Creatio Academy

Basic interface elements

Field

Version 7.18

Table of Contents

Field

Medium

You can add fields to the edit page in two ways:

1. Via section wizard (see the "Section wizard", "How to set up page fields" articles).

A base object (for example, the [*Activity*] object) replacing schema and a base edit page (for example, the `ActivityPageV2`) replacing schema will be created in the custom package as a result of the section wizard operation. A new column will be added in the object replacing schema. A configuration object with the settings of the new field location on a page will be added to the `diff` array in the edit page replacing schema.

Note.

When creating new sections via the wizard, it will create new schemas instead of replacing schemas in your current custom package.

You should implement additional edit page business logic or develop new client interface elements in the created edit page replacing schema.

2. Via creating replacing base object and replacing base page by developer tools.

Types of fields

Field with an image

There are certain peculiarities of adding a field with image (contact's photo, product picture, account logo, etc.) to an edit page:

1. The object column used for a field with image should have the "Link to image" type. Specify it as the [Image] system column of the object.
2. Add the default image to the edit page image schema collection.
3. A field with image is added to the `diff` edit page schema array with usage of the additional `image-edit-container` CSS-class container-wrapper.
4. The `values` property of the configuration object containing settings of a field with image must include the following properties:
   - `getSrcMethod` – method receiving image by link
   - `onPhotoChange` – method called upon image modification
   - `Readonly` – property defining the capability of image editing (changing, deleting)
   - `Generator` – control element generator. Indicate the `ImageCustomGeneratorV2.generateCustomImageControl` for the field with image

- `beforeFileSelected` – method called before opening the image selection dialog box

5. Add the following to the edit page schema collection:

- method receiving image by link
- method called upon image modification
- method saving the link to a modified image in the object column
- method called before opening the image selection dialog box

Calculated field

A calculated field is a page control element, whose value is generated based on the status and values in other elements on this page.

In Creatio, calculated fields are based on the Creatio client mechanism, which uses subscriptions to change events in view model schema attributes. For any attribute, you can set a configuration object and specify object schema column names. If the values in these columns change, the value of the calculated column will be updated. You can also specify the handler method for this event.

The general sequence of adding a calculated field is as follows:

1. Add a column for storing the values of the calculated field to the page object schema.
2. In the page view model, set up attribute dependencies by specifying column names from which the attribute depends and the handler name.
3. Add the implementation of the handler method to the method collection of the view model.
4. Set up the display of the calculated field in the `diff` property of the view model.

Setting up dependencies of the calculated field

In the `attributes` view model property, add an attribute for which the dependency is configured.

Declare a `dependencies` property, which is an array of configuration objects, each of which contains the following properties:

- `Columns` – an array of columns, whose values determine the value of the current column.
- `methodName` – handler method name.

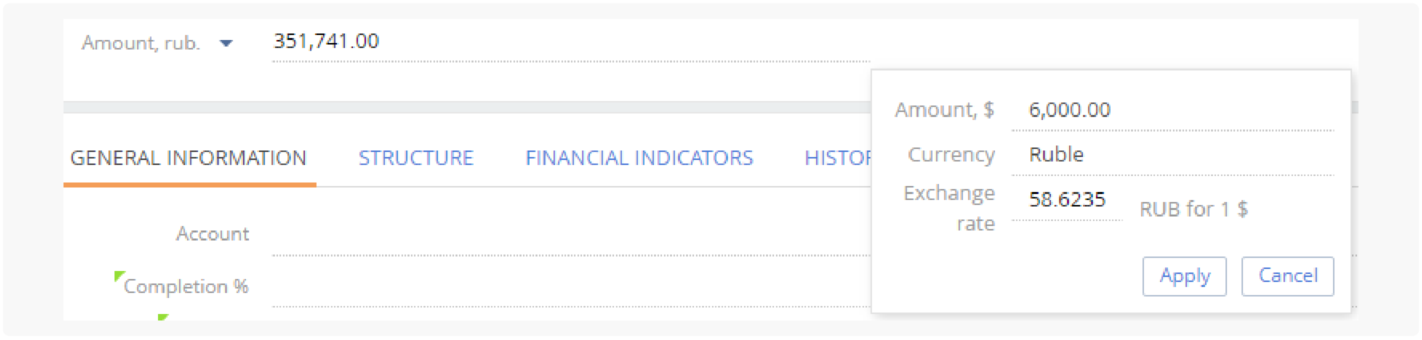
If the value of at least one of these columns changes in the view model, the event handler method (whose name is specified in the `methodName` property) will be called.

The handler method implementation must be added to the collection of the view methods.

Multi-currency field

One of the common Creatio configuration tasks is adding a [*Multi-currency field*] control on a page. This control enables users to specify the currency when entering monetary sums. It also enables fixing the sum equivalent in the base currency specified in the system settings. The sum is automatically converted as per the exchange rate when the currency changes. Fig.1 displays a multi-currency field in the Creatio interface.

Fig. 1. Multi-currency field



To add a multi-currency field on an edit page:

1. Add 4 columns to the object schema:

- [Currency] lookup column
- [Exchange rate] column
- [Amount] column to store the total sum in the selected currency
- [Amount in the base currency] column to store the sum in the base currency

Note.

The object schema should only contain one column – to store the total sum in the selected currency. The rest of columns can be virtual, unless the business task requires their values to be stored in the database. They can be determined as attributes in the view model schema.

2. Specify the following 3 modules as dependencies in declaring the view model class:

- `MoneyModule`,
- `MultiCurrencyEdit`,
- `MultiCurrencyEditUtilities`.

3. Connect the `Terrasoft.MultiCurrencyEditUtilities` mixin to the view model and initialize it in the `init()` overridden method.

4. Add a configuration object with the multi-currency field settings to the `diff` array of the edit page view model schema. In addition to common control properties, the `values` property must contain:

- `primaryAmount` – name of the column that contains the amount in the base currency
- `currency` – name of the column with reference to the currency lookup
- `rate` – name of the column that contains the currency exchange rate
- `generator` – control generator Specify `MultiCurrencyEditViewGenerator.generate` for a multi-currency field.

5. Add logic for recalculating the sum according to the currency. Apply the calculated field mechanism, as described in the Adding calculated fields article.

Actions on fields

Add the field validation

Validation is the verification of field values for their compliance with certain requirements. Values of the Creatio page fields are validated at the level of the page view model columns. The logic of the field value validation is implemented in the custom validation method.

Validator is the method of the view model where values of the view model column are analyzed for compliance with business requirements. This method must return validation results as an object with the following property:

- `invalidMessage` – a message string displayed under the field when making an attempt to save a page with an invalid field value and in the data window when saving a page with the field that did not passed validation.

If the value validation is successful, the validator method returns the object with empty string.

To start the field validation, the corresponding view model column must be bound to a specific validator. For this purpose, override the `setValidationConfig()` base method and call the `addColumnValidator()` method in it.

The `addColumnValidator()` method accepts two parameters:

- name of the view model column, to which the validator is bound.
- name of the column value validator method.

Attention.

If the field is validated in the replacement client schema of the base page, the parent implementation of the `setValidationConfig()` method must be called before calling the `addColumnValidator()` method to correctly initialize validators of the base page fields.

To add validation of field values:

1. Add the validator method to the collection of methods of the view model that will check a field value.
2. Override the `setValidationConfig()` method and connect the validator to the corresponding view model column in it.

Set a default value for a field

In Creatio, you can define the default values for edit page control elements.

You can set a default value in two ways:

1. Set the value on the business object column level. When creating a new object, certain page fields should be populated with some initially known values. In such cases, indicate these values for the corresponding object columns as the default values in object designer.

Types of default values.

Name	Description
`Set constant`	String, number, lookup value, Boolean.
`Set from system setting`	The complete list of system settings is available in the [System settings] section. It can be supplemented with custom system settings.
`Set from system variable`	Creatio system variables are global variables that store information about system-wide setting values. Unlike system settings, whose values can differ depending on different users, system variable values always remain the same for all users. The full list of system variables is implemented on the kernel level and cannot be changed by user: • New identifier • New sequential identifier • Current user • Contact of the current user • Account of the current user • Current date and time value • Current date value • Current time value
`Default value not set`	

2. Specify in the edit page source code. In some cases, it is impossible to set a default value via the object column properties. For example, these can be estimated values which are calculated by other column values of the object, etc. In such a case, you can set a default value only via programming means.

Use filtration for lookup fields

There are two methods of using the filtration in Creatio for lookup fields of the edit page:

1. The [FILTRATION] business rule.
2. Explicit indication of filters in the column description of the attributes model property.

The use of the [*FILTRATION*] business rule is expedient if a simple filter by a specific value or attribute must be used for the field. The business rules are detailed in the Setting the edit page fields using business rules. The detailed case for using the [*FILTRATION*] business rule is set forth in the The FILTRATION rule use case article.

If arbitrary filtration (sorting and addition of supplementary columns to a query when a drop-down list is displayed) is required, the explicit description should be used in the `attributes` model property.

Setting lookup field filters in the `attributes` model property:

1. The name of the column for which filters are set must be added to the `attributes` property of the view model.
2. The `lookupListConfig` property must be declared for this column. It represents a configuration item containing the following properties (not required):

   - `columns` – an array of column names to be added to a request in addition to Id and the primary display column.
   - `orders` – an array of configuration objects determining the data sorting when displayed.
   - `filter` – the method for returning the object of the `Terrasoft.BaseFilter` class or its inheritor, will be applied, in turn, to a request.
   - or `filters` – an array of filters (methods for returning collections of the `Terrasoft.FilterGroup` class).

Filters are added to a collection using the `add()` method which has the following parameters:

Name	**Data type**	**Description**
`key`	String	key
`item`	Mixed	Element.
`index`	Number	Index for insert. If not entered, the index to be inserted is not rated.

The object of the `Terrasoft.BaseFilter` class or its inheritor is the `item` parameter. The methods for creating filters with descriptions are given in Table 1 of the Filters handling article.

Attention.

Filters are combined by default in the collection using the AND logic operator. If the OR operator is to be used, this must be indicated explicitly in the `logicalOperation` property of the `Terrasoft.FilterGroup` object.

Block fields

During the development of the Creatio custom functions you may need to block all fields and details on the page when specific condition is met. Mechanism of blocking the edit page fields can simplify the process without creating a number of business rules.

More information about blocking of the page fields can be found in the Block fields article.

Attention.

Blocking mechanism is implemented in the Creatio version 7.11.1 or higher.

Add auto-numbering to the edit page field

You can add autonumbering for an object column. For instance, auto numbering is applied in the [*Documents*], [*Invoices*] and [*Contracts*] sections where a preformatted number is automatically generated when you add a new record.

There are two ways of implementing auto-numbering:

- Client-side implementation.
- Server-side implementation.

To implement auto-numbering `on the client side`, override the base virtual method `onEntityInitialized()` and call the `getIncrementCode()` method of the edit page base schema `BasePageV2`.

The `getIncrementCode()` method accepts two parameters:

- `callback` – the function that will run after receiving service response. The response must be passed to the corresponding column (attribute);
- `scope` – the context where the `callback` function will be run (optional parameter).

To implement auto-numbering `on the server side`, add the event handler [*Before record adding*] to the object, whose column will be auto-numbered. Set up number generating parameters in the business process, namely:

- Indicate the schema of the object for which generation will be performed.
- Call the [Generate ordinal number] action.
- Pass the generated value to the necessary object column.

Note.

This is not the only way to implement auto-numbering on the server-side. It can be implemented via custom means, for instance, by creating a custom service.

Regardless of the chosen solution, add two system settings to use auto-numbering:

- `[Entity]CodeMask` – object number mask.
- `[Entity]LastNumber` – current object number.

`Entity` – is the name of the object, whose column will be auto-numbered. For example, `InvoiceCodeMask` (Invoice number mask) and `InvoiceLastNumber` (Current invoice number).

Add a new field

Medium

Source code

You can download the package with case implementation using the following link.

Example 1

Case description

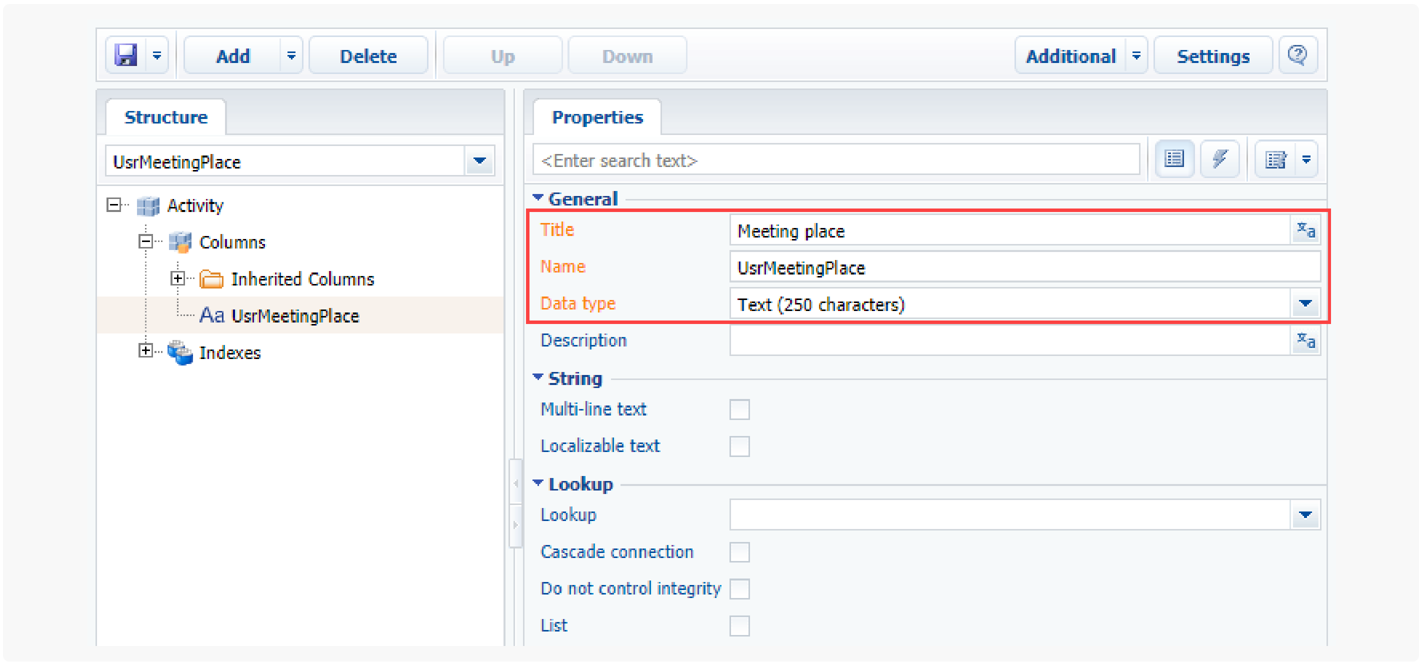
Manually add a new [*Meeting place*] field to the activity edit page.

Case implementation algorithm

1. Create a replacing object and add a new column to it.

Create an [*Activity*] replacing object and add the new [*Meeting place*] column of the “string” type to it (Fig. 1). Learn more about creating a replacing object schema in the “Create the entity schema” article.

Fig. 1. Adding a custom column to the replacing object

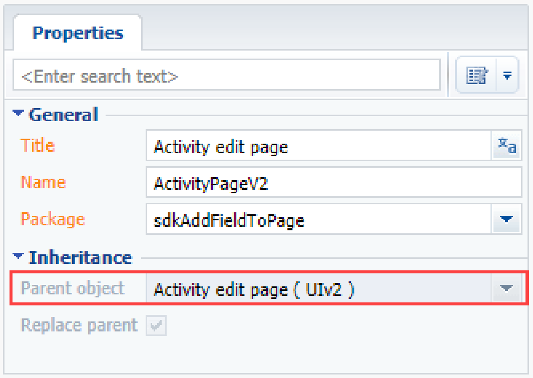


Title	Meeting place
Name	UsrMeetingPlace
Data type	Text (250 characters)
Description	

2. Create a replacing client module for the activity page

Create a replacing client module and specify the [*Activity edit page*] schema as parent object (Fig. 2). The procedure of creating a replacing page is covered in the “Create a client schema” article.

Fig. 2. Replacing edit page properties



General

Title	Activity edit page
Name	ActivityPageV2
Package	sdkAddFieldToPage

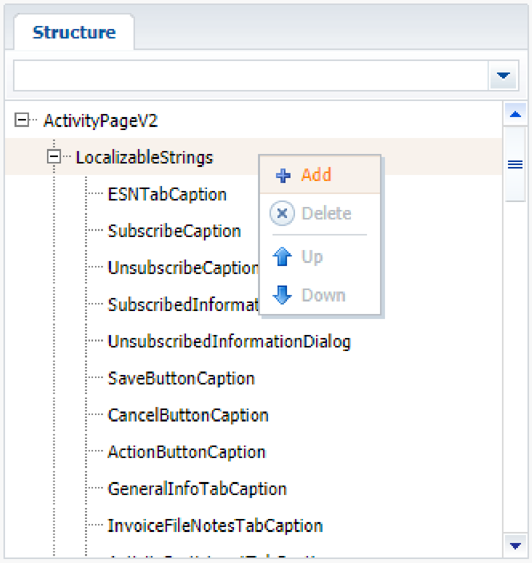
Inheritance

Parent object	Activity edit page (UIv2)
Replace parent	✓

3. Add a localized string with the field caption.

Add a string containing the added field caption to the localized string collection of the replacing page schema (fig.3).

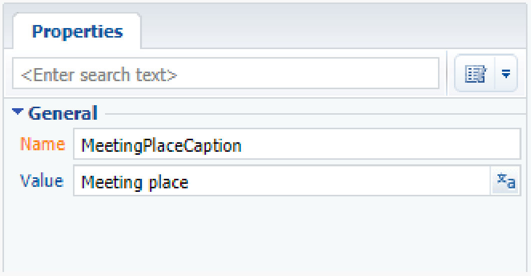
Fig. 3. Adding localized string to the schema



For the created string specify (Fig. 4):

- [Name] – "MeetingPlaceCaption";
- [Value] – “Meeting place”.

Fig. 4. Properties of the custom localized string



General

Name	MeetingPlaceCaption
Value	Meeting place

4. Add a new field to the activity edit page.

Add a configuration object containing the settings of the [*Meeting place*] field location on the page to the diff array.

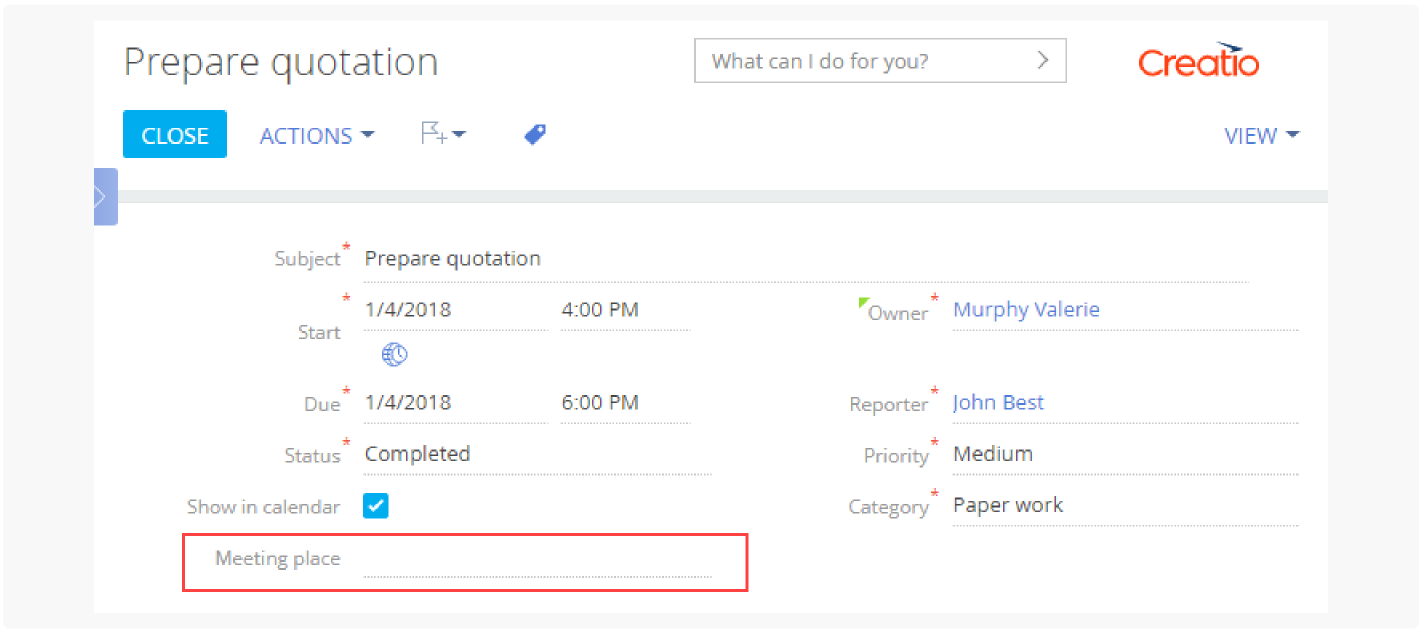
More information about the `diff` array properties is available in the "The "diff" array" article.

The replacing schema source code is as follows:

```
define("ActivityPageV2", [], function() {
    return {
        // Name of the edit page object schema.
        entitySchemaName: "Activity",
        // Displaying of a new field on the edit page.
        diff: /**SCHEMA_DIFF*/[
            // Meta data for adding the [Meeting place] field.
            {
                // Operation of adding a component to the page.
                "operation": "insert",
                // Meta name of a parent container where a field is added.
                "parentName": "Header",
                // The field is added to the component collection
                // of a parent element.
                "propertyName": "items",
                // The name of a schema column that the component is linked to.
                "name": "UsrMeetingPlace",
                "values": {
                    // Field caption.
                    "caption": {"bindTo": "Resources.Strings.MeetingPlaceCaption"},
                    // Field location.
                    "layout": {
                        // Column number.
                        "column": 0,
                        // String number.
                        "row": 5,
                        // Span of the occupied columns.
                        "colSpan": 12
                    }
                }
            }
        ]/**SCHEMA_DIFF*/
    };
});
```

After you save the schema and update the application page with clearing the cache, you will see a new field appear on the activity edit page (fig.5).

Fig. 5. Case result demonstration


Prepare quotation

Subject: Prepare quotation

Start: 1/4/2018 4:00 PM

Owner: Murphy Valerie

Due: 1/4/2018 6:00 PM

Reporter: John Best

Status: Completed

Priority: Medium

Show in calendar: ✓

Category: Paper work

Meeting place:

Example 2

Case description

Manually add a [*Country*] field to the contact profile edit page. The difference of this case is that you already have the [*Country*] column in your object schema.

Case implementation algorithm

1. Create a replacing contact page

Create a replacing client module and specify the [*Display schema — Contact card*], `ContactPageV2` schema as parent object. The procedure of creating a replacing page is covered in the“Create a client schema” article.

2. Add the [Country] field to the contact profile.

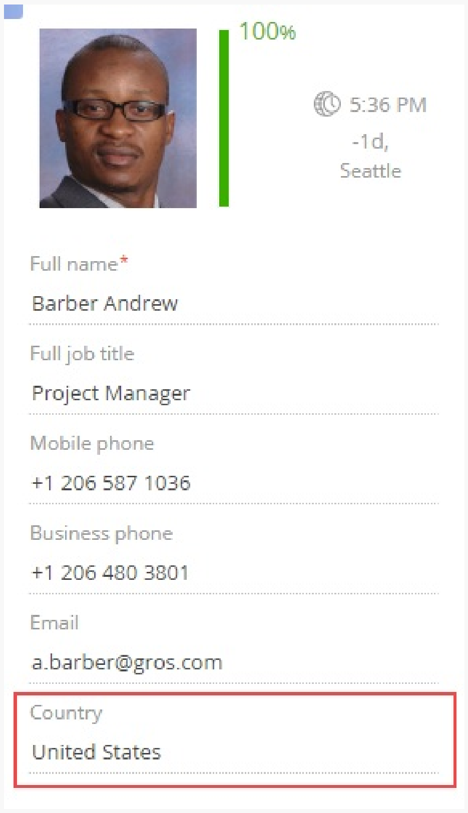
Add a configuration object containing the field property settings to the diff array. Indicate the `ProfileContainer.` element as a parent schema element where the field will be located.

The replacing schema source code is as follows:

```
define("ContactPageV2", [], function() {
    return {
        // Name of the edit page object schema.
        entitySchemaName: "Contact",
        diff: [
            // Meta data for adding the [Country] field.
            {
                // Operation of adding a component to the page.
                "operation": "insert",
                // Meta name of a parent container where a field is added.
                "parentName": "ProfileContainer",
                // The field is added to the component collection
                // of a parent element.
                "propertyName": "items",
                // The name of a schema column that the component is linked to.
                "name": "Country",
                "values": {
                    // Field type — lookup.
                    "contentType": Terrasoft.ContentType.LOOKUP,
                    // Field location.
                    "layout": {
                        // Column number.
                        "column": 0,
                        // String number.
                        "row": 6,
                        // Span of the occupied columns.
                        "colSpan": 24
                    }
                }
            }
        ]
    };
});
```

After you save the schema and update the application page with clearing the cache, you will see a new field appear on the contact edit page (fig.6).

Fig. 6. Case result demonstration



Full name*
Barber Andrew

Full job title
Project Manager

Mobile phone
+1 206 587 1036

Business phone
+1 206 480 3801

Email
a.barber@gros.com

Country
United States

Add a field with an image

Advanced

Case description

Adding a field with logo to the knowledge base article edit page.

