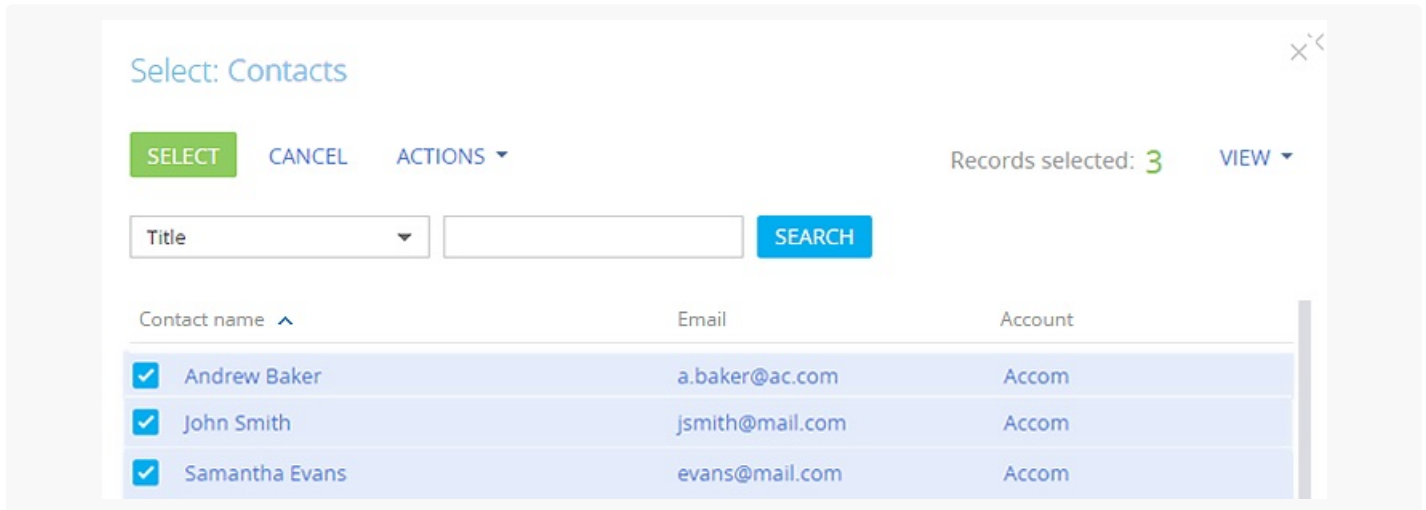
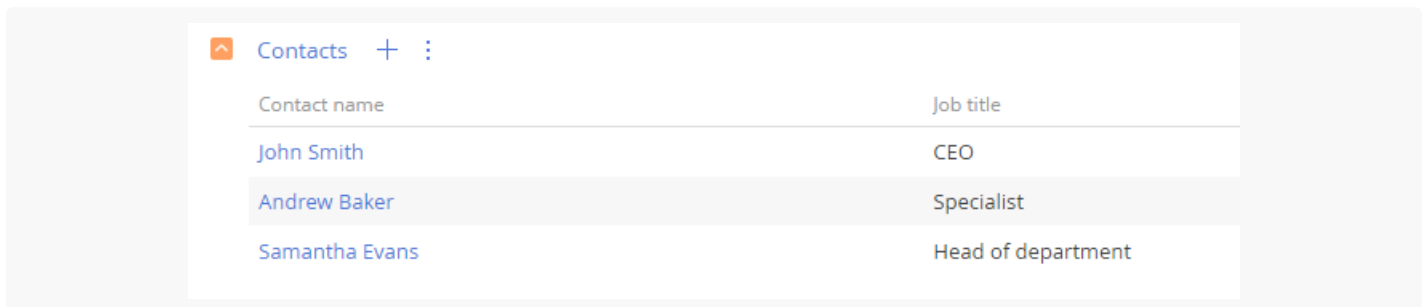


Fig. 5. Case result: All selected records are added to a detail



# Creating a custom detail with fields

 **Advanced**

## Introduction

A detail with fields can include multiple field groups. The base detail with fields is implemented in the `BaseFieldsDetail` schema of the `BaseFinance` package, which is available in Creatio bank customer journey, bank sales and lending. The detail record view model is implemented in the `BaseFieldRowViewModel` schema.

Base detail with fields enables you to:

- Add detail records without saving a page.
- Work with a detail like you would with an edit page.
- Use the base field validation with the ability to add a custom one.
- Add a virtual record.
- Expand record behavior logic.

A custom detail with fields can be [created](#) with the help of the base detail (a custom detail schema should be inherited from the base detail schema).

## Case description

Implement a custom detail with fields for document registration. The detail should be populated with records that include the document's [ *Number* ] and [ *Series* ] fields. The detail should be located on the [ *History* ] tab of the contact edit page.

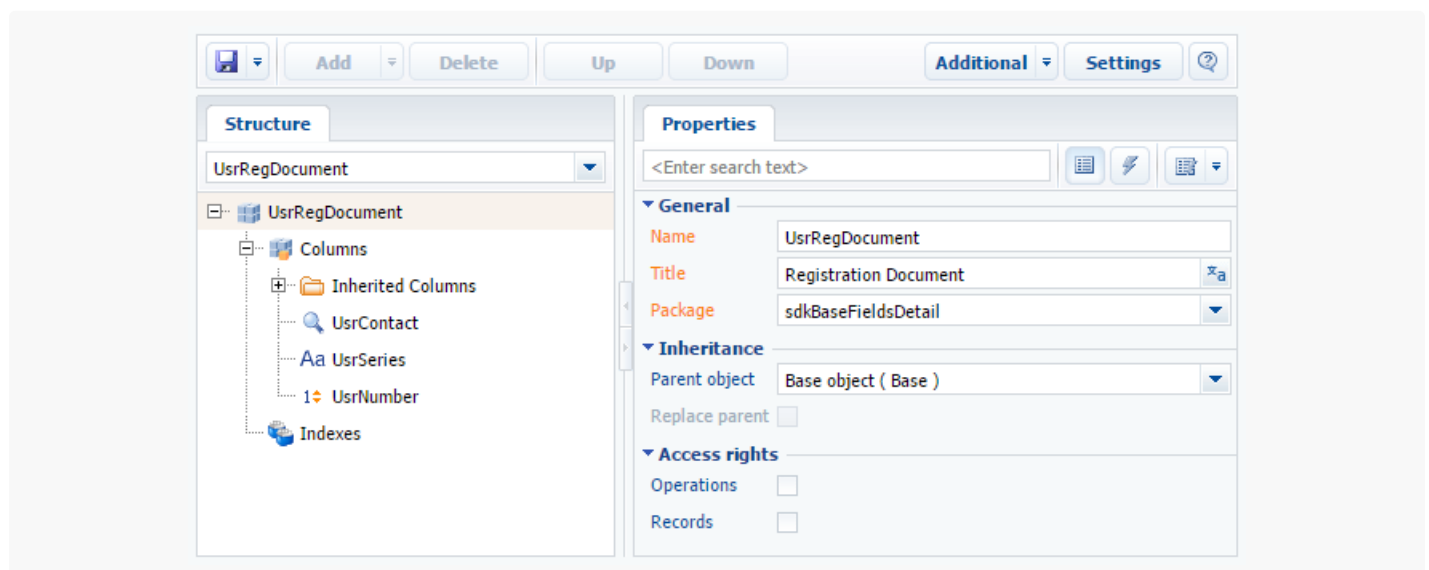
## Case implementation algorithm

### 1. Create a detail object schema

[Create a new object schema](#) in a custom package with the following property values:

- [Title] - "Registration document".
- [Name] - "UsrRegDocument".
- [Package] - the schema will be placed in this package after publishing. By default, this property contains the name of the package selected prior to creating a schema. It can be populated with any value from the drop-down list.
- [Parent object] - "Base object", implemented in the Base package.

Fig. 1. Object schema properties



Add three columns in the object structure. Column properties are listed in Table 1. Learn more about adding object schema columns in the [“Create the entity schema”](#) article.

