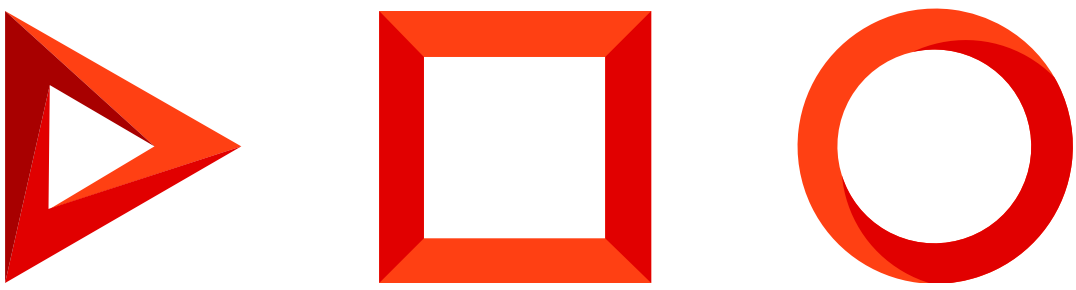
Creatio Academy

Record page

Action dashboard

Version 8.0



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

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

© 2023 Creatio. All rights reserved.

Table of Contents

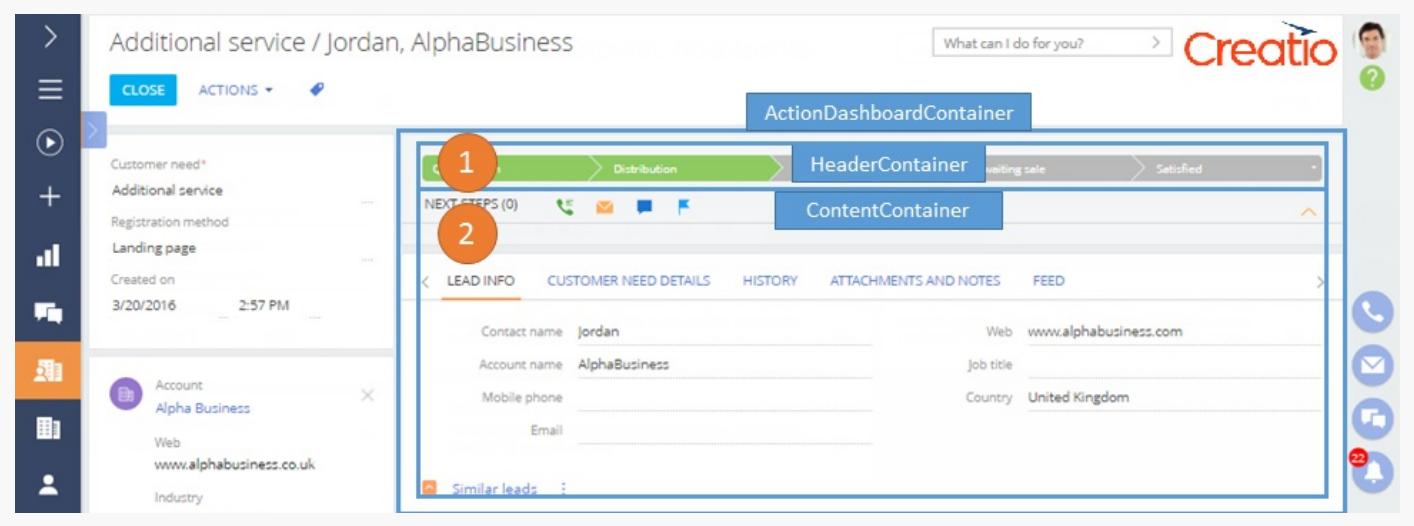
© 2023 Creatio. All rights reserved.

Action dashboard

Beginner

The action dashboard is designed to display information about the current state of a record and consists of two parts:

- The **Workflow bar** (1) — shows the business process stage status.
- The **Action panel** (2) — enables you to move on to the activity, work with email or feed, without leaving the section. The action dashboard displays business process activities that are connected to the section object by the corresponding field. The action panel can also display an auto-generated page, pre-configured page, question or object page.



The action dashboard is located in the `ActionDashboardContainer` container of the section record edit page. The workflow bar is located in the attached `HeaderContainer` container, and the action panel — in the `ContentContainer`.

The arrangement of the elements of the action dashboard is configured by the `BaseActionsDashboard` view model schema and the `SectionActionsDashboard` inherited schema of the `ActionsDashboard` package.

Starting with version 7.8.0, Creatio has a new edit page module – the "Action panel" (ActionsDashboard). An action panel displays information about the current status and actions for working with the current record.

General procedure of adding an action panel on a page:

1. Create a Schema of the Edit Page View Model inherited from the `SectionActionsDashboard` module.
2. Create a replacing page schema.
3. Set up the module in the `modules` property of the page view model.
4. In the "diff" array of the page view model, add the module on the page.

ActionsDashboard channels are a way of communicating with a contact. A channel is created for every section in which it's connected to, for example, a case, contact, or lead.

© 2023 Creatio. All rights reserved.

Add an action panel

Medium

Case description

Add an action panel to the order edit page.

Source code

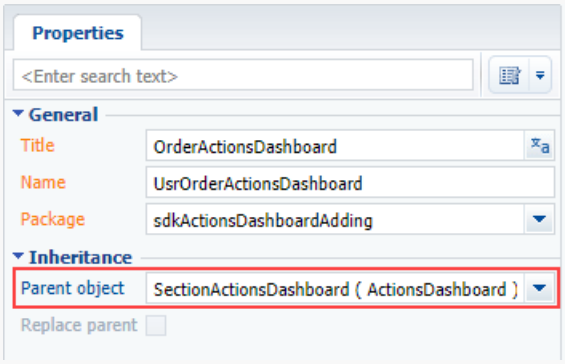
You can download the package with case implementation using the following link.

Case implementation algorithm

1. Create a client schema of the OrderActionsDashboard view model

Specify the `SectionActionsDashboard` schema as a parent object (Fig. 1).

Fig. 1. Properties of the client schema



The client schema source code is as follows:

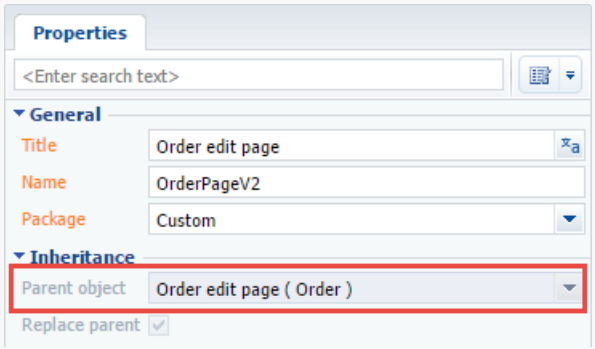
```
define("UsrOrderActionsDashboard", [], function () {
    return {
        details: /**SCHEMA_DETAILS*/{}/**SCHEMA_DETAILS*/,
        methods: {},
        diff: /**SCHEMA_DIFF*/[]/**SCHEMA_DIFF*/
    };
});
```

2. Create a replacing 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. 2). Creating a replacing page is covered in the “Create a client schema” article.

© 2023 Creatio. All rights reserved.

Fig. 2. Properties of the replacing edit page



3. Add a configuration object with the module settings in the modules collection of the page schema

Add the code of the page replacing module to the [*Source code*] tab: Add a configuration object with the module settings in it to the `modules` collection of the view model.

