Creatio Academy

Interface elements

Communication panel

Version 8.0

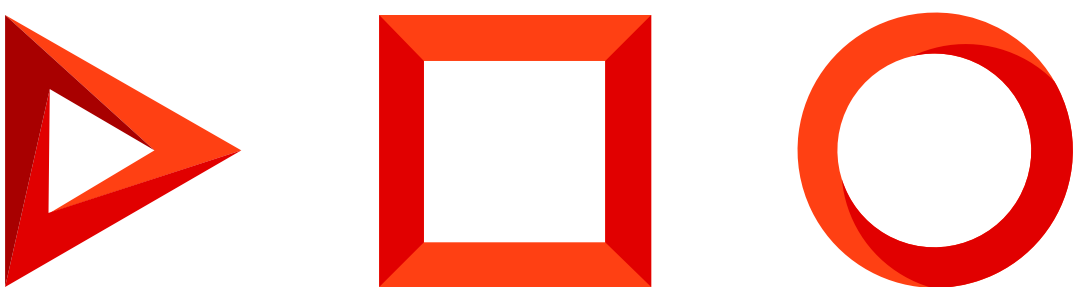


Table of Contents

Communication panel

Beginner

Communication panel is a tool that lets you talk to customers and coworkers, as well as receive Creatio notifications. Creatio displays the panel on the right.

Communication panel structure

The communication panel **tabs** are as follows:

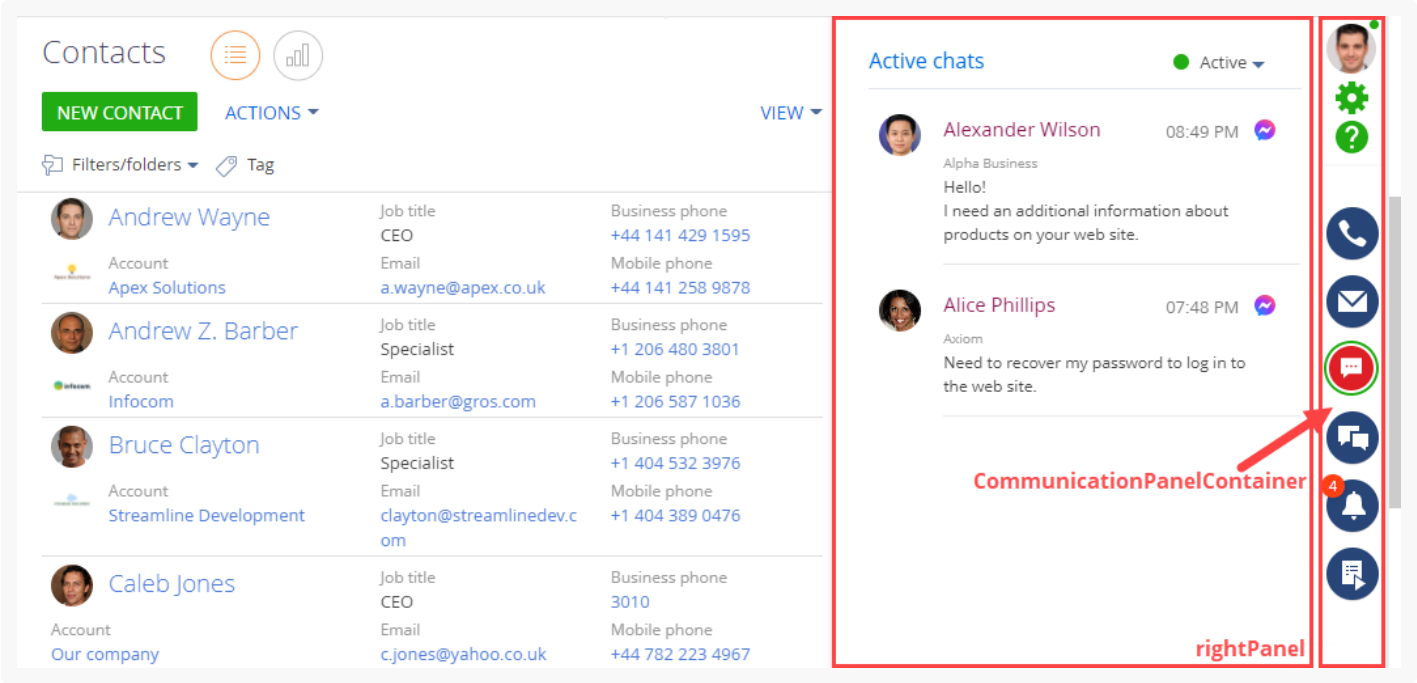
- [*Operation single window*]. Enables bank tellers to consult with customers. Available in Financial Services Creatio, customer journey edition. Tellers can search for bank customers by various criteria, initiate consultations, and postpone consultations.
- [*CTI panel*]. Lets you accept incoming and initiate outgoing calls directly in Creatio. One of Creatio phone integration tools. Learn more about the phone integration in the user documentation: Phone integration connectors, Manage calls.
- [*Email*]. Lets you send and receive emails. You can link emails to other Creatio objects. Learn more about managing emails in the user documentation: Work with emails.
- [*Feed*]. Displays messages of the [*Feed*] section. Lets you view messages from the channels you follow, as well as add new messages and comments. The functionality of the tab is identical to the functionality of the [*Feed*] section.
- [*Notification center*]. Displays notifications about various Creatio events. Learn more about managing notifications in the user documentation: Check notifications and process tasks.
- [*Business process tasks*]. Displays incomplete steps of running business processes. Learn more about managing running business processes in the user documentation: Check notifications and process tasks.

Communication panel containers

Creatio stores the UI elements related to the communication panel in the corresponding containers. Configure the communication panel containers in the `CommunicationPanel` base schema of the `Uiv2` package. Configure the [*CTI panel*] tab in the `CtiPanel` child schema of the `CTIBase` package.

> **Note.** Creatio uses HTML container meta names. It generates actual IDs that match the HTML elements of the communication panel based on the meta names.

View the main **communication panel containers** in the figure below.



- The button container of the communication panel (`CommunicationPanelContainer`) includes buttons that open communication panel tabs.
- The tab container of the communication panel (`rightPanel`) includes the content of the communication panel's current tab.

Create a custom reminder or notification

Creatio lets you display notifications in the following **ways**:

- immediately
- at the specified time

To **create a custom reminder or notification**:

1. Implement the **reminder text and dialog**.

   Create a [*Source code*] type schema. To do this, follow the guide in a separate article: Source code (C#).

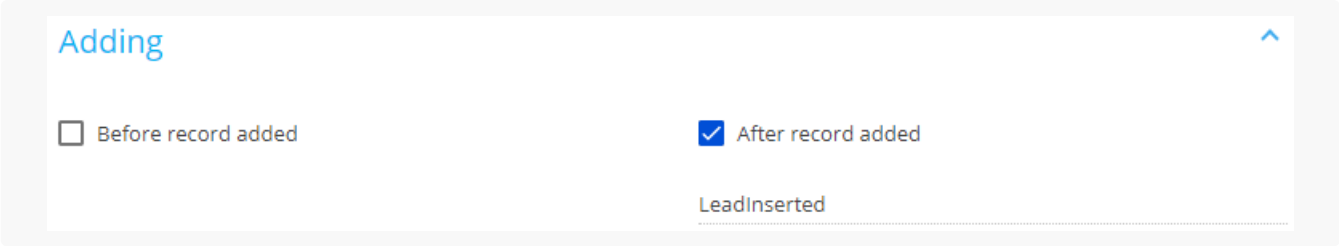
   a. Create a class that generates the reminder text and dialog.

   b. Implement the `IRemindingTextFormer` interface of the `Base` package's `IRemindingTextFormer` schema in the class.

2. Implement the **mechanism that sends the reminder**.

   a. Create a replacing object on which to send the reminder. To do this, follow the guide in a separate article: Object.