Source code

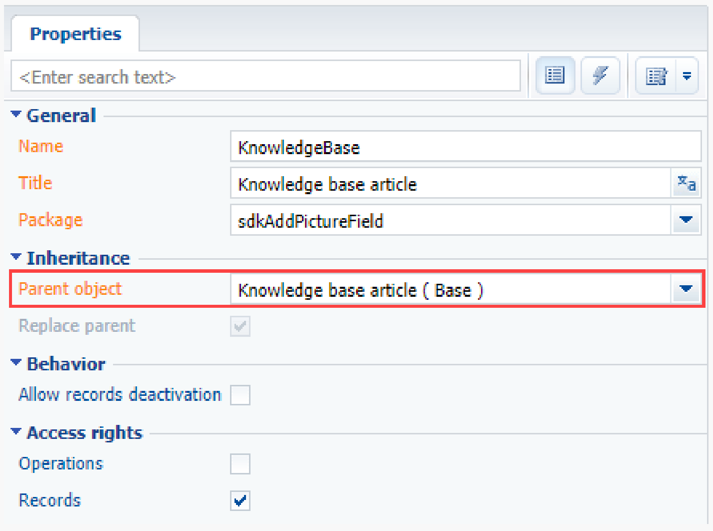
You can download the package with case implementation using the following link..

Case implementation algorithm

1. Creating the [Knowledge base] replacing object.

Create the [*Knowledge base article*] replacing object (Fig.1). Learn more about creating a replacing object in the "Create the entity schema" article.

Fig. 1. Properties of the object replacing schema



Properties

General

Name	KnowledgeBase
Title	Knowledge base article
Package	sdkAddPictureField

Inheritance

Parent object	Knowledge base article (Base)
Replace parent	✓

Behavior

Allow records deactivation	

Access rights

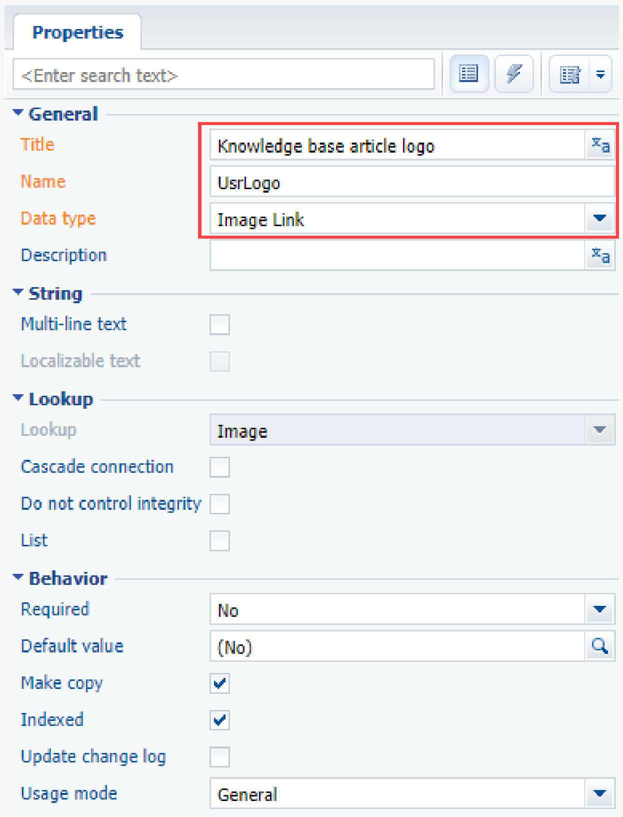
Operations	
Records	✓

2. Adding a new column to the replacing object.

For the created column specify (Fig. 2):

- [Title] – "Knowledge base article logo"
- [Name] – "UsrLogo"
- [Data type] – "Image Link"

Fig. 2. Adding a custom column to the replacing object



Properties

General

Title	Knowledge base article logo
Name	UsrLogo
Data type	Image Link
Description	

String

Multi-line text	
Localizable text	

Lookup

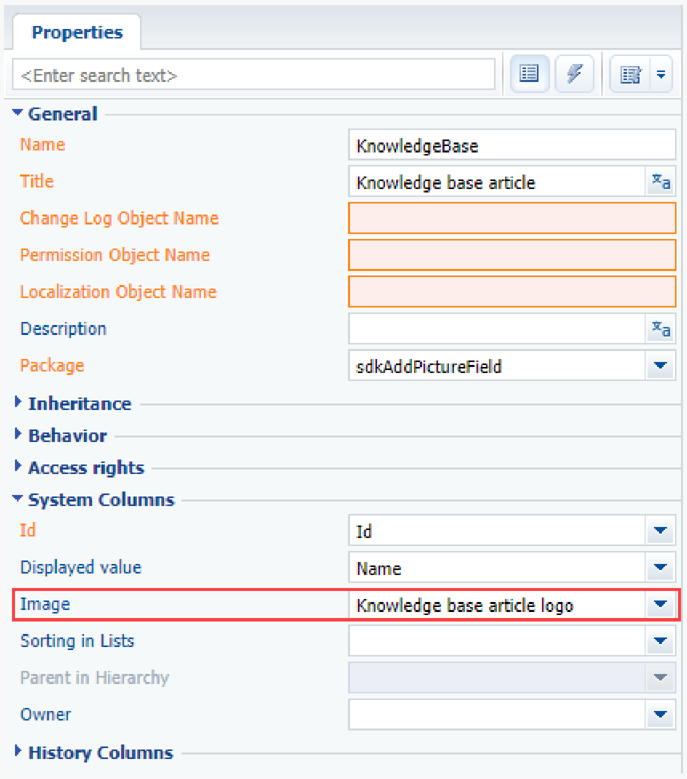
Lookup	Image
Cascade connection	
Do not control integrity	
List	

Behavior

Required	No
Default value	(No)
Make copy	✓
Indexed	✓
Update change log	
Usage mode	General

Indicate the created column as the [*Image*] object system column (Fig.3).

Fig. 3. Setting up the created column as the system column



Properties

General

Name	KnowledgeBase
Title	Knowledge base article
Change Log Object Name	
Permission Object Name	
Localization Object Name	
Description	
Package	sdkAddPictureField

Inheritance

Behavior

Access rights

System Columns

Id	Id
Displayed value	Name
Image	Knowledge base article logo
Sorting in Lists	
Parent in Hierarchy	
Owner	

History Columns

Note.

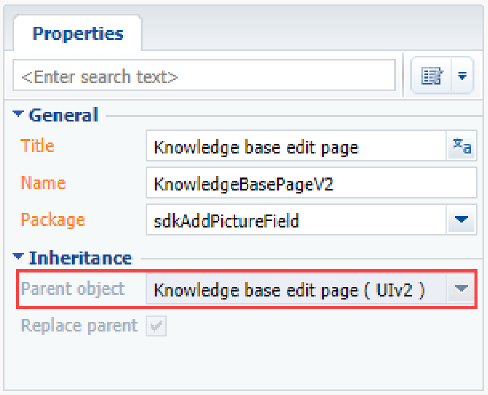
To view all object properties, switch to the object property advanced view mode.

Save and publish the object schema after you set up all properties.

3. Creating a replacing client module for the edit page.

Create a replacing client module and specify the [*Knowledge base edit page*] (`KnowledgeBasePageV2`) as the parent object in it (Fig. 4). The procedure of creating a replacing page is covered in the“Create a client schema” article.

Fig. 4. Properties of the replacing edit page


Properties

<Enter search text>

General

Title: Knowledge base edit page

Name: KnowledgeBasePageV2

Package: sdkAddPictureField

Inheritance

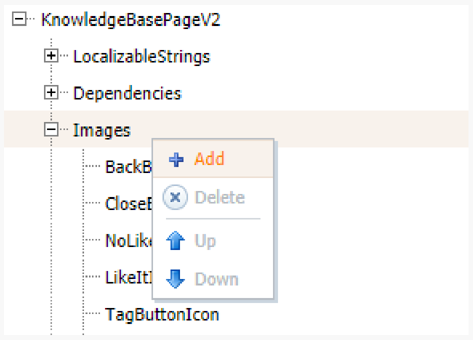
Parent object: Knowledge base edit page (UIv2)

Replace parent ✓

4. Adding a default image to the [Images] resources of the edit page schema.

Add the default image to the page replacing schema image collection (Fig.5).

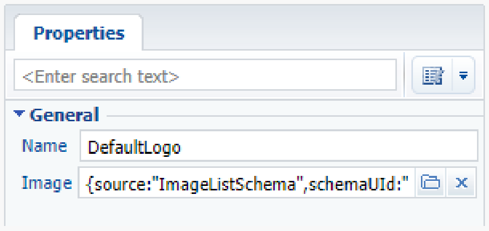
Fig. 5. Adding default image to the image schema resources



For the created image specify (Fig. 6):

- [Name] – "DefaultLogo"
- [Image] – file containing the default image (Fig.7)

Fig. 6. Schema resource properties


Properties

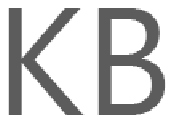
<Enter search text>

General

Name: DefaultLogo

Image: {source:"ImageListSchema",schemaUId:"

Fig. 7. Default image for the knowledge base article



5. Setting up displaying of a field with logo on the edit page

The field with logo should be placed in the upper part of the account edit page. In the base implementation the fields are placed in such a way that adding a logo can violate the page interface. That is why you need to rearrange the location of existing fields when locating a new field.

Add the configuration object of the filed with logo with the necessary parameters to the `diff` array property of the view model (see the source code below) and describe the modifications of the fields located in the upper part of the page: [*Name*], [*ModifiedBy*] and [*Type*].

The field with image is added to the page by using the additional `PhotoContainer` container-wrapper with the ["*image-edit-container*"] class.

6. Adding implementation of the following methods to the page view model method collection:

- `getPhotoSrcMethod()` – receives image by link
- `beforePhotoFileSelected()` – is called before opening the image selection dialog box
- `onPhotoChange` – is called upon image modification
- `onPhotoUploaded()` – saves the link to a modified image in the object column

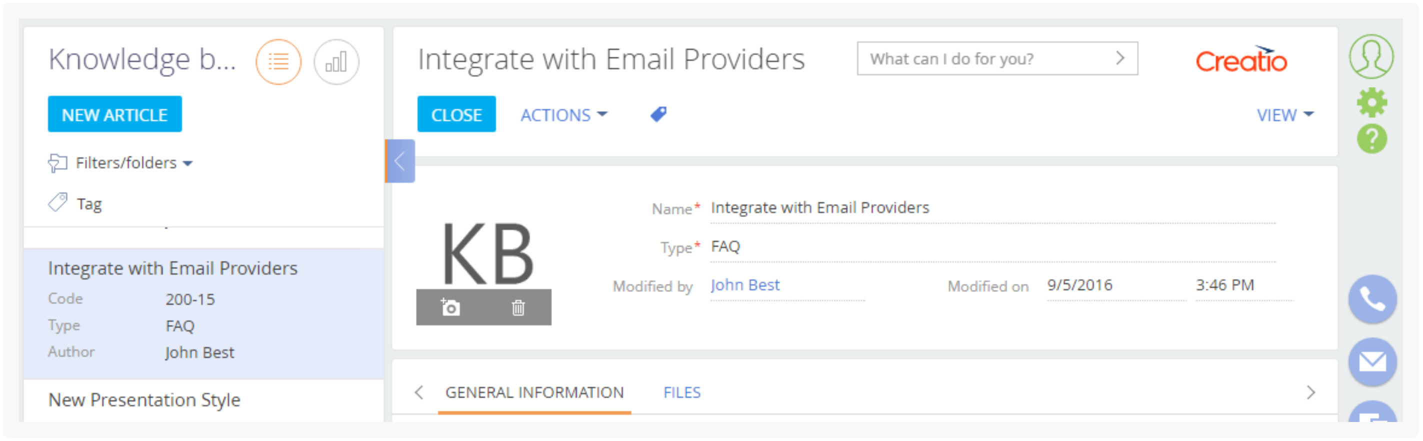
The replacing schema source code is as follows:

```
define("KnowledgeBasePageV2", ["KnowledgeBasePageV2Resources", "ConfigurationConstants"],
    function(resources, ConfigurationConstants) {
        return {
            // Name of the edit page object schema.
            entitySchemaName: "KnowledgeBase",
            // Edit page view model methods.
            methods: {
                // Called before opening the image selection dialog box.
                beforePhotoFileSelected: function() {
                    return true;
                },
                // Receives image by link.
                getPhotoSrcMethod: function() {
                    // Receiving a link to the image in the object column.
                    var imageColumnValue = this.get("UsrLogo");
                    // If the link is set, the method returns the url of the image file.
                    if (imageColumnValue) {
```

```
                    return this.getSchemaImageUrl(imageColumnValue);
                }
                // If the link is not set, it returns the default image.
                return this.Terrasoft.ImageUrlBuilder.getUrl(this.get("Resources.Images.Defa
            },
            // Processes the image modification.
            // photo – image file.
            onPhotoChange: function(photo) {
                if (!photo) {
                    this.set("UsrLogo", null);
                    return;
                }
                // The file is uploaded to the database. onPhotoUploaded is called when uplo
                this.Terrasoft.ImageApi.upload({
                    file: photo,
                    onComplete: this.onPhotoUploaded,
                    onError: this.Terrasoft.emptyFn,
                    scope: this
                });
            },
            // Saves the link to a modified image.
            // imageId – Id of the saved file from the database.
            onPhotoUploaded: function(imageId) {
                var imageData = {
                    value: imageId,
                    displayValue: "Image"
                };
                // The image column is assigned a link to the image.
                this.set("UsrLogo", imageData);
            }
        },
        //
        diff: /**SCHEMA_DIFF*/[
            // Container-wrapper that the component will be located in.
            {
                // Adding operation.
                "operation": "insert",
                // Parent container meta-name, where the component is added.
                "parentName": "Header",
                // The image is added to the component collection of the
                // parent container.
                "propertyName": "items",
                // Schema component meta-name, involved in the action.
                "name": "PhotoContainer",
                // Properties passed to the component structure.
                "values": {
                    // Element type – container.
                    "itemType": Terrasoft.ViewItemType.CONTAINER,
                    // CSS-class name.
                    "wrapClass": ["image-edit-container"],
                    // Locating in the parent container.
                    "layout": { "column": 0, "row": 0, "rowSpan": 3, "colSpan": 3 },
                    // Child element array.
                    "items": []
                }
            },
            // The [UsrLogo] field – the field with account logo.
            {
                "operation": "insert",
                "parentName": "PhotoContainer",
                "propertyName": "items",
                "name": "UsrLogo",
                "values": {
                    // Method receiving image by link.
                    "getSrcMethod": "getPhotoSrcMethod",
                    // Method called upon image modification.
                    "onPhotoChange": "onPhotoChange",
                    // Method called before opening the image selection dialog box.
                    "beforeFileSelected": "beforePhotoFileSelected",
                    // Property defining the capability of image editing (changing, deleting
                    "readonly": false,
                    // Control element view-generator.
                    "generator": "ImageCustomGeneratorV2.generateCustomImageControl"
                }
            },
            // Rearranging the location of the [Name] field.
            {
                // Merge operation.
                "operation": "merge",
                "name": "Name",
                "parentName": "Header",
                "propertyName": "items",
                "values": {
                    "bindTo": "Name",
                    "layout": {
                        "column": 3,
                        "row": 0,
                        "colSpan": 20
                    }
                }
            },
            // Rearranging the location of the [ModifiedBy] field.
            {
                "operation": "merge",
                "name": "ModifiedBy",
                "parentName": "Header",
                "propertyName": "items",
                "values": {
                    "bindTo": "ModifiedBy",
                    "layout": {
                        "column": 3,
                        "row": 2,
                        "colSpan": 20
                    }
                }
            },
            // Rearranging the location of the [Type] field.
            {
                "operation": "merge",
                "name": "Type",
                "parentName": "Header",
                "propertyName": "items",
                "values": {
                    "bindTo": "Type",
                    "layout": {
                        "column": 3,
                        "row": 1,
                        "colSpan": 20
                    },
                    "contentType": Terrasoft.ContentType.ENUM
                }
            }
        ]/**SCHEMA_DIFF*/
    };
});
```

The default logo will be displayed on the knowledge base article edit page after you save the schema and update the application page. When you hover over the image, you will see an action menu appear. You can use it to delete the image or set up a new one for a specific knowledge base article (Fig.9).

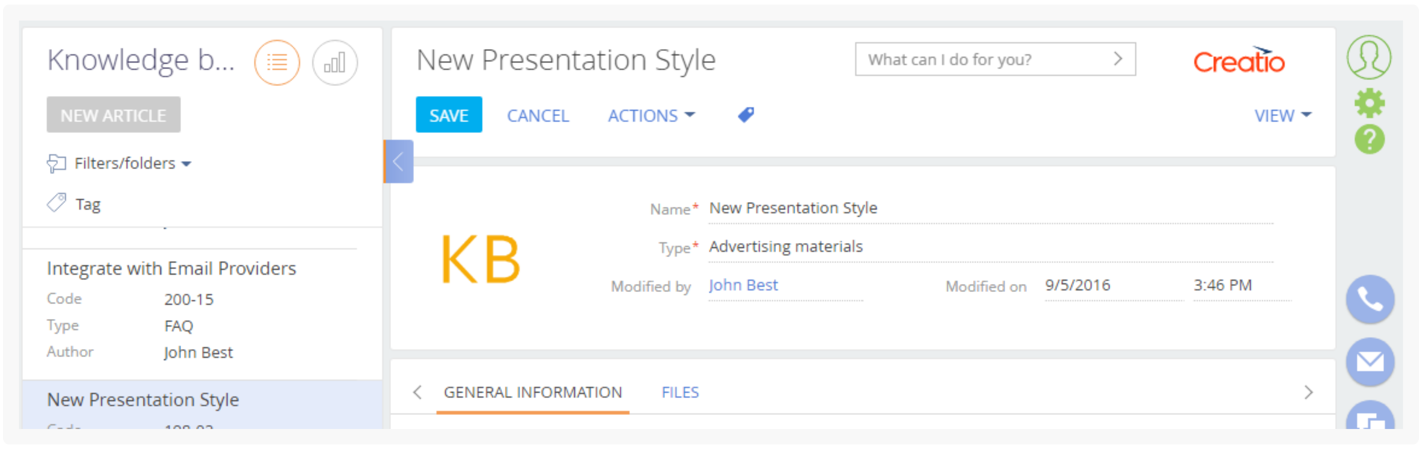
Fig. 8. Default logo



Integrate with Email Providers
Code 200-15
Type FAQ
Author John Best
New Presentation Style

Name* Integrate with Email Providers
Type* FAQ
Modified by John Best
Modified on 9/5/2016 3:46 PM

Fig. 9. Custom logo



Integrate with Email Providers

Code	200-15
Type	FAQ
Author	John Best

New Presentation Style

Name*	New Presentation Style		
Type*	Advertising materials		
Modified by	John Best	Modified on	9/5/2016 3:46 PM

Add calculated fields

Medium

Case description

Add [*Payment balance*] to display the balance between order amount and payment amount on the order edit page.

Note.

You can add fields to the edit page manually or using the section wizard. Learn more about adding fields to edit pages in the “Adding a new field to the edit page” article.

Source code

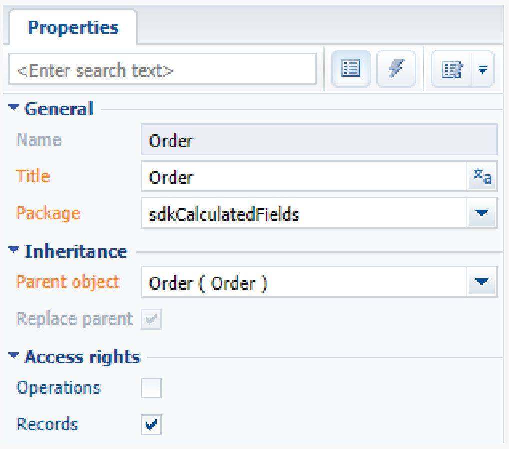
You can download the package with case implementation using the following link.

Case implementation algorithm

1. Create a replacing object

Select the custom package on the [*Schemas*] tab an click the [*Replacing object*] in the [*Add*] menu. Select the [*Order*] object as the parent object (Fig. 1).

Fig. 1. Properties of the [*Order*] replacing object



General

Name	Order
Title	Order
Package	sdkCalculatedFields

Inheritance

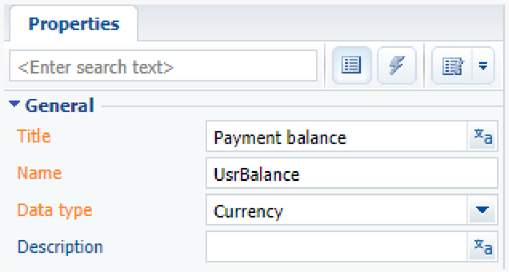
Parent object	Order (Order)
Replace parent	✓

Access rights

Operations	☐
Records	✓

Add a new [*Payment balance*] column of the [*Currency*] type to the replacing object (Fig. 2).

Fig. 2. Adding a custom column to the replacing object



General

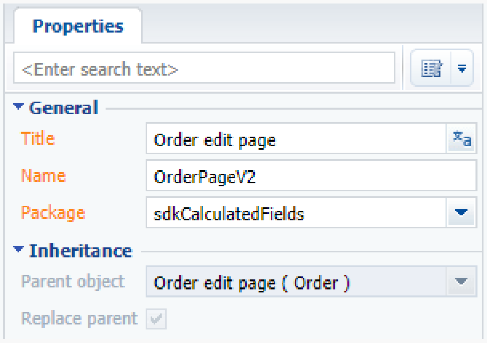
Title	Payment balance
Name	UsrBalance
Data type	Currency
Description	

Publish the object.

2. Create a replacing client module for the order edit page

A replacing client module must be created and [*Order Edit Page*] (`OrderPageV2`) must be specified as the parent object in it (Fig. 3). Creating a replacing page is covered in the “Create a client schema” article.

Fig. 3. Properties of the replacing edit page



General

Title	Order edit page
Name	OrderPageV2
Package	sdkCalculatedFields

Inheritance

Parent object	Order edit page (Order)
Replace parent	✓

3. Set up the display of the [Payment balance] field

Describe the configuration object with the required parameters in the `diff` property of the view model. Page schema source code is available below

4. Add the `UsrBalance` attribute to the view model schema

Add the `UsrBalance` attribute to the collection of the `attributes` property in the source code of the page view model. Specify dependency from the [*Amount*] and [*PaymentAmount*] columns, as well as the `calculateBalance()` handler method (which will be calculating the value of the [*UsrBalance*] column) in the configuration object of the `UsrBalance` attribute.

5. Add the needed methods in the `methods` collection of the view model

In the `methods` collection of the view model, add the `calculateBalance()` handler method that will handle the editing of the [*Amount*] and [*PaymentAmount*] columns. This method is used in the `UsrBalance` attribute.

Attention.

Make sure you take into consideration the type of data in the handler method that need to be returned in the calculation field as a result. For example, the [*Decimal (0.01)*] data type assumes a number with two decimals. Before recording the result in the object field, convert it via the toFixed() function. The example code in this case could look as follows:

// The calculation of difference between the [*Amount*] and [*PaymentAmount*] column values.
var result = amount - paymentAmount;
// The calculation result is assigned as a value to the [*UsrBalance*] column.
this.set("UsrBalance", result.toFixed(2));

Override the `onEntityInitialized()` base virtual method. The `onEntityInitialized()` method is triggered after the edit page object schema is initialized. Calling the `calculateBalance()` handler method to this method will ensure the calculation of the amount to be paid at the moment the order page opens and not only when the dependency columns are edited.

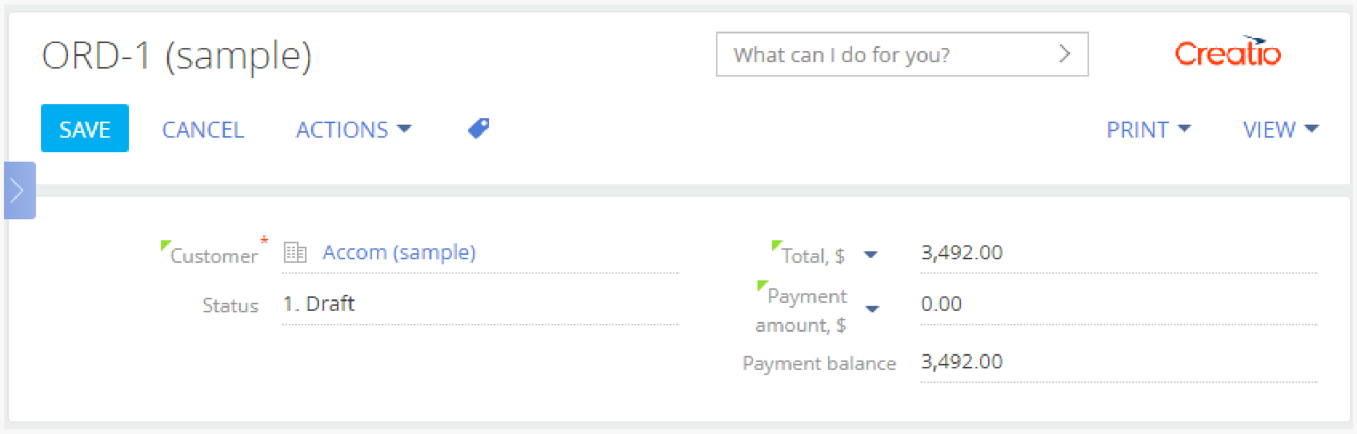
The complete source code of the module is available below:

```
define("OrderPageV2", [], function() {
    return {
        // Edit page object schema name.
        entitySchemaName: "Order",
        details: /**SCHEMA_DETAILS*/{}/**SCHEMA_DETAILS*/,
        // The attributes property of the view model.
        attributes: {
            // Name of the view model attribute.
            "UsrBalance": {
                // Data type of the view model column.
                dataValueType: Terrasoft.DataValueType.FLOAT,
                // Array of configuration objects that determines [UsrBalance] column dependenci
                dependencies: [
                    {
```

```
                    // The value in the [UsrBalance] column depends on the [Amount]
                    // and [PaymentAmount] columns.
                    columns: ["Amount", "PaymentAmount"],
                    // Handler method, which is called on modifying the value of the on of t
                    // and [PaymentAmount].
                    methodName: "calculateBalance"
                }
            ]
        }
    },
    // Collection of the edit page view model methods.
    methods: {
        // Overriding the base Terrasoft.BasePageV2.onEntityInitialized method, which
        // is triggerd after the edit page object schema has been initialized.
        onEntityInitialized: function() {
            // Method parent implementation is called.
            this.callParent(arguments);
            // Calling the handler method, which calculates the value in the [UsrBalance] co
            this.calculateBalance();
        },
        // Handler method that calculates the value in the [UsrBalance] column.
        calculateBalance: function() {
            // Checking whether the [Amount] and [PaymentAmount] columns are initialized
            // when the edit page is opened. If not, then zero values are set for them.
            var amount = this.get("Amount");
            if (!amount) {
                amount = 0;
            }
            var paymentAmount = this.get("PaymentAmount");
            if (!paymentAmount) {
                paymentAmount = 0;
            }
            // Calculating the margin between the values in the [Amount] and [PaymentAmount]
            var result = amount - paymentAmount;
            // The calculation result is set as the value in the [UsrBalance] column.
            this.set("UsrBalance", result);
        }
    },
    // Visual display of the [UsrBalance] column on the edit page.
    diff: /**SCHEMA_DIFF*/[
        {
            "operation": "insert",
            "parentName": "Header",
            "propertyName": "items",
            "name": "UsrBalance",
            "values": {
                "bindTo": "UsrBalance",
                "layout": {"column": 12, "row": 2, "colSpan": 12}
            }
            }
        ]/**SCHEMA_DIFF*/
    };
});
```

After saving the schema, updating the web page and clearing the cache, a new [*Payment balance*] will appear on the order page. The value in this field will be calculated based on the values in the [*Total*] and [*Payment amount*] fields (Fig. 4).