4. Add a configuration object with the settings determining the module position in the diff array

The replacing schema source code is as follows:

```
define("OrderPageV2", [],
    function () {
        return {
            entitySchemaName: "Order",
            attributes: {},
            modules: /**SCHEMA_MODULES*/{
                "ActionsDashboardModule": {
                    "config": {
                        "isSchemaConfigInitialized": true,
                        // Schema name.
                        "schemaName": "UsrOrderActionsDashboard",
                        "useHistoryState": false,
                        "parameters": {
                            // Configuration object of the view model.
                            "viewModelConfig": {
                                // Schema name of the page entity.
                                "entitySchemaName": "Order",
                                // Configuration object of the Actions block.
                                "actionsConfig": {
                                    // Schema name for loading items to Actions.
                                    "schemaName": "OrderStatus",
                                     // Column name in the parent schema that references the sch
                                     // If not specified, takes the schemaName value.
                                    "columnName": "Status",
```

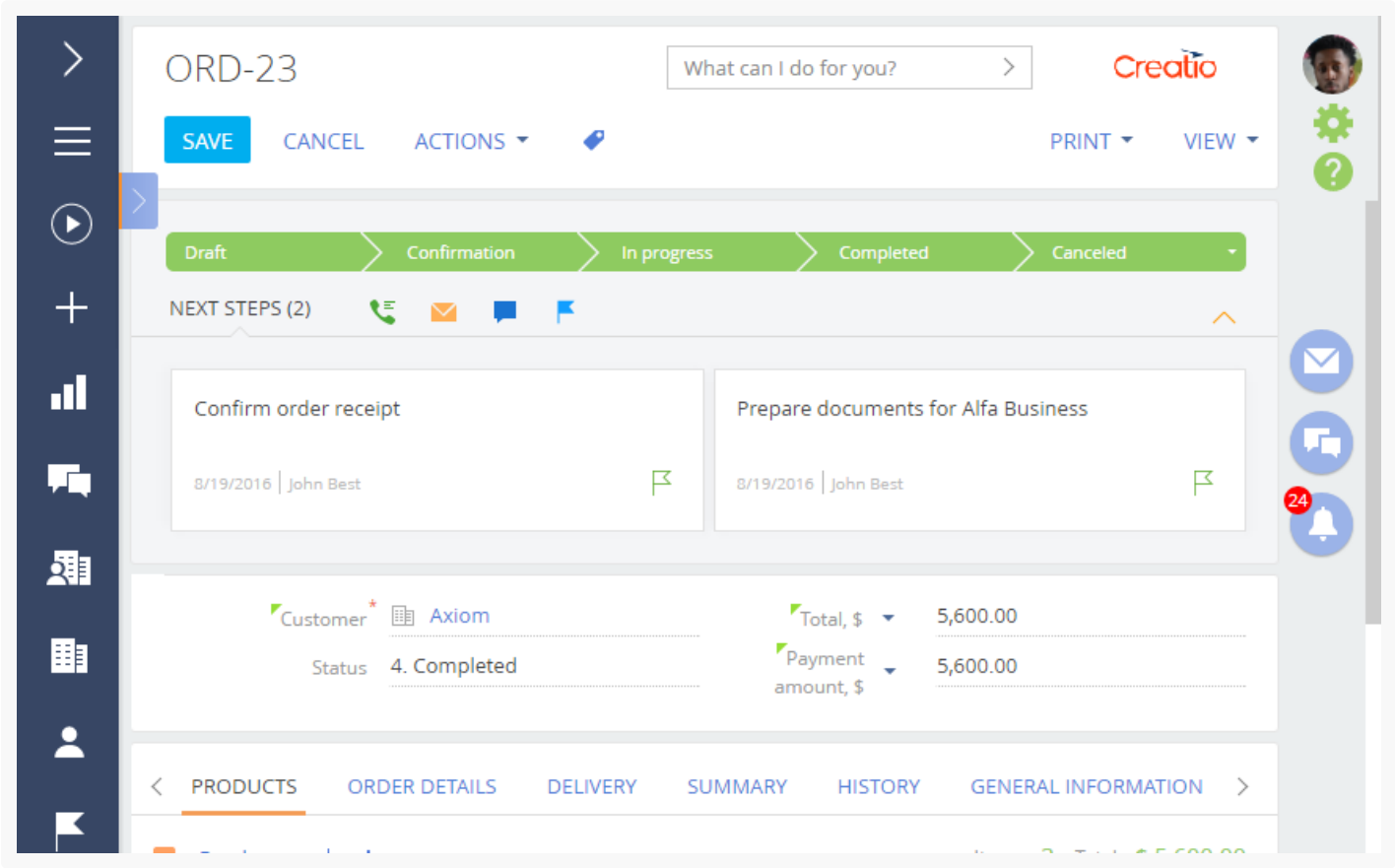
© 2023 Creatio. All rights reserved.

```
                                    // Column name for element sorting.
                                    "orderColumnName": "Position",
                                    // Column name for item sorting in the item menu.
                                    "innerOrderColumnName": "Position"
                                },
                                // Displaying the action panel module, the value is [true] by de
                                "useDashboard": true,
                                // Displaying the Content block, [true] by default.
                                "contentVisible": true,
                                // Risplaying the Header block, [true] by default.
                                "headerVisible": true,
                                // The configuration object of the panel elements.
                                "dashboardConfig": {
                                    // Connection between activities and page object.
                                    "Activity": {
                                        // Page object column name.
                                        "masterColumnName": "Id",
                                        // Clumn name in the [Activity] object.o
                                        "referenceColumnName": "Order"
                                    }
                                }
                            }
                        }
                    }
                }
            }/**SCHEMA_MODULES*/,
            details: /**SCHEMA_DETAILS*/{}/**SCHEMA_DETAILS*/,
            methods: {},
            diff: /**SCHEMA_DIFF*/[
                {
                    "operation": "insert",
                    "name": "ActionsDashboardModule",
                    "parentName": "ActionDashboardContainer",
                    "propertyName": "items",
                    "values": {
                        "classes": { wrapClassName: ["actions-dashboard-module"] },
                        "itemType": Terrasoft.ViewItemType.MODULE
                    }
                }
            ]/**SCHEMA_DIFF*/
        };
    });
```

After saving the schema and updating the Creatio web page, the action panel will be added to the order page. The action panel will contain the order status and connected uncompleted activities (Fig. 3).

© 2023 Creatio. All rights reserved.

Fig. 3. Demonstrating the case implementation result



Add a new channel to the action panel

Advanced

Case description

Add a new custom channel to the action dashboard of the contact edit page. The channel must have the same functionality as the call results channel (`CallMessagePublisher` channel).

Source code

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

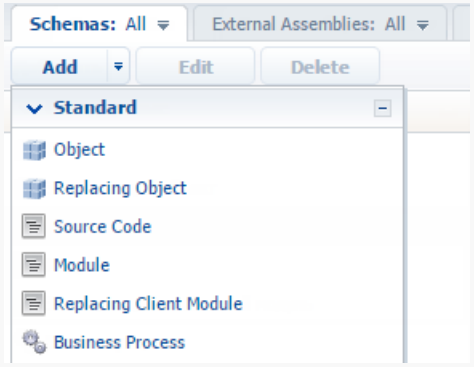
Case implementation algorithm

1. Add the `UsrCallsMessagePublisher` source code schema

Perform the [*Add*] > [*Source code*] menu command on the [*Schema*] tab in the [*Configuration*] section (Fig. 1).

Fig. 1. Adding source code schema

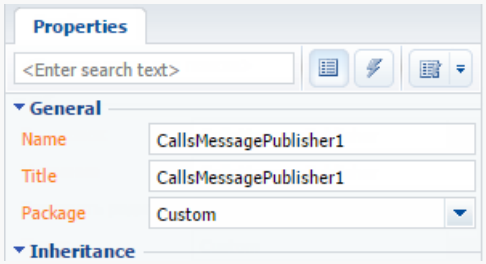
© 2023 Creatio. All rights reserved.



For the created schema specify (Fig. 2):

- [Title] – "Call message logging publisher"
- [Name] – “UsrCallsMessagePublisher”

Fig. 2. The Source code schema properties



In the created schema, add the new `CallsMessagePublisher` class inherited from the `BaseMessagePublisher` class to the `Terrasoft.Configuration` namespace. The `BaseMessagePublisher` class contains the basic logic to save an object in the database and the basic logic of event handlers. The inheritor class will contain the logic for a particular sender, for example, filling of columns of the `Activity` object and the subsequent sending of the message.

To implement the new `CallsMessagePublisher` class, you must add the following source code in the created schema.

```
using System.Collections.Generic;
using Terrasoft.Core;

namespace Terrasoft.Configuration
{
    // The BaseMessagePublisher heir class.
    public class CallsMessagePublisher : BaseMessagePublisher
    {
        // Class constructor.
        public CallsMessagePublisher(UserConnection userConnection, Dictionary<string, string> e
            : base(userConnection, entityFieldsData) {
            //The schema the CallsMessagePublisher works with.
            EntitySchemaName = "Activity";
```

© 2023 Creatio. All rights reserved.