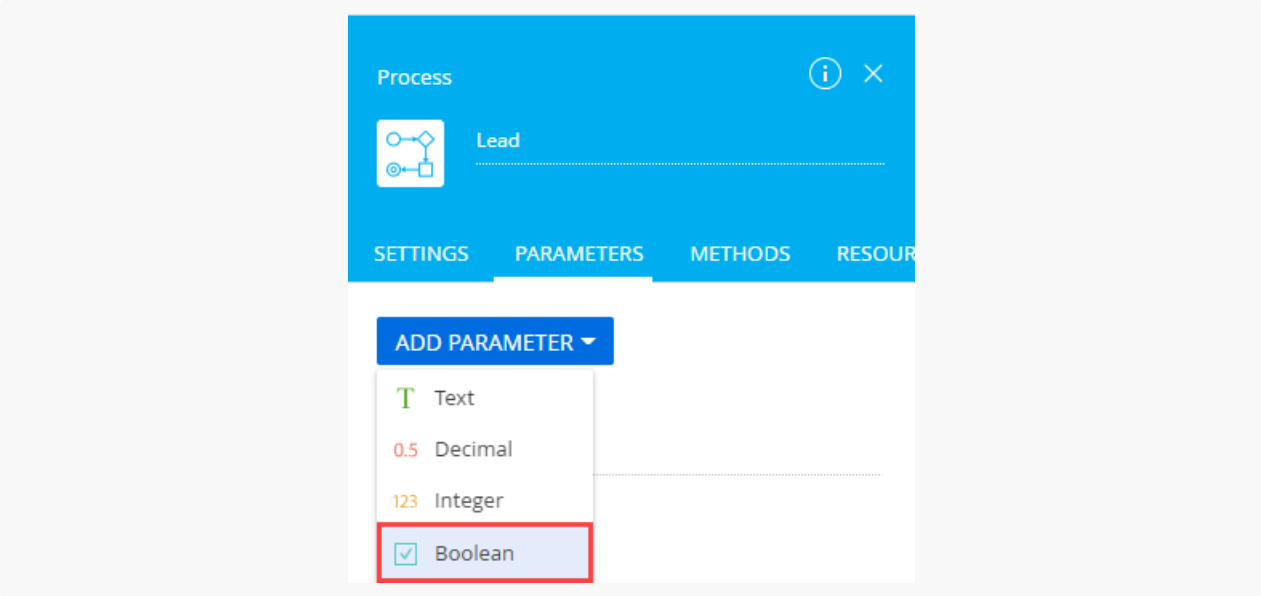
   b. Create an **object event**. To do this, go to the [*Events*] node and select the [*After record added*] checkbox in the [*Adding*] block.



c. Set up the **parameter the business process uses to generate the reminder** in the object schema.

a. Add a **parameter**.

a. Open the [*Parameters*] tab in the setup area and click [*Add parameter*] → [*Boolean*].



b. Fill out the **parameter properties** in the Object Designer.

b. Overload the **methods**.

- Overload the **method called after the object is saved**. To do this, open the [*Methods*] tab of the setup area and add the source code of the overloaded method.

  Creatio can display the notification in the following **ways**:

  - The reminder tab of the notification center. To set this up, assign `RemindingConsts.NotificationTypeRemindingId` to the `remindingConfig.NotificationTypeId` constant.
  - The service message tab of the notification center. To set this up, assign `RemindingConsts.NotificationTypeNotificationId` to the `remindingConfig.NotificationTypeId` constant.

- Overload the **method called before the object is saved**. To do this, open the [*Methods*] tab of the setup area and add the source code of the overloaded method.

3. Implement the **mechanism that displays the reminder**.

   Create a replacing view model. To do this, follow the guide in a separate article: Client module. Select

`ReminderNotificationsSchema` as the parent object.

Create a case based on a message from the internal feed of another case

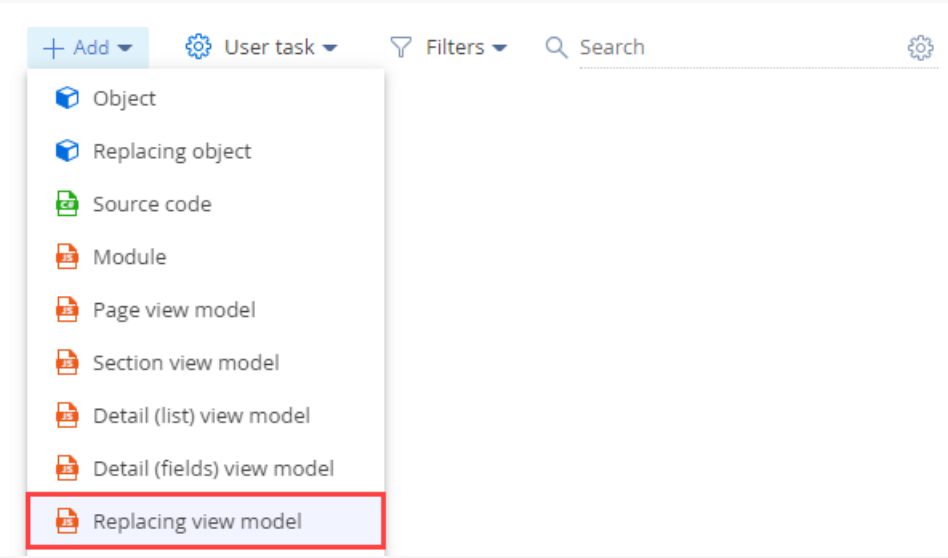
Advanced

The example is relevant to Service Creatio products.

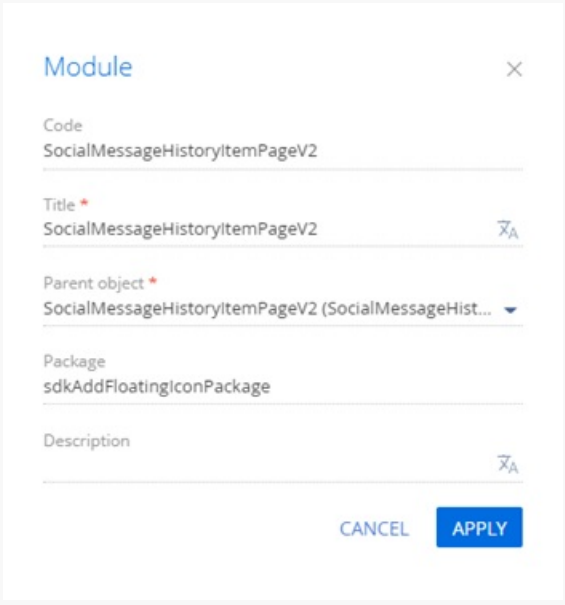
Example. Add a pop-up button to the [*Processing*] tab of the case page. Display the button when you select text in emails and self-service portal messages sent from the internal case feed. Create a new case on button click. Populate the [*Subject*] and [*Description*] fields automatically. Set the field values to selected text.

Create a replacing view model schema of the case page

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



3. Fill out the **schema properties**.
   - Set [*Code*] to "`SocialMessageHistoryItemPageV2`."
   - Set [*Title*] to "`SocialMessageHistoryItemPageV2`."
   - Select "`SocialMessageHistoryItemPageV2`" in the [*Parent object*] property.



4. Implement the **pop-up button logic**.

   - Implement the following **methods** in the `methods` property:

     - `onSelectedTextChanged()`. Passes the selected text to the `HighlightedHistoryMessage` attribute. Triggered on text selection.
     - `onSelectedTextButtonClick()`. Creates a case, retrieves the case subject from the `HighlightedHistoryMessage` attribute. The `BaseMessageHistory` parent schema defines the case creation logic. Triggered on pop-up button click.
     - `getMessageFromHistory()`. An overloaded parent schema method that retrieves the subject of the selected message.

   - Add a configuration object that contains the settings of the the `Message` package's `SelectionHandlerMultiLineLabel` element to the `diff` array of modifications. The element implements the mechanism that creates a new case based on selected text.

   View the source code of the replacing view model schema of the case page below.