Fig. 4. Case result demonstration



Customer*	Accom (sample)	Total, $	3,492.00
Status	1. Draft	Payment amount, $	0.00
		Payment balance	3,492.00

Add multi-currency field

Advanced

Case description

Add a multi-currency [*Amount*] field on the project edit page.

Source code

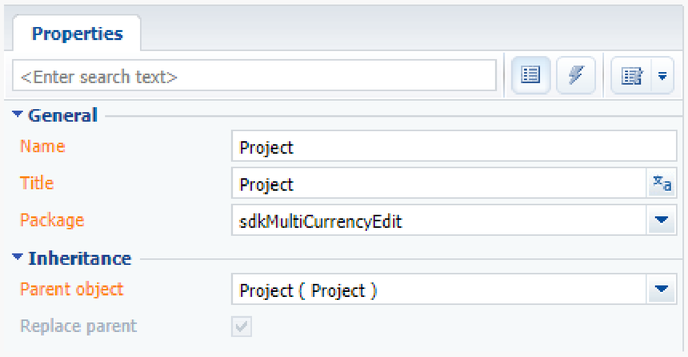
You can download the package with case implementation using the following link.

Case implementation algorithm

1. Add the necessary columns to the object replacing schema

Create the replacing schema of the [*Project*] object in the custom package (Fig. 2). More information about creating a replacing object and adding columns is available in the “Adding a new field to the edit page” article.

Fig. 2. Properties of the object replacing schema



Properties

<Enter search text>

General

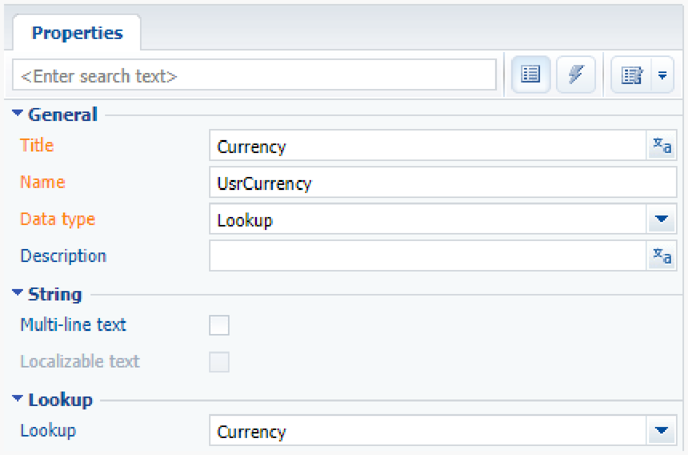
Name	Project
Title	Project
Package	sdkMultiCurrencyEdit

Inheritance

Parent object	Project (Project)
Replace parent	✓

Add 4 columns with properties (see Fig. 3 – Fig. 6) to the replacing schema. Column properties in the Object Designer are displayed in the extended mode.

Fig. 3. The [*UsrCurrency*] column properties



Properties

<Enter search text>

General

Title	Currency
Name	UsrCurrency
Data type	Lookup
Description	

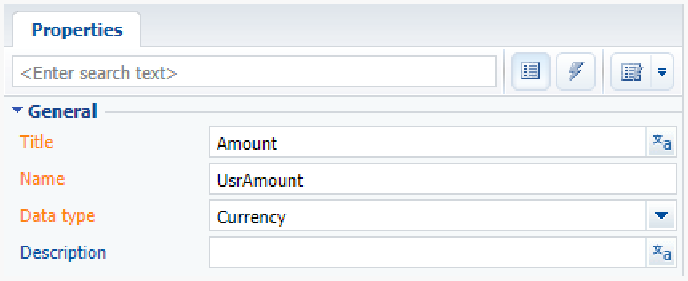
String

Multi-line text	
Localizable text	

Lookup

Lookup	Currency

Fig. 4. The [*UsrAmount*] column properties



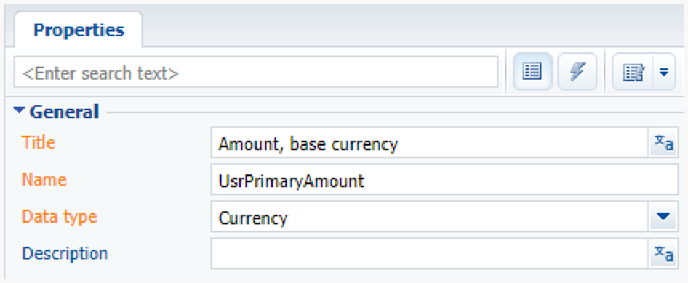
Properties

<Enter search text>

General

Title	Amount
Name	UsrAmount
Data type	Currency
Description	

Fig. 5. The [*UsrPrimaryAmount*] column properties



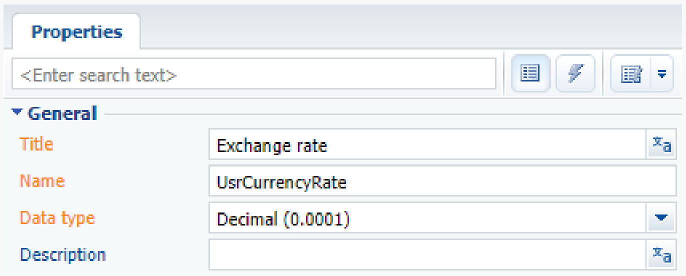
Properties

<Enter search text>

General

Title	Amount, base currency
Name	UsrPrimaryAmount
Data type	Currency
Description	

Fig. 6. The [*UsrCurrencyRate*] column properties



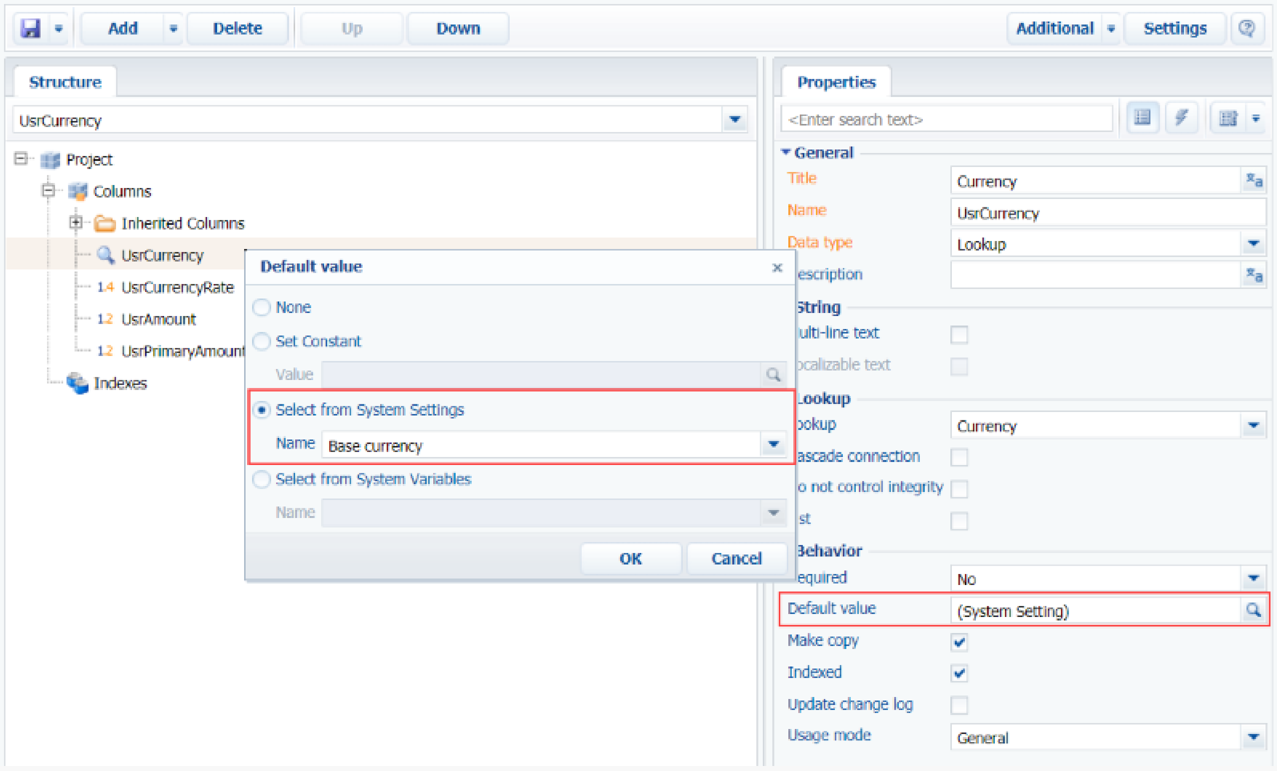
Properties

General

Title	Exchange rate
Name	UsrCurrencyRate
Data type	Decimal (0.0001)
Description	

To the [*UsrCurrency*] column, add the default value – the [*Base currency*] system setting (Fig.7).

Fig. 7. The default value for the [*UsrCurrency*] column



Default value

None

Set Constant

Value

[● Select from System Settings]

Name: Base currency

Select from System Variables

Name

OK Cancel

General

Title	Currency
Name	UsrCurrency
Data type	Lookup
Description	

String

Multi-line text	☐
Localizable text	☐

Lookup

Lookup	Currency
Cascade connection	☐
Do not control integrity	☐
List	☐

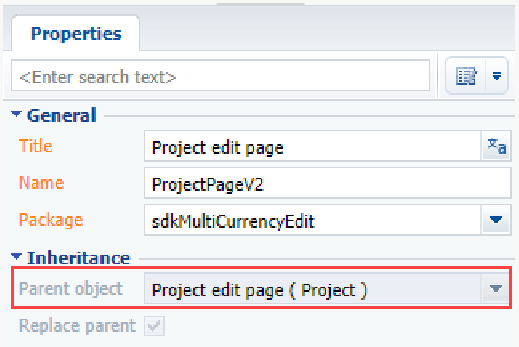
Behavior

Required	No
Default value	(System Setting)
Make copy	✓
Indexed	✓
Update change log	☐
Usage mode	General

2. Create a project replacing edit page in custom package

Create a replacing client module and specify the [*Project edit page*], `ProjectPageV2` schema as its parent object (Fig. 8). The procedure of creating a replacing page is covered in the "Create a client schema" article.

Fig. 8. Properties of the [*Projects*] replacing edit page



Title: Project edit page
Name: ProjectPageV2
Package: sdkMultiCurrencyEdit
Parent object: Project edit page (Project)
Replace parent: ✓

Add the following modules as dependencies when declaring view model class: `MoneyModule`, `MultiCurrencyEdit`, `MultiCurrencyEditUtilities` (see the source code below).

3. Add the necessary attributes

Specify the `UsrCurrency`, `UsrCurrencyRate`, `UsrAmount` and `UsrPrimaryAmount` attributes that correspond to the added columns of the object schema in the `attributes` property of the edit page view model schema.

The multi-currency module operates only with the `Currency` column, so in addition you need to create a `Currency` attribute and declare a virtual column in it. Connect this column wit the previously created `UsrCurrency` column via a handler method.

To ensure the correct operation of the multi-currency module, add the `CurrencyRateList` (currency rate collection) and `CurrencyButtonMenuList` (collection for the button of selecting the currency) attributes (see the source code below).

5. Connect the Terrasoft.MultiCurrencyEditUtilities mixin to the view model

Declare the `Terrasoft.MultiCurrencyEditUtilities` mixin in the `mixins` property of the page view model schema. Initialize it in the `init()` overridden method of the view model schema (see the following code below).

6. Implement the recalculation logic according to the currency

Add handler methods of the attribute dependencies in the `methods` collection of the page view model schema (see the following code below).

7. Add the multi-currency field on the page

Add the configuration object with the multi-currency field settings to the `diff` array of the edit page view model schema.

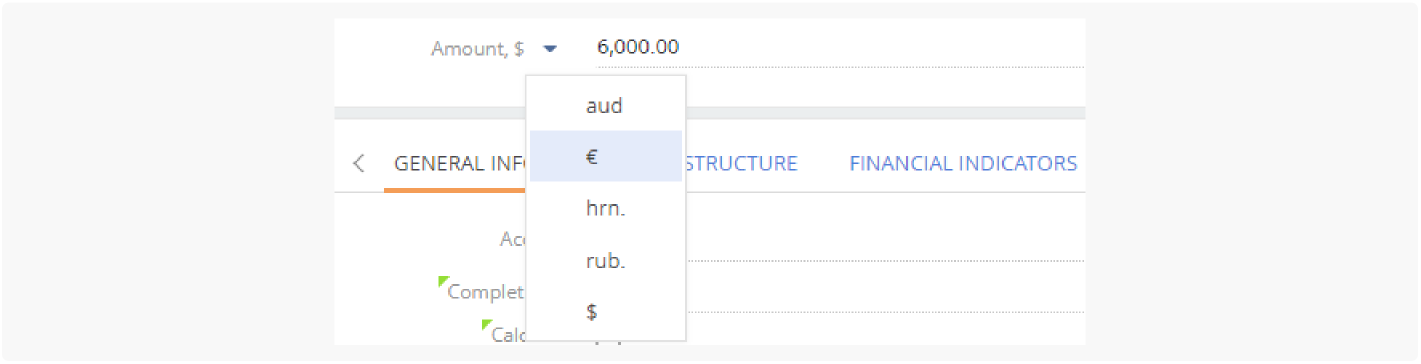
The replacing schema source code is as follows:

```
// Specify modules as dependencies
// MoneyModule, MultiCurrencyEdit and MultiCurrencyEditUtilities.
```

```
define("ProjectPageV2", ["MoneyModule", "MultiCurrencyEdit", "MultiCurrencyEditUtilities"],
    function(MoneyModule, MultiCurrencyEdit, MultiCurrencyEditUtilities) {
        return {
            // Name of the edit page object schema.
            entitySchemaName: "Project",
            // Attributes  of the view model.
            attributes: {
                // Currency.
                "UsrCurrency": {
                    // Attribute data type is a lookup.
                    "dataValueType": this.Terrasoft.DataValueType.LOOKUP,
                    // Configuration of the lookup.
                    "lookupListConfig": {
                        "columns": ["Division", "Symbol"]
                    }
                },
                // Exchange rate.
                "UsrCurrencyRate": {
                    "dataValueType": this.Terrasoft.DataValueType.FLOAT,
                    // Attribute dependencies.
                    "dependencies": [
                        {
                            // Columns on which the attribute depends
                            "columns": ["UsrCurrency"],
                            // Handler method.
                            "methodName": "setCurrencyRate"
                        }
                    ]
                },
                // Amount.
                "UsrAmount": {
                    "dataValueType": this.Terrasoft.DataValueType.FLOAT,
                    "dependencies": [
                        {
                            "columns": ["UsrCurrencyRate", "UsrCurrency"],
                            "methodName": "recalculateAmount"
                        }
                    ]
                },
                // Amount in base currency.
                "UsrPrimaryAmount": {
                    "dependencies": [
                        {
                            "columns": ["UsrAmount"],
                            "methodName": "recalculatePrimaryAmount"
                        }
                    ]
                },
                // Currency is a virtual column for compatibility with the MultiCurrencyEditUtil
                "Currency": {
                    "type": this.Terrasoft.ViewModelColumnType.VIRTUAL_COLUMN,
                    "dataValueType": this.Terrasoft.DataValueType.LOOKUP,
                    "lookupListConfig": {
                        "columns": ["Division"]
                    },
                    "dependencies": [
                        {
                            "columns": ["Currency"],
                            "methodName": "onVirtualCurrencyChange"
                        }
                    ]
                },
                // Currency rate collection
                "CurrencyRateList": {
                    dataValueType: this.Terrasoft.DataValueType.COLLECTION,
                    value: this.Ext.create("Terrasoft.Collection")
                },
                // Collection for the currency button menu
                "CurrencyButtonMenuList": {
                    dataValueType: this.Terrasoft.DataValueType.COLLECTION,
                    value: this.Ext.create("Terrasoft.BaseViewModelCollection")
                }
            },
            // View model mixins.
            mixins: {
                // Mixin that controls multicurrency on the edit page.
                MultiCurrencyEditUtilities: "Terrasoft.MultiCurrencyEditUtilities"
            },
            // Methods of the page view model
            methods: {
                // Overriding the Terrasoft.BasePageV2.onEntityInitialized() basic method.
                onEntityInitialized: function() {
                    // Calling the parent implementation of the onEntityInitialized method.
                    this.callParent(arguments);
                    this.set("Currency", this.get("UsrCurrency"), {silent: true});
                    // Initialization of the mixin controlling the multi-currency.
                    this.mixins.MultiCurrencyEditUtilities.init.call(this);
                },
                // Sets the exchange rate.
                setCurrencyRate: function() {
                    //Loads the exchange rate at the beginning of the project.
                    MoneyModule.LoadCurrencyRate.call(this, "UsrCurrency", "UsrCurrencyRate", th
                },
                // Recalculates the amount.
                recalculateAmount: function() {
                    var currency = this.get("UsrCurrency");
                    var division = currency ? currency.Division : null;
                    MoneyModule.RecalcCurrencyValue.call(this, "UsrCurrencyRate", "UsrAmount", "
                },
                // Recalculates the amount in base currency.
                recalculatePrimaryAmount: function() {
                    var currency = this.get("UsrCurrency");
                    var division = currency ? currency.Division : null;
                    MoneyModule.RecalcBaseValue.call(this, "UsrCurrencyRate", "UsrAmount", "UsrP
                },
                // The handler of the currency virtual column change.
                onVirtualCurrencyChange: function() {
                    var currency = this.get("Currency");
                    this.set("UsrCurrency", currency);
                }
            },
            // Setting up the visualization of a multi-currency field on the edit page.
            diff: /**SCHEMA_DIFF*/[
                // Metadata for adding the[Amount] field.
                {
                    // Adding operation.
                    "operation": "insert",
                    // The meta-name of the parent container to which the component is added.
                    "parentName": "Header",
                    // The field is added to the collection of the
                    // parent container.
                    "propertyName": "items",
                    // The meta-name of the schema component above which the action is performec
                    "name": "UsrAmount",
                    // Properties passed to the constructor of the component.
                    "values": {
                        // The name of the column of the view model to which the binding is perf
                        "bindTo": "UsrAmount",
                        // Element location in the container.
                        "layout": { "column": 0, "row": 2, "colSpan": 12 },
                        // The name of the column that contains the amount in the base currency.
                        "primaryAmount": "UsrPrimaryAmount",
                        // The name of the column that contains the currency of the amount.
                        "currency": "UsrCurrency",
                        // The name of the column that contains the exchange rate.
                        "rate": "UsrCurrencyRate",
                        // The property that defines whether the amount field is enabled and edi
                        "primaryAmountEnabled": false,
                        // Generator of the control view.
                        "generator": "MultiCurrencyEditViewGenerator.generate"
                    }
                }
            ]/**SCHEMA_DIFF*/
        };
    });
```

After saving the schema and updating the application page the [*Amount*] multi-currency field will be displayed on the project edit page (Fig .1). The value of the field will be automatically recalculated after selecting a currency from the drop-down list (Fig. 9).

Fig. 9. Drop-down list of currencies



Add the field validation

Medium

Source code

You can download the package with case implementation using the following link.

Example 1

Case description

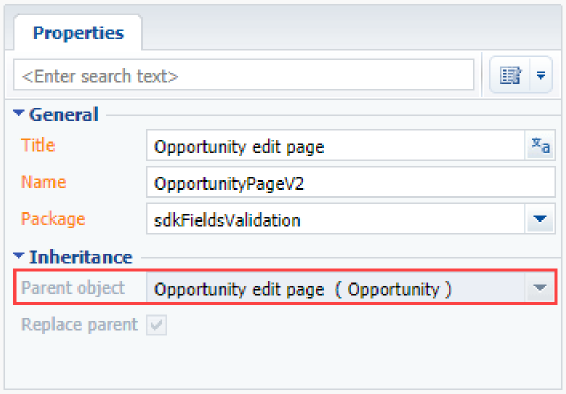
Set the validation on the opportunity page as follows: the date in the [*Created on*] field must be earlier than the date in the [*Closed on*] field.

Case implementation algorithm

1. Create a replacement client module of the opportunity edit page

A replacing client module must be created and `[OpportunityPageV2]` must be specified as the parent object in it (Fig. 1). Creating a replacing page is covered in the “Create a client schema” article.

Fig. 1. Properties of the replacing edit page



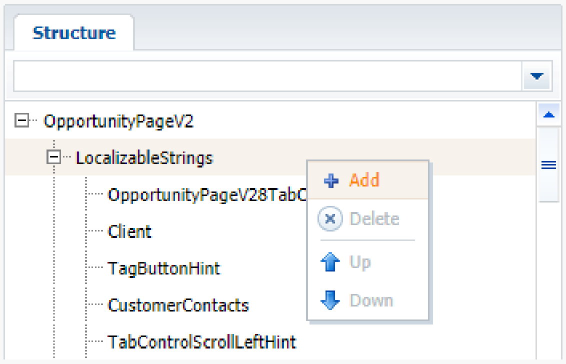
Title	Opportunity edit page
Name	OpportunityPageV2
Package	sdkFieldsValidation

Parent object	Opportunity edit page (Opportunity)
Replace parent	✓

2. Add an error string to the collection of localizable strings of the page replacing schema

Create a new localizable string (Fig. 2).

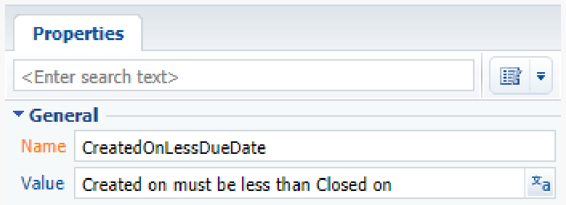
Fig. 2. Adding localized string to the schema



For the created string, specify (Fig. 3):

- [Name] – “CreatedOnLessDueDate”.
- [Value] – “Created on must be less than Closed on”.

Fig. 3. Properties of the custom localizable string



Name	CreatedOnLessDueDate
Value	Created on must be less than Closed on

3. Add the implementation of methods in the `methods` collection of the view model

- `dueDateValidator()` – validator method that determines if the condition is fulfilled.

- `setValidationConfig()` – an overridden base method in which the validator method is bound to the [DueDate] and [CreatedOn] columns.

The replacing schema source code is as follows:

```
define("OpportunityPageV2", [], function() {
    return {
        // Name of the edit page object schema.
        entitySchemaName: "Opportunity",
        methods: {
            // Validate method for values in the [DueDate] and [CreatedOn] columns.
            dueDateValidator: function() {
                // Variable for storing a validation error message.
                var invalidMessage = "";
                // Checking values in the [DueDate] and [CreatedOn] columns.
                if (this.get("DueDate") < this.get("CreatedOn")) {
                     // If the value of the [DueDate] column is less than the value
                     // of the [CreatedOn] column a value of the localizable string is
                     // placed into the variable along with the validation error message
                     // in the invalidMessage variable.
                    invalidMessage = this.get("Resources.Strings.CreatedOnLessDueDate");
                }
                 // Object whose properties contain validation error messages.
                 // If the validation is successful, empty strings are returned to the
                 // object.
                return {
                    // Validation error message.
                    invalidMessage: invalidMessage
                };
            },
            // Redefining the base method initiating custom validators.
            setValidationConfig: function() {
                // This calls the initialization of validators for the parent view model.
                this.callParent(arguments);
                // The dueDateValidator() validate method is added for the [DueDate] column.
                this.addColumnValidator("DueDate", this.dueDateValidator);
                // The dueDateValidator() validate method is added for the [CreatedOn] column.
                this.addColumnValidator("CreatedOn", this.dueDateValidator);
            }
        }
    };
});
```

After you save the schema and refresh Creatio page, a string with the corresponding message (Fig. 4) will appear on the opportunity edit page when entering the date of closing or date of creation which does not satisfy the validation condition (the date of creation must be before than the date of closing). The data window will appear when making an attempt to save the opportunity (Fig. 5).