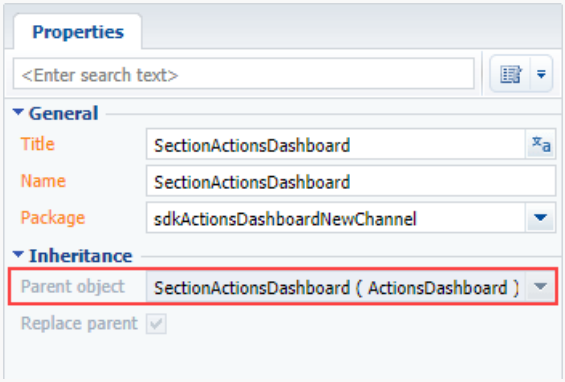
```
        }
    }
}
```

Save and publish the schema.

2. Create the SectionActionsDashboard replacing client schema

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

Fig. 3. Properties of the replacing schema



Note.

If you want to add a channel to only one edit page, you must create a new module named [*section_name*] `SectionActionsDashboard` (e.g. `BooksSectionActionsDashboard`) and set `SectionActionsDashboard` as the parent schema.

Specify the module which should be rendered in this channel on one of the tabs in the replacing schema `diff` property. Set the operations of inserting the `CallsMessageTab` tab and message container in this property. The new channel will be visible on the edit pages of those sections, which are connected to `SectionActionsDashboard`.

In the `methods` property override the `getSectionPublishers()` method that will add the new channel to the list of message publishers, and the `getExtendedConfig()` method, in which the tab settings are configured.

For the `getExtendedConfig()` method to run correctly, you must upload the channel icon and specify it in the `ImageSrc` parameter. The icons used in this example can be downloaded here.

You should also override the `onGetRecordInfoForPublisher()` method and add the `getContactEntityParameterValue()` method that defines the contact value from the edit page.

The replacing schema source code is as follows:

```
define("SectionActionsDashboard", ["SectionActionsDashboardResources", "UsrCallsMessagePublisher
```

© 2023 Creatio. All rights reserved.

```
function(resources) {
    return {
        attributes: {},
        messages: {},
        methods: {
            // Method sets the channel tab display settings in the action dashboard.
            getExtendedConfig: function() {
                // Parent method calling.
                var config = this.callParent(arguments);
                var lczImages = resources.localizableImages;
                config.CallsMessageTab = {
                    // Tab image.
                    "ImageSrc": this.Terrasoft.ImageUrlBuilder.getUrl(lczImages.CallsMessage
                    // Marker value.
                    "MarkerValue": "calls-message-tab",
                    // Alignment.
                    "Align": this.Terrasoft.Align.RIGHT,
                    // Tag.
                    "Tag": "UsrCalls"
                };
                return config;
            },
            // Redefines the parent object and adds the contact value from the edit page
            // of the section that contains the action dashboard.
            onGetRecordInfoForPublisher: function() {
                var info = this.callParent(arguments);
                info.additionalInfo.contact = this.getContactEntityParameterValue(info.relat
                return info;
            },
            // Defines the contact value from the section edit page
            // that contains the action dashboard.
            getContactEntityParameterValue: function(relationSchemaName) {
                var contact;
                if (relationSchemaName === "Contact") {
                    var id = this.getMasterEntityParameterValue("Id");
                    var name = this.getMasterEntityParameterValue("Name");
                    if (id && name) {
                        contact = {value: id, displayValue: name};
                    }
                } else {
                    contact = this.getMasterEntityParameterValue("Contact");
                }
                return contact;
            },
            //Adds the created channel to the message publisher list.
            getSectionPublishers: function() {
                var publishers = this.callParent(arguments);
                publishers.push("UsrCalls");
                return publishers;
```

© 2023 Creatio. All rights reserved.

```
            }
        },
        // An array of modifications, with which the representation of the module is built i
        diff: /**SCHEMA_DIFF*/[
            // Adding the CallsMessageTab tab.
            {
                // operation type — insertion.
                "operation": "insert",
                // Tab name.
                "name": "CallsMessageTab",
                // Parent element name.
                "parentName": "Tabs",
                // Property name.
                "propertyName": "tabs",
                // Property configuration object.
                "values": {
                    // Child elements array.
                    "items": []
                }
            },
            // Adding message container.
            {
                "operation": "insert",
                "name": "CallsMessageTabContainer",
                "parentName": "CallsMessageTab",
                "propertyName": "items",
                "values": {
                    // Element type — container.
                    "itemType": this.Terrasoft.ViewItemType.CONTAINER,
                    // Container CSS class.
                    "classes": {
                        "wrapClassName": ["calls-message-content"]
                    },
                    "items": []
                }
            },
            // Adding the UsrCallsMessageModule module.
            {
                "operation": "insert",
                "name": "UsrCallsMessageModule",
                "parentName": "CallsMessageTab",
                "propertyName": "items",
                "values": {
                    // Tab module CSS class.
                    "classes": {
                        "wrapClassName": ["calls-message-module", "message-module"]
                    },
                    // Element type — module.
```

© 2023 Creatio. All rights reserved.

```
                        "itemType": this.Terrasoft.ViewItemType.MODULE,
                        // Module name.
                        "moduleName": "UsrCallsMessagePublisherModule",
                        // Binding the method executed after the element has been rendered.
                        "afterrender": {
                            "bindTo": "onMessageModuleRendered"
                        },
                        // Binding the method executed after the element has been rerendered.
                        "afterrerender": {
                            "bindTo": "onMessageModuleRendered"
                        }
                    }
                }
            ]/**SCHEMA_DIFF*/
        };
    }
);
```

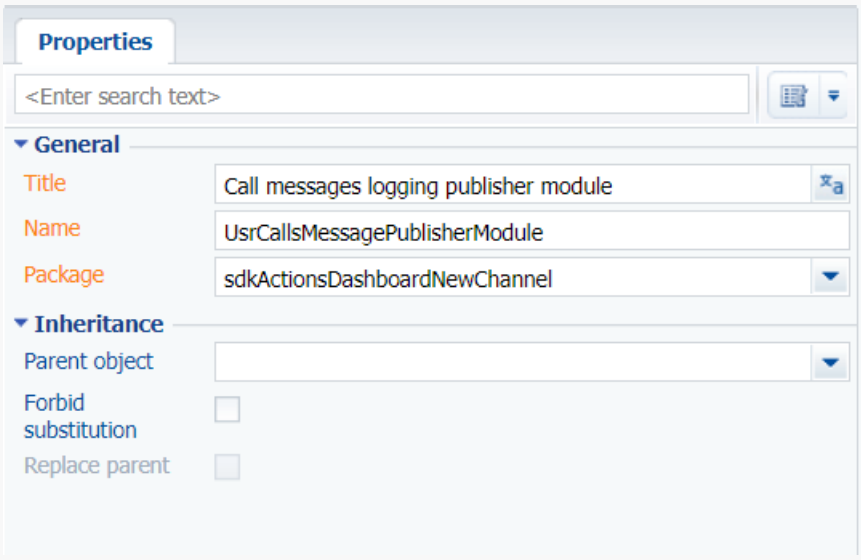
3. Create the UsrCallsMessagePublisherModule module

The `UsrCallsMessagePublisherModule` serves as container that renders the `SectionActionsDashboard` page with implemented logic of added channel in the `UsrCallsMessagePublisherPage`.

Set following properties for the module (Fig. 4):

- [Title] – "Call messages logging publisher module"
- [Name] – “UsrCallsMessagePublisherModule”
- [Parent object] – `BaseMessagePublisherModule.`

Fig. 4. Properties of the module



The module source code:

© 2023 Creatio. All rights reserved.

```
define("UsrCallsMessagePublisherModule", ["BaseMessagePublisherModule"],
    function() {
        // Defining the class.
        Ext.define("Terrasoft.configuration.UsrCallsMessagePublisherModule", {
            // Basic class.
            extend: "Terrasoft.BaseMessagePublisherModule",
            // Short class name.
            alternateClassName: "Terrasoft.UsrCallsMessagePublisherModule",
            // Initialization of the page that will be rendered in this module.
            initSchemaName: function() {
                this.schemaName = "UsrCallsMessagePublisherPage";
            }
        });
        // Returns the class object defined in the module.
        return Terrasoft.UsrCallsMessagePublisherModule;
    });
```

4. Create the UsrCallsMessagePublisherPage page

For the created page set the `BaseMessagePublisherPage` schema of the `MessagePublisher` package as parent object. Set the "UsrCallsMessagePublisherPage" value as the title and name.