SocialMessageHistoryItemPageV2

```
define("SocialMessageHistoryItemPageV2", ["SocialMessageConstants", "css!SocialMessageHistory
    return {
        /* The name of the record page object's schema. */
        entitySchemaName: "BaseMessageHistory",
        /* The details of the record page's view model. */
        details: /**SCHEMA_DETAILS*/{}/**SCHEMA_DETAILS*/,
        /* The methods of the record page's view model. */
        methods: {
            /* Overload the base method. Retrieve the subject of the selected message. */
            getMessageFromHistory: function() {
                var message = this.get("HighlightedHistoryMessage");
```

```
                if (this.isHistoryMessageEmpty(message)) {
                    message = this.get("[Activity:Id:RecordId].Body");
                }
                return message;
            },
            /* The handler of the text selection event. */
            onSelectedTextChanged: function(text) {
                this.set("HighlightedHistoryMessage", text);
            },
            /* The handler of the pop-up button click. */
            onSelectedTextButtonClick: function() {
                /* Prepare case data from history. */
                this.prepareCaseDataFromHistory();
            }
        },
        /* Display the button on the record page. */
        diff: /**SCHEMA_DIFF*/[
            /* The metadata to add the pop-up button to the page. */
            {
                /* Execute the operation that modifies the existing element. */
                "operation": "merge",
                /* The meta name of the component to change. */
                "name": "MessageText",
                /* The properties to pass to the element's constructor. */
                "values": {
                    /* The properties of the view generator. */
                    "generator": function() {
                        return {
                            /* The value of the id HTML tag. */
                            "id": "MessageText",
                            /* The value of the marker. */
                            "markerValue": "MessageText",
                            /* The name of the component class. */
                            "className": "Terrasoft.SelectionHandlerMultilineLabel",
                            /* Set up the CSS styles. */
                            "classes": {
                                "multilineLabelClass": ["messageText"]
                            },
                            /* The caption. */
                            "caption": {"bindTo": "Message"},
                            "showLinks": true,
                            /* Bind the change event of the selected text to the handler meth
                            "selectedTextChanged": {"bindTo": "onSelectedTextChanged"},
                            /* Bind the click event of the selected text's pop-up button to t
                            "selectedTextHandlerButtonClick": {"bindTo": "onSelectedTextButto
                            /* Flag the pop-up button as visible. */
                            "showFloatButton": true
                        };
                    }
```

```
                }
            }
        ]/**SCHEMA_DIFF*/
    };
});
```

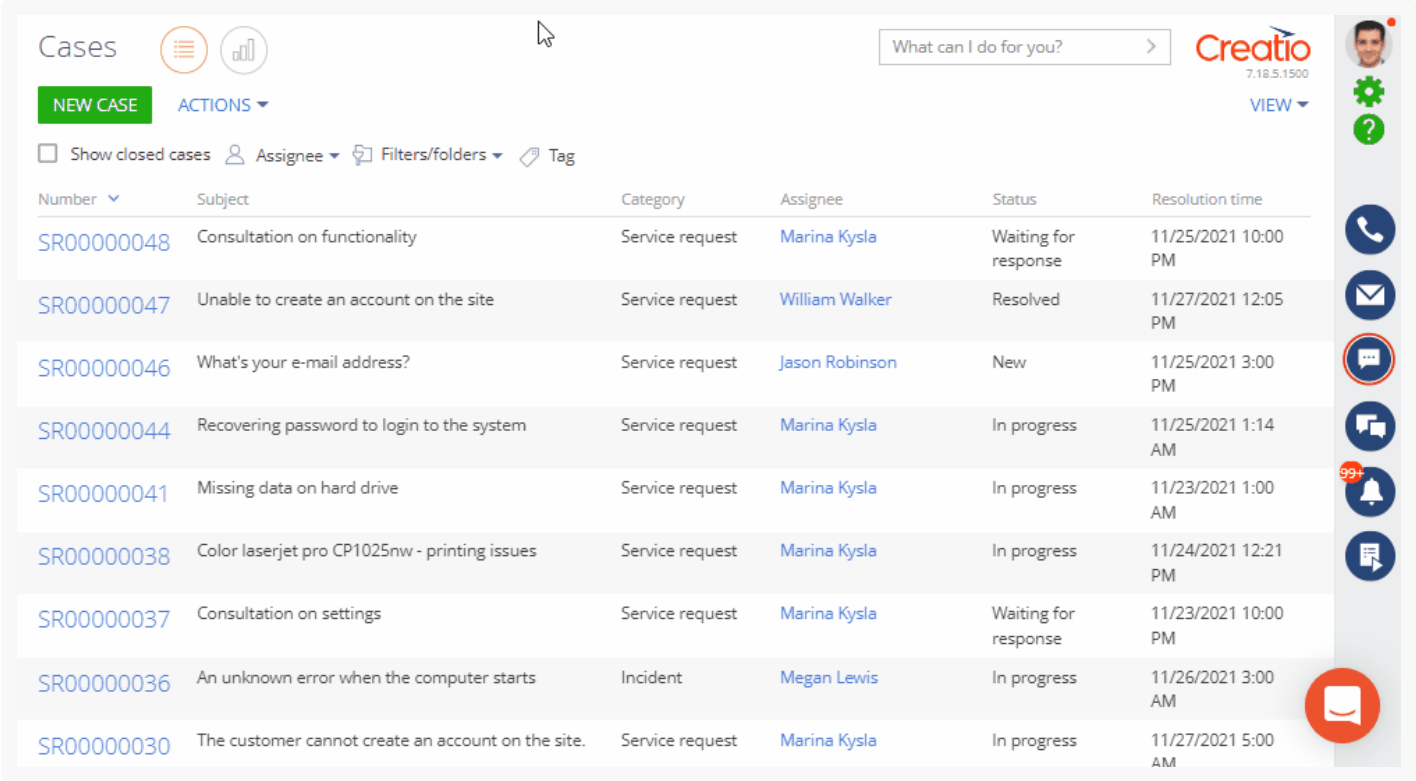
5. Click [*Save*] on the Designer's toolbar.

Outcome of the example

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

1. Refresh the [*Cases*] section page.
2. Open a case page.

As a result, Creatio will add a pop-up button to the [*Processing*] tab of the case page. The button appears when you select text from emails and self-service portal messages sent from the internal case feed. Click the button to create a new case. Creatio populates the [*Subject*] and [*Description*] fields automatically. The field values are set to selected text.



Hide feed area in the agent desktop

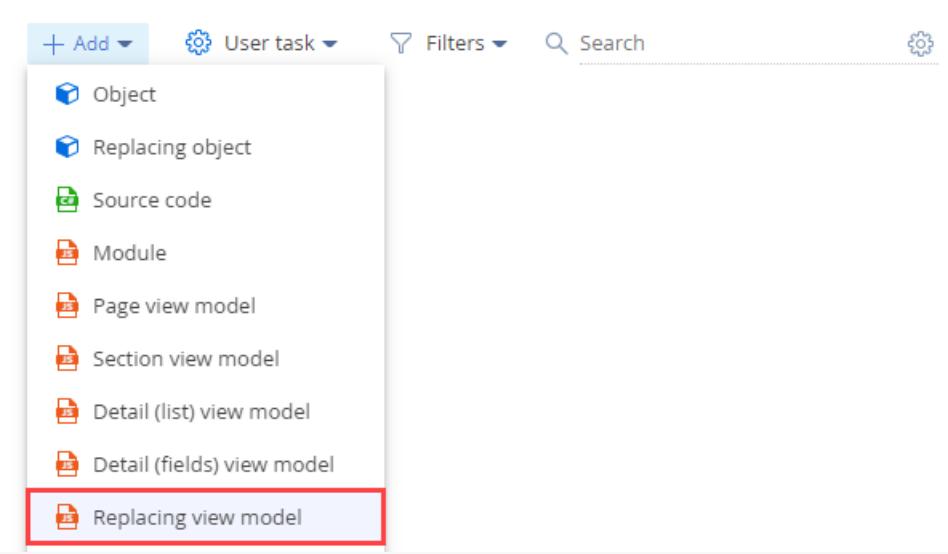
Medium

The example is relevant to the [*Contact center*] workplace.

Example. Hide the feed area from the [*Agent desktop*] section.

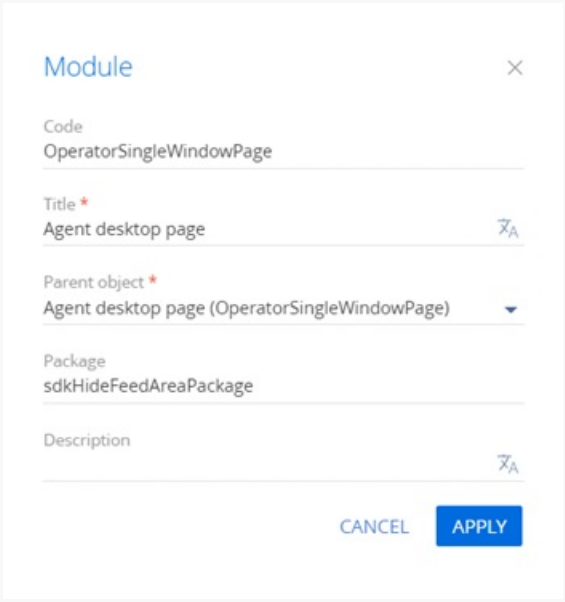
Create a schema of the replacing section view model

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



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

   - Set [*Code*] to "`OperatorSingleWindowPage`."
   - Set [*Title*] to "`Agent desktop page`."
   - Select "`OperatorSingleWindowPage`" in the [*Parent object*] property.