Table 1. — Column properties of the UsrRegDocument detail object schema

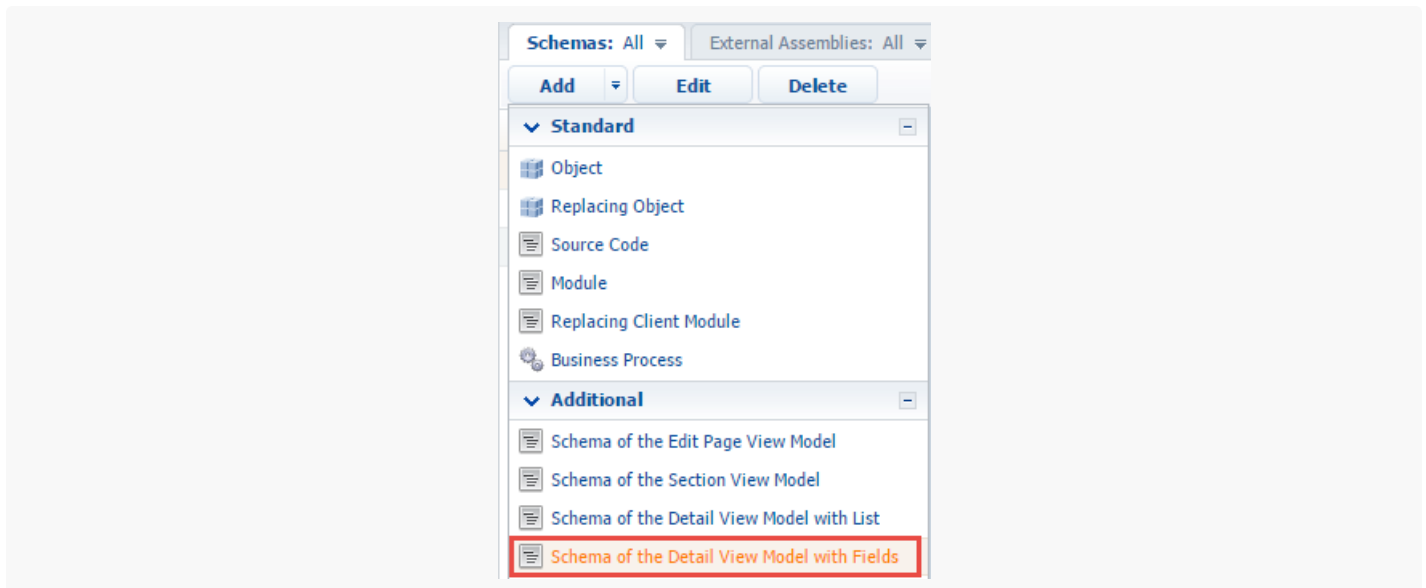
Title	Name	Data Type
Contact	UsrContact	Lookup
Series	UsrSeries	Text (50 characters)
Number	UsrNumber	Integer

Publish the schema to apply changes.

## 2. Create a view model schema for the custom detail with fields.

[Add a custom schema](#)([ *Schema of the Detail View Model with Fields* ]) in a custom package (Fig. 2).

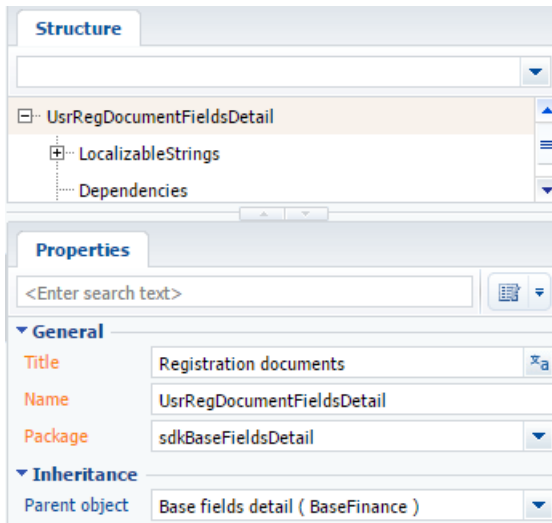
Fig. 2. Adding a custom view model schema for the custom detail with fields



Property values for the created schema (Fig. 3):

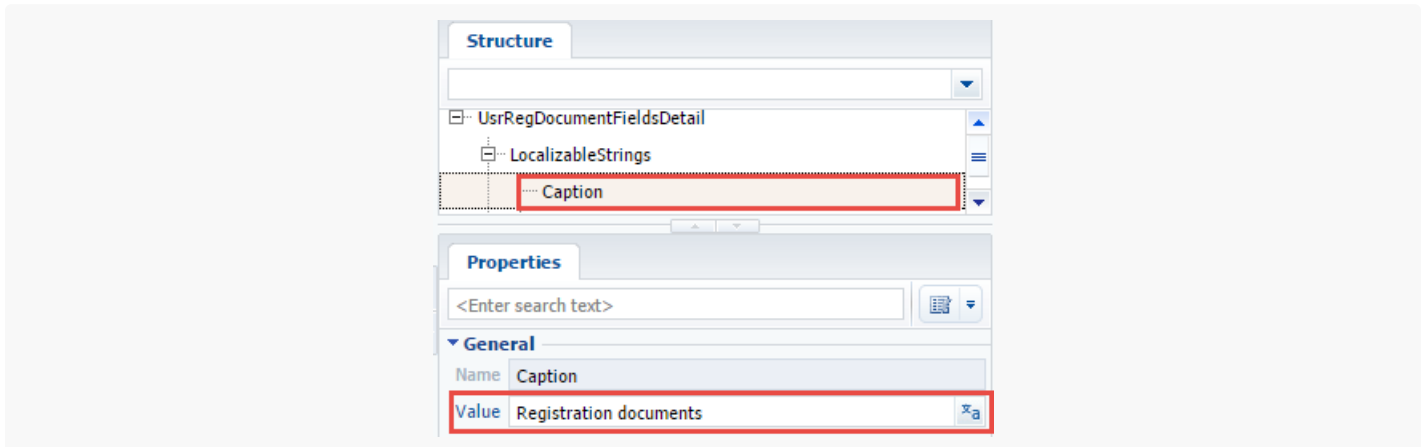
- [Title] – “Registration documents”.
- [Name] – “UsrRegDocumentFieldsDetail”.
- [Package] – the schema will be placed in this package after publishing. By default, this property contains the name of the package selected prior to creating a schema. It can be populated with any value from the drop-down list.
- [Parent object] – “Base fields detail”, implemented in the BaseFinance package.

Fig. 3. UsrRegDocumentFieldsDetail client schema properties



Assign the “Registration documents” value to the localizable [ *Caption* ] string of the [ *Value* ] property.

Fig. 4. Localizable string properties



[Create a module description](#) and redefine the base `getDisplayColumns()` method, which returns column names that are displayed as detail fields. By default, this method returns all required columns, as well as the column with the set [ *Displayed value* ] checkbox in the object schema.

Schema source code:

```
define("UsrRegDocumentFieldsDetail", [],
function() {
return {
entitySchemaName: "UsrRegDocument",
diff: /**SCHEMA_DIFF*/ [], /**SCHEMA_DIFF*/
methods: {
getDisplayColumns: function() {
return ["UsrSeries", "UsrNumber"];
}
}
};
});
```