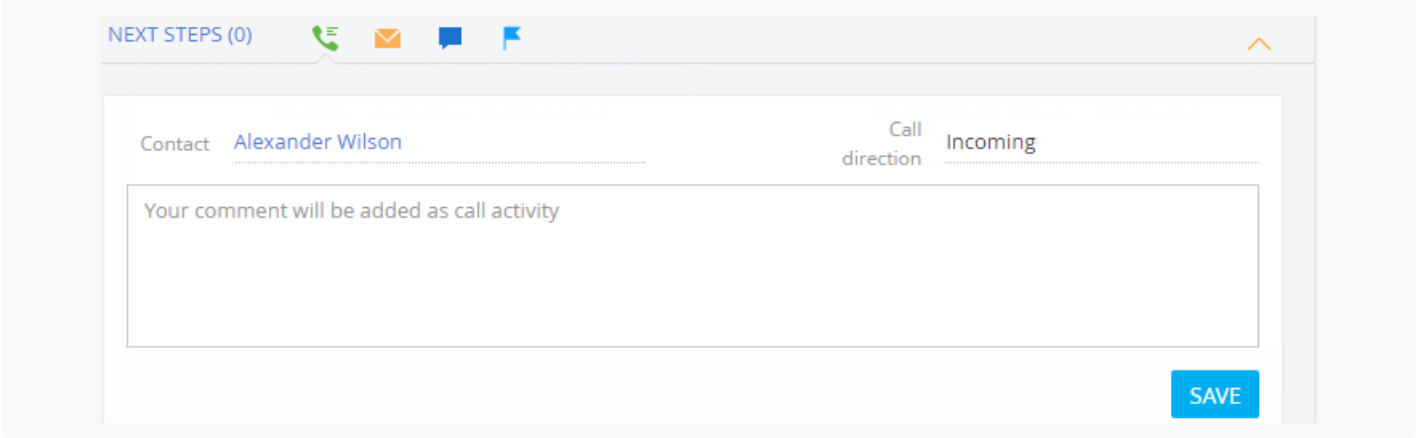
In the source code page, specify the schema name of the object that will run along with the page (in this case, `Activity`), implement the logic of message publication and override the `getServiceConfig` method, in which you must set the class name from the configuration.

```
//Sets the class that will work with this page.
getServiceConfig: function() {
   return {
      className: "Terrasoft.Configuration.CallsMessagePublisher"
  };
}
```

Implementation of message publication logic contains big number of methods, attributes and properties. The full source code of the `UsrCallsMessagePublisherPage` schema can be found in the sdkActionsDashboardNewChannel package. The source code shows the implementation of the `CallMessagePublisher` channel that is used for logging incoming and outgoing calls.
As a result you will get the new channel in the `SectionActionsDashboard` (Fig. 5).

Fig. 5. An example of a custom CallsMessagePublisher channel in the SectionActionsDashboard of the [*Contacts*] section.

© 2023 Creatio. All rights reserved.



Add multi-language email templates to a custom section

Advanced

You can set up custom logic for selecting languages of multi-language email templates. You can select email templates in the needed language using the action dashboard of a section record. The selection is based on special rules that can be specified for the section. If special rules are not defined, the selection is based on the contact, bound to the edited record (the [*Contact*] column). If a section object does not have a column for connecting with a contact, the `DefaultMessageLanguage` system setting value is used.

To add custom logic for selecting multi-language templates (localization):

1. Create a class or classes inherited from `BaseLanguageRule` and define the language selection rules (one class defines one rule).

2. Create a class inherited from `BaseLanguageIterator`. Define the `LanguageRules` property in the class constructor as a class instance array created on the previous step. The sequence corresponds to the rule priority.

3. Create a class inherited from `AppEventListenerBase` that will bind the class defining the language selection rules to the section.

4. Add the necessary multi-language templates to the [*Email templates*] lookup.

Case description

Add logic of selecting an email template language to a custom section based on the `UsrContact` column of the primary section object. Use English and Spanish languages.

Source code

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

Attention.

© 2023 Creatio. All rights reserved.

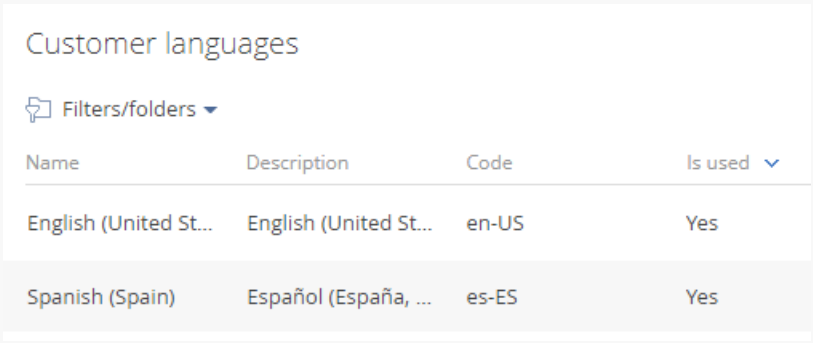
You can install the package for Creatio products, containing the `EmailTemplates` package. Make sure all the below described preliminary settings are performed after you install the package.

Preliminary settings

For correct case implementation:

1. Make sure that the [*Customer languages*] lookup contains English and Spanish languages (Fig.1).

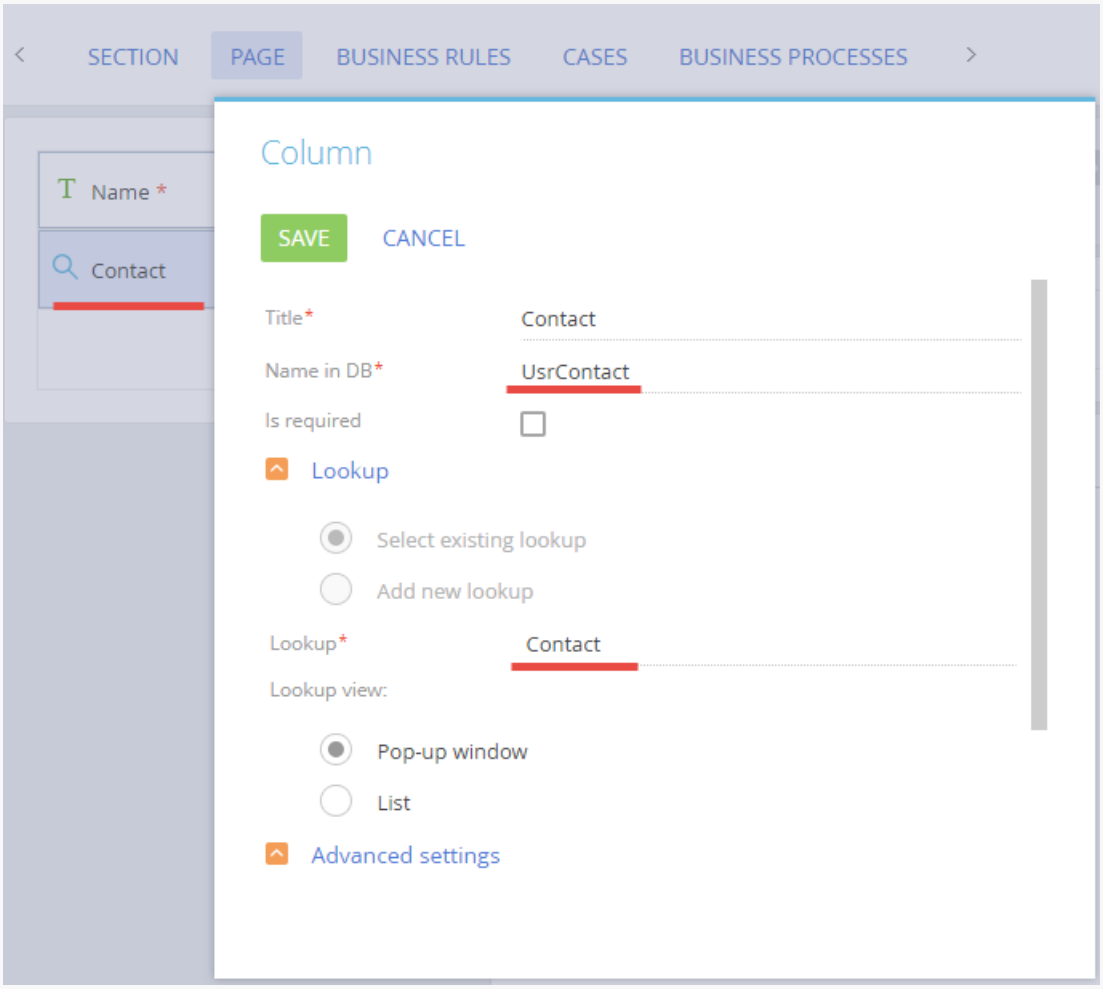
Fig. 1. [*Customer languages*] lookup



2. Use the section wizard to check that there is the `UsrContact` column bound to the [*Contact*] lookup on the edit page of the custom section record (Fig.2).