4. Implement the **mechanism that hides the feed area**.

   - Implement the `loadContent()` method in the `methods` property. This is an overloaded base method that excludes the `ESNFeedModule` feed module from the index of loaded modules.
   - Add a configuration object that deletes the element from the page to the `diff` array of modifications.

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

OperatorSingleWindowPage

```
define("OperatorSingleWindowPage", [], function() {
    return {
        /* The methods of the section view model. */
        methods: {
            /* Replace the base method to exclude the ESNFeedModule feed module from the inde
            loadContent: function() {
                /* Since the centerContainer container is deleted, you do not need to load th
                this.loadModule("ESNFeedModule", "centerContainer"); */
                /* Load the modules. */
                this.loadModule("SectionDashboardsModule", "rightContainer");
                this.loadModule("OperatorQueuesModule", "leftContainer");
            }
        },
        /* Display the container in the section. */
        diff: /**SCHEMA_DIFF*/[
            /* The metadata to remove the container from the section. */
            {
                /* Execute the operation that deletes the existing element. */
                "operation": "remove",
                /* The meta name of the component to delete. */
                "name": "centerContainer"
```

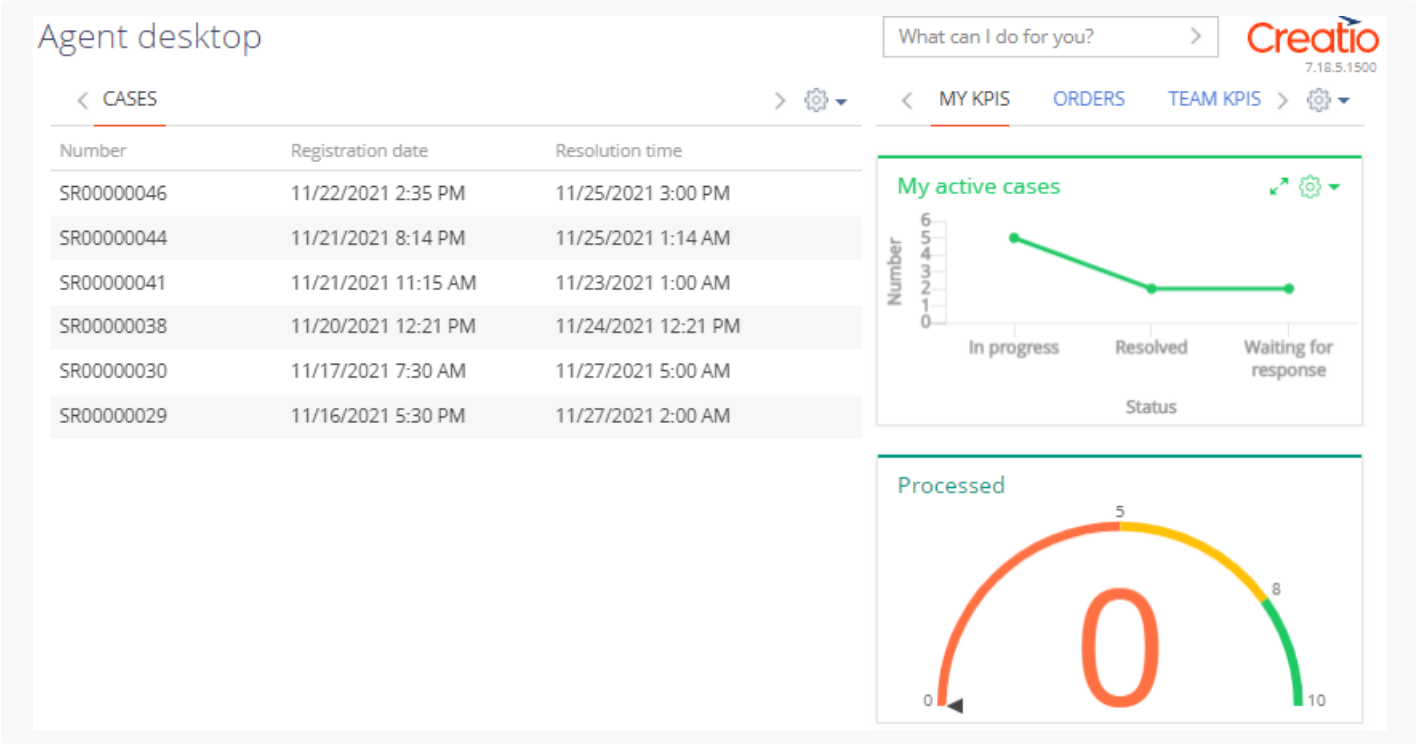
```
            }
        ]/**SCHEMA_DIFF*/
    };
});
```

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

Outcome of the example

To **view the outcome of the example**, refresh the [*Agent desktop*] section page.

As a result, Creatio will hide the feed area from the [*Agent desktop*] section.



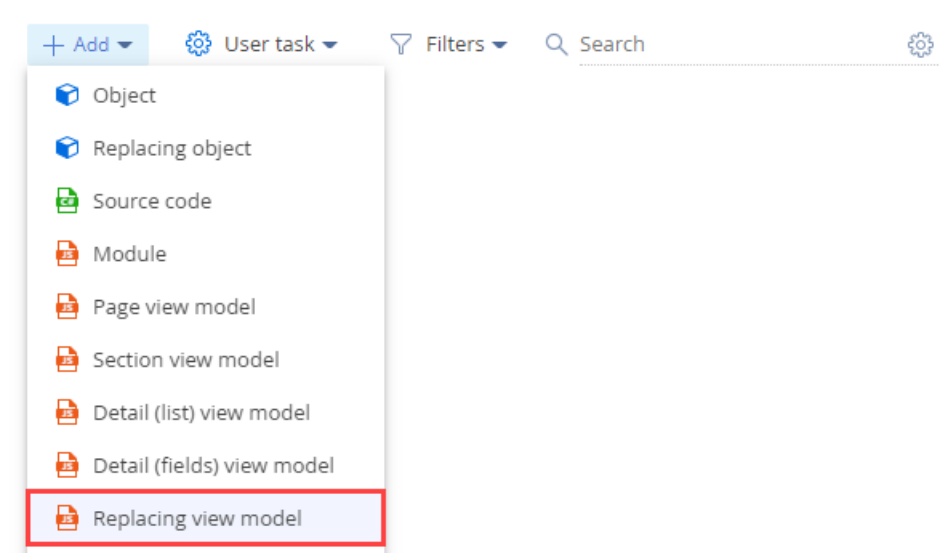
Display the time zone of a contact on a communication panel tab

Advanced

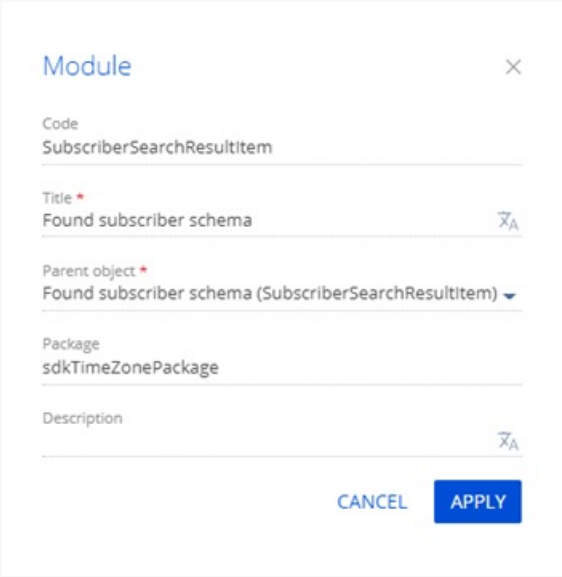
Example. Display the time zone of a contact on the [*CTI panel*] tab of the communication panel when you search for a contact. Use the current time of a contact.