Fig. 4. Case results: invalid date message

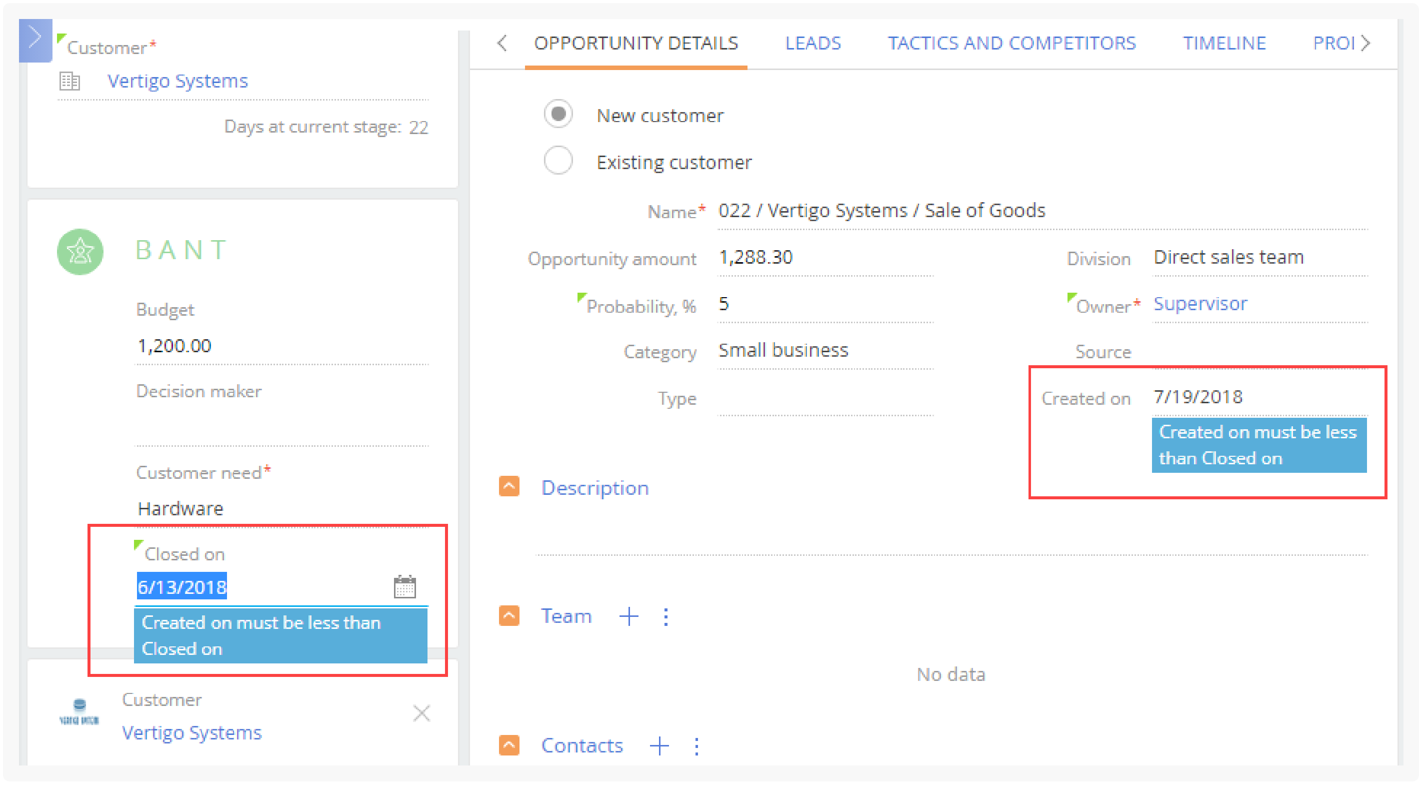
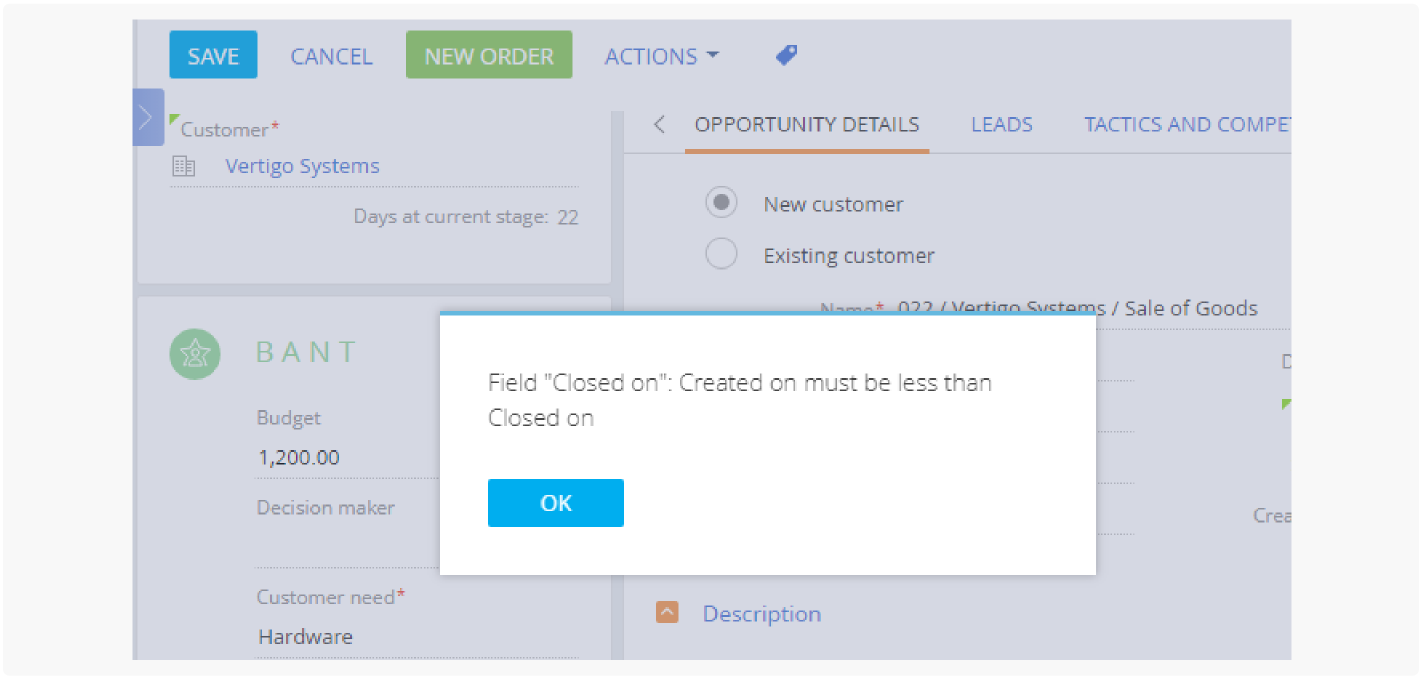


Fig. 5. Case results: message when saving



Field "Closed on": Created on must be less than Closed on

Example 2

Case description

Set the [*Business phone*] field validation as follows: phone number must correspond to the following mask: +44

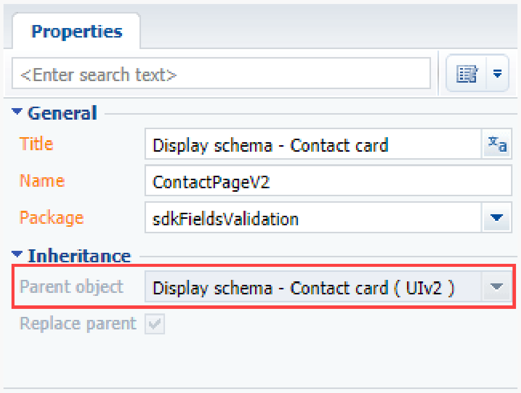
XXX XXX XXXX, otherwise the “Enter the number in the “+44 XXX XXX XXXX” format ” message appears.

Case implementation algorithm

1. Create a replacing client module

Create a replacing client module and specify the [*Display schema – Contact card*] (`ContactPageV2`) schema as parent object (Fig. 6). Creating a replacing page is covered in the “Create a client schema” article.

Fig. 6. Properties of the replacing edit page



Title	Display schema - Contact card
Name	ContactPageV2
Package	sdkFieldsValidation
Parent object	Display schema - Contact card (UIv2)
Replace parent	✓

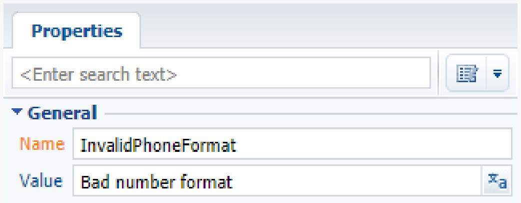
2. Add an error string to the collection of localizable strings of the page replacing schema

Create a new localizable string (Fig. 2).

For the created string, specify (Fig. 7):

- [Name – "InvalidPhoneFormat".
- [Value] – "Enter the number in the “+44 XXX XXX XXXX” format”.

Fig. 7. Properties of the custom localizable string



Name	InvalidPhoneFormat
Value	Bad number format

3. Add the implementation of methods in the `methods` collection of the view model

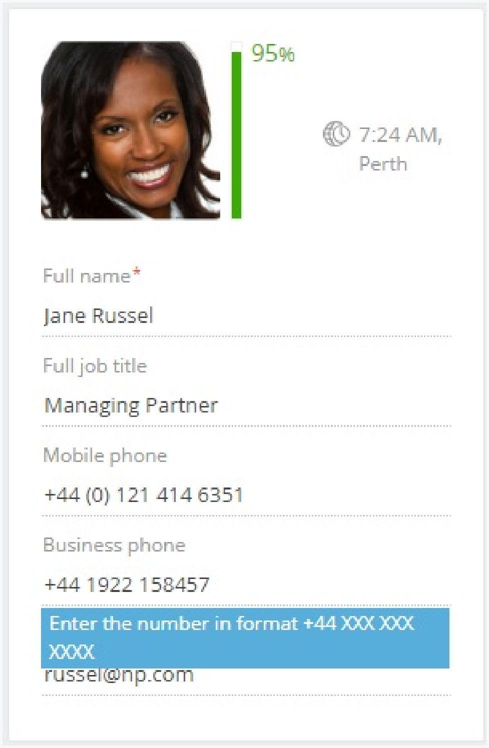
- `phoneValidator()` – validator method that determines if the condition is fulfilled.
- `setValidationConfig()` – an overridden base method in which the validator method is bound to the [Phone] column.

The replacing schema source code is as follows:

```
define("ContactPageV2", ["ConfigurationConstants"], function(ConfigurationConstants) {
    return {
        entitySchemaName: "Contact",
        methods: {
            // Redefining the base method initiating custom validators.
            setValidationConfig: function() {
                // Calls the initialization of validators for the parent view model.
                this.callParent(arguments);
                // The phoneValidator() validate method is added to the [Phone] column.
                this.addColumnValidator("Phone", this.phoneValidator);
            },
            phoneValidator: function(value) {
                // Variable for stroing a validation error message.
                var invalidMessage = "";
                // Variable for stroing the number check result.
                var isValid = true;
                // Variable for the phone number.
                var number = value || this.get("Phone");
                // Determining the correctness of the number format using a regular expression.
                isValid = (Ext.isEmpty(number) ||
                    new RegExp("^\\+44\\s[0-9]{3}\\s[0-9]{3}\\s[0-9]{4}$").test(number));
                // If the format of the number is incorrect, then an error message is filled in.
                if (!isValid) {
                    invalidMessage = this.get("Resources.Strings.InvalidPhoneFormat");
                }
                // Object which properties contain validation error messages.
                // If the validation is successful, empty strings are returned to the object.
                return {
                    invalidMessage: invalidMessage
                };
            }
        }
    };
});
```

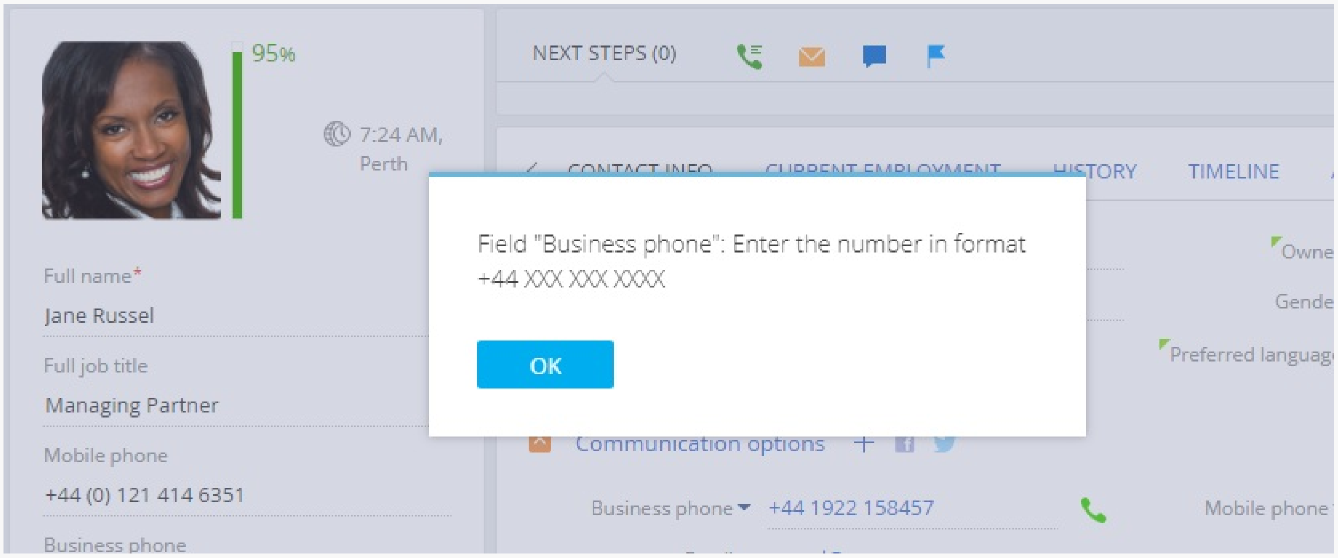
When the schema is saved and the system web-page is updated, the verification of the number format validity will be preformed on the contact or account edit page when a new phone number is added. If the format is incorrect, a string with a corresponding message will appear (Fig. 8, 9).

Fig. 8. Case results: Message about the incorrect format



Jane Russel
Managing Partner
+44 (0) 121 414 6351
+44 1922 158457
Enter the number in format +44 XXX XXX XXXX
russel@np.com

Fig. 9. Case results: message when saving



Jane Russel
Managing Partner
+44 (0) 121 414 6351
Field "Business phone": Enter the number in format +44 XXX XXX XXXX
+44 1922 158457

Set a default value for a field

Medium

Source code

You can download the package with case implementation using the following link.

Example of setting a field default value via object column properties

Case description

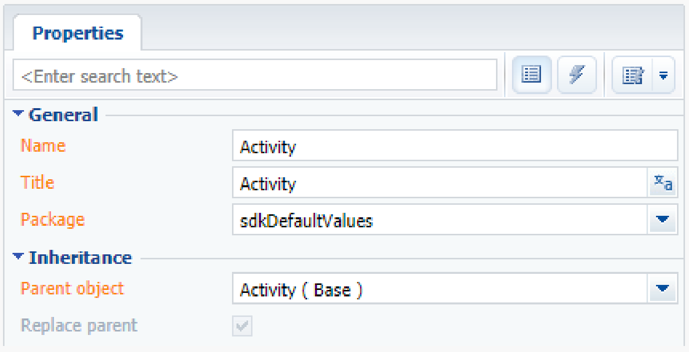
When creating a new activity, the [*Show in calendar*] checkbox should be set by default.

Case implementation algorithm

1. Creating the [Activity] replacing object in the custom package

Create the [*Activity*] replacing object (Fig.1). Learn more about creating a replacing object in the "Create the entity schema" article.

Fig. 1. The [*Activity*] replacing object properties

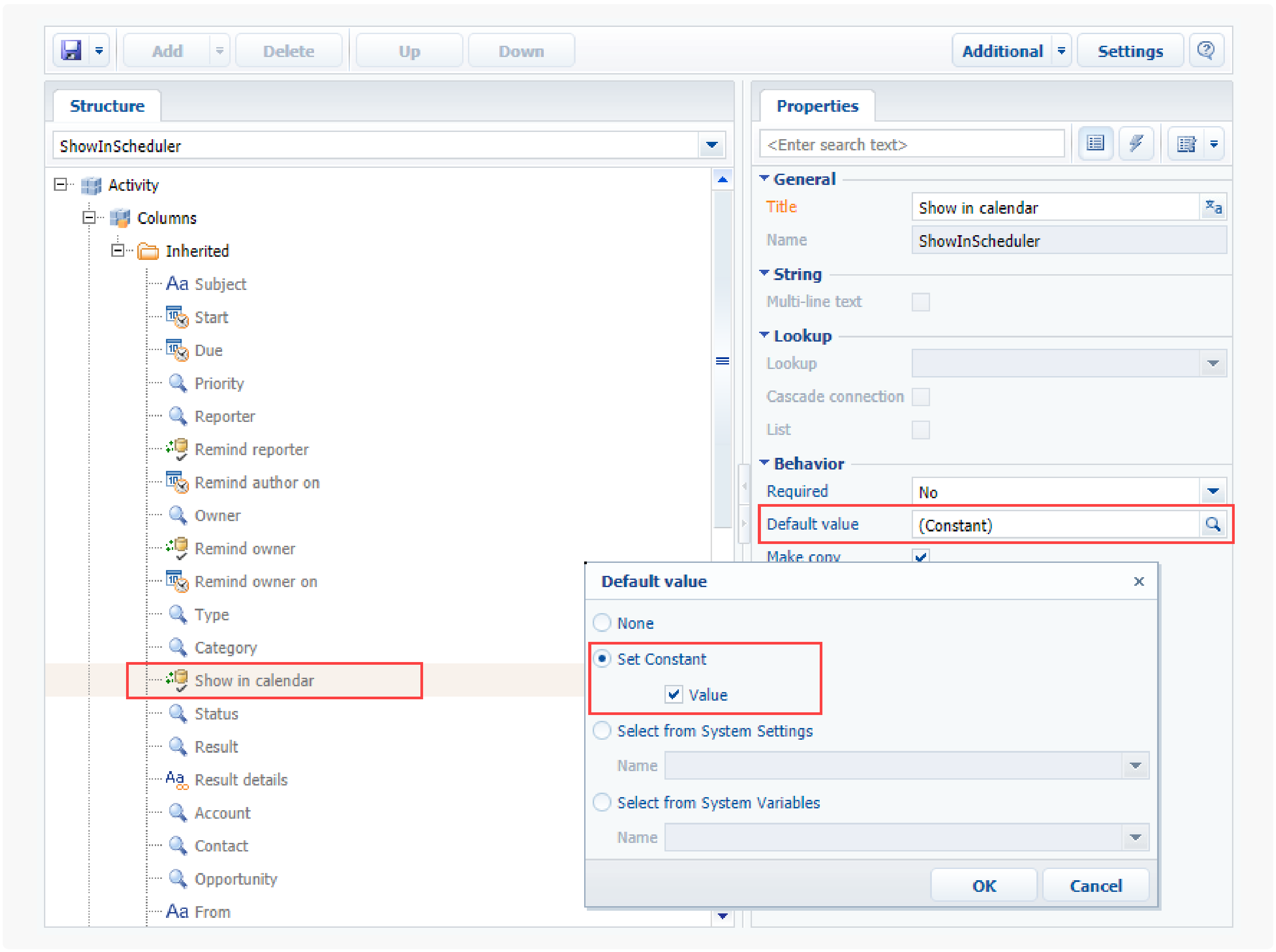


Name	Activity
Title	Activity
Package	sdkDefaultValues
Parent object	Activity (Base)
Replace parent	✓

2. Setting the default value for the [Show in calendar] column

Select the [*Show in calendar*] column from the inherited column list and edit its [*Default value*] property as shown in fig.2. To implement the case, select a constant value as the default one.

Fig. 2. Setting the default value for the [*Show in calendar*] column

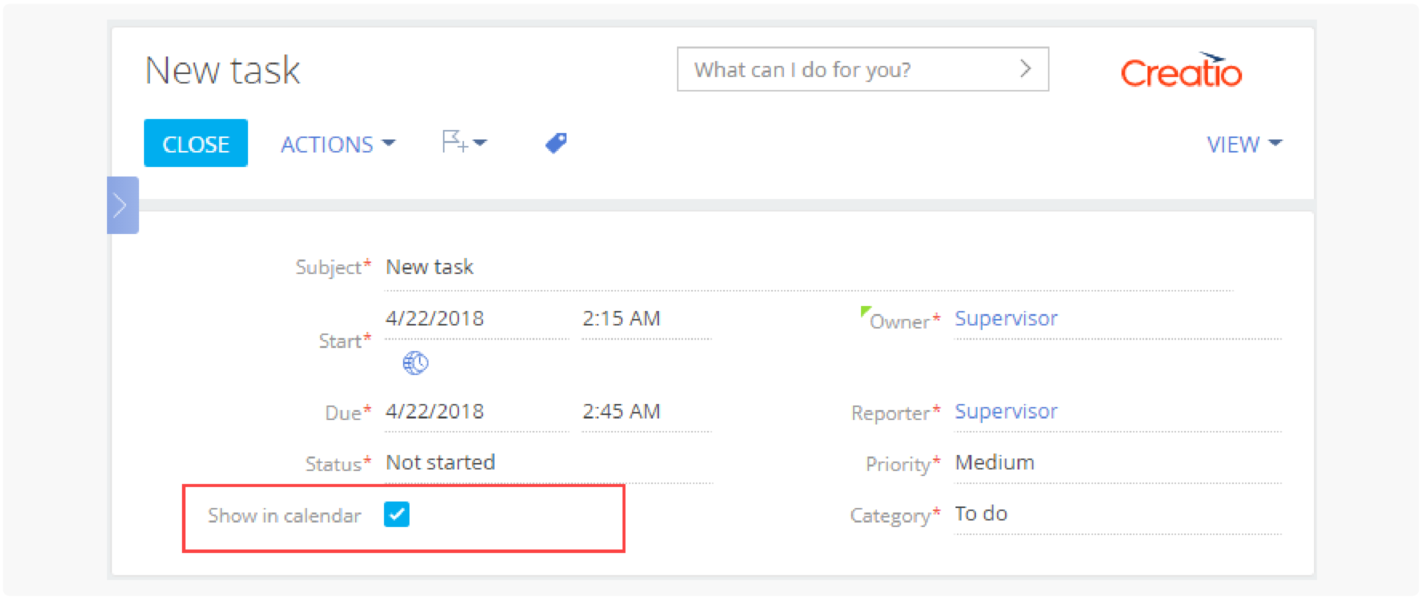


Title: Show in calendar
Name: ShowInScheduler
Multi-line text
Lookup
Cascade connection
List
Required: No
Default value: (Constant)
Make copy: ✓

Default value
None
[✓ Set Constant]
Value ✓
Select from System Settings
Name
Select from System Variables
Name

After you publish the schema, update the page and clear the cache. The [*Show in calendar*] field will be selected on the activity edit page when adding a new activity.

Fig. 3. Demonstration of setting the default value



Subject*	New task		
Start*	4/22/2018 2:15 AM	Owner*	Supervisor
Due*	4/22/2018 2:45 AM	Reporter*	Supervisor
Status*	Not started	Priority*	Medium
Show in calendar	✓	Category*	To do

Example of setting a default value in the edit page source code

Case description

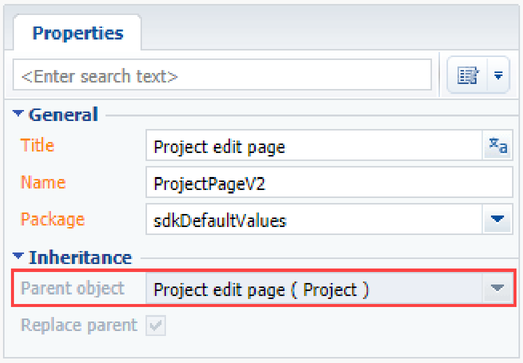
The default value in the [*Deadline*] field on the project edit page should be as follows: the [*Start*] field value plus 10 days.

Case implementation algorithm

1. Create a project replacing edit page in custom package

Create a replacing client module and specify the [*Project edit page*], `ProjectPageV2` schema as its parent object (Fig. 4). The procedure of creating a replacing page is covered in the "Create a client schema" article.

Fig. 4. Replacing edit page properties



Title: Project edit page
Name: ProjectPageV2
Package: sdkDefaultValues
Parent object: Project edit page (Project)
Replace parent: ✓

2. Add the implementation of the following methods to the method collection of the page view model

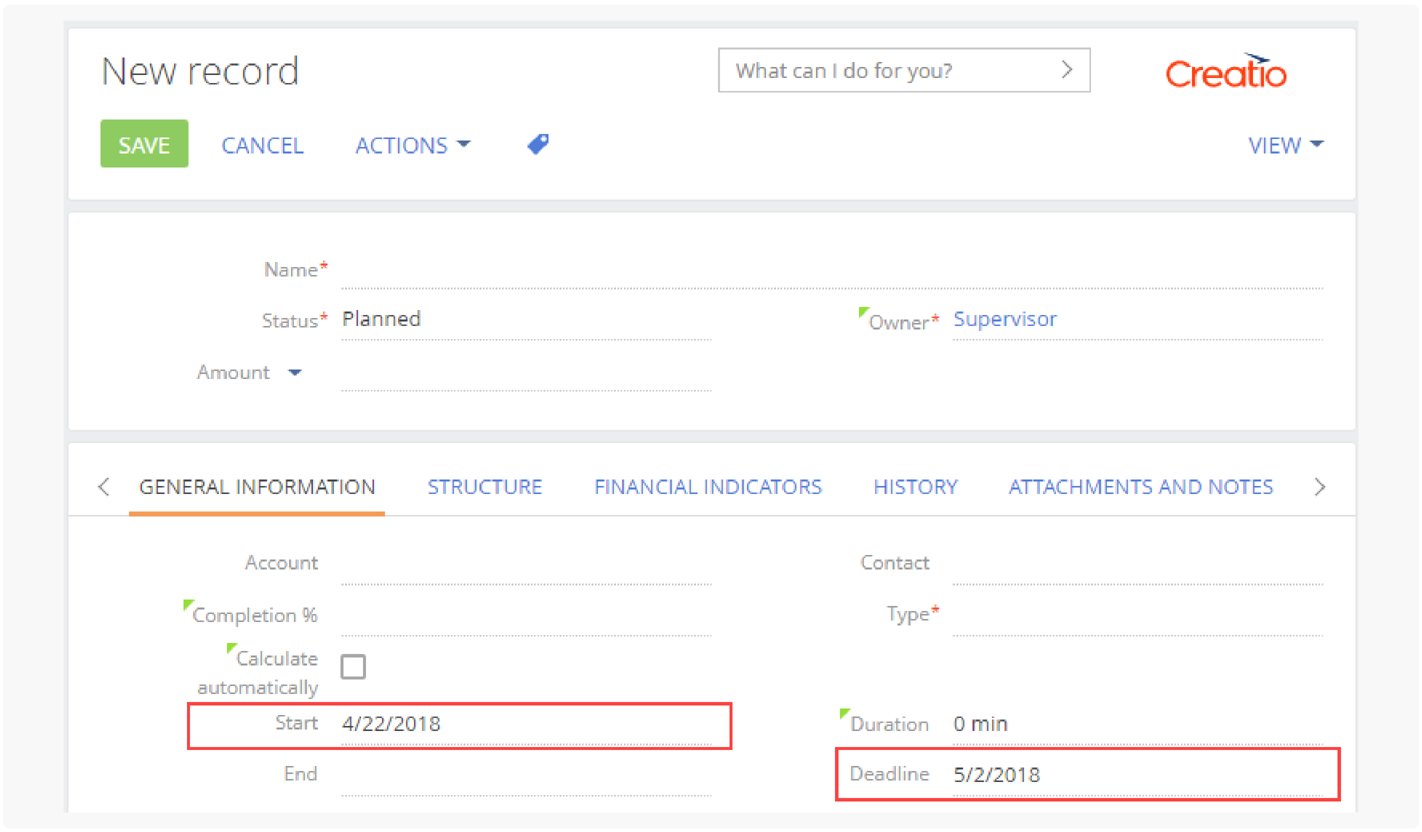
- `setDeadline() –` handler method. Calculates the [Deadline] field value.
- `onEntityInitialized()` – an overridden base virtual method. Triggered upon termination of object schema initialization. Add the handler method call to set the [Deadline] field value to it when opening the edit page.