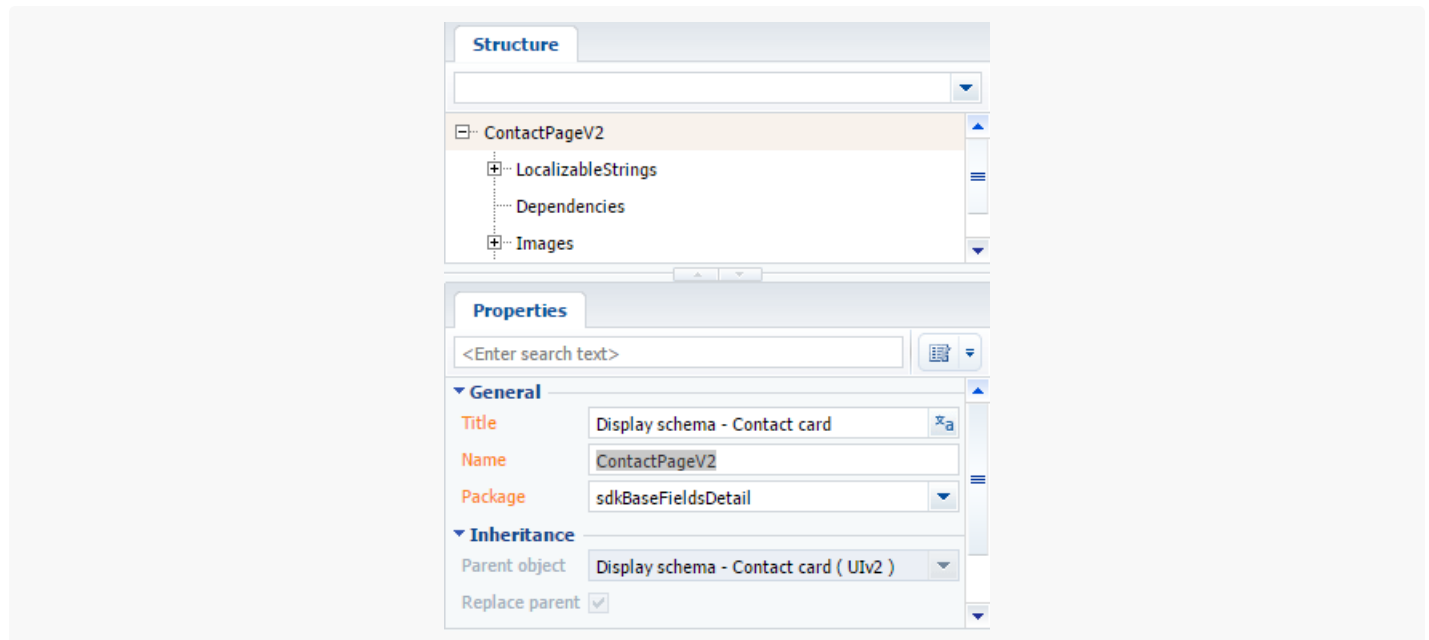
Save the schema to apply changes.

### 3. Create a replacing schema for the edit page

To do this, create a [replacing schema](#) of the contact edit page ( `ContactPageV2` ). Main replacing schema properties (Fig. 5):

- [Title] - "Display schema - Contact card".
- [Name] - "ContactPageV2".
- [Package] - the schema will be placed in this package after publishing. By default, this property contains the name of the package selected prior to creating a schema. It can be populated with any value from the drop-down list.
- [Parent object] - "Display schema - Contact card", implemented in the `UIv2` package.

Fig. 5. The ContactPageV2 replacing schema properties



Make the following adjustments to the source code:

- [Create a module description](#)
- Add a detail in the ["details" property](#)
- Add a configuration object of the detail's view model to the ["diff" modification array](#).

Schema source code:

```
define("ContactPageV2", [], function() {
    return {
        entitySchemaName: "Contact",
        details: /**SCHEMA_DETAILS*/ {
```

```

// Adding a field with details.
"UshrRegDocumentFieldsDetail": {
    // Name of the custom detail schema.
    "schemaName": "UshrRegDocumentFieldsDetail",
    // Filtering current contact's record details.
    "filter": {
        // Detail object column.
        "detailColumn": "UshrContact",
        // Contact's Id column.
        "masterColumn": "Id"
    }
}
} /**SCHEMA_DETAILS*/ ,
diff: /**SCHEMA_DIFF*/ [{
    // Adding a new element.
    "operation": "insert",
    // Element name.
    "name": "UshrRegDocumentFieldsDetail",
    // Value configuration object.
    "values": {
        // Element type.
        "itemType": Terrasoft.ViewItemType.DETAIL
    },
    // Container element name.
    "parentName": "HistoryTab",
    // Property name of the container element with the collection of nested elements.
    "propertyName": "items",
    // The index of the element added to the collection.
    "index": 0
}] /**SCHEMA_DIFF*/
});
});

```

Save the schema to apply changes.

The [ *History* ] tab of the contact edit page should now have the custom detail with the [ *Registration documents* ] fields (Fig. 6).

Fig. 6. Case result

<

CONTACT INFO

ADDITIONAL INFORMATION

CURRENT EMPLOYMENT

HISTORY

ATTACHMENTS >

Registration documents

+

Series

AA

Number

123,456

✕

Series

BB

Number

234,567

✕

Series

CC

Number

345678

✕

**Attention.**

Register the detail with fields (similarly to the [detail with the editable grid](#)) in the system to make it visible in detail and section wizards.

# Advanced settings of a custom detail with fields



## Introduction

A detail with fields can include multiple field groups. The base detail with fields is implemented in the `BaseFieldsDetail` schema of the `BaseFinance` package, which is available in Creatio bank customer journey, bank sales and lending. The detail record view model is implemented in the `BaseFieldRowViewModel` schema.

The process of creating a custom detail with fields is described in a separate [Creating a custom detail with fields](#).

## Adding custom styles

### Case description

Redefine the field signature style for a detail implemented in the “[Creating a custom detail with fields](#)” article. The field signatures should be displayed in blue.

**Note.**

You can override the basic CSS style classes for displaying detail records by using the `getLeftRowContainerWrapClass()` and `getRightRowContainerWrapClass()` methods.

### Case implementation algorithm

## 1. Create a module schema and define record view styles

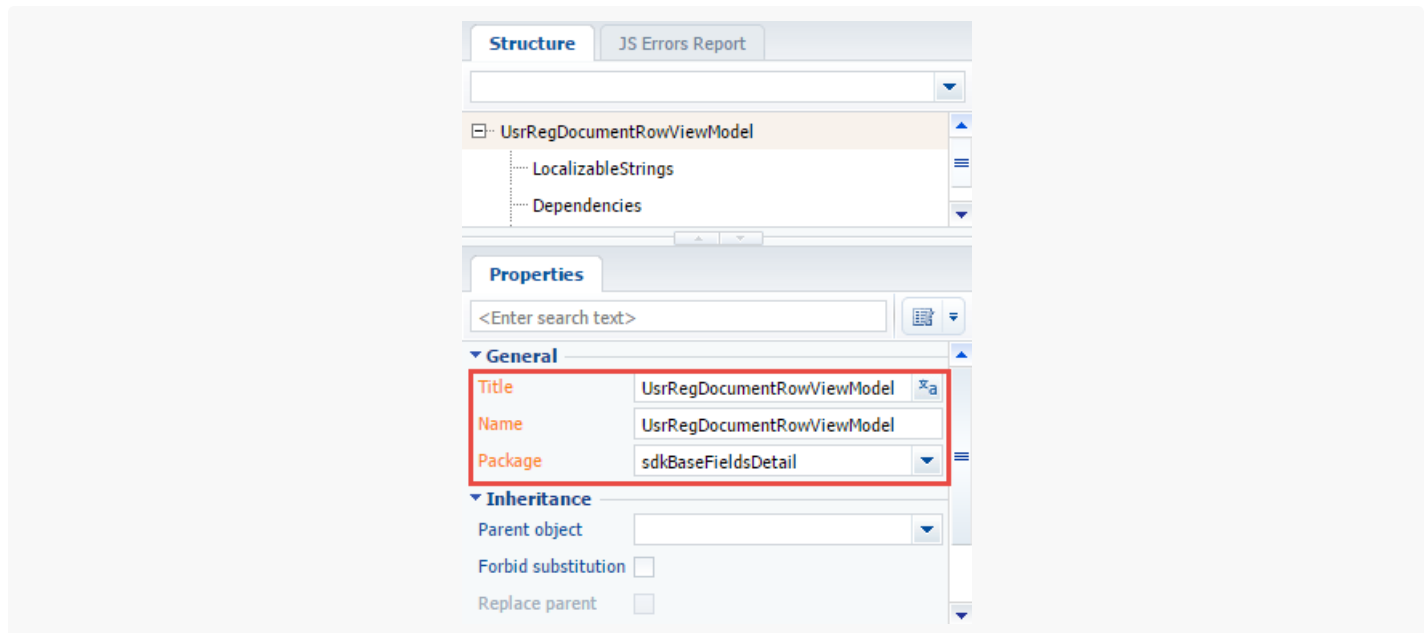
### Attention.

You can not set the styles in a view model schema of the edit page. It is necessary to create a new module schema, define the styles and add the created module to module dependencies of a detail.

[Create a new module schema](#) in a custom package with the following property values:

- [Title] – “UsrRegDocumentRowViewModel”.
- [Name] – “UsrRegDocumentRowViewModel”.
- [Package] – the schema will be placed in this package after publishing. By default, this property contains the name of the package selected prior to creating a schema. It can be populated with any value from the drop-down list.