1. Create a replacing view model schema of the page for found subscribers

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



3. Fill out the **schema properties**.
   - Set [*Code*] to "`SubscriberSearchResultItem`."
   - Set [*Title*] to "`Found subscriber schema`."
   - Select "`SubscriberSearchResultItem`" in the [*Parent object*] property.



4. Add the `TimezoneGenerator` and `TimezoneMixin` modules as dependencies to the declaration of the view model class. The `TimezoneGenerator` module creates an element that displays the time zone of the contact. The `TimezoneMixin` module searches for the time zone of the contact.
5. Implement a **mechanism that displays the time zone**.
   - Add the `IsShowTimeZone` attribute to the `attributes` property. The attribute determines the visibility of the

time zone element.

- Add the `TimezoneMixin` mixin to the `mixins` property. To run the search for the contact time zone, pass the contact ID to the `init` method of the `TimezoneMixin` mixin. As a result, Creatio will set the following **attributes**:
  - `TimeZoneCaption`. The name of the contact's time zone and current time.
  - `TimeZoneCity`. The name of the city in the time zone.
- Implement the following **methods** in the `methods` property:
  - `constructor()`. The class constructor.
  - `isContactType()`. Returns the flag that indicates that the subscriber is a contact.
- Add a configuration object with the display settings of the contact's time zone to the `diff` array of modifications.
  - `index` property. Configures the element layout.

    `SubscriberSearchResultItemContainer` container **elements**:
    - `0` index: subscriber photo.
    - `1` index: subscriber info.
    - `2` index: subscriber phones.

    Assign `2` to the `index` property of the array of modifications to display the contact time zone between the subscriber info and the index of phone numbers.
  - `wrapClass` property. Controls the styles. An element generator. Use the `subscriber-data` CSS class to define styles for text elements in the schema.

View the source code of the replacing view model schema of the page for found subscribers below.

SubscriberSearchResultItem

```
define("SubscriberSearchResultItem", ["TimezoneGenerator", "TimezoneMixin"], function() {
    return {
        /* The attributes of the page view model. */
        attributes: {
            /* The name of the attribute that determines the visibility of the time zone elem
            "IsShowTimeZone": {
                /* The data type of the view model column. */
                "dataValueType": Terrasoft.DataValueType.BOOLEAN,
                /* The attribute type. */
                "type": Terrasoft.ViewModelColumnType.VIRTUAL_COLUMN,
                /* The default value. */
                "value": true
            }
        },
        /* The mixins of the page view model. */
```

```
        mixins: {
            /* Enable the mixin. */
            TimezoneMixin: "Terrasoft.TimezoneMixin"
        },
        /* The methods of the page view model. */
        methods: {
            /* The class constructor. */
            constructor: function() {
                /* Call the base constructor. */
                this.callParent(arguments);
                /* The flag that indicates that the subscriber is a contact. */
                var isContact = this.isContactType();
                /* If the subscriber is a contact, display the element. */
                this.set("IsShowTimeZone", isContact);
                /* If the subscriber is a contact. */
                if (isContact) {
                    /* The contact ID. */
                    var contactId = this.get("Id");
                    /* Search for the contact time zone. */
                    this.mixins.TimezoneMixin.init.call(this, contactId);
                }
            },
            /* Return the flag that indicates that the subscriber is a contact. */
            isContactType: function() {
                /* The subscriber type. */
                var type = this.get("Type");
                /* Return the comparison result. */
                return type === "Contact";
            }
        },
        /* The array of modifications of the page view model. */
        diff: [
            {
                /* Add the element to the page. */
                "operation": "insert",
                /* The meta name of the parent container to add the element. */
                "parentName": "SubscriberSearchResultItemContainer",
                /* Add the element to the parent element's collection of elements. */
                "propertyName": "items",
                /* The meta name of the element to add. */
                "name": "TimezoneContact",
                /* The properties to pass to the element constructor. */
                "values": {
                    /* The element type. */
                    "itemType": Terrasoft.ViewItemType.CONTAINER,
                    /* Call the method of the generator to create the view configuration. */
                    "generator": "TimezoneGenerator.generateTimeZone",
                    /* Bind the container visibility to the attribute. */
                    "visible": {"bindTo": "IsShowTimeZone"},
```

```
                    /* The name of the CSS class. */
                    "wrapClass": ["subscriber-data", "timezone"],
                    /* Bind the caption to the attribute. */
                    "timeZoneCaption": {"bindTo": "TimeZoneCaption"},
                    /* Bind the city to the attribute. */
                    "timeZoneCity": {"bindTo": "TimeZoneCity"}
                },
                /* The element layout in the parent container. */
                "index": 2
            }
        ]
    };
});
```

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

As a result, Creatio will display the current time and city of the contact.

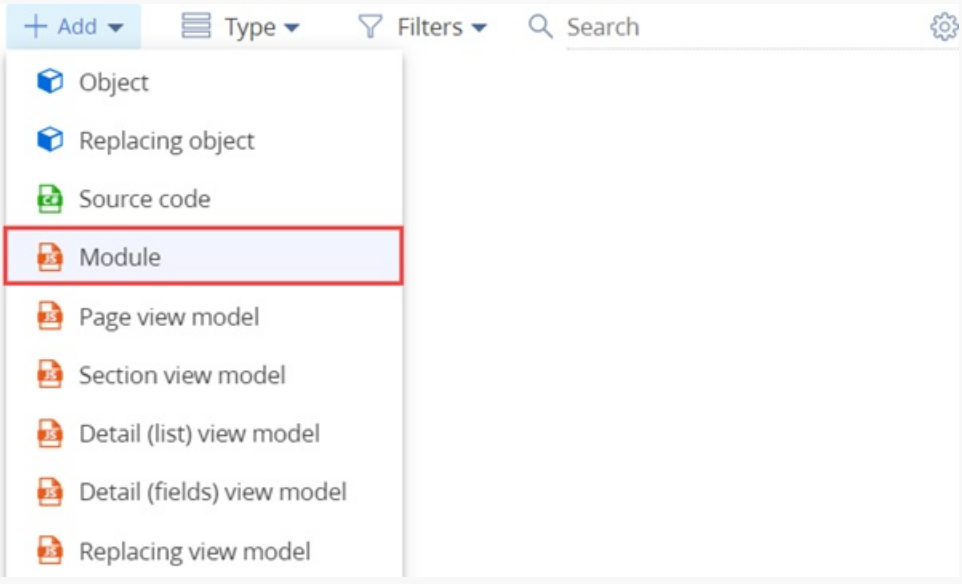
2. Add display styles of the time zone

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

1. Create a module schema. Define the styles in it.
2. Add the style module to the dependencies of the page for found subscribers.

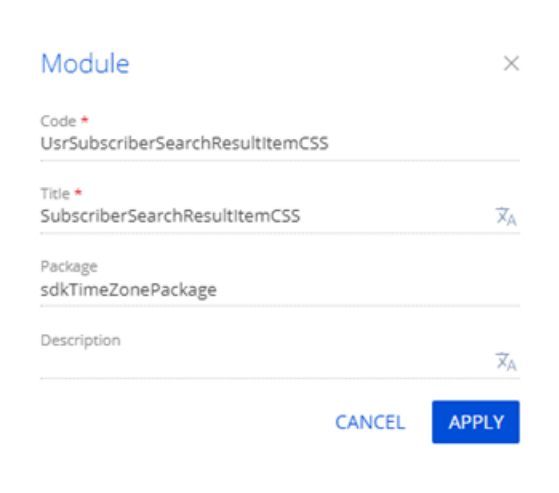
1. Create a module schema

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



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

   - Set [*Code*] to "`UsrSubscriberSearchResultItemCSS`."
   - Set [*Title*] to "`SubscriberSearchResultItemCSS`."



   Click [*Apply*] to apply the properties.