Fig. 2. The UsrContact column

© 2023 Creatio. All rights reserved.



Case implementation algorithm

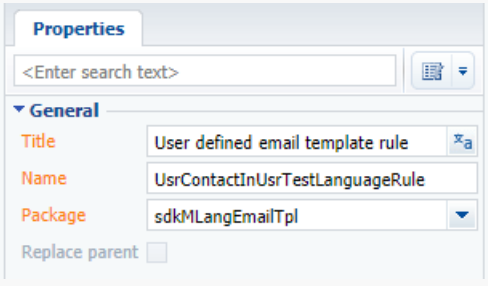
1. Adding a language selection rule

Create a [*Source code*] schema in the custom package (see “Create the [*Source code*] schema”).

For the created schema specify (Fig. 3):

- [Name] – "UsrContactInUsrTestLanguageRule"
- [Title] – "User defined email template rule”

Fig. 3. The [*Source code*] schema properties



© 2023 Creatio. All rights reserved.

Add the following source code to the schema:

```
namespace Terrasoft.Configuration
{
    using System;
    using Terrasoft.Core;
    using Terrasoft.Core.Entities;
    public class ContactInUsrTestLanguageRule : BaseLanguageRule
    {
        public ContactInUsrTestLanguageRule (UserConnection userConnection) : base(userConnectic
        {
        }
        // Defines the user preferred language identifier.
        // recId – current record identifier.
        public override Guid GetLanguageId(Guid recId)
        {
            // Creating the EntitySchemaQuery instance for the custom section primary object.
            var esq = new EntitySchemaQuery(UserConnection.EntitySchemaManager, "UsrMLangEmailTp
            // Defining the contact language column name.
            var languageColumnName = esq.AddColumn("UsrContact.Language.Id").Name;
            // Obtaining current record instance.
            Entity usrRecEntity = esq.GetEntity(UserConnection, recId);
            // Obtaining the value of user preferred language identifier.
            Guid languageId = usrRecEntity.GetTypedColumnValue<Guid>(languageColumnName);
            return languageId;
        }
    }
}
```

Publish the schema.

2. Defining the order sequence of language selection rules

Create a [*Source code*] schema in the custom package (see “Create the [*Source code*] schema”).

For the created schema specify (Fig. 3):

- [Name] – "UsrTestLanguageIterator”
- [Title] – "User defined language iterator”

Add the following source code to the schema:

```
namespace Terrasoft.Configuration
{
    using Terrasoft.Core;
    public class UsrTestLanguageIterator: BaseLanguageIterator
    {
```

© 2023 Creatio. All rights reserved.

```
        public UsrTestLanguageIterator(UserConnection userConnection): base(userConnection)
        {
            // Language selection rule array.
            LanguageRules = new ILanguageRule[] {
                // Custom rule.
                new ContactInUsrTestLanguageRule (UserConnection),
                // Default rule.
                new DefaultLanguageRule(UserConnection),
            };
        }
    }
}
```

`DefaultLanguageRule.` is the second array element The rules uses the `DefaultLanguage` system setting for obtaining the language and is used by default if the language was not detected by higher priority rules.

Publish the schema.

3. Binding language selection iterator to the section

Create a [*Source code*] schema in the custom package (see “Create the [*Source code*] schema”).

For the created schema specify (Fig. 3):

- [Name] – "UsrTestMLangBinder”
- [Title] – "UsrTestMLangBinder"

Add the following source code to the schema:

```
namespace Terrasoft.Configuration
{
    using Terrasoft.Core.Factories;
    using Terrasoft.Web.Common;
    public class UsrTestMLangBinder: AppEventListenerBase
    {
        public override void OnAppStart(AppEventContext context)
        {
            // Calling the basic logics.
            base.OnAppStart(context);
            // Binding iterator to a custom section.
            // UsrMLangEmailTpl – name of the section primary object.
            ClassFactory.Bind<ILanguageIterator, UsrTestLanguageIterator>("UsrMLangEmailTpl");
        }
    }
}
```

Publish the schema.

© 2023 Creatio. All rights reserved.

4. Adding the necessary multi-language templates

Add a new record (Fig.4) to the [*Email Templates*] lookup and define the email templates in the necessary languages (Fig.5).

Fig. 4. A new record in the [*Email templates*] lookup

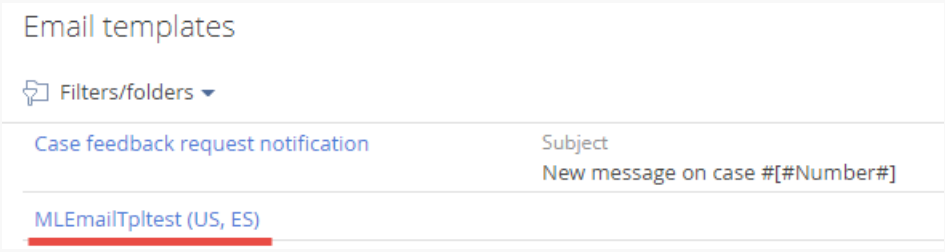
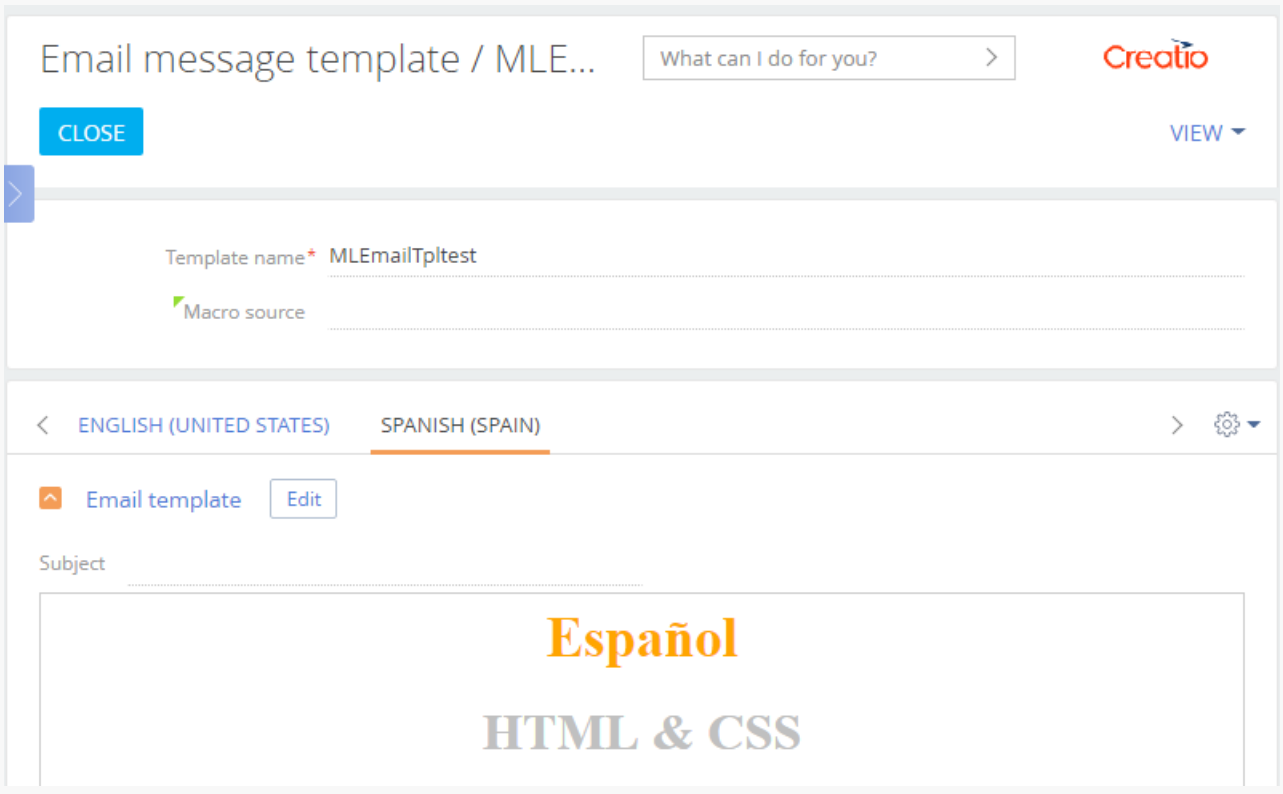


Fig. 5. Adding templates in the necessary languages



As a result of case implementation, in the action dashboard panel (Fig. 6. 1) of the custom section record edit page (Fig.6) the email templates (Fig. 6. 2) will be selected automatically in the language (Fig. 6. 3) specified as the contact's preferred language (Fig.7).