Fig. 1. UsrRegDocumentFieldsDetail custom schema properties



Create a module description, and define the `Terrasoft.configuration.UsrRegDocumentRowViewModel` class which is inherited from `Terrasoft.configuration.BaseFieldRowViewModel`.

The source code of the schema:

```
define("UsrRegDocumentRowViewModel", ["BaseFieldRowViewModel"], function() {
    Ext.define("Terrasoft.configuration.UsrRegDocumentRowViewModel", {
        extend: "Terrasoft.BaseFieldRowViewModel",
        alternateClassName: "Terrasoft.UsrRegDocumentRowViewModel"
    });
    return Terrasoft.UsrRegDocumentRowViewModel;
});
```

Define the CSS view classes for the correct display of detail records. To do this, add the following CSS classes to the LESS tab of the module designer:

```
.reg-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;
    width;

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

Save the schema to apply changes.

## 2. Modifying a replacing view model schema of a detail

To use the created module and its styles in a detail schema, add it to the dependency of the module defined in the detail schema.

Additionally, add the following methods to a detail schema module:

- `getRowViewModelClassName()` – returns the name of the record view model class to the detail.
- `getLeftRowContainerWrapClass()` – returns the string array with CSS class names, used to generate the views of record signature field containers.

The source code of the modified schema:

```

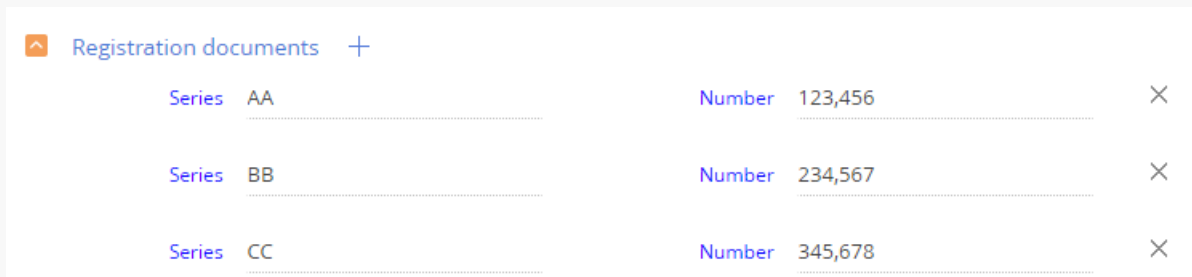
define("UsrRegDocumentFieldsDetail", ["UsrRegDocumentRowViewModel", "css!UsrRegDocumentRowViewMc
function() {
  return {
    entitySchemaName: "UsrRegDocument",
    diff: /**SCHEMA_DIFF*/ [], /**SCHEMA_DIFF*/
    methods: {
      getDisplayColumns: function() {
        return ["UsrSeries", "UsrNumber"];
      },
      getRowViewModelClassName: function() {
        return "Terrasoft.UsrRegDocumentRowViewModel";
      },
      getLeftRowContainerWrapClass: function() {
        return ["reg-document-left-row-container", "field-detail-row"];
      }
    }
  };
});

```

Save the schema to apply changes.

On the [ *History* ] tab of the contact edit page, the detail names will be displayed in blue (Fig. 2).

Fig. 2. Case result



Series	Number	
AA	123,456	×
BB	234,567	×
CC	345,678	×

## Adding additional custom logic for detail records

### Case description

Add the field validation to the [ *Number* ] field of the detail, implemented in the [“Creating a custom detail with fields”](#) article. The field value can not be negative.

### Case implementation algorithm

## 1. Adding a localizable string with the error message

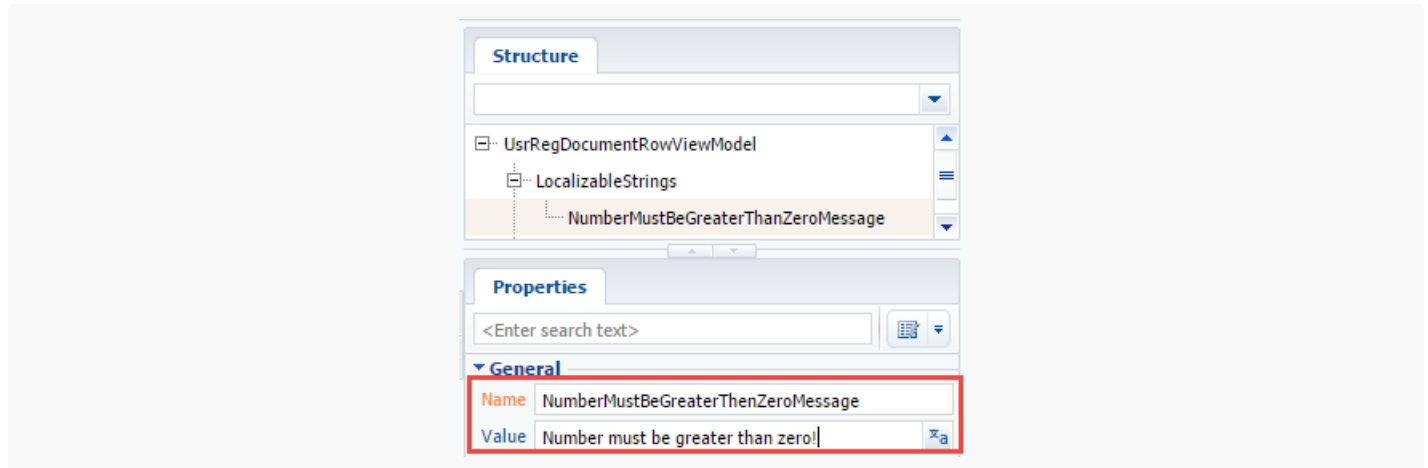
In the module designer, add the localizable string on the [ *Structure* ] tab of the opened

`UsrRegDocumentRowViewModel` schema with the following property values

(Fig. 3):

- [Name] – “NumberMustBeGreaterThanZeroMessage”.
- [Value] – “Number must be greater than zero!”.

Fig. 3. Case result



### Note.

A localized string is a schema resource. In order for its values to appear in the client part of the application, add the `UsrRegDocumentRowViewModelResources` resource module to the dependencies of the `UsrRegDocumentRowViewModel` module.

Save the schema to apply changes.

## 2. Adding the validation program logic

Add the following methods to the `UsrRegDocumentRowViewModel` module to implement the validation program logic:

- `validateNumberMoreThenZero()` – contains the validation logic of the field value.
- `setValidationConfig()` – connects the [Number] column and the `validateNumberMoreThenZero()` validation method.
- `Init()` – an overridden base method that calls the base logic and the `setValidationConfig()` method.

Source code of the modified schema:

```
define("UsrRegDocumentRowViewModel", ["UsrRegDocumentRowViewModelResources", "BaseFieldRowViewMc
function(resources) {
    Ext.define("Terrasoft.configuration.UsrRegDocumentRowViewModel", {
        extend: "Terrasoft.BaseFieldRowViewModel",
```

```

alternateClassName: "Terrasoft.UsrRegDocumentRowViewModel",
validateNumberMoreThenZero: function(columnValue) {
    var invalidMessage;
    if (columnValue < 0) {
        invalidMessage = resources.localizableStrings.NumberMustBeGreaterThanZeroMes
    }
    return {
        fullInvalidMessage: invalidMessage,
        invalidMessage: invalidMessage
    };
},
setValidationConfig: function() {
    this.addColumnValidator("UsrNumber", this.validateNumberMoreThenZero);
},
init: function() {
    this.callParent(arguments);
    this.setValidationConfig();
}
});
return Terrasoft.UsrRegDocumentRowViewModel;
});

```

Save the schema to apply changes.

When a negative value is entered in the [ *Number* ] field on the [ *History* ] tab of the contact edit page, a warning message will be displayed (Fig. 4).

Fig. 4. Case result

Series	Number	Action
AA	-1	×
BB	234,567	×
CC	345,678	×

## Adding a virtual record

When a detail is loaded, adding virtual records enables you to display a field edit card immediately, without pressing the [ *Add* ] button.

**That requires defining the `useVirtualRecord()` method (returns true) in the `UsrRegDocument..`**