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

Set up the visual styles of the time zone

```
/* Set up the visual styles of the element to add. */
.ctiPanelMain .search-result-items-list-container .timezone {
    /* Top padding. */
    padding-top: 13px;
    /* Bottom margin. */
    margin-bottom: -10px;
}
/* Set up the visual styles of the contact time. */
.ctiPanelMain .search-result-items-list-container .timezone-caption {
    /* Left padding. */
    padding-left: 10px;
    /* Text color. */
    color: rgb(255, 174, 0);
    /* Set the text font to bold. */
    font-weight: bold;
}
/* Set up the visual styles of the contact city. */
.ctiPanelMain .search-result-items-list-container .timezone-city {
    /* Left padding. */
    padding-left: 10px;
}
```

5. Go to the [*JS*] node of the object structure and add the module code.

UsrSubscriberSearchResultItemCSS

```
define("UsrSubscriberSearchResultItemCSS", [], function() {
    return {};
});
```

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

2. Edit the view model schema of the page for found subscribers

To **use the module and its styles** in the schema of the page for found subscribers:

1. Open the `SubscriberSearchResultItem` view model schema of the page for found subscribers.
2. Add the `UsrSubscriberSearchResultItemCSS` module to the dependencies of the `SubscriberSearchResultItem` schema.

   View the source code of the edited schema of the page for found subscribers.

SubscriberSearchResultItem

```
define("SubscriberSearchResultItem", ["TimezoneGenerator", "TimezoneMixin", "css!UsrSubscribe
    return {
        /* The attributes of the page view model. */
        attributes: {
            /* The name of the attribute that controls the visibility of the time zone elemen
            "IsShowTimeZone": {
                /* The data type of the view model column. */
                "dataValueType": Terrasoft.DataValueType.BOOLEAN,
                /* The attribute type. */
                "type": Terrasoft.ViewModelColumnType.VIRTUAL_COLUMN,
                /* The default value. */
                "value": true
            }
        },
        /* The mixins of the page view model. */
        mixins: {
            /* Enable the mixin. */
            TimezoneMixin: "Terrasoft.TimezoneMixin"
        },
        /* The methods of the page view model. */
        methods: {
            /* The class constructor. */
            constructor: function() {
                /* Call the base constructor. */
```

```
                this.callParent(arguments);
                /* The flag that indicates that the subscriber is a contact. */
                var isContact = this.isContactType();
                /* If the subscriber is a contact, display the element. */
                this.set("IsShowTimeZone", isContact);
                /* If the subscriber is a contact. */
                if (isContact) {
                    /* The contact ID. */
                    var contactId = this.get("Id");
                    /* Search for the contact time zone. */
                    this.mixins.TimezoneMixin.init.call(this, contactId);
                }
            },
            /* Return the flag that indicates that the subscriber is a contact. */
            isContactType: function() {
                /* The subscriber type. */
                var type = this.get("Type");
                /* Return the comparison result. */
                return type === "Contact";
            }
        },
        /* The array of modifications of the page view model. */
        diff: [
            {
                /* Add the element to the page. */
                "operation": "insert",
                /* The meta name of the parent container to add the element. */
                "parentName": "SubscriberSearchResultItemContainer",
                /* Add the element to the parent element's collection of elements. */
                "propertyName": "items",
                /* The meta name of the element to add. */
                "name": "TimezoneContact",
                /* The properties to pass to the element constructor. */
                "values": {
                    /* The element type. */
                    "itemType": Terrasoft.ViewItemType.CONTAINER,
                    /* Call the method of the generator to create the view configuration. */
                    "generator": "TimezoneGenerator.generateTimeZone",
                    /* Bind the container visibility to the attribute. */
                    "visible": {"bindTo": "IsShowTimeZone"},
                    /* The name of the CSS class. */
                    "wrapClass": ["subscriber-data", "timezone"],
                    /* Bind the caption to the attribute. */
                    "timeZoneCaption": {"bindTo": "TimeZoneCaption"},
                    /* Bind the city to the attribute. */
                    "timeZoneCity": {"bindTo": "TimeZoneCity"}
                },
                /* The element layout in the parent container. */
                "index": 2
```

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

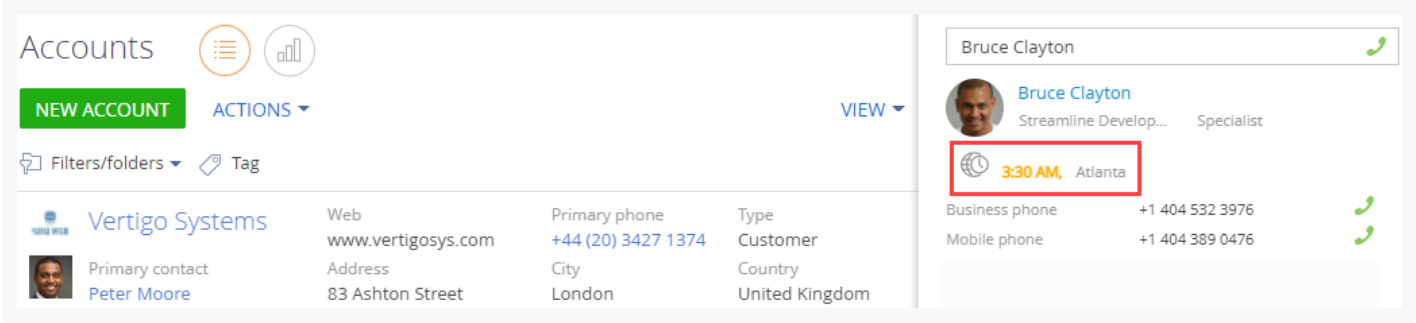
3. Click [*Save*] on the Designer's toolbar.

Outcome of the example

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

1. Open the [*CTI panel*] tab of the communication panel.
2. Search for a subscriber.

As a result, when you search for a contact, Creatio will display their time zone on the [*CTI panel*] tab of the communication panel. The current time of the contact will be used.



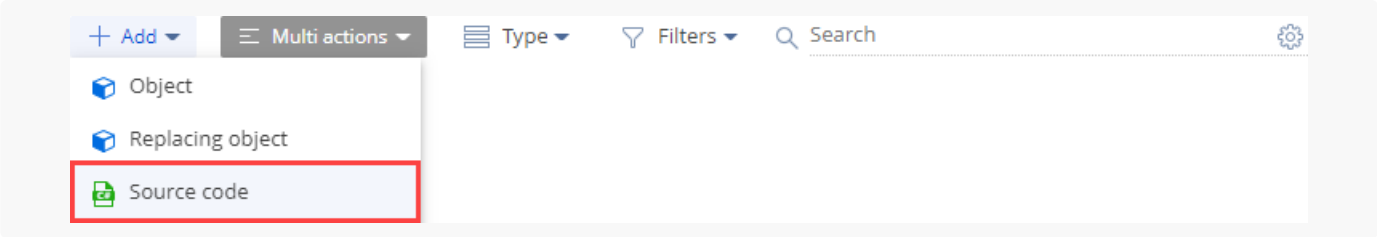
Create a custom reminder

Medium

Example. Create a custom reminder for the opportunity update date in a lead. Creatio displays the date in the [*Next actualization date*] field on the [*Opportunity info*] tab. Display the reminder on the reminder tab of the notification center.

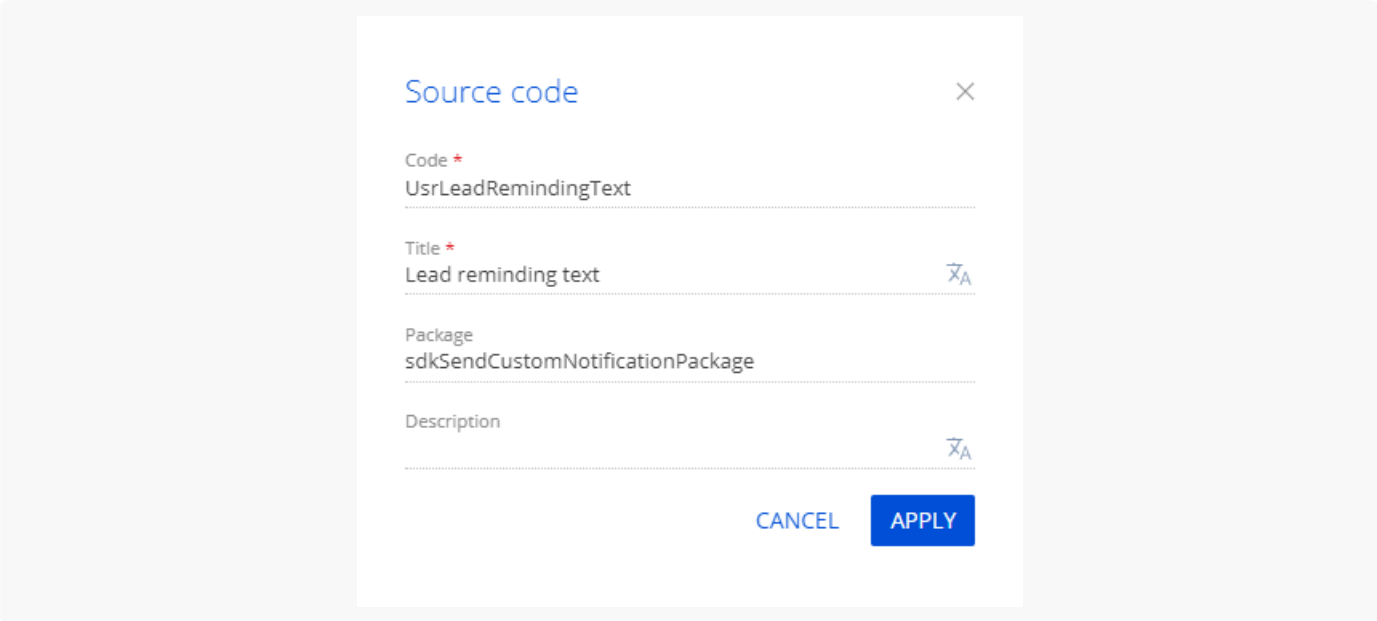
1. Implement the reminder text and dialog