Fig. 6. Case result

© 2023 Creatio. All rights reserved.

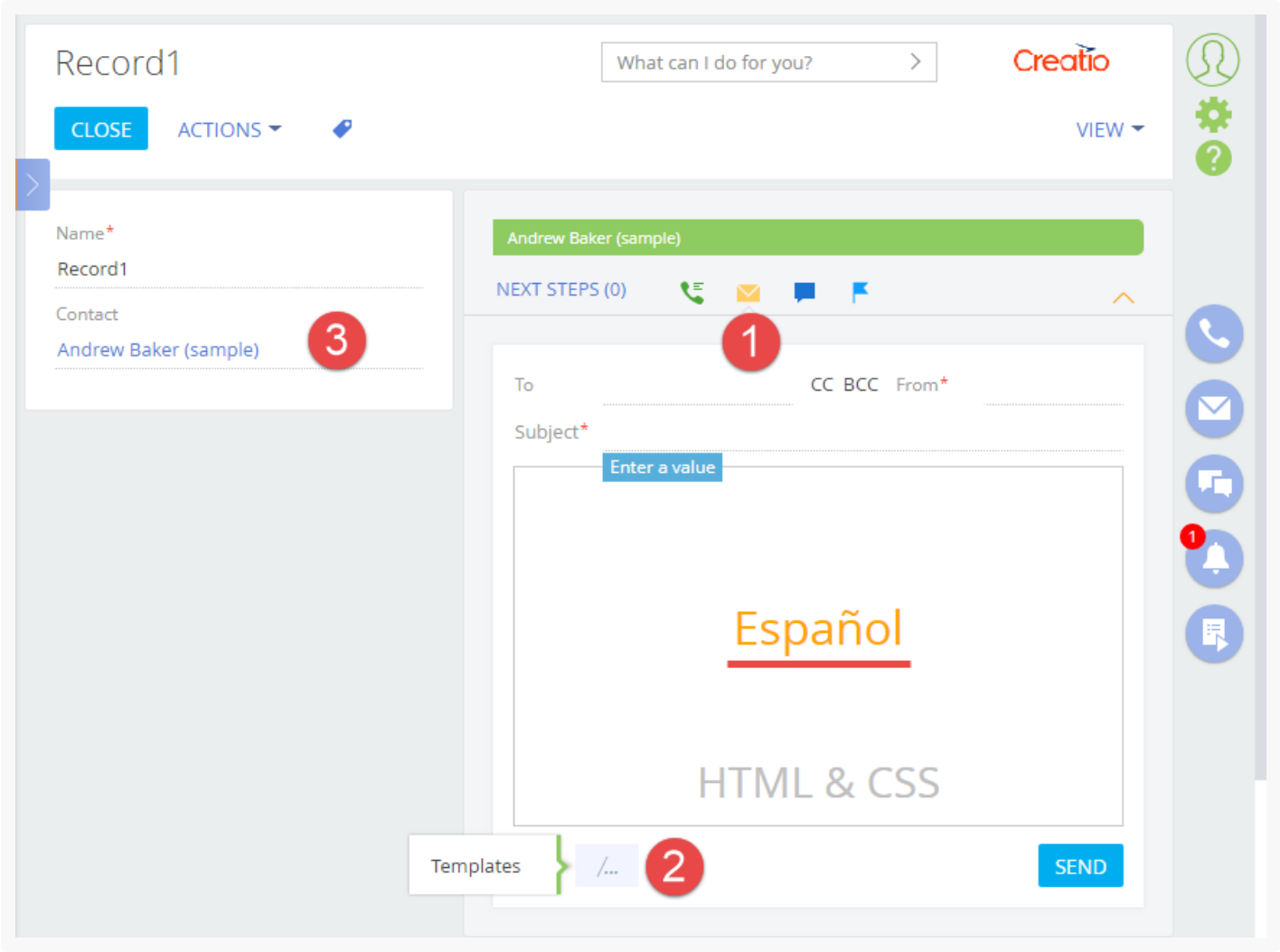
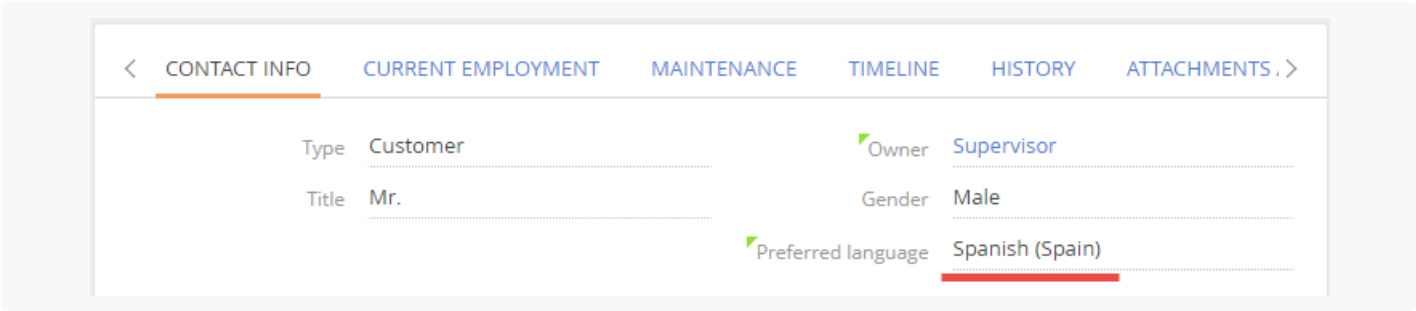