The replacing schema source code is as follows:

```
define("ProjectPageV2", [], function() {
    return {
        // Name of the edit page object schema.
        entitySchemaName: "Project",
        methods: {
            // Overriding the Terrasoft.BasePageV2.onEntityInitialized() base method.
            // Triggered upon termination of edit page object schema initialization.
            onEntityInitialized: function() {
                // Calling of method parent implementation.
                this.callParent(arguments);
                // Calling of handler method that calculates the [Deadline] field value.
                this.setDeadline();
            },
            // Handler method. Calculates the [Deadline] field value.
            setDeadline: function() {
                // The [Deadline] column value.
                var deadline = this.get("Deadline");
                // Is a new record mode set?
                var newmode = this.isNewMode();
                // If the value is not set and the new record mode is set.
                if (!deadline && newmode) {
                    // Receipt of the [Start] column value.
                    var newDate = new Date(this.get("StartDate"));
                    newDate.setDate(newDate.getDate() + 10);
                    // Setting of the [Deadline] column value.
                    this.set("Deadline", newDate);
                }
            }
        }
    };
});
```

Save the schema and update the application page. A date that equals the [*Start*] field date plus 10 days will be set in the [*Deadline*] field (Fig.5).

Fig. 5. Demonstration of setting the calculated default value



Name*
Status* Planned
Owner* Supervisor
Amount
Account
Contact
Completion %
Type*
Calculate automatically
Start 4/22/2018
Duration 0 min
End
Deadline 5/2/2018

Make a field required based on a specific condition

Medium

Note.

In Creatio, you can configure business rules using developer tools as well as the section wizard. For more information please refer to the "Setting up business rules" article.

Case description

Set up the contact edit page fields so that the [*Business phone*] field is required on condition that the [*Contact type*] field is populated with the "Customer" value.

Source code

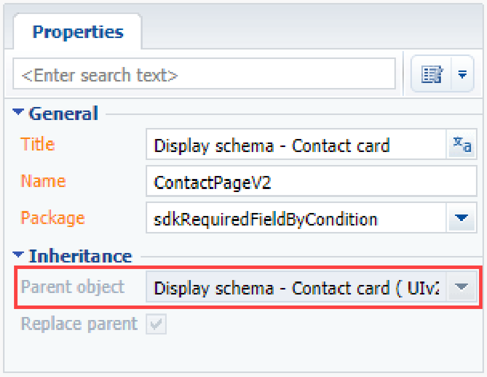
You can download the package with case implementation using the following link.

Case implementation algorithm

1. Create a replacing client module for the contact edit page.

Create a replacing client module and specify the [*Display schema — Contact card*] schema as parent object (Fig. 1). The procedure for creating a replacing page is covered in the "Create a client schema" article.

Fig. 1. Replacing edit page properties



Title: Display schema - Contact card
Name: ContactPageV2
Package: sdkRequiredFieldByCondition
Parent object: Display schema - Contact card (UIv2
Replace parent: ✓

2. Add a rule to the "rules" property of the page view model

Add the BINDPARAMETER type rule for the `Phone` column to the `rules` property of the page view model. Set the `property` rule value to BusinessRuleModule.enums.Property.REQUIRED. Add a rule execution condition to the `conditions` array – the `Type` column value of the model should be equal to the `ConfigurationConstants.ContactType.Client` configuration constant.

Note.

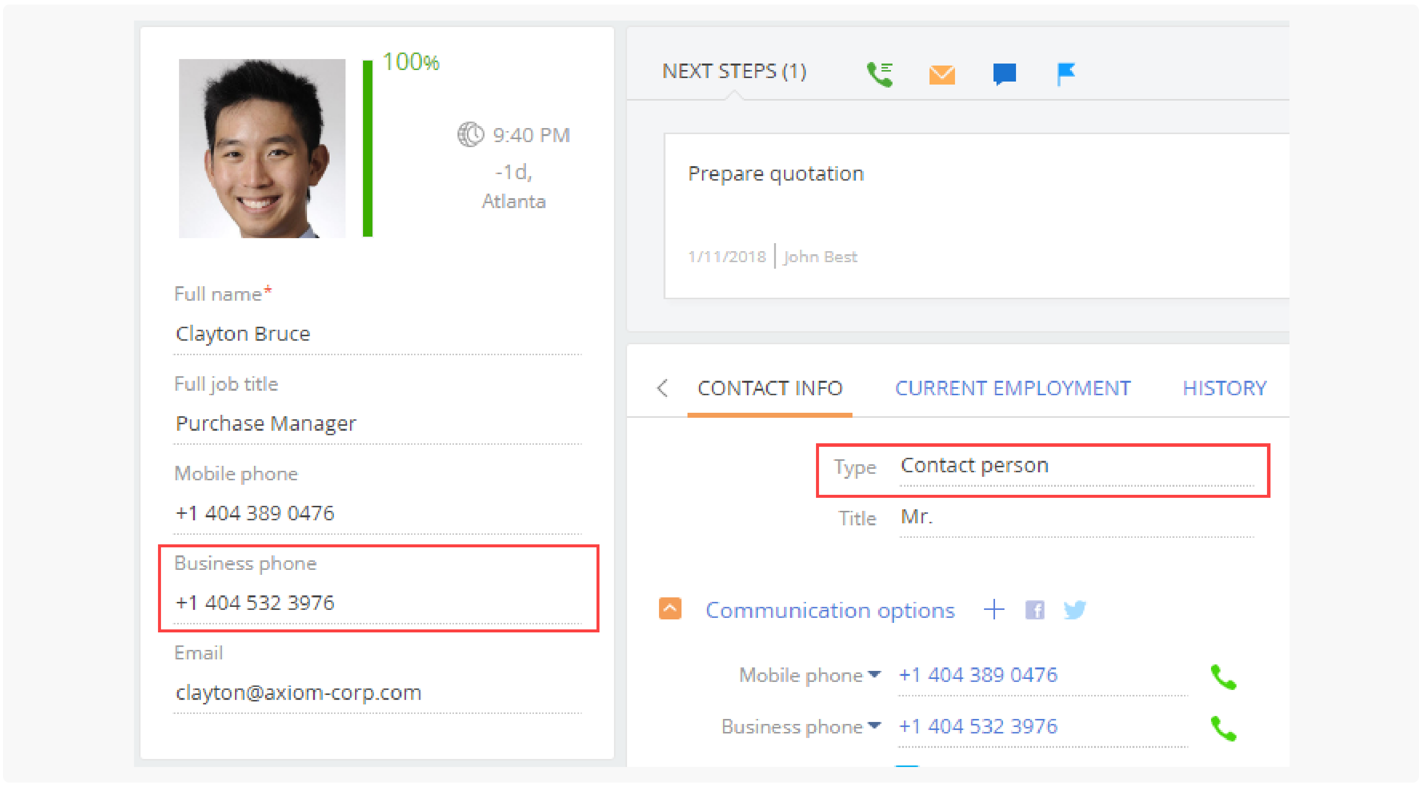
The `ConfigurationConstants.ContactType.Client` configuration constant contains the "Client" record identifier of the [*Contact type*] lookup.

The replacing schema source code is as follows:

```
// Add the BusinessRuleModule and ConfigurationConstants modules to the module dependency list.
define("ContactPageV2", ["BusinessRuleModule", "ConfigurationConstants"],
    function(BusinessRuleModule, ConfigurationConstants) {
        return {
            // Name of the edit page object schema.
            entitySchemaName: "Contact",
            // Rules of the edit page view model.
            rules: {
                // Set of rules of the [Business rule] view model column.
                "Phone": {
                    // Dependency of the [Business phone] field "required" property on the [Type
                    "BindParameterRequiredAccountByType": {
                        // BINDPARAMETER rule type.
                        "ruleType": BusinessRuleModule.enums.RuleType.BINDPARAMETER,
                        // The rule regulates the REQUIRED property.
                        "property": BusinessRuleModule.enums.Property.REQUIRED,
                        // Condition array, whose performance triggers the rule execution.
                        // Defines if the [Type] column value is equal to the "Client" value.
                        "conditions": [{
                            // Expression of the left side of the condition.
                            "leftExpression": {
                                // Expression type — view model attribute(column).
                                "type": BusinessRuleModule.enums.ValueType.ATTRIBUTE,
                                // Name of the view model column whose value is compared in the
                                "attribute": "Type"
                            },
                            // Comparison operation type.
                            "comparisonType": Terrasoft.ComparisonType.EQUAL,
                            // Expression of the right side of the condition.
                            "rightExpression": {
                                // Expression type – constant value.
                                "type": BusinessRuleModule.enums.ValueType.CONSTANT,
                                // The comparison value for the left side of the expression.
                                "value": ConfigurationConstants.ContactType.Client
                            }
                        }]
                    }
                }
            }
        };
    });
```

After you save the schema and update the application web page, the [*Business phone*] filed of the contact edit page will be required on condition the contact type is the “Customer”.

Fig. 2. Case result The [*Business phone*] field – optional


100%
9:40 PM
-1d,
Atlanta

Full name*
Clayton Bruce

Full job title
Purchase Manager

Mobile phone
+1 404 389 0476

Business phone
+1 404 532 3976

Email
clayton@axiom-corp.com

NEXT STEPS (1)

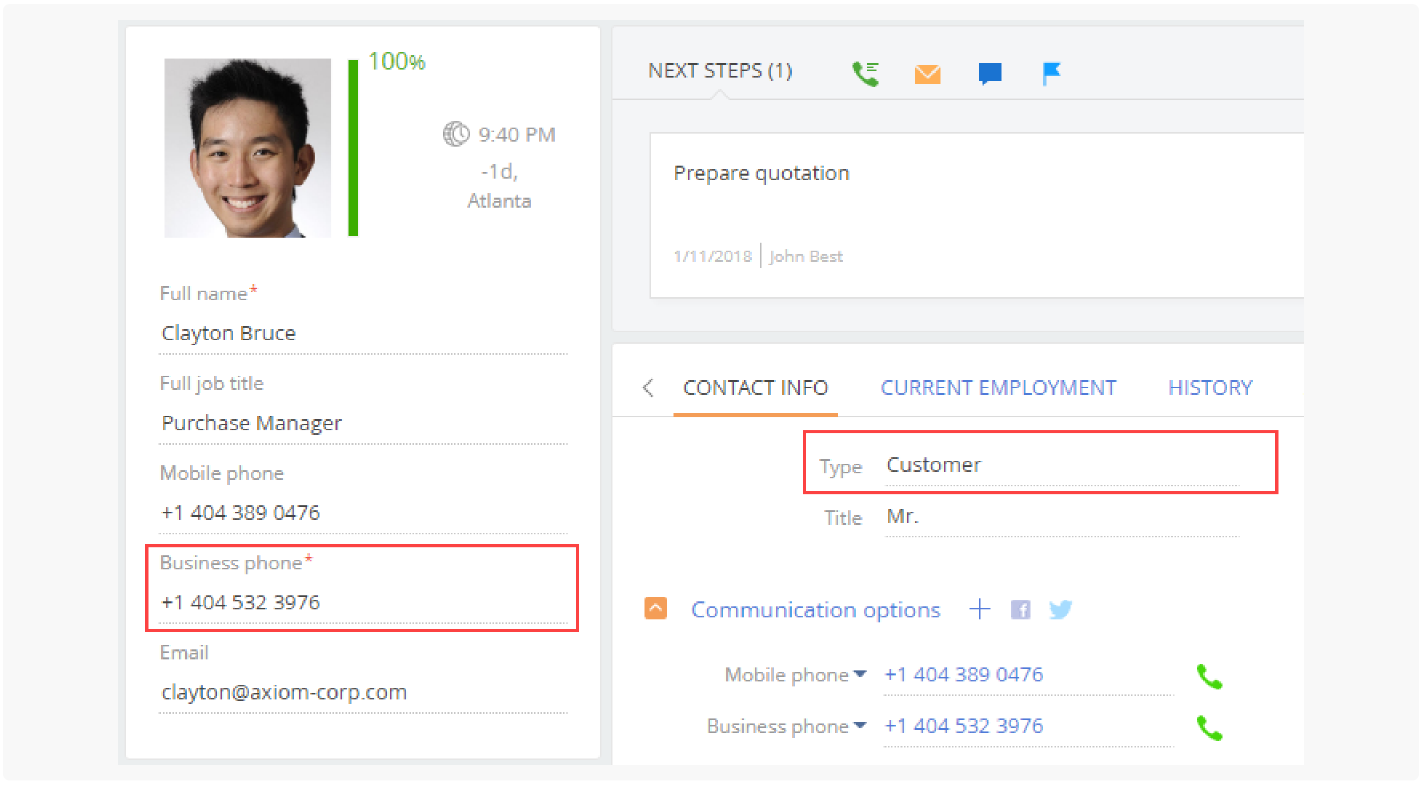
Prepare quotation
1/11/2018 | John Best

CONTACT INFO | CURRENT EMPLOYMENT | HISTORY

Type: Contact person
Title: Mr.

Communication options
Mobile phone: +1 404 389 0476
Business phone: +1 404 532 3976

Fig. 3. Case result The [*Business phone*] field – required


100%
9:40 PM
-1d,
Atlanta

Full name*
Clayton Bruce

Full job title
Purchase Manager

Mobile phone
+1 404 389 0476

Business phone*
+1 404 532 3976

Email
clayton@axiom-corp.com

NEXT STEPS (1)

Prepare quotation
1/11/2018 | John Best

CONTACT INFO | CURRENT EMPLOYMENT | HISTORY

Type: Customer
Title: Mr.

Communication options
Mobile phone: +1 404 389 0476
Business phone: +1 404 532 3976

Display the difference between dates on the page

Easy

Creatio uses the capabilities of the standard `Date` JavaScript-object to work with dates on the client's side of the application. For example, the `Date.prototype.getDate()` method is used to display the day of the month for a specific date in accordance with the local time, and the `Date.prototype.setDate()` method is used to set the day of the month relative to the current month. All properties and methods of the `Date` object may be found in the documentation.

For example, when creating a new contract, the [*End Date*] field should display a date that is 5 days longer than the [*Start Day*] field. To do this:

1. Create a replacing `ContactPageV2` edit page schema of the [*Contracts*] section. The procedure for creating a replacing client schema is covered in the "Create a client schema".

2. Add the following code to the created module:

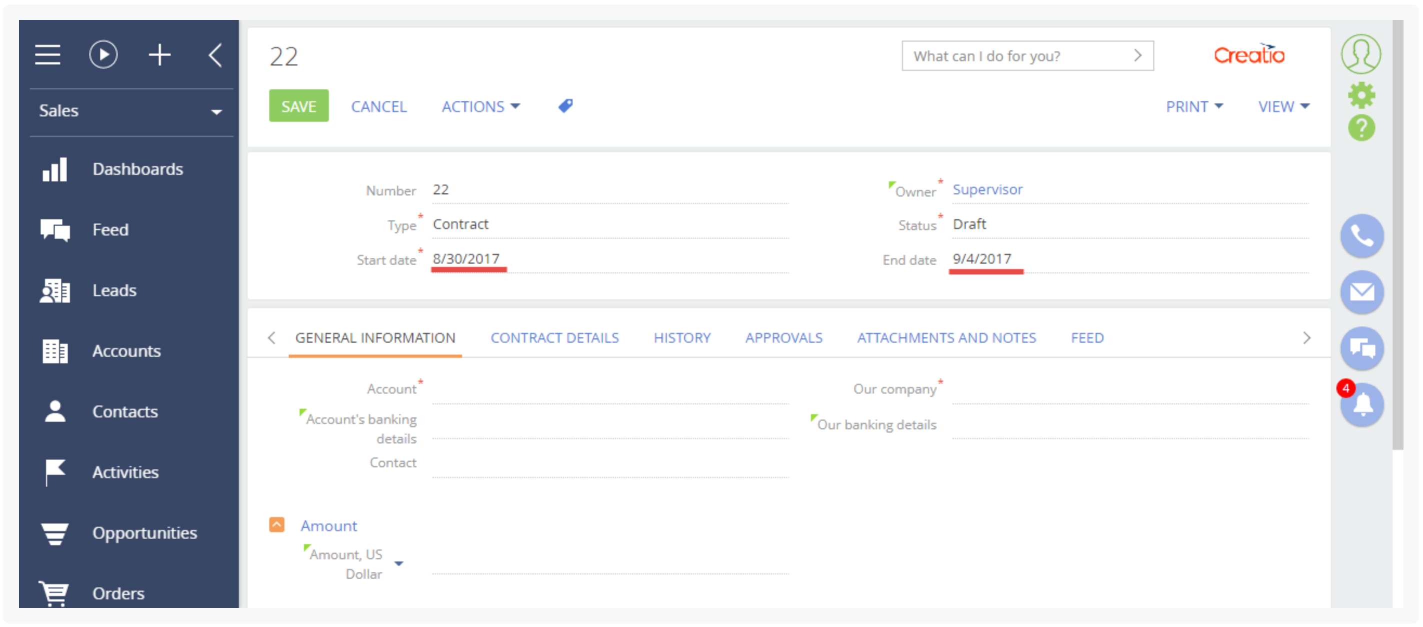
```
define("ContractPageV2", [], function() {
    return {
        entitySchemaName: "Contract",
        methods: {
            // The date is set after the object is initialized.
            onEntityInitialized: function() {
                // Checking the mode of the new record.
                if ((this.isAddMode() && this.Ext.isEmpty(this.get("EndDate")))) {
                    // Calling an auxiliary method.
                    this.setEndDate(this.get("StartDate"), 5);
                }
                // Calling the base functionality.
                this.callParent(arguments);
            },
            // Auxiliary method for setting the date.
            setEndDate: function(date, dateOffsetInDays) {
                var offsetDate = new Date();
                offsetDate.setDate(date.getDate() + dateOffsetInDays);
                this.set("EndDate", offsetDate);
            }
        }
    };
});
```

3. Save the changes.

4. Refresh browser page.

As a result, while adding a new contract. its end date will be 5 days ahead of its start date.

Fig. 1. Case result



Number 22
Type Contract
Start date 8/30/2017
Owner Supervisor
Status Draft
End date 9/4/2017

Note.

In order for the date in the [*End Date*] field to be recalculated automatically when the user changes the [*Start Day*] field, it is necessary to use the functionality of the computed fields. Please refer to the "Adding calculated fields" article for more details.

Filter the lookup

Medium

Case description

Add the [*Country*], [*State/Province*] and [*City*] fields to the page. If the [*Country*] field is populated, the values in the [*State/Province*] field must include only states and provinces of that country. If the [*State/Province*] field is populated, the values in the [*City*] field must include only cities located in that state or province. If the [*City*] field is populated first, the [*Country*] and [*State/Province*] fields must be automatically populated with the corresponding values.

Note.

The base contact page schema already has a rule for filtering cities by country. Therefore, if the [*Country*] field is not added, only cities from the country specified for a contact can be selected.

Source code

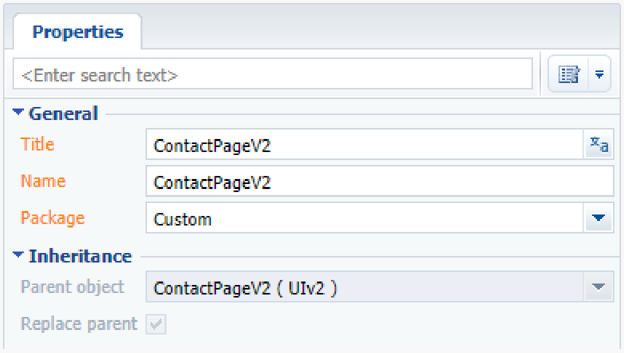
Use this link to download the case implementation package.

Case implementation algorithm

1. Create a replacing contact page

Create a replacing client module and specify the [*Display schema — Contact card*] schema as parent object (Fig. 1). The procedure for creating a replacing page is covered in the “Create a client schema” article.

Fig. 1. Order edit page replacing schema properties



Title: ContactPageV2
Name: ContactPageV2
Package: Custom
Parent object: ContactPageV2 (UIv2)
Replace parent: ✓

2. Add the [Country], [State/Province] and [City] fields to the page.

To do this, add three configuration objects with the settings for the corresponding field properties to the `diff` array.

3. Add FILTRATION-type rules to the [City] and [State/Province] columns.

To do this, add two rules of the `BusinessRuleModule.enums.RuleType.FILTRATION` type to the `rules` property for the [*City*] and [*Region*] columns. To enable reverse filtering (i.e., to automatically populate the [*Country*] and [*State/Province*] fields based on the selected city), set the `autocomplete` property to `true`.

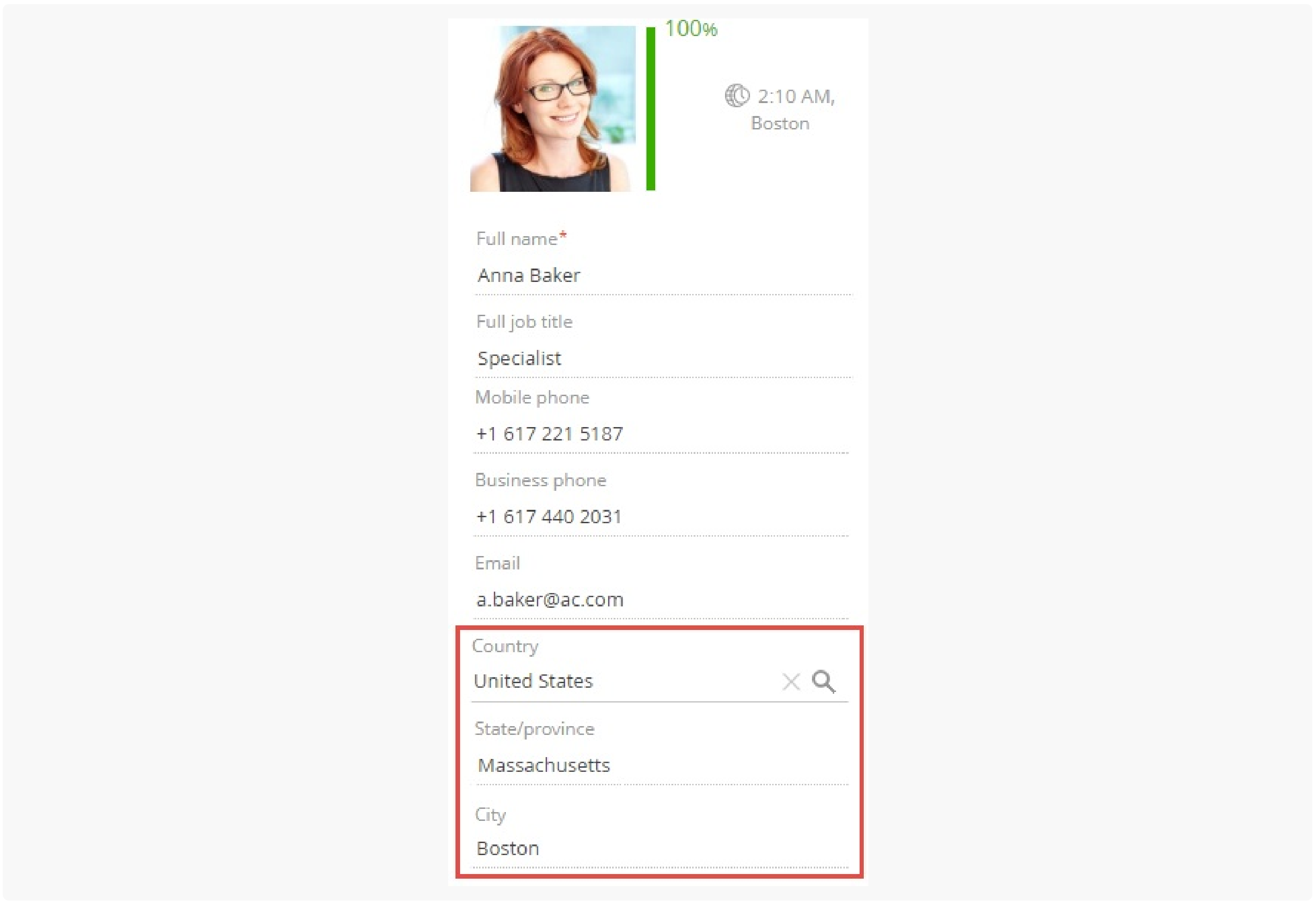
The replacing schema source code is as follows:

```
// Add the module BusinessRuleModul to the dependency list of the module.
define("ContactPageV2", ["BusinessRuleModule"],
    function(BusinessRuleModule) {
        return {
            // Name of the schema of the edit page object.
            entitySchemaName: "Contact",
            // A property that contains a collection of business rules for the schema of the pag
            rules: {
                // A set of rules for the [City] column of the view model..
                "City": {
                    // The rule for filtering the [City] column by the value of the [Region] col
                    "FiltrationCityByRegion": {
                        //  FILTRATION rule type.
                        "ruleType": BusinessRuleModule.enums.RuleType.FILTRATION,
                        // Reverse filtering will be performed.
                        "autocomplete": true,
                        // The value will be cleared when the value of the [Region] column chang
                        "autoClean": true,
                        // The path to the column for filtering in the [City] reference schema,
                        // which is referenced by the [City] column of the
                        // edit page view model.
                        "baseAttributePatch": "Region",
                        // The type of the comparison operation in the filter.
                        "comparisonType": Terrasoft.ComparisonType.EQUAL,
                        // The column (attribute) of the view model will be the comparison value
                        "type": BusinessRuleModule.enums.ValueType.ATTRIBUTE,
                        // The column name of the view model of the edit page,
                        // the value of which will be filtered.
                        "attribute": "Region"
                    }
                },
                // A set of rules for the [Region] column of the view model.
                "Region": {
                    "FiltrationRegionByCountry": {
                        "ruleType": BusinessRuleModule.enums.RuleType.FILTRATION,
                        "autocomplete": true,
                        "autoClean": true,
                        "baseAttributePatch": "Country",
                        "comparisonType": Terrasoft.ComparisonType.EQUAL,
                        "type": BusinessRuleModule.enums.ValueType.ATTRIBUTE,
                        "attribute": "Country"
                    }
                }
            },
            // Setting up the visualization of the [Country], [State/Province] and [City] fields
            diff: [
                // Metadata for adding the [Country] field.
                {
                    "operation": "insert",
                    "parentName": "ProfileContainer",
                    "propertyName": "items",
                    "name": "Country",
                    "values": {
                        "contentType": Terrasoft.ContentType.LOOKUP,
                        "layout": {
                            "column": 0,
                            "row": 6,
                            "colSpan": 24
                        }
                    }
                },
                // Metadata for adding the [State/Province] field.
                {
                    "operation": "insert",
                    "parentName": "ProfileContainer",
                    "propertyName": "items",
                    "name": "Region",
                    "values": {
                        "contentType": Terrasoft.ContentType.LOOKUP,
                        "layout": {
                            "column": 0,
                            "row": 7,
                            "colSpan": 24
                        }
                    }
                },
                // Metadata for adding the [City] field.
                {
                    "operation": "insert",
                    "parentName": "ProfileContainer",
                    "propertyName": "items",
                    "name": "City",
                    "values": {
                        "contentType": Terrasoft.ContentType.LOOKUP,
                        "layout": {
                            "column": 0,
                            "row": 8,
                            "colSpan": 24
                        }
                    }
                }
            ]
        };
    });
```

4. Save the created replacing page schema

After saving the schema and updating the application page, three new fields will be added to the contact profile (Fig. 2). Their values will be filtered based on the values entered in any of these fields. The filtering also works in the lookup selection window (Fig. 4).