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



3. Fill out the **schema properties** in the Source Code Designer:

   - Set [*Code*] to "`UsrLeadRemindingText`."
   - Set [*Title*] to "`Lead reminding text`."



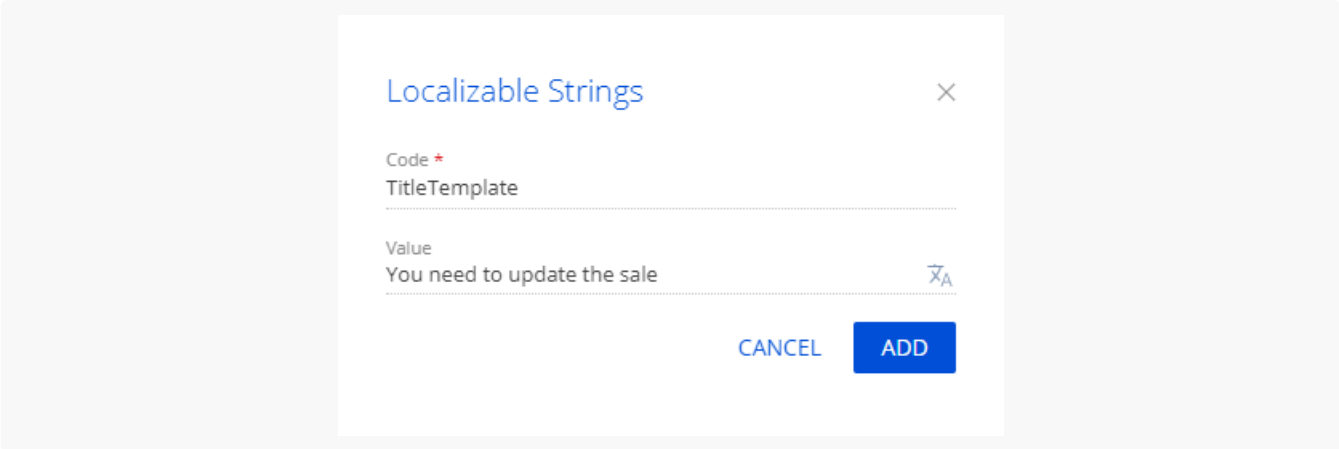
   Click [*Apply*] to apply the properties.

4. Add a **localizable string** that contains the reminder dialog caption.

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

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

      - Set [*Code*] to "`TitleTemplate`."
      - Set [*Value*] to "`You need to update the sale`."



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

5. Add a **localizable string** that contains the reminder text similarly.

   View the properties of the localizable string to add in the table below.

   Localizable string properties

[***Code***]	[***Value***]
"BodyTemplate"	"Lead {0} requires update of sales information"

6. Implement the **reminder text and dialog**.

   View the source code of the source code schema below.

UsrLeadRemindingText

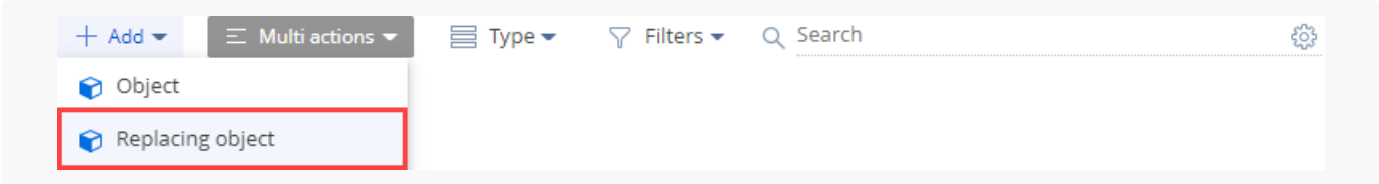
```
namespace Terrasoft.Configuration
{
    using System.Collections.Generic;
    using Terrasoft.Common;
    using Terrasoft.Core;

    public class UsrLeadRemindingText : IRemindingTextFormer
    {
        private const string ClassName = nameof(UsrLeadRemindingText);
        protected readonly UserConnection UserConnection;

        public UsrLeadRemindingText(UserConnection userConnection) {
            UserConnection = userConnection;
        }
        /* Generate the reminder text using the collection of incoming parameters and BodyTem
        public string GetBody(IDictionary<string, object> formParameters) {
            formParameters.CheckArgumentNull("formParameters");
            var bodyTemplate = UserConnection.GetLocalizableString(ClassName, "BodyTemplate")
            var leadName = (string)formParameters["LeadName"];
```

```
            var body = string.Format(bodyTemplate, leadName);
            return body;
        }
        /* Generate the reminder caption using the class name and TitleTemplate localizable s
        public string GetTitle(IDictionary<string, object> formParameters) {
            return UserConnection.GetLocalizableString(ClassName, "TitleTemplate");
        }
    }
}
```

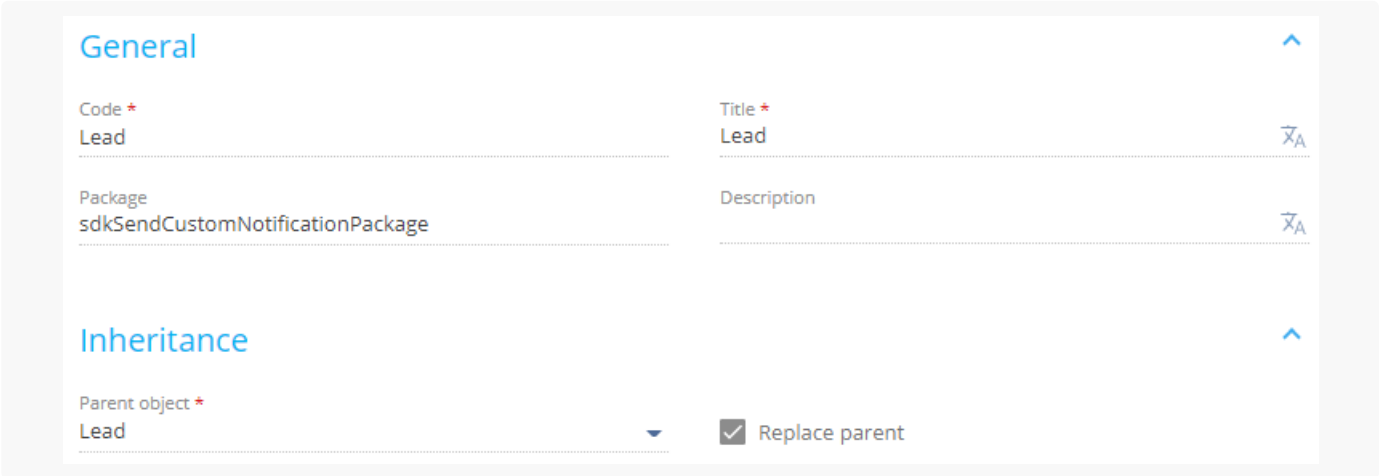
7. Click [*Save*] on the Source Code Designer's toolbar to save the changes to Creatio metadata temporarily.
8. Click [*Publish*] on the Source Code Designer's toolbar to apply the changes to the database level.