Fig. 7. Contact preferred language



Create custom verification action page

Medium

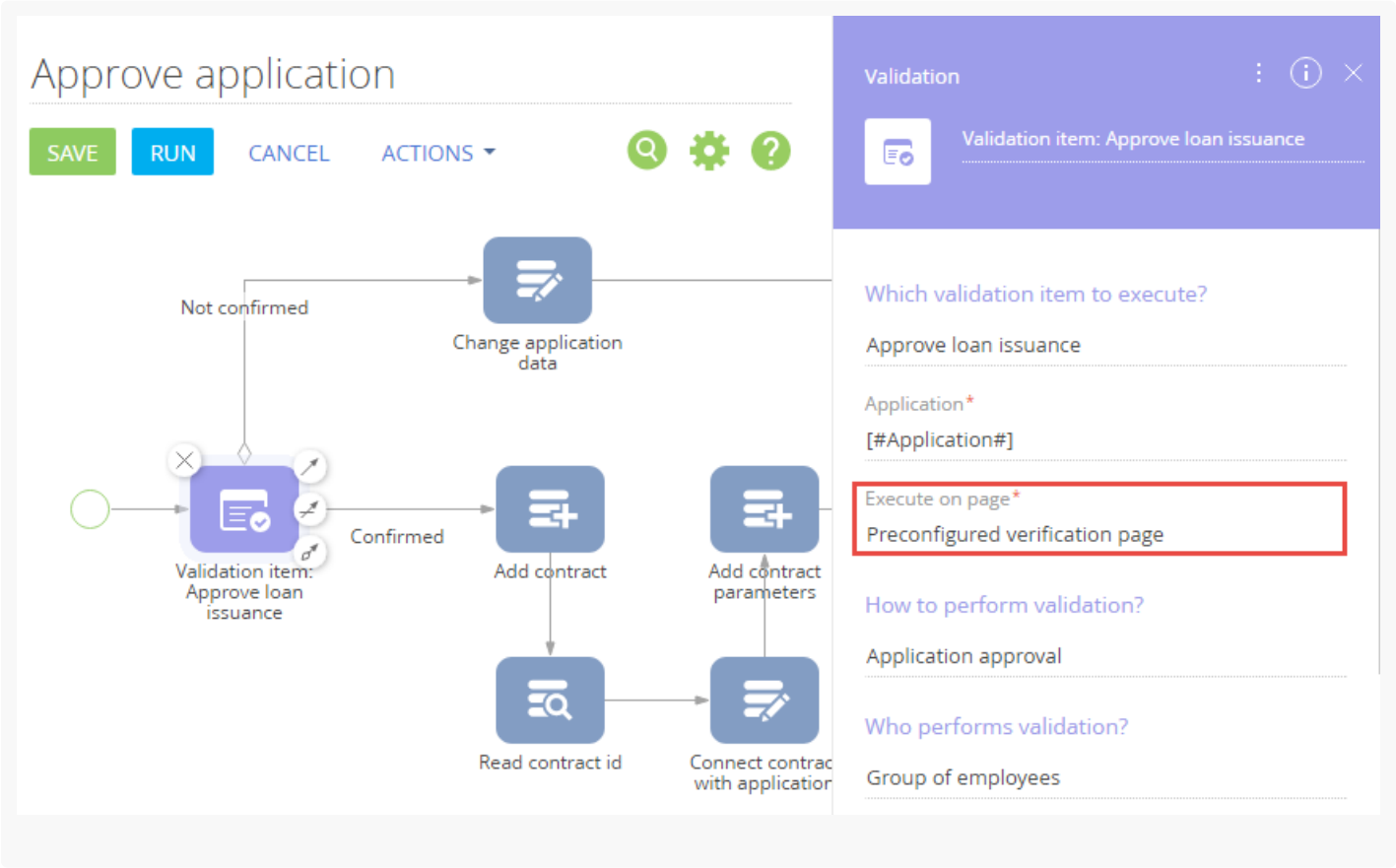
Verification action is a confirmation that the data in the application form corresponds to the requirements of the application. The verification action is performed to check the data provided by the clients when they fill out their application forms.

When creating a custom verification action page, for example, in the `Approve application` business process, you

© 2023 Creatio. All rights reserved.

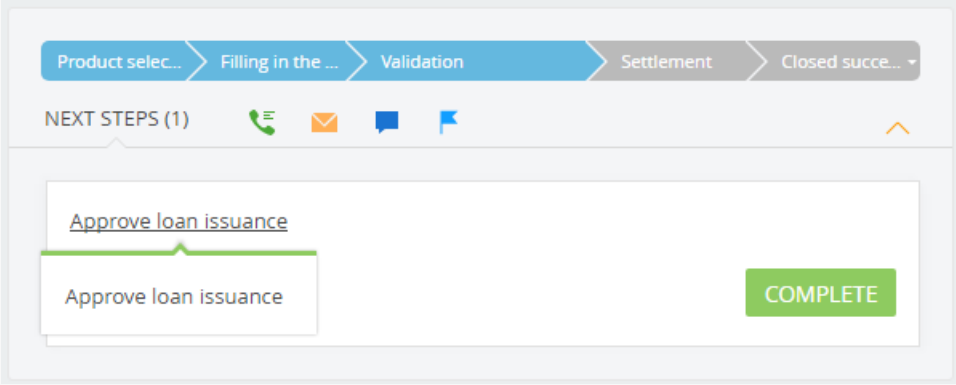
can select the [*Preconfigured verification page*].

Fig. 1 Selecting the verification page



The page is displayed after clicking the [*Complete*] button of the [*Approve loan issuance*] activity that is created when the application is moved to the [*Validation*] stage (fig. 2).

Fig. 2 Activity on the [*Validation*] stage



Preconfigured verification page contains (Fig. 3):

1. Buttons for selecting the result of the verification action.
2. The [Comment] field – comments to the verification action.

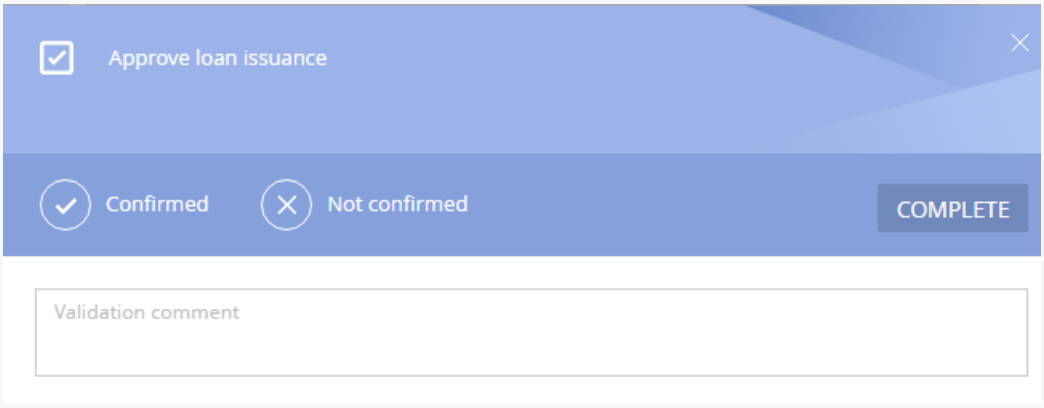
© 2023 Creatio. All rights reserved.

3. The [Conversation script] detail – contains a hints for the verifiers who call customers in the process of verification. Read only.
4. The [Attachments] detail – contains files and links attached to the validation stage. Read only.
5. The [Checklist] detail – contains control questions and answers to them.

Attention.

If a detail has no attached data, it will not be displayed to save page space.

Fig. 3 Verification page



You can create custom verification pages, inheriting them from the preconfigured page. To create a custom page:

1. Create a custom schema of the verification action page.
2. Use the schema created in the business process.

Case description

Create a verification action page where the [*Comment*] field is hidden.

Case implementation algorithm

1. Create a schema of the verification action page

To do this, go to the [*Configuration*] section and select a custom package. Then execute the [*Add*] > [*Schema of the Edit Page View Model*] command. The process of creating custom schema of the view model is covered in the "Create a client schema" article.

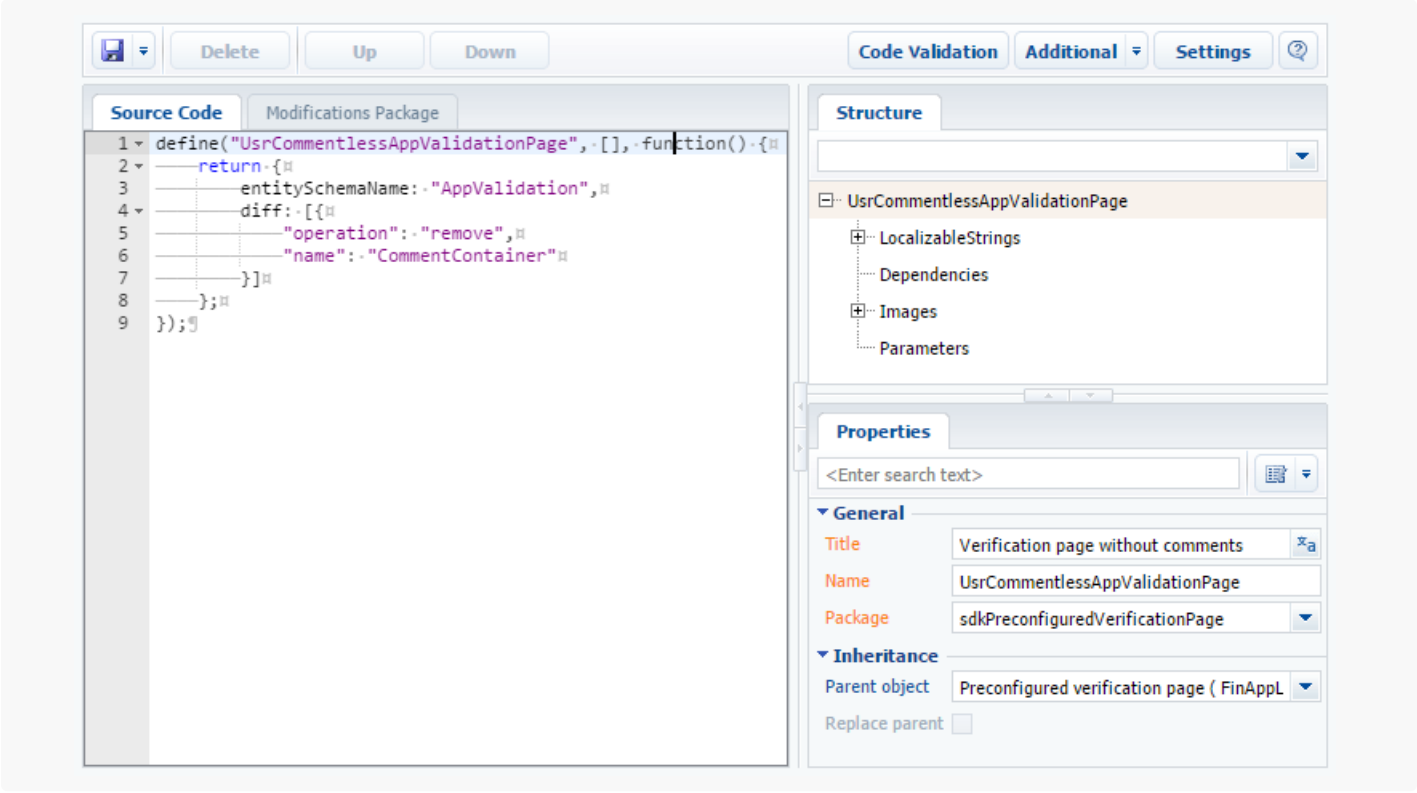
You need to assign the following properties for the created schema (Fig. 4):