```

useVirtualRecord: function() {

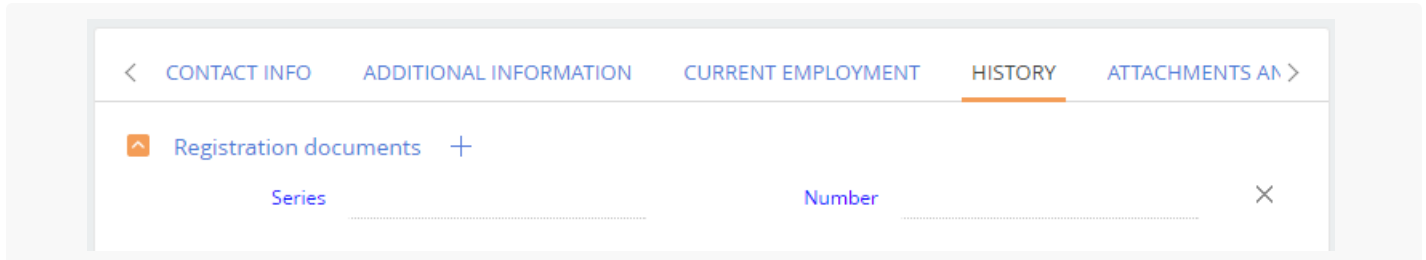
```



```
return true;
}
```

When opening a tab with a detail, a virtual record will be displayed (Fig. 5).

Fig. 5. Displaying a virtual record



## Adding the [Attachments] detail

 Advanced

### Introduction

The [ *Attachments* ] detail is designed for storing files and links to web resources and knowledge base articles related to a section record. The detail is available in all Creatio sections (see: “[Attachments](#)”). The main functionality of the detail is implemented in the `FileDetailV2` schema of the `UIv2` package.

To add a detail to the edit page of a custom section record, do the following:

1. Create a regular detail in the detail wizard using the object schema of the `Section object nameFile` section (see: “[Creating a new section](#)”).
2. Change the parent for the detail list schema.
3. In the wizard, add the detail to the the edit page of a custom section record.
4. Perform additional detail configurations.

### Case description

Add the [ *Attachments* ] details to the record edit page of the custom [ *Photos* ] section. All schemas of the [ *Photos* ] section and the [ *Attachments* ] details should be stored in a custom package (for example, `sdkDetailAttachment` ).

#### Note.

To create a section, use the section wizard (see: “[Creating a new section](#)” and “[Section wizard](#)”).

**Attention.**

If the development needs to be carried out in a [custom package](#), it needs to be specified in the [ *Current package* ] system setting. Otherwise, the wizard will save the changes to the [ *Custom* ] package.

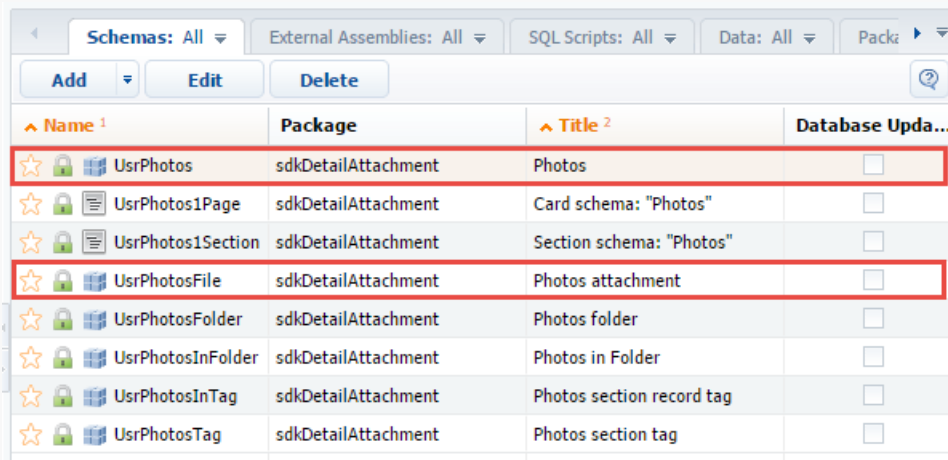
## Case implementation algorithm

### 1. Create a detail, using the [Section object name]File section object schema.

As a result (Fig. 1), a number of client modules and object schemas will be created in the custom package. The name of the schema of the main section object is `UsrPhotos` (Fig. 2). The schema name of the section object that you want to use for the details is `UsrPhotosFile`.

Fig. 1. The [ *Photos* ] section properties in the section wizard

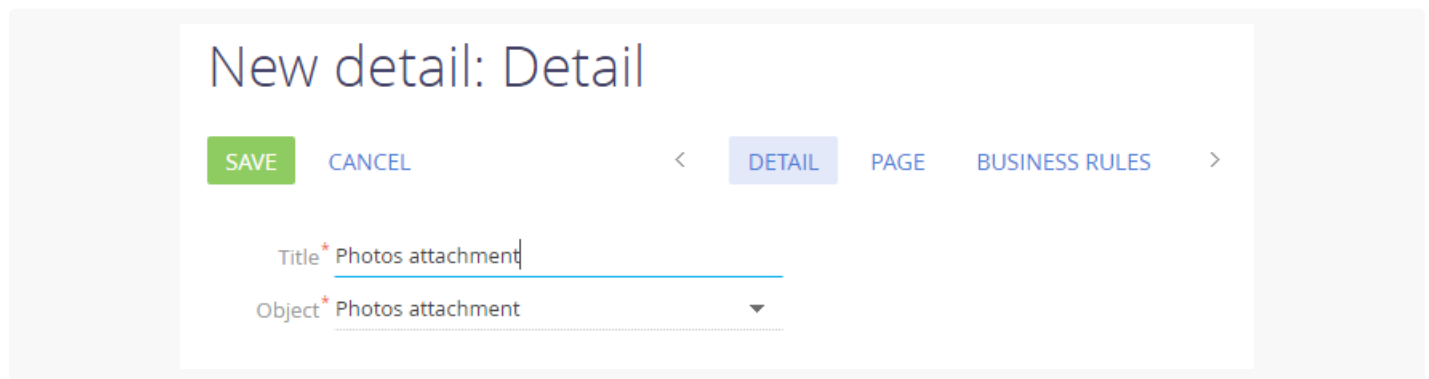
Fig. 2. Schemas created by the section wizard in the user package



Name <sup>1</sup>	Package	Title <sup>2</sup>	Database Upda...
UsrPhotos	sdkDetailAttachment	Photos	<input type="checkbox"/>
UsrPhotos1Page	sdkDetailAttachment	Card schema: "Photos"	<input type="checkbox"/>
UsrPhotos1Section	sdkDetailAttachment	Section schema: "Photos"	<input type="checkbox"/>
UsrPhotosFile	sdkDetailAttachment	Photos attachment	<input type="checkbox"/>
UsrPhotosFolder	sdkDetailAttachment	Photos folder	<input type="checkbox"/>
UsrPhotosInFolder	sdkDetailAttachment	Photos in Folder	<input type="checkbox"/>
UsrPhotosInTag	sdkDetailAttachment	Photos section record tag	<input type="checkbox"/>
UsrPhotosTag	sdkDetailAttachment	Photos section tag	<input type="checkbox"/>

Creating details in a wizard is described in more detail in the “[Creating a detail in wizards](#)” article. Choose the detail title and the [ *Photos attachment* ] object as the default one in the detail wizard (Fig. 3). The transition to the second step in the wizard is optional.

Fig. 3. Detail properties in the wizard



## New detail: Detail

SAVE
CANCEL
<
DETAIL
PAGE
BUSINESS RULES
>

Title\* Photos attachment

Object\* Photos attachment

As a result, the custom package will have a schema of the detail list client module and a schema of the detail edit page (Fig. 4).

Fig. 4. Schemas created by the section wizard in the user package

Schemas: All External Assemblies: All SQL Scripts: All Data: All Package Dependencies			
Add Edit Delete			
Name <sup>1</sup>	Package	Title <sup>2</sup>	Database Update Required
UsrPhotos	sdkDetailAttachment	Photos	<input type="checkbox"/>