Fig. 2. New fields in the contact profile



100%

2:10 AM, Boston

Full name*

Anna Baker

Full job title

Specialist

Mobile phone

+1 617 221 5187

Business phone

+1 617 440 2031

Email

a.baker@ac.com

Country

United States

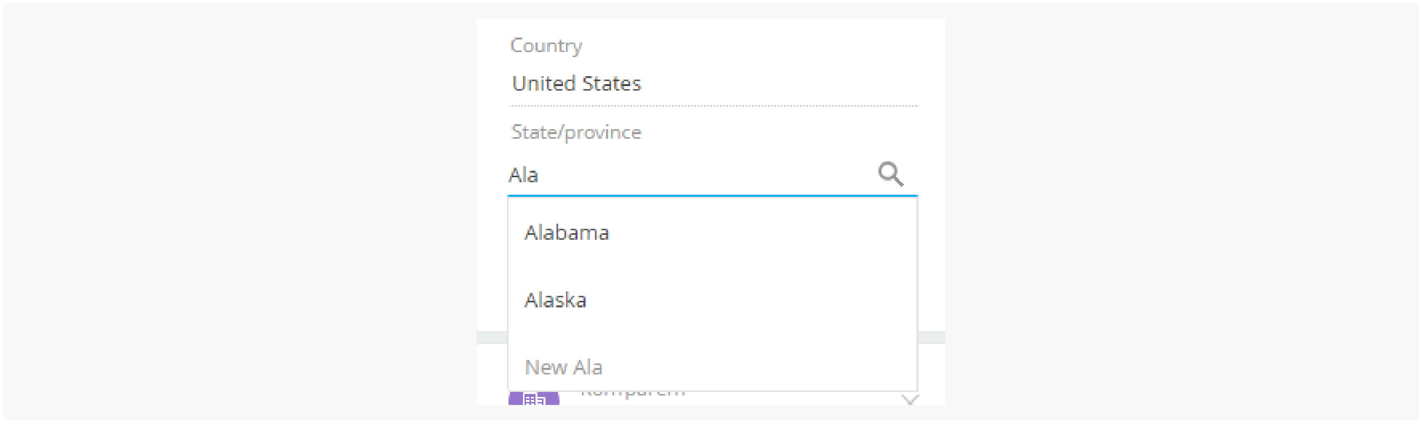
State/province

Massachusetts

City

Boston

Fig. 3. Filtering



Country

United States

State/province

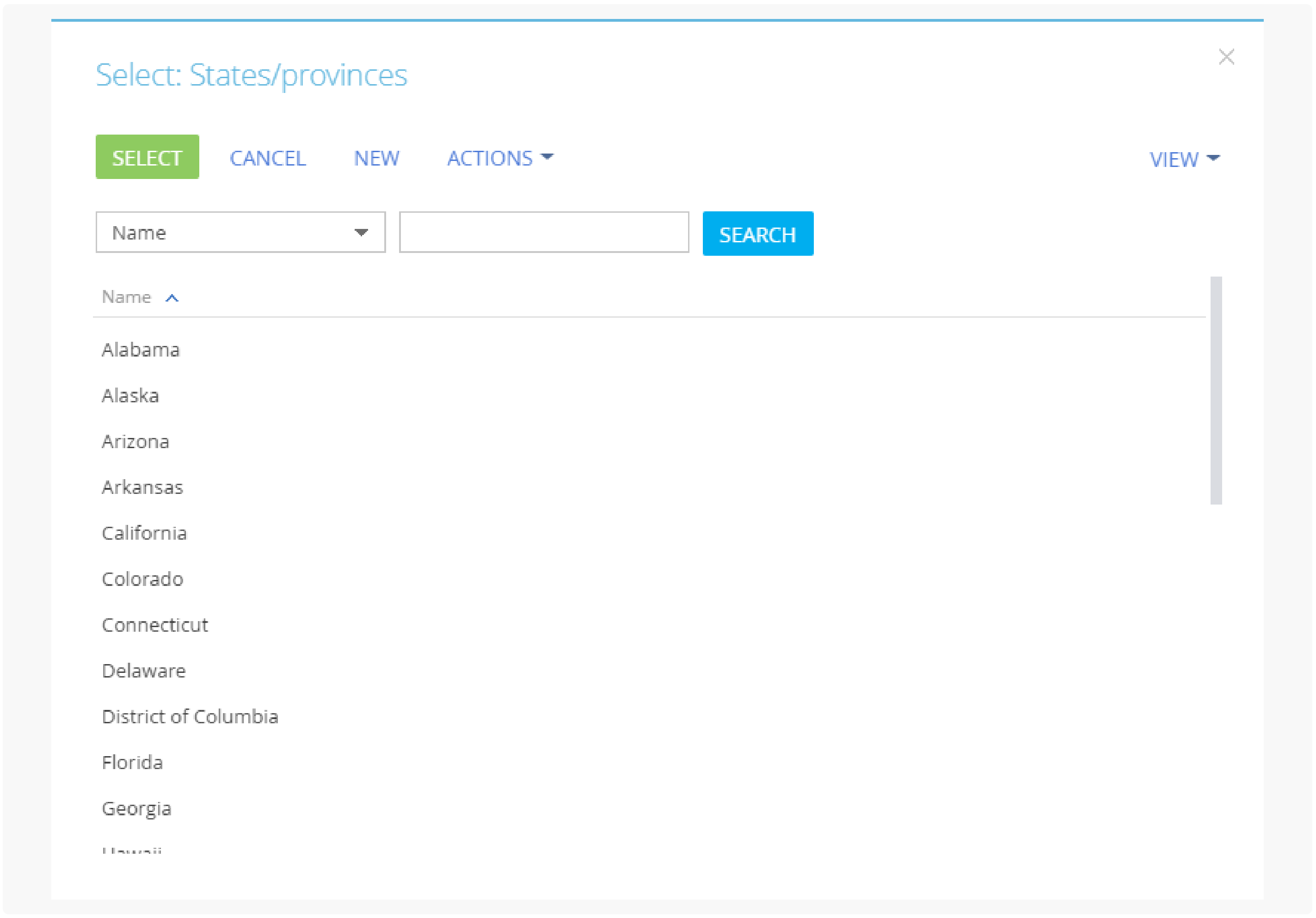
Ala

Alabama

Alaska

New Ala

Fig. 4. Filtered values in the lookup selection window


Select: States/provinces

Alabama
Alaska
Arizona
Arkansas
California
Colorado
Connecticut
Delaware
District of Columbia
Florida
Georgia

Use filtration for lookup fields

Medium

Case description

When a value is added to the [*Owner*] field of the account edit page, display only those contact lookup values for which the following conditions are fulfilled:

- a system user associated with this contact is available
- this user is active.

Source code

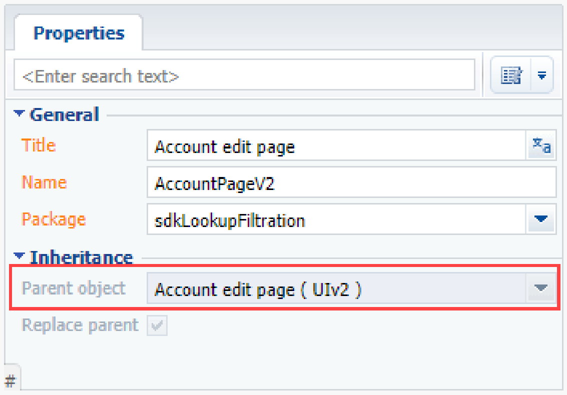
You can download the package with case implementation using the following link.

Case implementation algorithm

1. Create a replacing account edit page

A replacing client module must be created and `[AccountPageV2]` must be specified as the parent object in it (Fig. 1). The procedure of creating a replacing page is covered in the"Create a client schema" article.

Fig. 1. Properties of the replacing edit page



Title: Account edit page
Name: AccountPageV2
Package: sdkLookupFiltration
Parent object: Account edit page (UIv2)
Replace parent: ✓

2. Add the attribute with filtration to the attributes property of the view model

Specify the type of column data – the `Terrasoft.DataValueType.LOOKUP` in the configuration object, in the [*Owner*] attribute and describe the configuration object of the `lookupListConfig` lookup field. Add the filters property to `lookupListConfig`, which represents the function for returning the `filters` collection. Add a function that returns a collection of filters to the array.

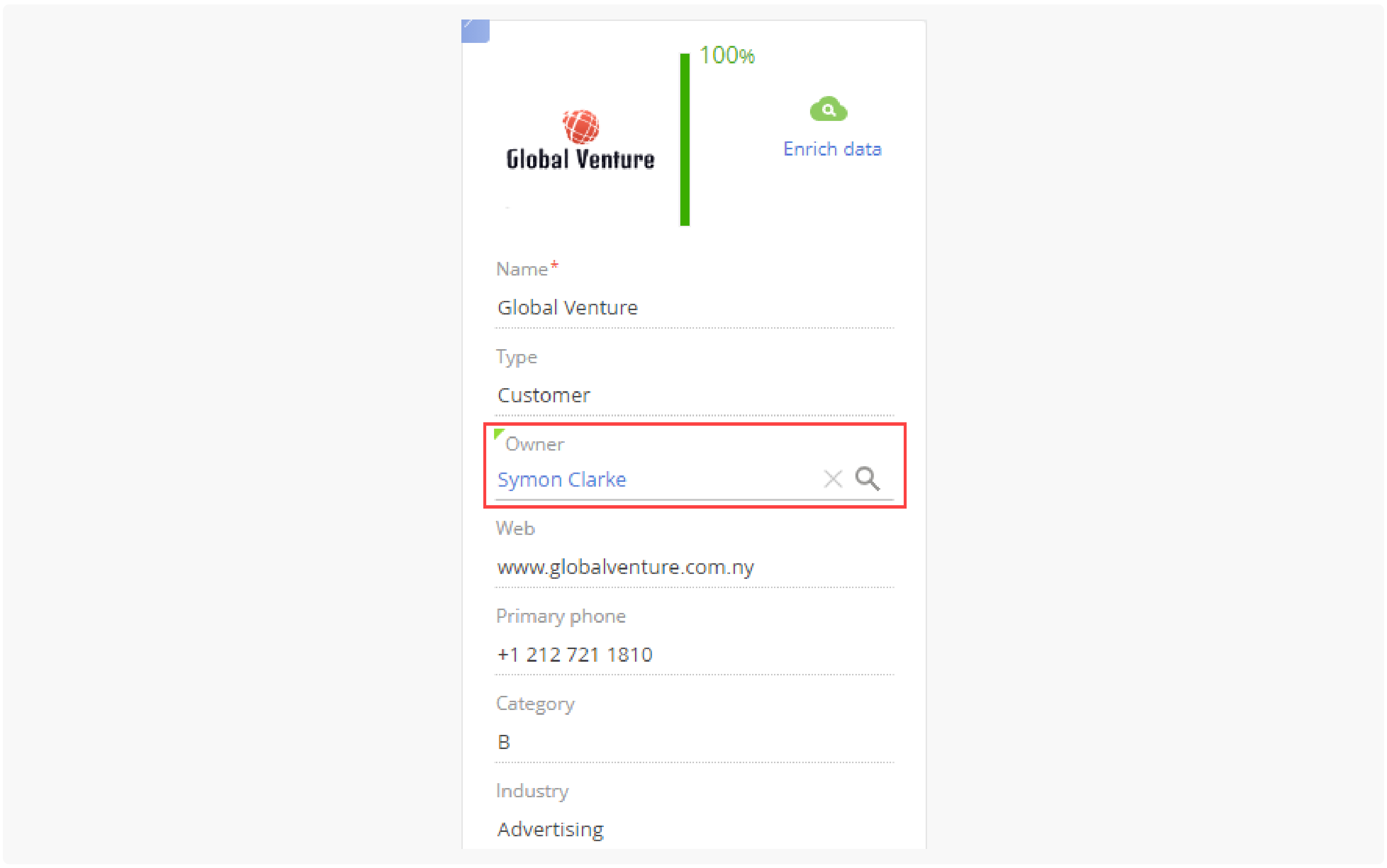
The replacing schema source code is as follows:

```
define("AccountPageV2", [], function() {
    return {
        // Name of the edit page object schema
        "entitySchemaName": "Account",
        // List of the schema attributes.
        "attributes": {
            // Name of the view model column.
            "Owner": {
                // Attribute data type.
                "dataValueType": Terrasoft.DataValueType.LOOKUP,
                // The configuration object of the LOOKUP type.
                "lookupListConfig": {
                    // Array of filters used for the query that forms the lookup field data.
                    "filters": [
                        function() {
                            var filterGroup = Ext.create("Terrasoft.FilterGroup");
                            // Adding the "IsUser" filter to the resulting filters collection.
                            // The filter provides for the selection of all records in the Conta
                            // to which the Id column from the SysAdminUnit schema is connected,
                            // Id is not equal to null.
                            filterGroup.add("IsUser",
                                Terrasoft.createColumnIsNotNullFilter("[SysAdminUnit:Contact].I
                                                        // Adding the "IsActive" filter to the r
                            // The filter provides for the selection of all records from the cor
                            // Contact to which the Id column from the SysAdminUnit schema, for
                            // Active=true, is connected.
                            filterGroup.add("IsActive",
                                Terrasoft.createColumnFilterWithParameter(
                                    Terrasoft.ComparisonType.EQUAL,
                                    "[SysAdminUnit:Contact].Active",
                                    true));
                            return filterGroup;
                        }
                    ]
                }
            }
        }
    };
});
```

When the schema is saved and the system web-page is updated, only values from the contact lookup which comply with custom conditions will be displayed on the account edit page when adding a value to the [*Owner*] field on the account edit page (Fig. 2, Fig. 3). I.e:

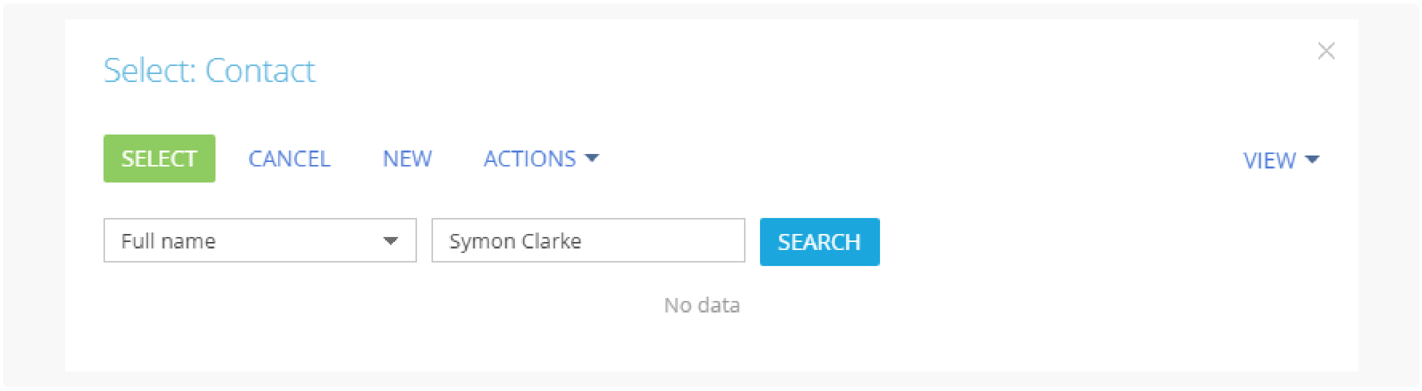
- a system user associated with this contact is available
- this user is active.

Fig. 2. Account profile with the owner



Name*
Global Venture
Type
Customer
Owner
Symon Clarke
Web
www.globalventure.com.ny
Primary phone
+1 212 721 1810
Category
B
Industry
Advertising

Fig. 3. The owner is disable in the filtered contact lookup



Full name
Symon Clarke
No data

Block fields

Medium

Case description

Block all the fields on the invoice edit page if the invoice is on the [*Paid*] stage. The [*Payment status*] field and the [*Activities*] detail should stay editable.

Attention.

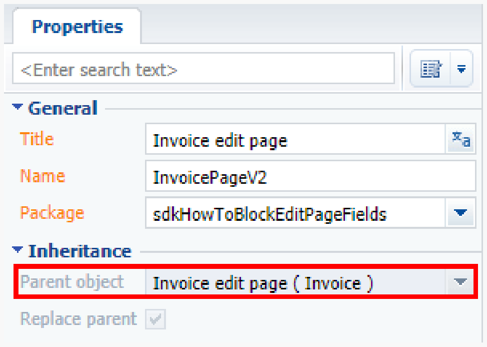
If the field has binding for the `enabled` property in the `diff` array element or in the BINDPARAMETER business rule, the mechanism will not block this field.

Case implementation algorithm

1. Create a replacing schema of the invoice edit page

Create a replacing client module and specify the [*Invoice edit page*] schema as parent (Fig. 1). The procedure for creating a replacing client schema is covered in the “Create a client schema” article.

Fig. 1. The properties of the [*Invoice edit page*] schema



Properties
<Enter search text>
General
Title: Invoice edit page
Name: InvoicePageV2
Package: sdkHowToBlockEditPageFields
Inheritance
Parent object: Invoice edit page (Invoice)
Replace parent: ✓

2. Add schema source code

Add the source code of implementation of the [*Invoice edit page*] replacing schema on the [*Source code*] panel of schema designer. The source code is available below:

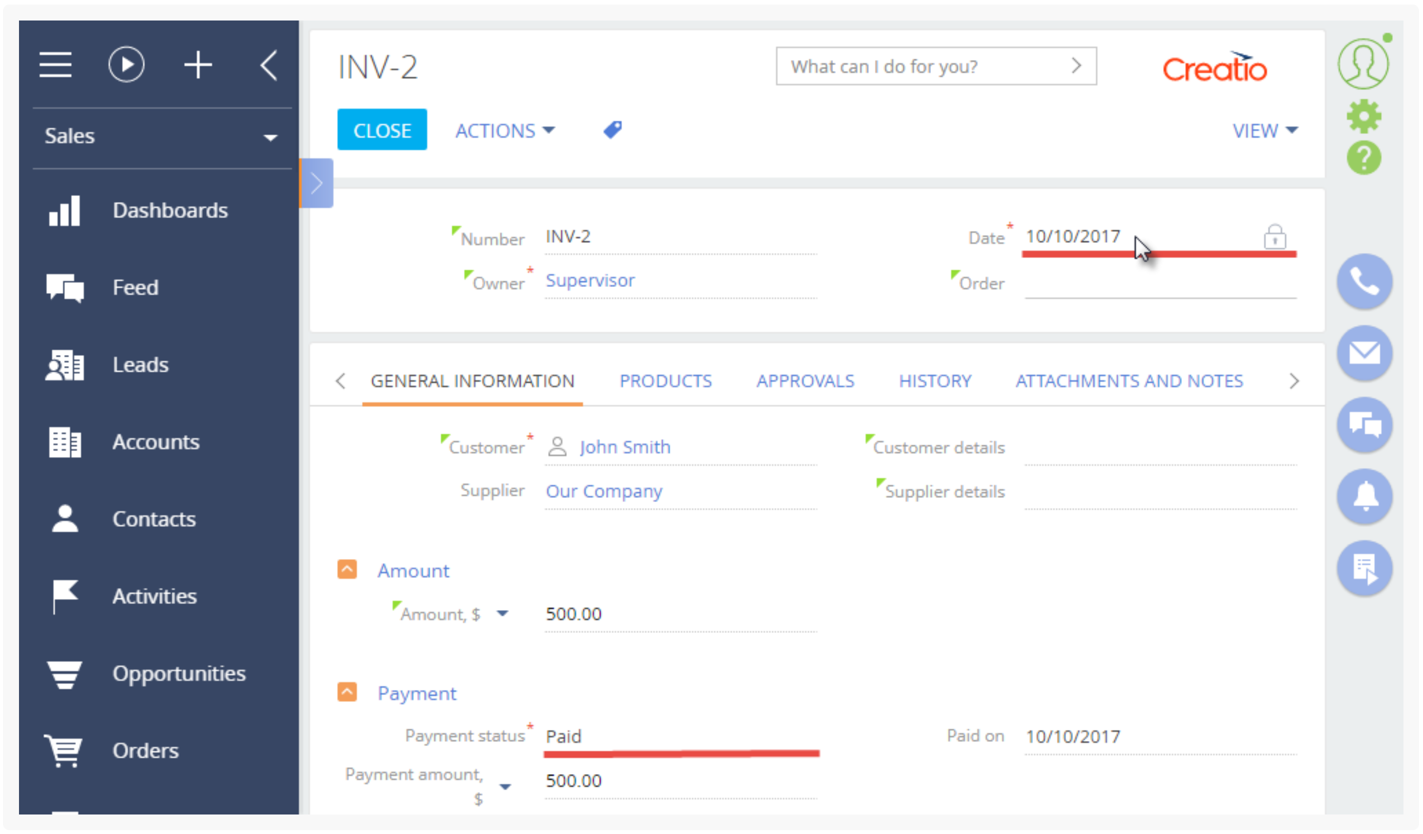
```
define("InvoicePageV2", ["InvoiceConfigurationConstants"], function(InvoiceConfigurationConstant
    return {
        entitySchemaName: "Invoice",
        attributes: {
            // Status of the blocking of fields.
            "IsModelItemsEnabled": {
                dataValueType: Terrasoft.DataValueType.BOOLEAN,
                value: true,
                dependencies: [{
                    columns: ["PaymentStatus"],
                    methodName: "setCardLockoutStatus"
                }]
            }
        },
        methods: {
            getDisableExclusionsColumnTags: function() {
                // The [Payment status] field should not be blocked.
                return ["PaymentStatus"];
            },

            getDisableExclusionsDetailSchemaNames: function() {
                // Also, the "Activity" detail is not blocked.
                return ["ActivityDetailV2"];
            },
            setCardLockoutStatus: function() {
                // Get current invoice status.
                var state = this.get("PaymentStatus");
                // If the current account status is "paid", then block the fields.
                if (state.value === InvoiceConfigurationConstants.Invoice.PaymentStatus.Paid) {
                    // Set a property that stores the field lock flag.
                    this.set("IsModelItemsEnabled", false);
                } else {
                    // Otherwise, unlock the fields.
                    this.set("IsModelItemsEnabled", true);
                }
            },
            onEntityInitialized: function() {
                this.callParent(arguments);
                // Set the status of the blocking of fields.
                this.setCardLockoutStatus();
            }
        },
        diff: /**SCHEMA_DIFF*/[
            {
                "operation": "merge",
                "name": "CardContentWrapper",
                "values": {
                    "generator": "DisableControlsGenerator.generatePartial"
                }
            }
        ]/**SCHEMA_DIFF*/
    };
});
```

The `IsModelItemsEnabled` attribute will be defined and methods for blocking and unlocking the field of the invoice edit page will be implemented. The `CardContentWrapper` container of the edit page is used as the container of the blocking mechanism.

Save the schema after adding the source code. Then clear the browser cache.

As a result, the most of the invoice fields will be blocked after changing the status to the [*Paid*]. Fields and details specified in the exceptions for blocking will stay unlocked. The fields that have the `enabled` property explicitly set to `true` will stay unlocked

Fig. 2. Case result



INV-2

Number INV-2
Date* 10/10/2017
Owner* Supervisor
Order
Customer* John Smith
Customer details
Supplier Our Company
Supplier details
Amount, $ 500.00
Payment status* Paid
Paid on 10/10/2017
Payment amount, $ 500.00

Lock a field on a specific condition

Medium

Note.