- [Title] – Verification page without comments.
- [Name] – UsrCommentlessAppValidationPage.

© 2023 Creatio. All rights reserved.

- [Package] – Custom (or another custom package).
- [Parent object] – Preconfigured verification page of the `FinAppLending` package.

Fig. 4 Schema properties of the view model page



Add the following source code to the [*Source code*] tab:

```
define("UsrCommentlessAppValidationPage", [], function() {
    return {
        entitySchemaName: "AppValidation",
        diff: [{
            "operation": "remove",
            "name": "CommentContainer"
        }]
    };
});
```

The [*Comment*] field is removed from the parent element in the diff array.

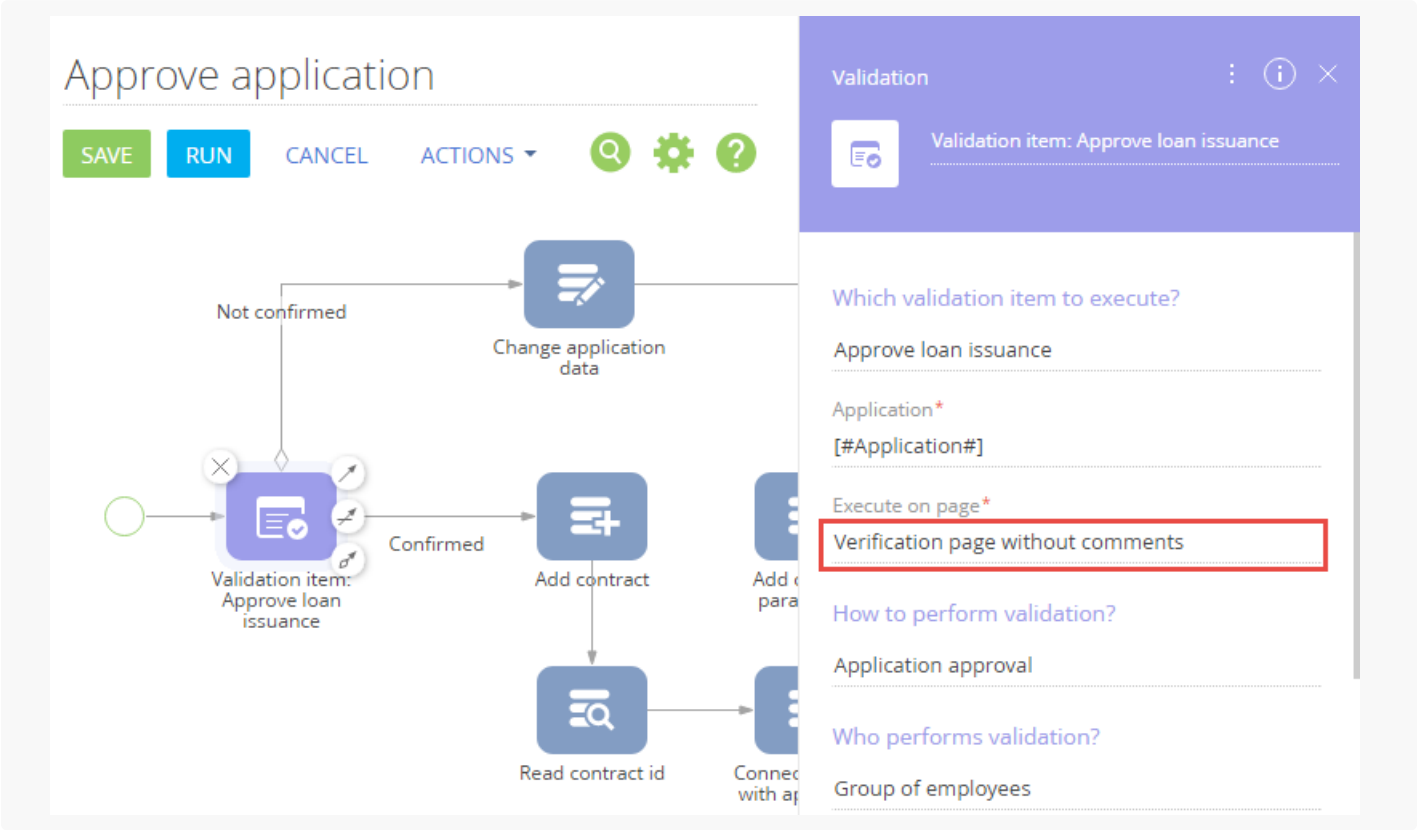
Save the schema to apply the changes.

2. Use the schema created in the business process.

To use the created schema, specify it in the [*Execute on page*] field of the [*Validation item*] item of the business process. This schema can be used in both new and existing business processes, such as `Approve application` (Fig. 5).

© 2023 Creatio. All rights reserved.

Fig. 5 Specifying the custom verification page



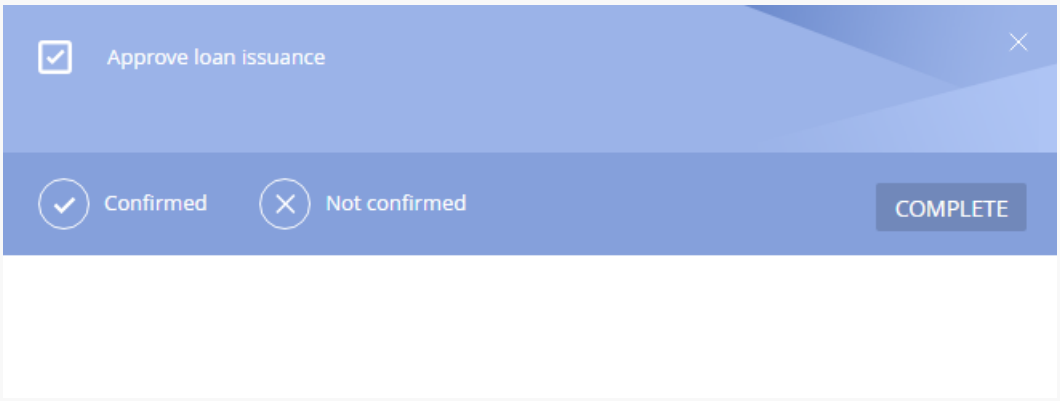
Save the business process to apply the changes.

Attention.

Restart the application in IIS for the changes to take effect.

After the changes are applied, the previous verification page (Fig. 3) will be replaced with a custom page that does not contain the [*Comment*] field (Fig. 6).

Fig. 6 Verification page without the [*Comment*] field.



© 2023 Creatio. All rights reserved.

© 2023 Creatio. All rights reserved.