UsrPhotos1Page	sdkDetailAttachment	Card schema: "Photos"	<input type="checkbox"/>
UsrPhotos1Section	sdkDetailAttachment	Section schema: "Photos"	<input type="checkbox"/>
UsrPhotosFile	sdkDetailAttachment	Photos attachment	<input type="checkbox"/>
UsrPhotosFolder	sdkDetailAttachment	Photos folder	<input type="checkbox"/>
UsrPhotosInFolder	sdkDetailAttachment	Photos in Folder	<input type="checkbox"/>
UsrPhotosInTag	sdkDetailAttachment	Photos section record tag	<input type="checkbox"/>
UsrPhotosTag	sdkDetailAttachment	Photos section tag	<input type="checkbox"/>
UsrSchema3Detail	sdkDetailAttachment	Detail schema: "Photos attachment"	<input type="checkbox"/>
UsrUsrPhotosFile1Page	sdkDetailAttachment	Card schema: "Photos attachment"	<input type="checkbox"/>

**Note.**

Schema names are generated automatically and may be different from those shown on Fig. 4.

## 2. Change the parent for the detail list schema.

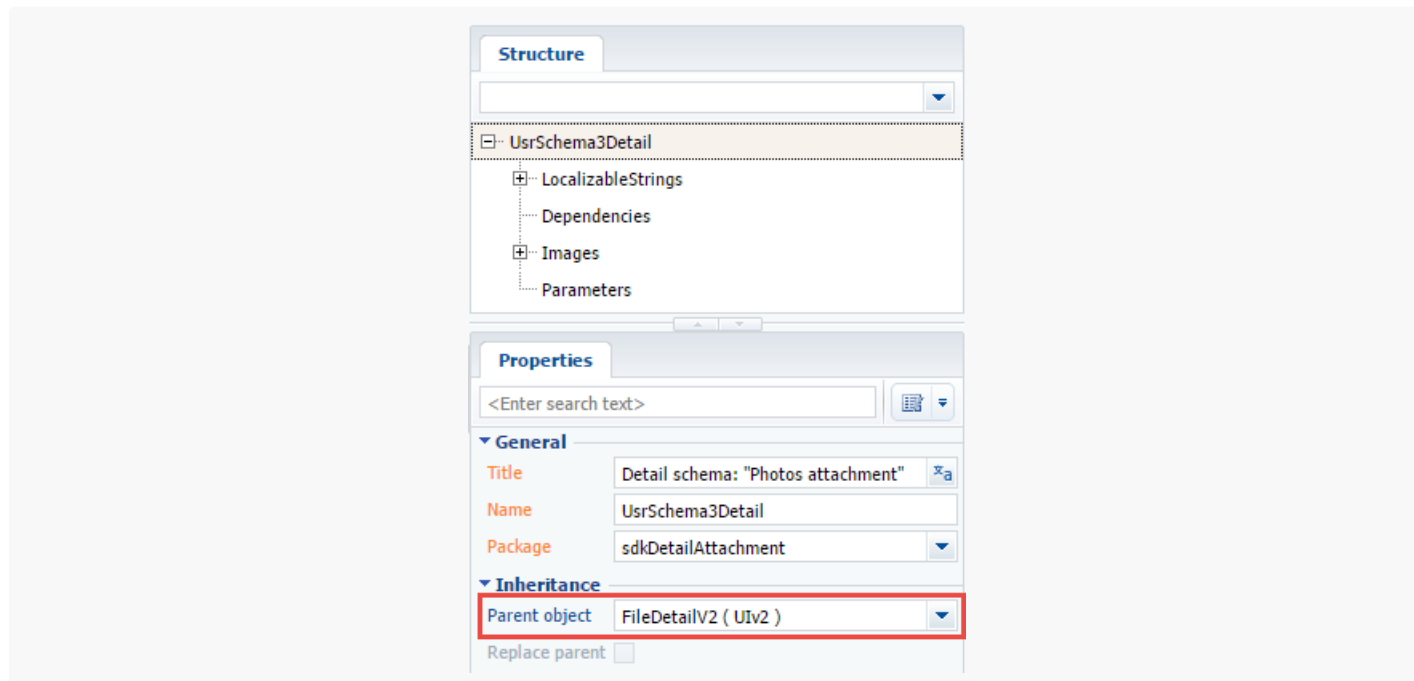
The detail with [an edit page](#) is created. The [ *Base schema - Detail with list* ] schema of the `NUI` package is the parent object of the client module schema of the `UsrSchema3Detail` detail list (Fig. 5).

Fig. 5. A default parent object

The screenshot shows the 'Properties' window for the 'UsrSchema3Detail' schema. The 'Structure' tab is active, showing a tree view with 'UsrSchema3Detail' as the root, containing 'LocalizableStrings', 'Dependencies', 'Images', and 'Parameters'. The 'Properties' tab is also visible, showing the 'General' section with fields for 'Title' (Detail schema: "Photos attachment"), 'Name' (UsrSchema3Detail), and 'Package' (sdkDetailAttachment). The 'Inheritance' section is expanded, showing 'Parent object' set to 'Base schema - Detail with list ( NUI )', which is highlighted with a red box. A 'Replace parent' checkbox is also present.

To implement the [ *Files and Links* ] detail functionality in the custom detail, specify the `FileDetailV2` schema as the parent object of the `UsrSchema3Detail` schema (Fig. 6).

Fig. 6. The parent object – the FileDetailV2 schema



**Note.**

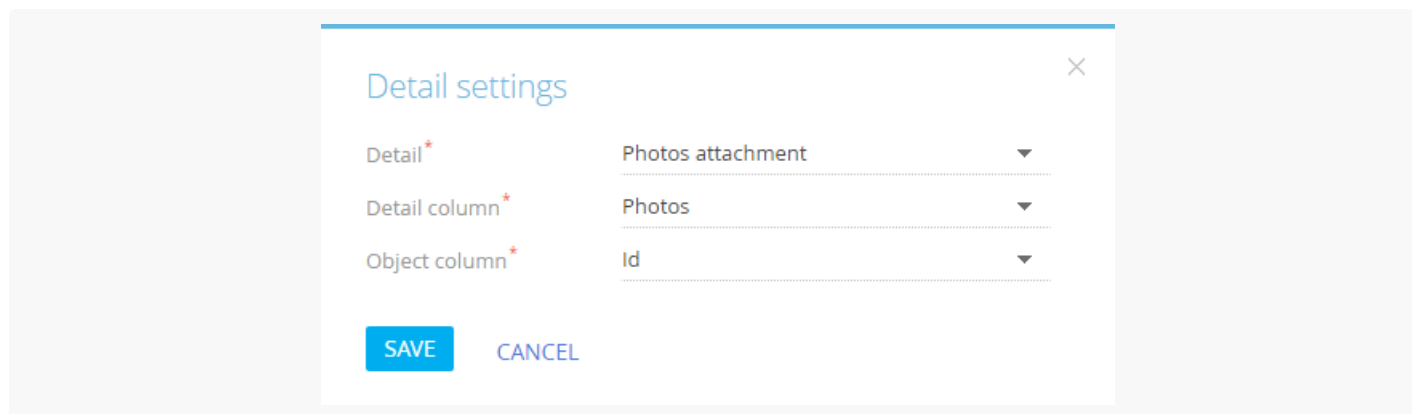
The procedure of specifying a parent object is covered in the [“Create a client schema”](#).

Save the schema to apply changes.

### 3. In the wizard, add the detail to the the edit page of a custom section record.

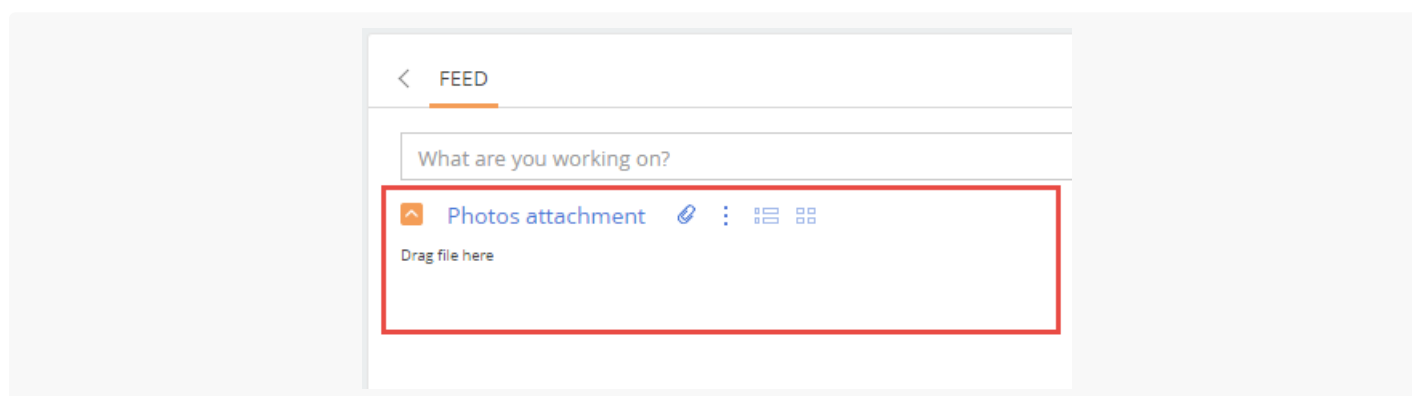
Adding a detail to the record edit page using the detail wizard is described in the [“Creating a detail in wizards”](#) article. When you add a detail in the wizard, configure the link between the detail columns and the main section object (Fig. 7).

Fig. 7. Configuring the link between the detail columns and the main section object



Upon refreshing, the detail will be displayed on a record edit page (Fig. 8).

Fig. 8. A detail on an edit page



#### Note.

You can add details to any edit page tab. Additionally, you can create a separate [ [Attachments](#) ] tab for the detail.

## 4. Perform additional detail configurations

After completing the previous steps, the detail is fully functional, but looks different from the [ *Attachments* ] detail of the base application sections. To make the custom detail look similar to the standard one, you need to define CSS styles for it.

#### Attention.

The client module schema of the `UsrSchema3Detail` detail list is a schema of the view model. Defining styles in it is impossible. It is necessary to create a new module schema, define the styles and add the created module to module dependencies of a detail.

[Create a new module schema](#) in a custom package with the following property values:

- [Title] - "UsrSchema3DetailCSS".

- [Name] - "UsrSchema3DetailCSS".
- [Package] - "sdkDetailAttachment".

Add the following CSS selectors to the LESS tab of the module designer:

```
div[id*="UsrSchema3Detail"] {
  .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;
        }
    }
}

```

**Note.**

The styles defined in the source code above almost completely coincide with the styles defined in the schema of the `FileDetailCssModule` module in the `Uiv2` package. They are intended for the `FileDetailV2` schema used as the parent object .

You can not use the `FileDetailCssModule` module directly, because the markers and identifiers of the HTML elements of the standard detail and the detail created by the wizard are different.

Save the schema to apply changes.

To use the created module and its styles in a detail `UsrSchema3Detail` detail schema, add it to the dependency of the module defined in the detail schema.

The source code of the modified schema:

```

div[id*="UsrSchema3Detail"] {
    .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;

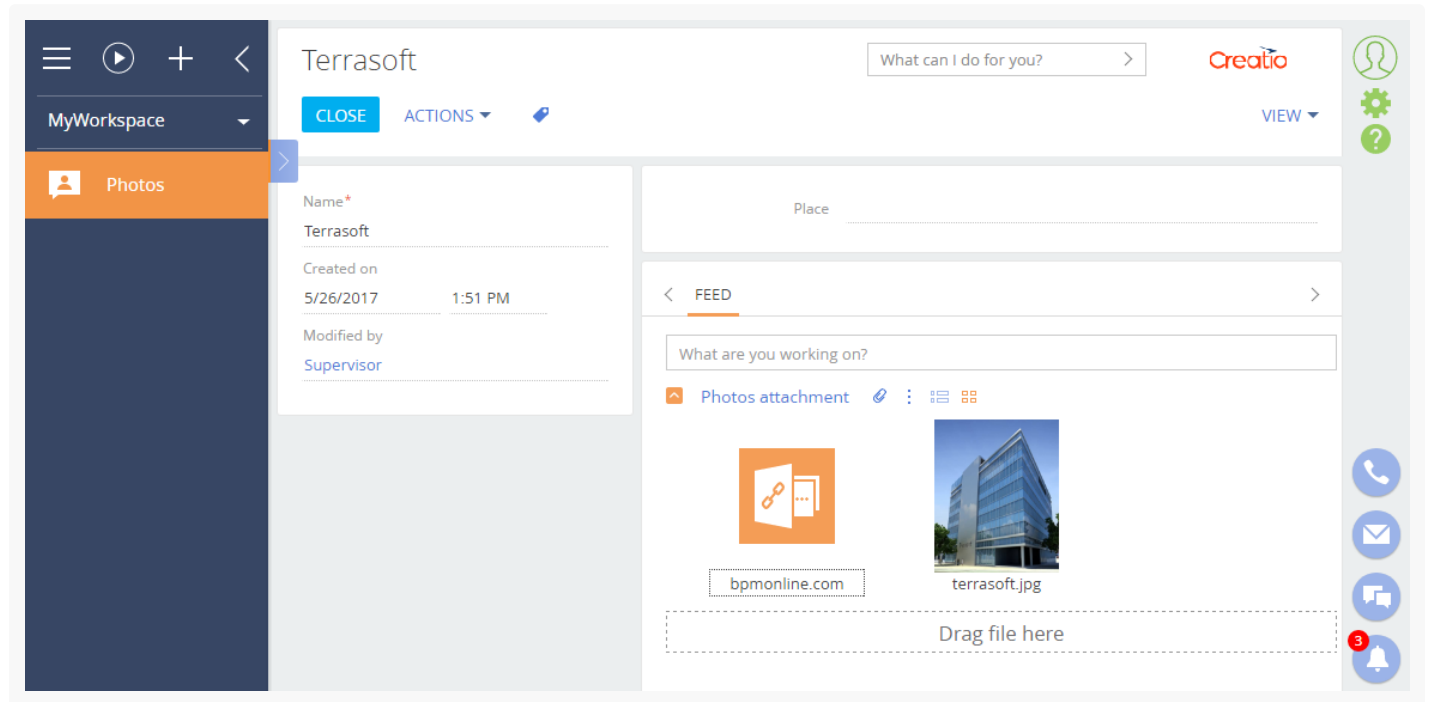
    .t-btn-image {
        background-position: 0 16px !important;
    }
}
div[data-item-marker="tiled"] {
    .tiled-mode-button {
        .button-pressed;
    }
}
div[data-item-marker="listed"] {
    .listed-mode-button {
        .button-pressed;
    }
}
}
}

```

Save the schema to apply changes.

As a result, the edit page of the custom [ *Photos* ] section will display an [ *Attachments* ] detail, almost identical to the base one (Fig. 9).

Fig. 9. Case result



## Displaying additional columns on the [Attachments] tab

 Advanced

### Introduction

The [ *Attachments* ] detail is designed for storing files and links to web resources and knowledge base articles related to the section record. The detail is available in all Creatio section (see: “[How to work with attachments and notes](#)”). The main functionality of the detail is implemented in the `FileDetailV2` scheme of the `UIV2` package.

By default, the [ *Attachments* ] detail is set to have only the [ *Name* ] and [ *Version* ] columns in the list view. The [ *Description* ] column is available while adding a new link. However, it is not displayed in the list view.

Fig. 1. The [ *Description* ] field

Andrew Baker (sa... What can I do for you? > Creatio

SAVE CANCEL

Name \* bpm'online.com

Description bpm'online web-page

**Note.**

The tile view of the [ *Attachments* ] detail only features the [ *Name* ] column and a file or a link.

Use the columns setup page to set up the detail's list columns (see: "[Setting up columns](#)"). However, the [ *Attachments* ] detail does not have this configuration option by default.

To add this configuration option, do the following:

- Create a replacing schema of the `FileDetailV2` detail.
- Call the method of opening the column configuration page in the replacing schema.

## Case description

Add a new [ *Columns setup* ] command to the [ *Actions* ] menu for the [ *Attachments* ] detail.

## Case implementation algorithm

### 1. Replace the FileDetailV2 detail schema

The procedure for creating a replacing client schema is covered in the "[Create a client schema](#)". Create a replacement schema with the following properties in a custom package:

- [Title]— "FileDetailV2".
- [Name] — "FileDetailV2".
- [Package] – the schema will be placed in this package after publishing. By default, this property contains the name of the package selected prior to creating a schema. It can be populated with any value from the drop-down list.
- [Parent object] – "FileDetailV2".

### 2. Call the method of opening the column configuration page in the replacing schema.

To do this, go to the [module description](#) in the source code of the replacement schema, and redefine the

base `getGridSettingsMenuItem()` method, which returns the detail menu item, associated with the call to the base `openGridSettings()` method defined in the `GridUtilities` mixin.

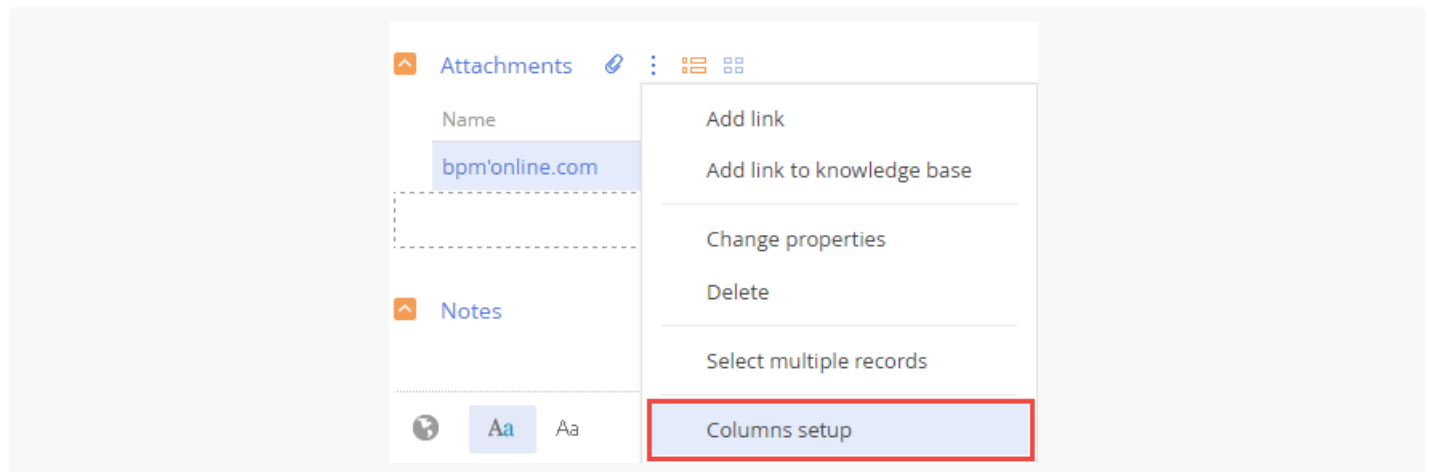
Schema source code:

```
define("FileDetailV2", [], function() {
  return {
    methods: {
      getGridSettingsMenuItem: function() {
        return this.getButtonItem({
          Caption: {"bindTo": "Resources.Strings.SetupGridMenuCaption"},
          Click: {"bindTo": "openGridSettings"}
        });
      }
    }
  };
});
```

Save the schema to apply changes.

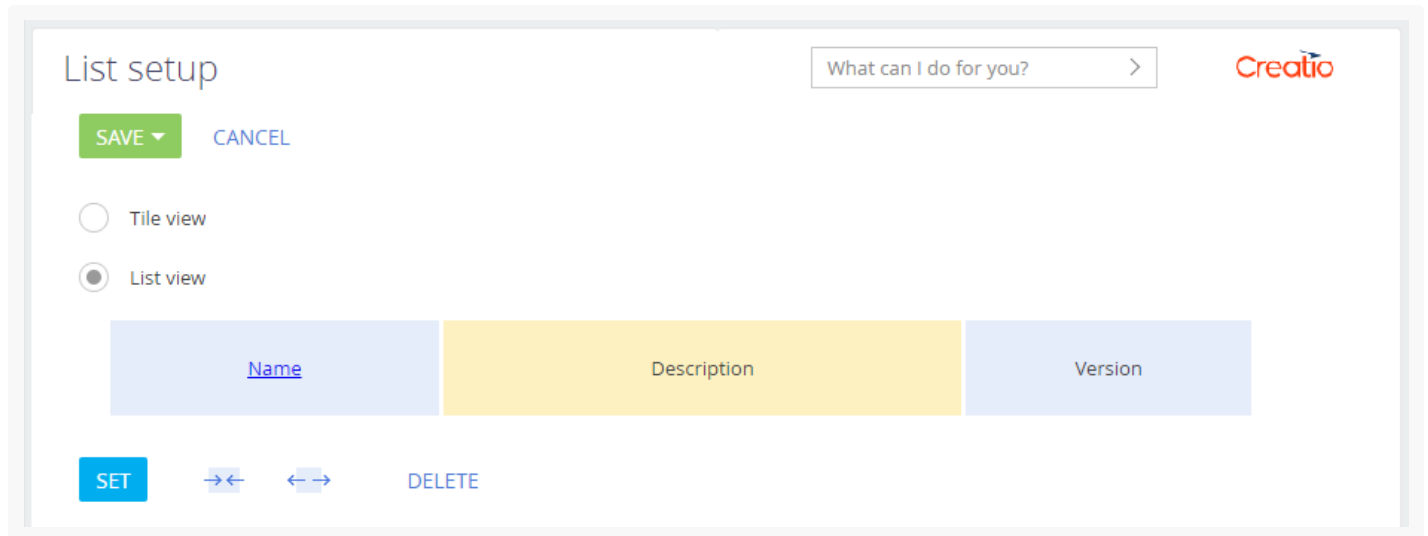
As a result, a new [ *Columns setup* ] command is displayed in the [ *Actions* ] menu (Fig. 2).

Fig. 2. The [ *Columns setup* ] command



This command enables the user to configure the list column view, and add the [ *Description* ] column to the detail.

Fig. 3. The column configuration page



The 'List setup' dialog box is shown. It has a search bar at the top right with the placeholder text 'What can I do for you?' and the Creatio logo. Below the search bar are 'SAVE' and 'CANCEL' buttons. Underneath are radio buttons for 'Tile view' and 'List view', with 'List view' being selected. A table preview is shown with three columns: 'Name' (blue header), 'Description' (yellow header), and 'Version' (blue header). At the bottom are 'SET', two arrow buttons, and a 'DELETE' button.

List setup

What can I do for you? >

SAVE CANCEL

☐ Tile view

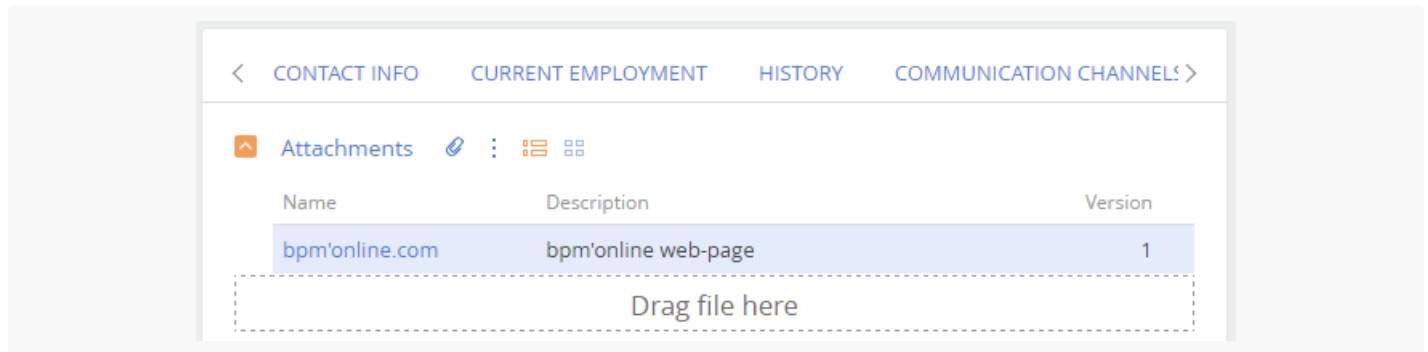
☒ List view

Name	Description	Version
------	-------------	---------

SET → ← DELETE

After saving the settings in the detail list view, the column will be displayed (Fig. 4).

Fig. 4. Case result



The screenshot shows the 'Attachments' tab in a case detail view. The top navigation bar includes 'CONTACT INFO', 'CURRENT EMPLOYMENT', 'HISTORY', and 'COMMUNICATION CHANNELS'. Below the tab header, there are icons for file upload, download, and view. The table has three columns: 'Name', 'Description', and 'Version'. A single row is visible with the values 'bpm'online.com', 'bpm'online web-page', and '1'. Below the table is a dashed box with the text 'Drag file here'.

Name	Description	Version
bpm'online.com	bpm'online web-page	1

Drag file here