In Creatio, you can configure business rules using developer tools the as well as the section wizard. For more information please refer to the “Setting up the business rules”.

Case description

Configure fields on the contact edit page to make the [*Business phone*] field editable only if the [*Mobile phone*] field is filled.

Source code

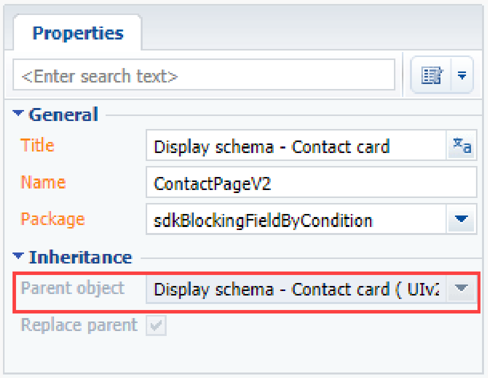
Use this link to download the case implementation package.

Case implementation algorithm

1. Create a replacing client module for the contact page

Create a replacing client module and specify the [*Display schema — Contact card*] schema as parent object (Fig. 1). The procedure for creating a replacing page is covered in the "Create a client schema" article.

Fig. 1. Order edit page replacing schema properties



Title: Display schema - Contact card
Name: ContactPageV2
Package: sdkBlockingFieldByCondition
Parent object: Display schema - Contact card (UIv2
Replace parent: ✓

2. In the rules property of the page view model, add the rule

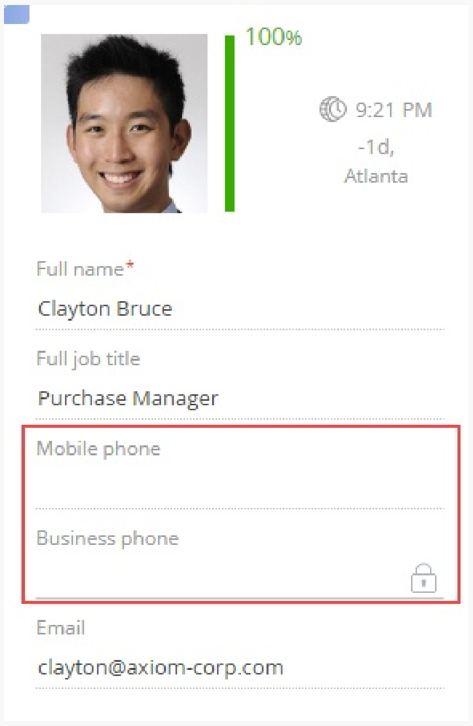
For the `Phone` column, add the rule with the BINDPARAMETER type to `rules` property of the page view model. Set the BusinessRuleModule.enums.Property.ENABLED. Value for the rule's `property`. Add the following condition for rule execution to the `conditions` array: the value in the `MobilePhone` column should be filled.

The replacing schema source code is as follows:

```
// Add the module BusinessRuleModule to the list of dependent modules.
define("ContactPageV2", ["BusinessRuleModule"], function(BusinessRuleModule) {
    return {
        // Name of the page schema of the edit page.
        entitySchemaName: "Contact",
        // Rules of the edit page view model.
        rules: {
            // A set of rules for the [Business phone] column of the view model.
            "Phone": {
                // Dependence of the availability of the [Business phone] field from the value c
                "BindParameterEnabledPhoneByMobile": {
                    // The type of the BINDPARAMETER rule.
                    "ruleType": BusinessRuleModule.enums.RuleType.BINDPARAMETER,
                    // The rule regulates the ENABLED property.
                    "property": BusinessRuleModule.enums.Property.ENABLED,
                    // An array of conditions in which the rule is triggered.
                    // Determines whether the [Mobile Phone] field is populated.
                    "conditions": [{
                        // Expression of the left side of the condition.
                        "leftExpression": {
                            // The type of the expression is the attribute (column) of the view
                            "type": BusinessRuleModule.enums.ValueType.ATTRIBUTE,
                            // The name of the column in the view model, whose value is comparec
                            "attribute": "MobilePhone"
                        },
                        // The type of comparison operation is "not equal to".
                        "comparisonType": Terrasoft.ComparisonType.NOT_EQUAL,
                        // Expression of the right side of the condition.
                        "rightExpression": {
                            // The expression type is a constant value.
                            "type": BusinessRuleModule.enums.ValueType.CONSTANT,
                            // The value with which the left side expression is compared.
                            "value": ""
                        }
                    }]
                }
            }
        }
    };
});
```

After saving the schema and refreshing the application page, the [*Business phone*] field will be non-editable until the [*Mobile phone*] field is empty (Fig. 2).

Fig. 2. Example result demonstration



Full name*
Clayton Bruce
Full job title
Purchase Manager
Mobile phone
Business phone
Email
clayton@axiom-corp.com

Hide a field on a specific condition

Medium

Note.

In Creatio, you can configure business rules using developer tools as well as the section wizard. For more information please refer to the "Setting up the business rules".

Case description

Add a new [*Meeting place*] field to the activity page. The field will be available only for activities of the [*Meeting*] type.

Note.

You can add fields to the edit page manually or via the section wizard.

For more on adding fields to edit pages see the "Adding a new field to the edit page" article.

Source code

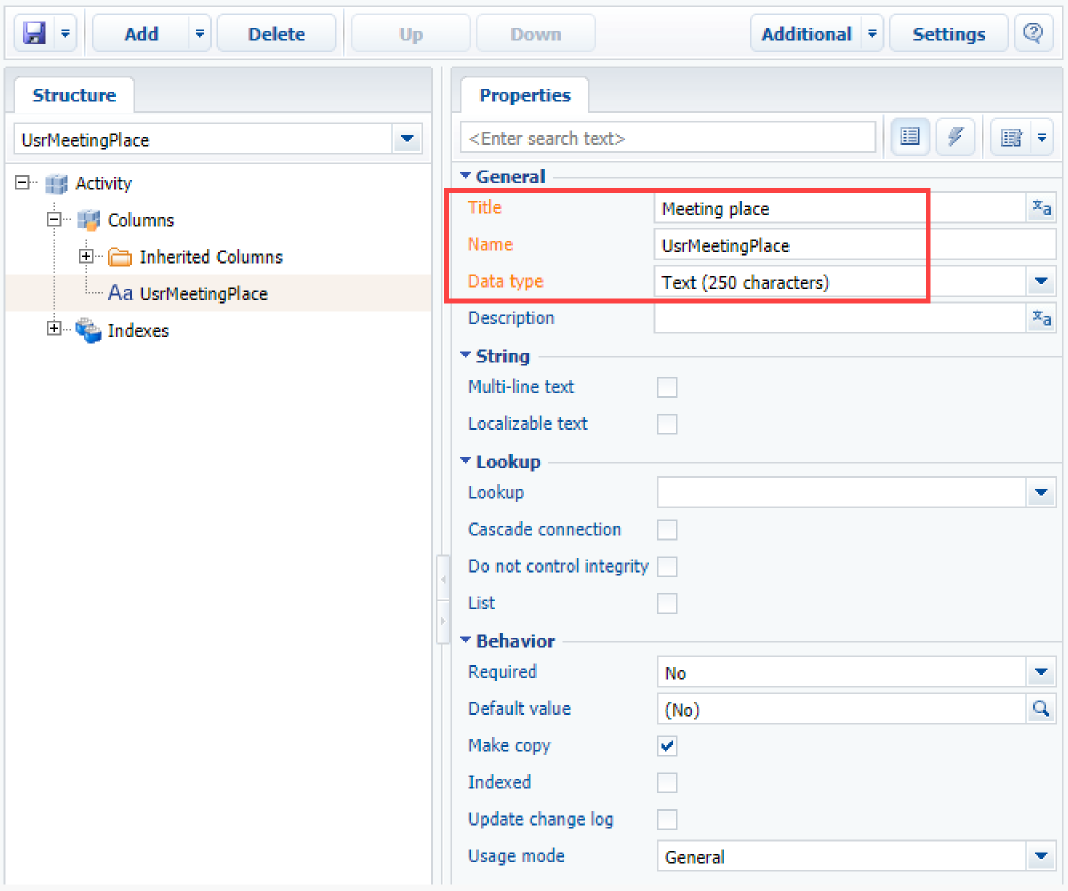
Use this link to download the case implementation package.

Case implementation algorithm

1. Create a replacing object and add a new column to it.

Create an [*Activity*] replacing object and add a new [*Meeting place*] column of the "string" type to it (Fig. 1). Learn more about creating a replacing object schema in the "Create the entity schema" article.

Fig. 1. Adding a custom column to the replacing object



Add | Delete | Up | Down | Additional | Settings

Structure

UsrMeetingPlace

- Activity
  - Columns
    - Inherited Columns
    - UsrMeetingPlace
  - Indexes

Properties

General

Title	Meeting place
Name	UsrMeetingPlace
Data type	Text (250 characters)
Description	

String

Multi-line text	☐
Localizable text	☐

Lookup

Lookup	
Cascade connection	☐
Do not control integrity	☐
List	☐

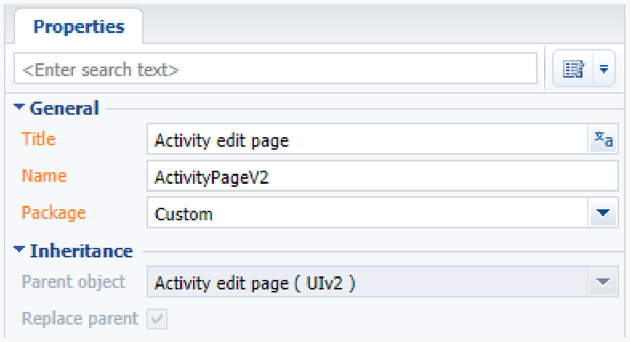
Behavior

Required	No
Default value	(No)
Make copy	✓
Indexed	☐
Update change log	☐
Usage mode	General

2. Create a replacing client module for the activity page

Create a replacing client module and specify the [*Activity edit page*] schema as parent object (Fig. 2). The procedure for creating a replacing page is covered in the “Create a client schema” article.

Fig. 2. Replacing edit page properties



Properties

General

Title	Activity edit page
Name	ActivityPageV2
Package	Custom

Inheritance

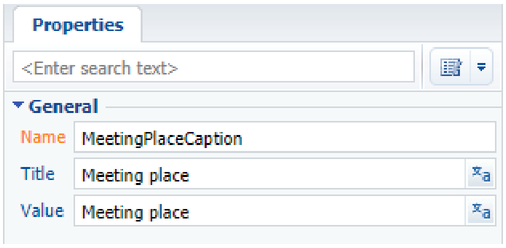
Parent object	Activity edit page (UIv2)
Replace parent	✓

3. Add a new field to the activity edit page.

Add a configuration object with the [*Meeting place*] field properties on the page to the diff array. The process of adding fields to pages is covered in the “Adding a new field to the edit page” article.

To enable localization of this field, add a localizable string (Fig. 3) and bind it to the field title.

Fig. 3. Localizable string properties



Name: MeetingPlaceCaption
Title: Meeting place
Value: Meeting place

4. Add a rule to the “rules” property of the page view model

For the `UsrMeetingPlace` column, add the rule with the BINDPARAMETER type to `rules` property of the page view model. Set the `BusinessRuleModule.enums.Property.VISIBLE` value for the rule’s `property`. Add the following condition for rule execution to the `conditions` array: the value in the `ActivityCategory` column of the model should be equal to the `ConfigurationConstants.Activity.ActivityCategory.Meeting` configuration constant.

Note.

The `ConfigurationConstants.Activity.ActivityCategory.Meeting` configurational constant contain the id of the “Meeting” record of the [*Activity category*] lookup.

The replacing schema source code is as follows:

```
// Add the module BusinessRuleModule and ConfigurationConstants to the dependency list of the mo
define("ActivityPageV2", ["BusinessRuleModule", "ConfigurationConstants"],
    function(BusinessRuleModule, ConfigurationConstants) {
        return {
            // Name of the page schema of the edit page.
            entitySchemaName: "Activity",
            // Displaying a new field on the edit page.
            diff: /**SCHEMA_DIFF*/[
                // Metadata for adding a field [Meeting place].
                {
                    // The operation of adding a component to a page.
                    "operation": "insert",
                    // The meta name of the parent container to which the field is added.
                    "parentName": "Header",
                    // The field is added to the parent
                    // component's collection.
                    "propertyName": "items",
                    // The name of the column of the schema to which the component is bound.
                    "name": "UsrMeetingPlace",
                    "values": {
                        // Field title.
                        "caption": {"bindTo": "Resources.Strings.MeetingPlaceCaption"},
                        // Location of the field.
                        "layout": { "column": 0, "row": 5, "colSpan": 12 }
                    }
                }
            ]/**SCHEMA_DIFF*/,
            // Rules of the edit page view model.
            rules: {
                // A set of rules for the [Meeting place] column of the view model.
                "UsrMeetingPlace": {
                    // The dependence of visibility of the [Meeting Place] field from the value
                    "BindParametrVisibilePlaceByType": {
                        // The type of the BINDPARAMETER rule.
                        "ruleType": BusinessRuleModule.enums.RuleType.BINDPARAMETER,
                        // Rule regulates the VISIBLE property.
                        "property": BusinessRuleModule.enums.Property.VISIBLE,
                        // An array of conditions in which the rule is triggered.
                        // Determines whether the value in the [Category] column is equal to the
                        "conditions": [{
                            // Expression of the left side of the condition.
                            "leftExpression": {
                                //The type of the expression is the attribute (column) of the vi
                                "type": BusinessRuleModule.enums.ValueType.ATTRIBUTE,
                                // Name of the view model column which value is compared in the
                                "attribute": "ActivityCategory"
                            },
                            // The type of comparison operation.
                            "comparisonType": Terrasoft.ComparisonType.EQUAL,
                            // Expression of the right side of the condition.
                            "rightExpression": {
                                // Type of expression is a constant value.
                                "type": BusinessRuleModule.enums.ValueType.CONSTANT,
                                // The value with which the left side expression is compared.
                                "value": ConfigurationConstants.Activity.ActivityCategory.Meetir
                            }
                        }]
                    }
                }
            }
        };
    });
```

After saving the schema and refreshing the application page, an additional [*Meeting place*] field will appear on the activity page if the activity category is “Meeting” (Fig. 5, 5).

Fig. 4. Case result. Activity type is “To do”, the [*Meeting place*] field is not visible

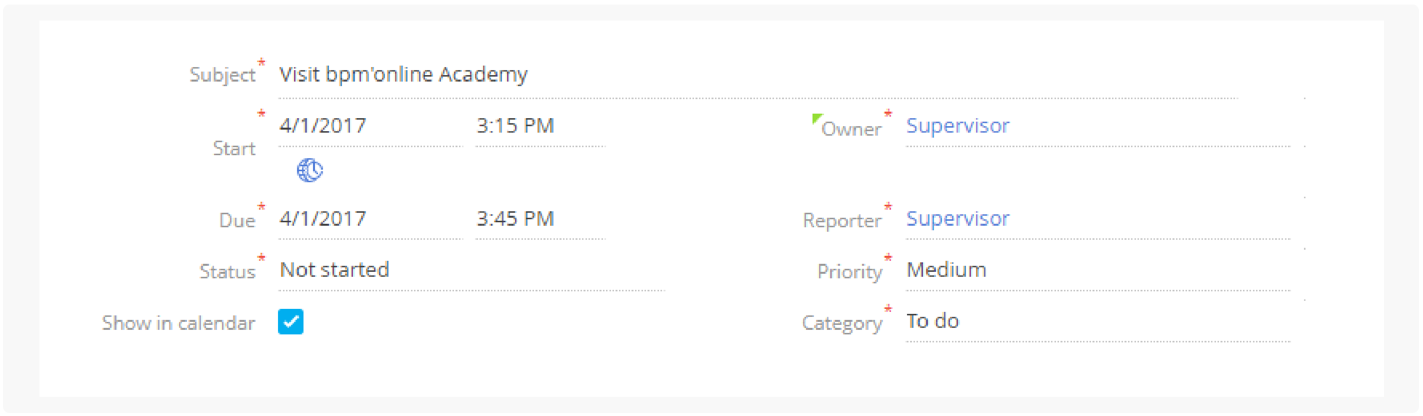
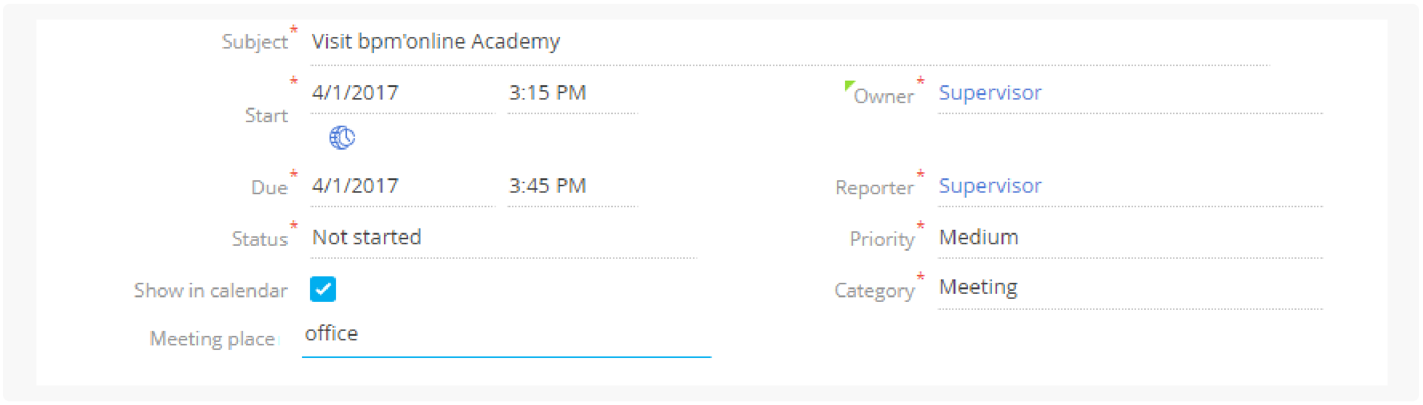

Subject* Visit bpm'online Academy
Start* 4/1/2017 3:15 PM
Owner* Supervisor
Due* 4/1/2017 3:45 PM
Reporter* Supervisor
Status* Not started
Priority* Medium
Show in calendar ✓
Category* To do

Fig. 5. Case result. Activity type is “To do”, the [*Meeting place*] field is visible


Subject* Visit bpm'online Academy
Start* 4/1/2017 3:15 PM
Owner* Supervisor
Due* 4/1/2017 3:45 PM
Reporter* Supervisor
Status* Not started
Priority* Medium
Show in calendar ✓
Category* Meeting
Meeting place office

Add auto-numbering to the field

Advanced

Case description

Set up auto numbering for the [*Code*] field in the [*Products*] section. The product code format must be as follows: ART_00001, ART_00002 and so on.

Attention.

We covered two alternative ways of case implementation: client- and server-side.

Source code

You can download the package with case implementation using the following link.

Attention.

The package does not contain the bound system settings `ProductCodeMask` and `ProductLastNumber`. You will need to add them manually.

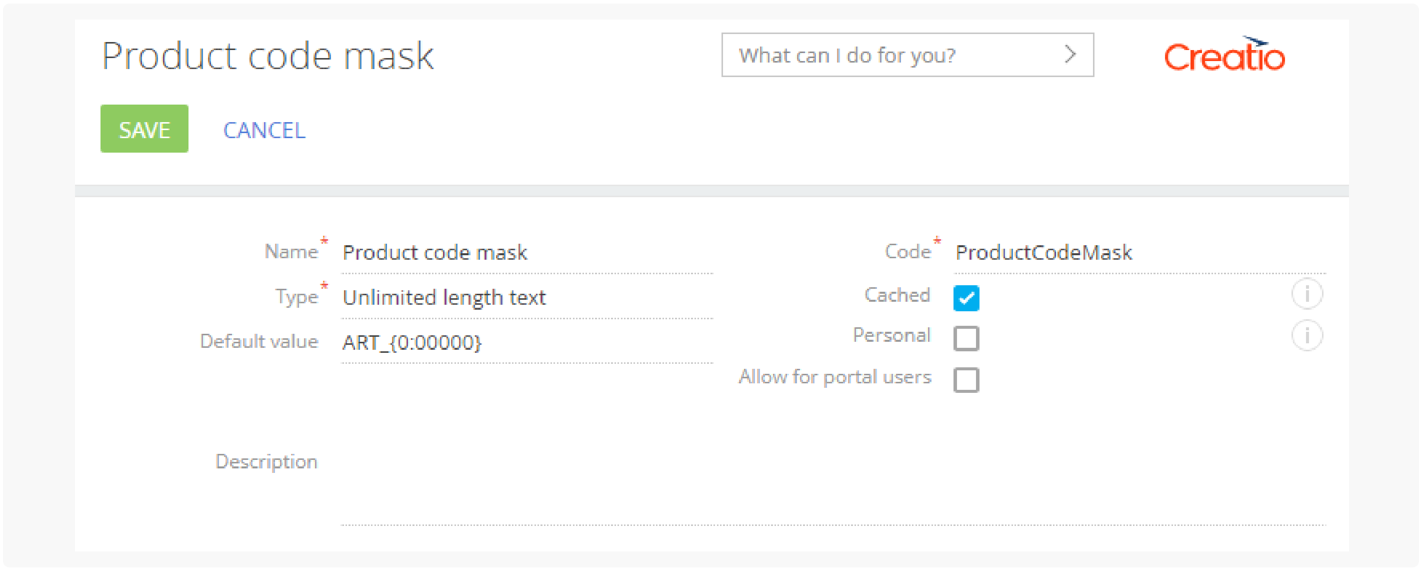
Case implementation algorithm: client-side

1. Create two system settings

Create the [*Product code mask*] system setting with the following number mask: “ART_{0:00000}” (Fig.1) Populate the following fields:

- [Name] – “Product code mask”.
- [Code] – “ProductCodeMask”.
- [Type] – a string, whose length depends on the number of characters in the mask. In most cases 50 characters are enough. In this example, we use a string of unlimited length.
- [Default value] – "ART_{0:00000}".

Fig. 1. The [*Product code mask*] system setting

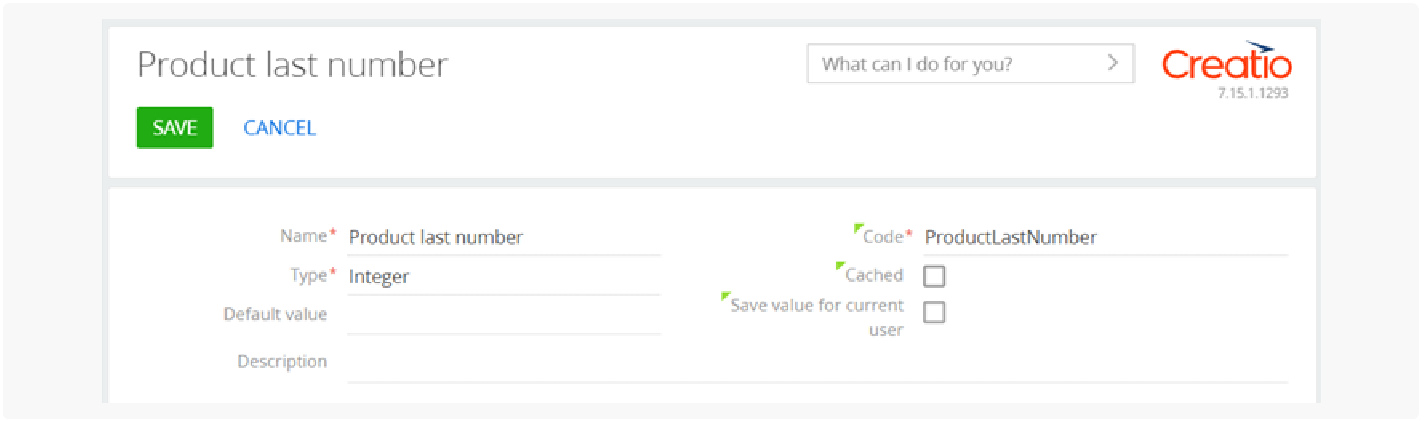


Name*	Product code mask	Code*	ProductCodeMask
Type*	Unlimited length text	Cached	✓
Default value	ART_{0:00000}	Personal	☐
		Allow for portal users	☐
Description			

Create the [*Product last number*] system setting (Fig.2). Populate its properties:

- [Name] – “Product last number”.
- [Code] – “ProductLastNumber”.
- [Type] – “Integer”.

Fig. 2. The [*Product last number*] system setting



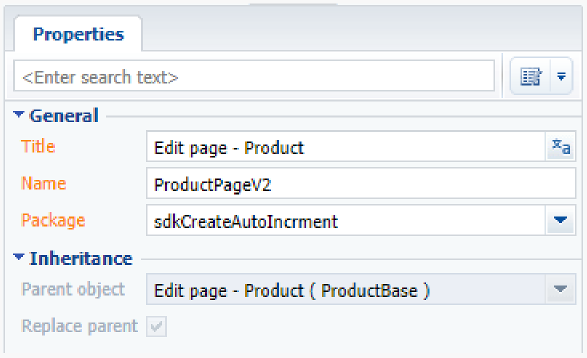
Product last number

Name*	Product last number	Code*	ProductLastNumber
Type*	Integer	Cached	☐
Default value		Save value for current user	☐
Description			

2. Create a replacing schema in the custom package

Create a replacing client module and specify the `ProductPageV2` schema as parent object (Fig. 3). The procedure for creating a replacing page is covered in the "Create a client schema" article.

Fig. 3. Properties of the product edit page replacing schema



General

Title	Edit page - Product
Name	ProductPageV2
Package	sdkCreateAutoIncrment

Inheritance

Parent object	Edit page - Product (ProductBase)
Replace parent	✓

3. Override the onEntityInitialized() method

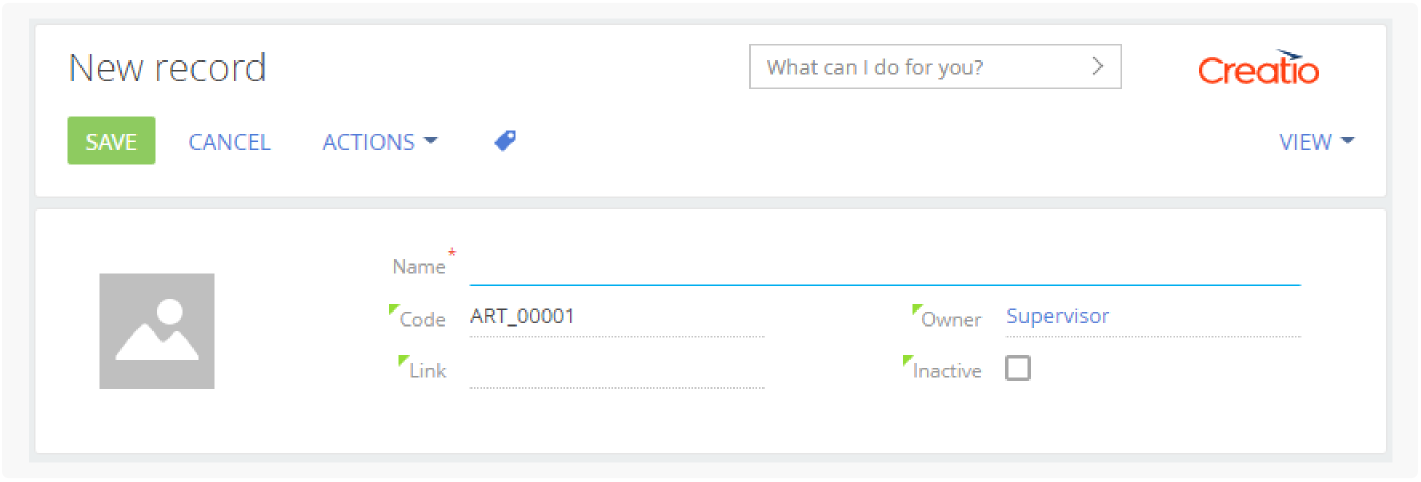
In the collection of edit page view model methods, override the `onEntityInitialized()` method. In `onEntityInitialized()` method, call the `getIncrementCode()` method and fill in the generated number in the [*Code*] column of its callback function. The replacing schema source code is as follows:

```
define("ProductPageV2", [], function() {
    return {
        // The name of edit page object schema.
        entitySchemaName: "Product",
        //  The collection of edit page view model methods.
        methods: {
            // Overriding Terrasoft.BasePageV2.onEntityInitialized base method, that
            // is triggered when the initialization of the edit page object schema is finished.
            onEntityInitialized: function() {
                // onEntityInitialized method parent realization is called.
                this.callParent(arguments);
                // The code is generated only in case we create a new element or a copy of the e
```

```
                    if (this.isAddMode() || this.isCopyMode()) {
                        //  Call of the Terrasoft.BasePageV2.getIncrementCode base method, that gene
                        // according to the previously set mask.
                        this.getIncrementCode(function(response) {
                            // The generated number is stored in [Code] column.
                            this.set("Code", response);
                        });
                    }
                }
            }
        };
    });
```

After saving the schema, clearing the browser cache and updating the application page, the automatically generated product code will be displayed when you add a new product (Fig.4).

Fig. 4. The result of case implementation on the client side



Code ART_00001
Owner Supervisor

Case implementation algorithm: server-side

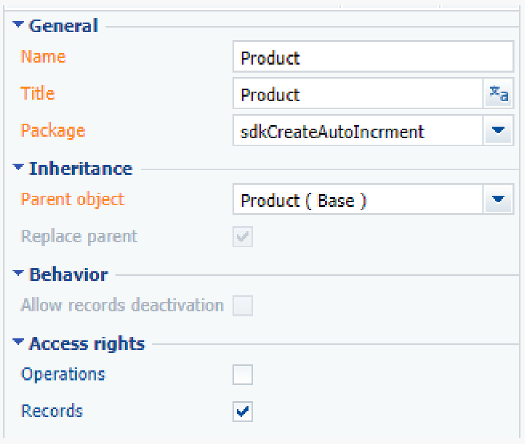
1. Create two system settings

This step is absolutely identical to the first step of case implementation algorithm on the client side.

2. Create a replacing schema of the [Product] object

Select a custom package and execute the [*Add*] – [*Replacing object*] menu command on the [*Schemas*] tab. Specify the [*Product*] object as the parent object in the new object properties (Fig. 5).

Fig. 5. Properties of the product replacing schema

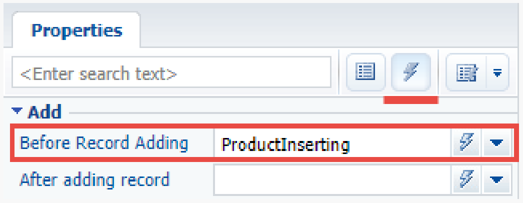


General
Name: Product
Title: Product
Package: sdkCreateAutoIncrment
Inheritance
Parent object: Product (Base)
Replace parent: ✓
Behavior
Allow records deactivation: ☐
Access rights
Operations: ☐
Records: ✓

3. Add the [Before record adding] event handler to the object schema

Add a new event handler in the object properties displayed in the object designer. To do this, go to the event tab and double-click the [*Before Record Adding*] field or click the event icon in this field (Fig.6).

Fig. 6. The [*Before record adding*] product event handler



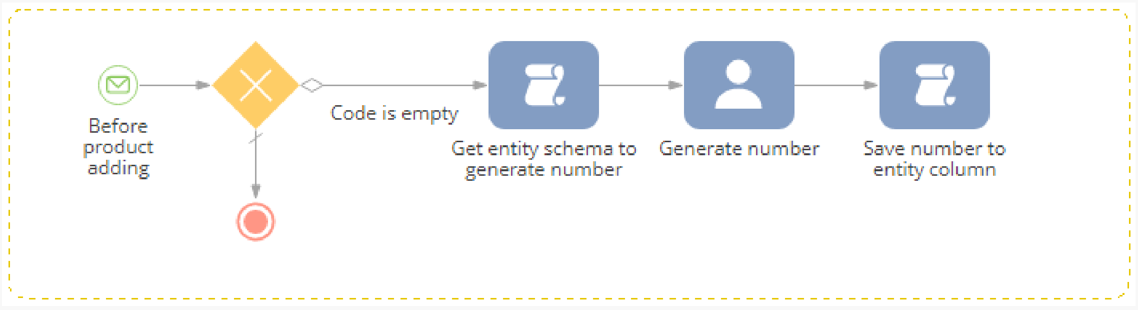
Properties
<Enter search text>
Add
Before Record Adding: ProductInserting
After adding record:

The object’s process designer will open.

4. Add an event sub-process

To implement the [*Before Record Adding*] event handler, add event sub-process to the working area of the object process designer. Set up a business process for number generation (Fig.7).

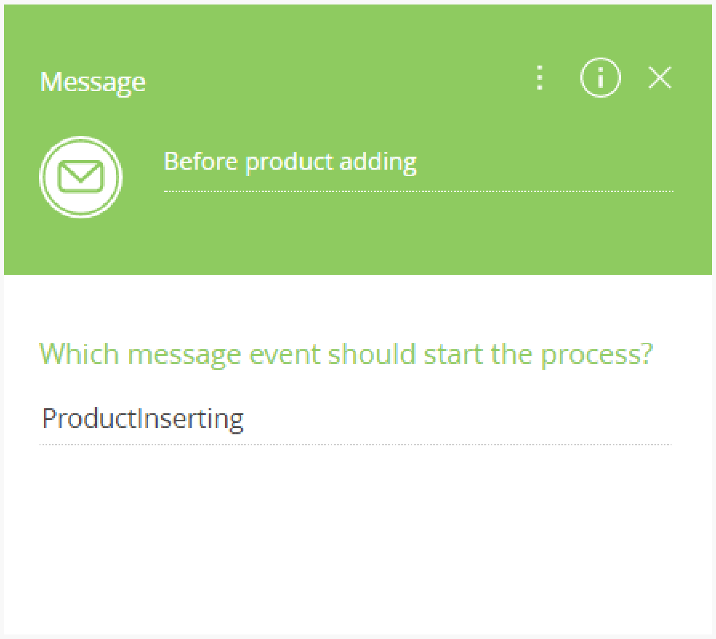
Fig. 7. The sub-process for the [*Before Record Adding*] event handler


Before product adding
Code is empty
Get entity schema to generate number
Generate number
Save number to entity column

Event sub-process elements

1. Initial message [*Before product adding*] (Fig.8) – the sub-process will be run upon receiving the `ProductInserting` message added at step3.

Fig. 8. The [*Before product adding*] initial message properties

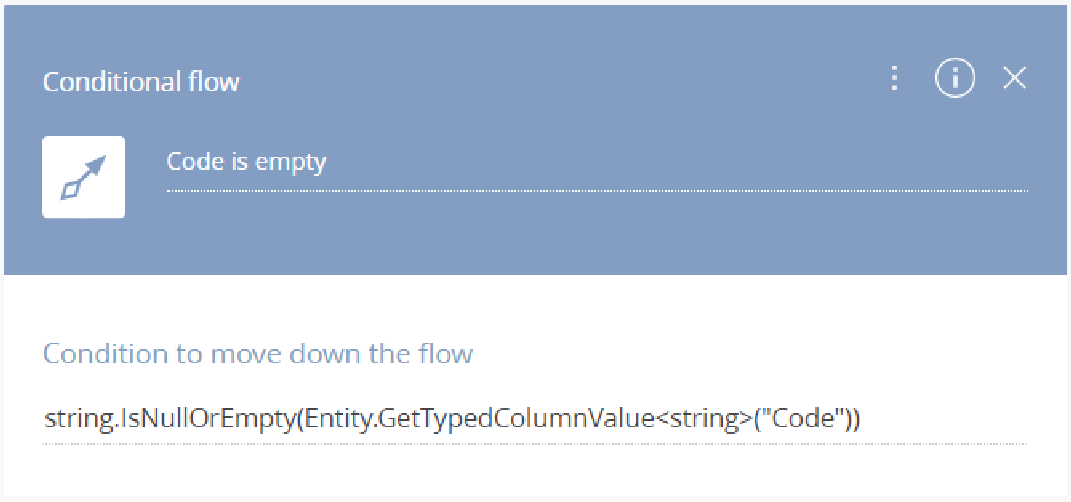


ProductInserting

2. [*Exclusive gateway (OR)*], which branches the process into two flows:

- Default flow – the transition down this flow will occur if the condition flow cannot be implemented. This branch finishes with the [Terminate] event.
- Condition flow [Code is empty] – checks whether the [Code] column is populated (Fig.9). The further execution of the sub-process can only be possible if the column is not populated.

Fig. 9. The [*Code is empty*] condition flow properties



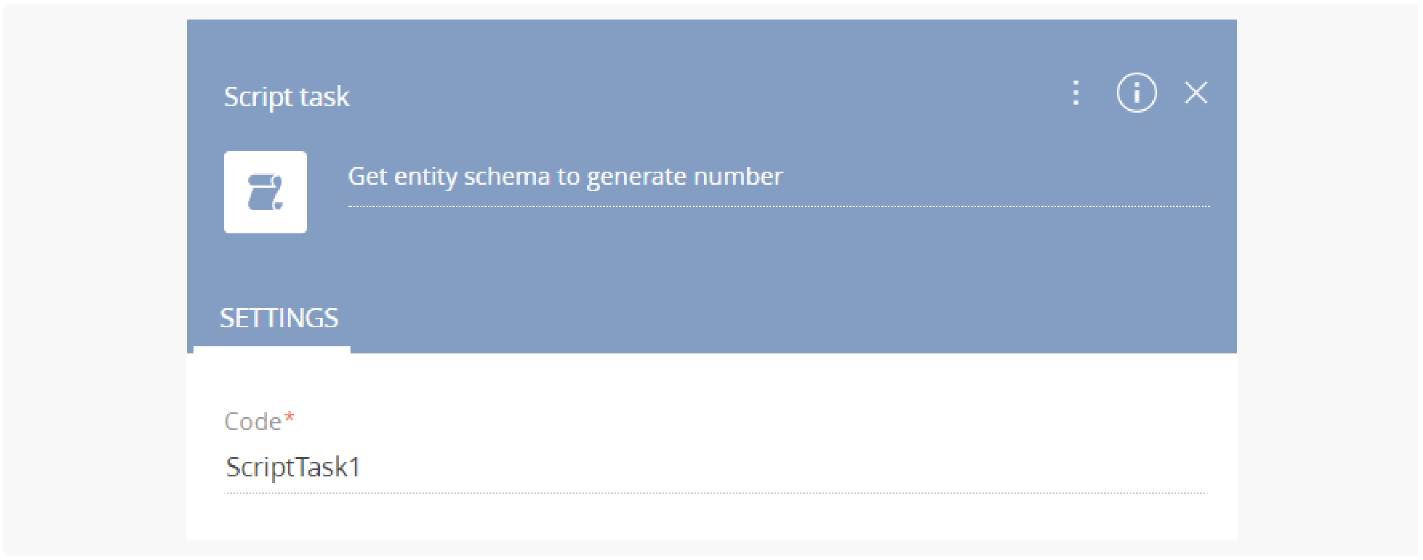
string.IsNullOrEmpty(Entity.GetTypedColumnValue<string>("Code"))

Add the following code to the Condition field of the condition flow

```
string.IsNullOrEmpty(Entity.GetTypedColumnValue<string>("Code"))
```

3. Script task [*Get entity schema to generate number*] (Fig.10).

Fig. 10. [*Get entity schema to generate number*] script task properties



Script task

Get entity schema to generate number

SETTINGS

Code*

ScriptTask1

Program code C# script is executed in this element. To add it double-click the element. Add...

```
//Setting the schema for number generation.
UserTask1.EntitySchema = Entity.Schema;
return true;
```

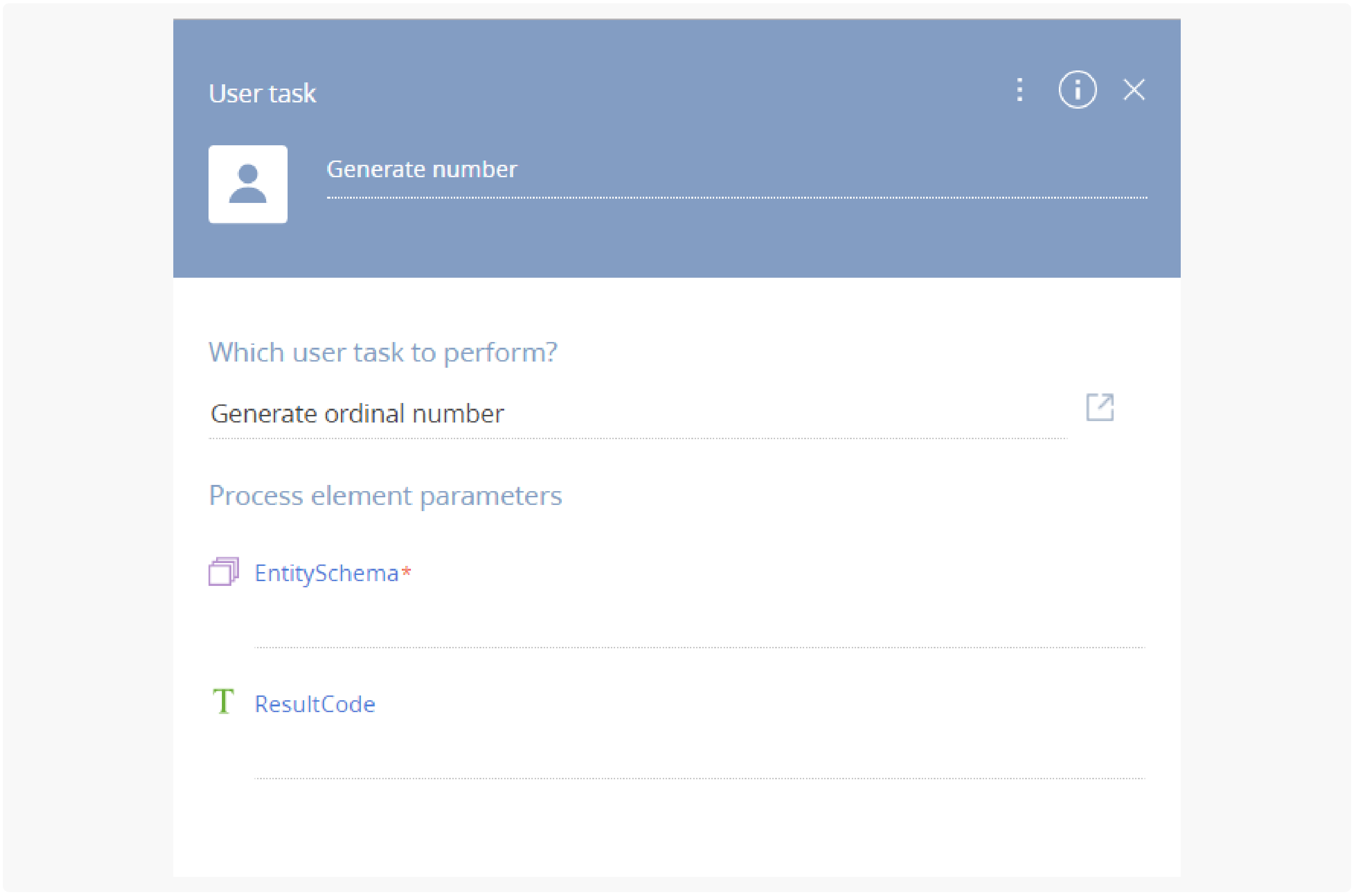
Note that “UserTask1” here is the name of the [*Generate number*] user action.

Attention.

Save the script after adding the source code. To do this, select the [*Save*] menu action.

4. User task [*Generate number*] (Fig.11).

Fig. 11. The [*Generate number*] user task properties


User task

Generate number

Which user task to perform?

Generate ordinal number

Process element parameters

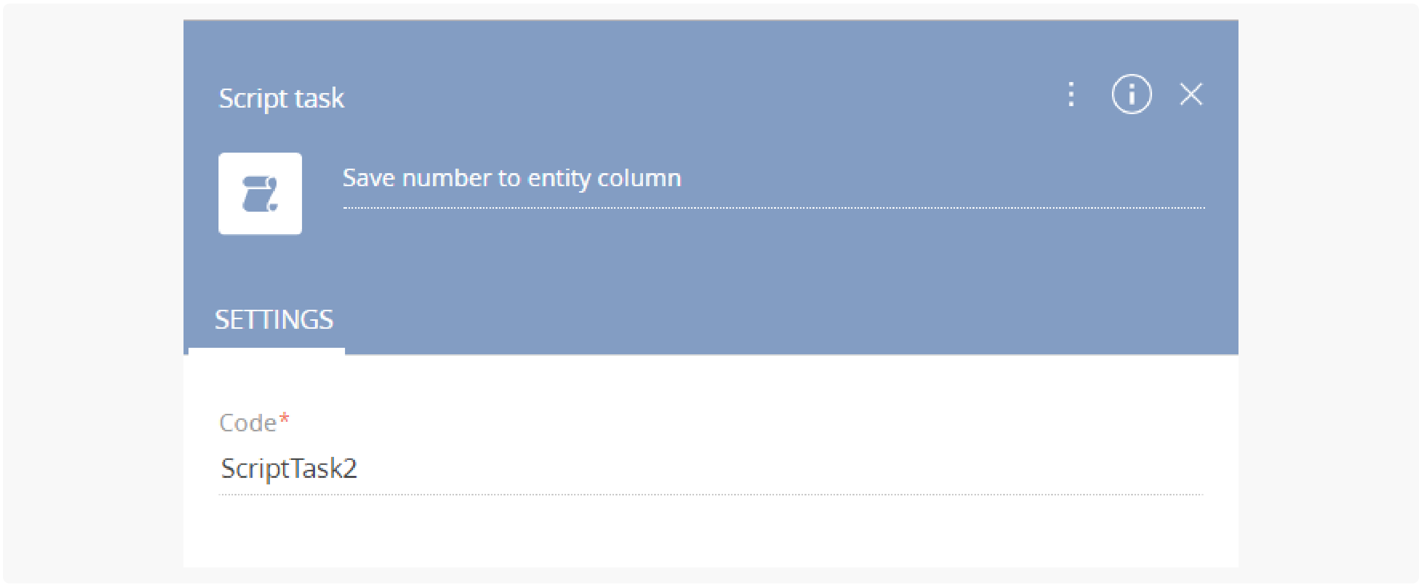
EntitySchema*

ResultCode

This element performs the [*Generate ordinal number*] system action. It is the [*Generate ordinal number*] system action, which generates the current ordinal number in accordance with the `ProductCodeMask` mask set in the system settings.

5. Script task [*Save number to entity column*] (Fig.12).

Fig. 12. The [*Save number to entity column*] script task properties


Script task

Save number to entity column

SETTINGS

Code*

ScriptTask2

Program code C# script is executed in this element. The value generated by the UserTask1 ...

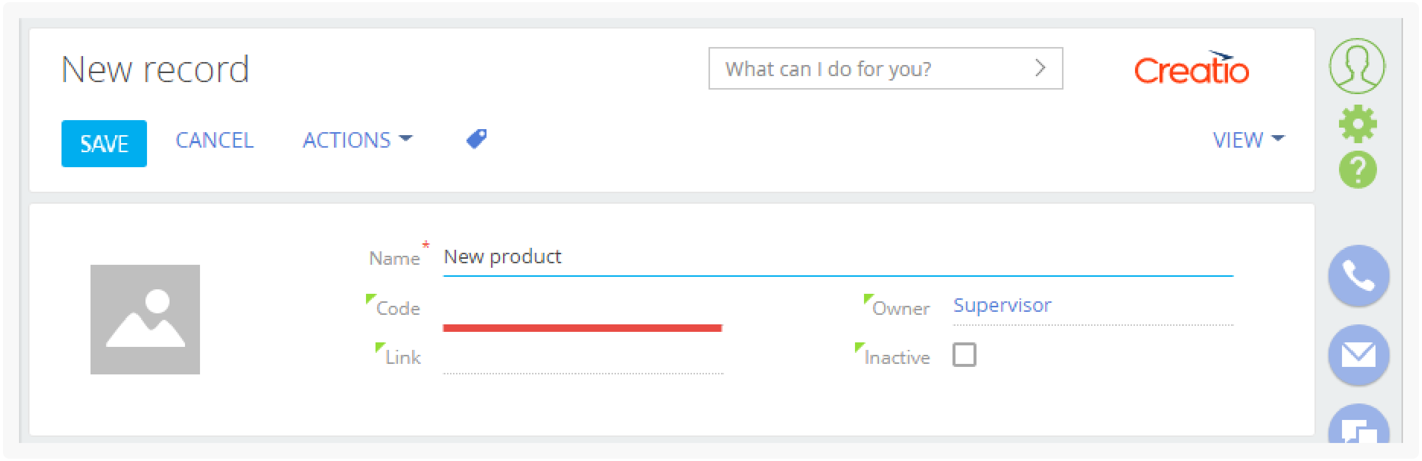
```
Entity.SetColumnValue("Code", UserTask1.ResultCode);
return true;
```

Save and close the default process designer and publish the [*Product*] object schema. As a result, after saving the new product, the [*Code*] field will be automatically populated on the product page (Fig.13, Fig.14).

Attention.

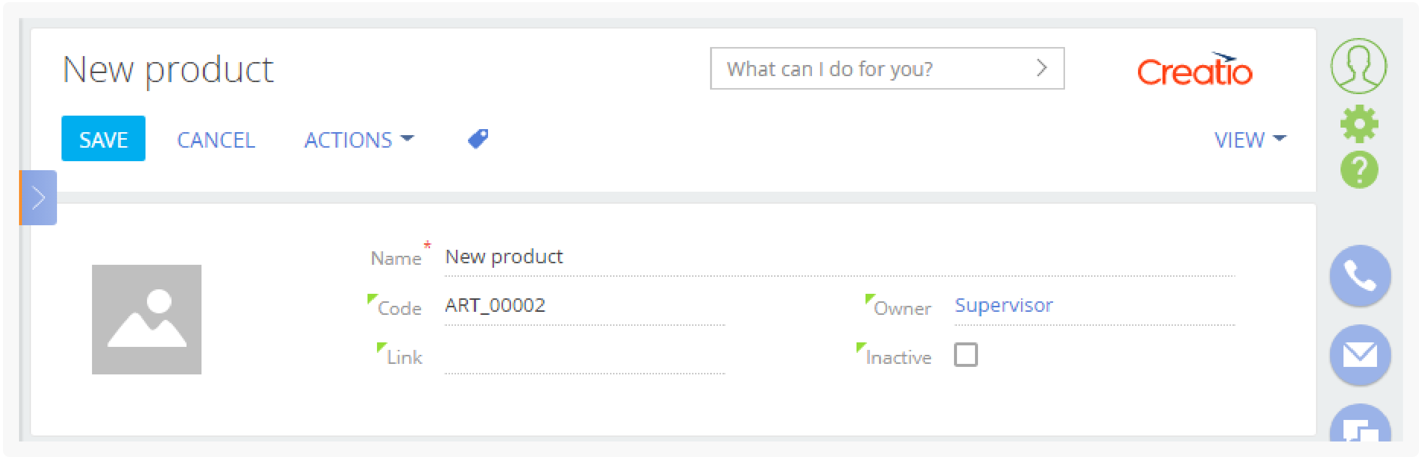
Since code auto generation and saving in the column is performed on the server side when the [*Before saving record*] event occurs, it is impossible to view the code value on the product page immediately. This is because the [*Before saving record*] event occurs on the server side after sending the request to add a record from the application client part.

Fig. 13. The code is not displayed when creating a product



Name* New product
Code
Owner Supervisor
Link
Inactive ☐

Fig. 14. The code is displayed in the saved product



Name* New product
Code ART_00002
Owner Supervisor
Link
Inactive ☐