2. Implement the mechanism that sends the reminder

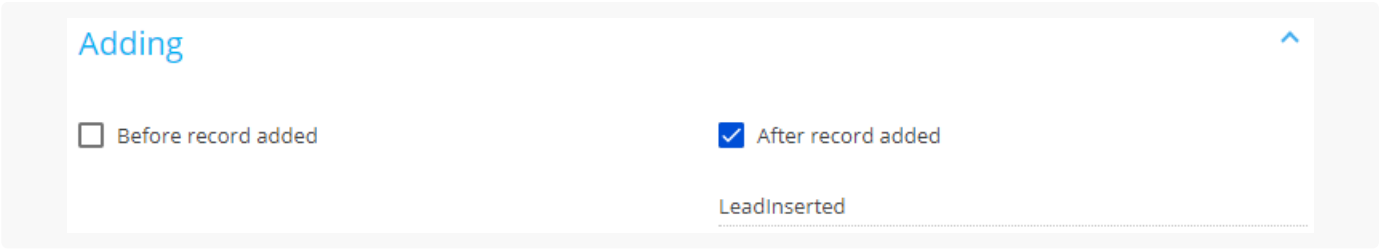
1. Go to the [*Configuration*] section and select a custom package to add the schema.
2. Click [*Add*] → [*Replacing object*] on the section list toolbar.



3. Select "Lead" in the [*Parent object*] property of the Object Designer.



4. Create an **object event**.
   a. Open the [*Events*] node.
   b. Go to the [*Adding*] block → select the [*After record added*] checkbox.



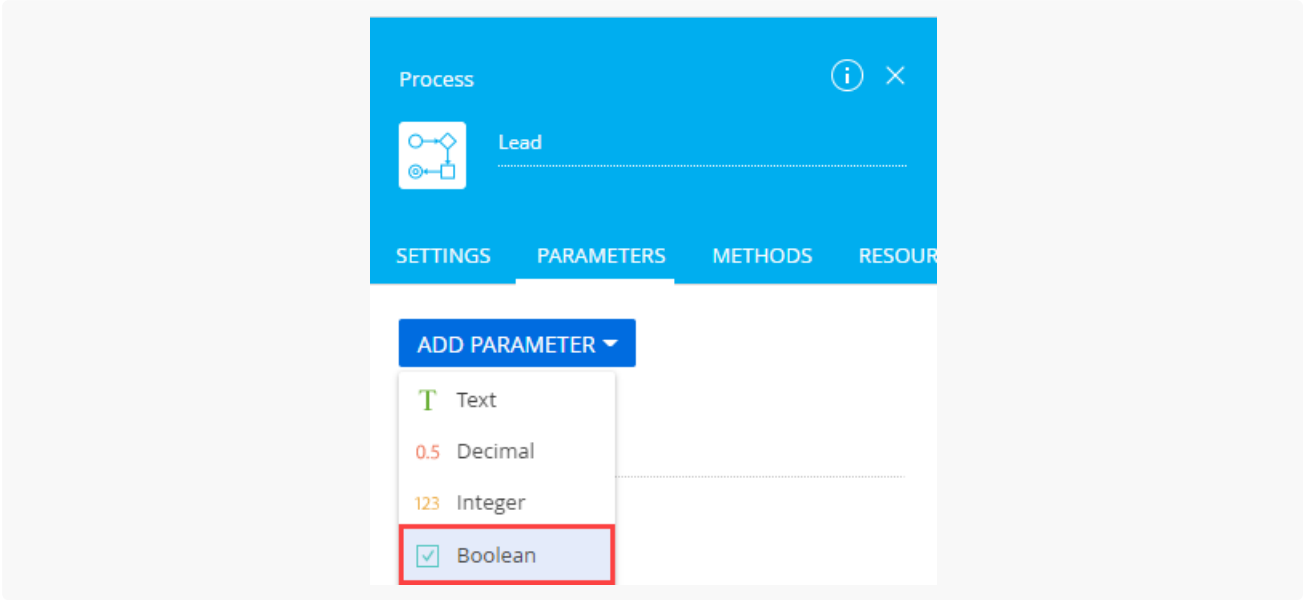
5. Set up the **parameter the business process uses to generate the reminder** in the object schema.

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

   b. Click [*Open process*] on the Object Designer's toolbar.

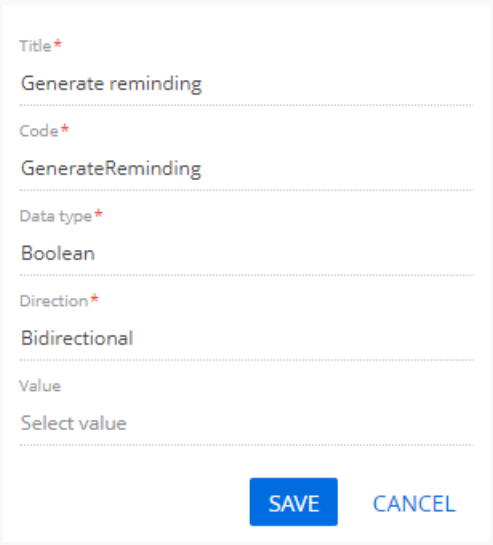
   c. Add a **parameter**.

      a. Open the [*Parameters*] tab in the setup area and click [*Add parameter*] → [*Boolean*].



      b. Fill out the **parameter properties** in the Object Designer.

      - Set [*Title*] to "`Generate reminding`."
      - Set [*Code*] to "`GenerateReminding`."



e. Click [*Save*].

d. Overload the **methods**.

- Overload the **method called after the object is saved**. To do this, open the [*Methods*] tab of the setup area and add the source code of the overloaded `LeadSaved()` method.

LeadSaved()

```
public override void LeadSaved() {
    base.LeadSaved();
    /* Check whether the reminder must be generated. */
    if (!GenerateReminding) {
        return;
    }
    DateTime remindTime = Entity.NextActualizationDate;
    if (Entity.OwnerId.Equals(Guid.Empty) || remindTime.Equals(default(DateTime))) {
        return;
    }
    /* Create an instance of the UsrLeadRemindingText class. */
    IRemindingTextFormer textFormer = ClassFactory.Get<UsrLeadRemindingText>(new Constru
    /* Retrieve the lead name. */
    string leadName = Entity.LeadName;
    /* Retrieve the reminder text. */
    string subjectCaption = textFormer.GetBody(new Dictionary<string, object> {
        {"LeadName", leadName}
    });
    /* Retrieve the reminder caption. */
    string popupTitle = textFormer.GetTitle(null);
    /* Configure the reminder. */
    var remindingConfig = new RemindingConfig(Entity);
    /* Set the message author to the current contact. */
    remindingConfig.AuthorId = UserConnection.CurrentUser.ContactId;
```

```
    /* Set the target recipient to the lead owner. */
    remindingConfig.ContactId = Entity.OwnerId;
    /* Set the type to reminding. */
    remindingConfig.NotificationTypeId = RemindingConsts.NotificationTypeRemindingId;
    /* Set the date for sending a reminder to the opportunity update date in the lead. *
    remindingConfig.RemindTime = remindTime;
    /* Reminder text. */
    remindingConfig.Description = subjectCaption;
    /* Reminder caption. */
    remindingConfig.PopupTitle = popupTitle;
    /* Create a utility class of the reminder. */
    var remindingUtilities = ClassFactory.Get<RemindingUtilities>();
    /* Create a reminder. */
    remindingUtilities.CreateReminding(UserConnection, remindingConfig);
}
```

The `RemindingConsts.NotificationTypeRemindingId` value of the `remindingConfig.NotificationTypeId` constant lets you display the notification on the reminder tab of the notification center. If you set the constant to `RemindingConsts.NotificationTypeNotificationId`, Creatio displays the reminder on the service message tab of the notification center.

- Overload the **method called before the object is saved**. To do this, open the [*Methods*] tab and add the source code of the overloaded `LeadSavingMethod()` method.

`LeadSavingMethod()`

```
public override void LeadSavingMethod() {
    /* Call the base method implementation. */
    base.LeadSavingMethod();
    /* Retrieve the owner ID. */
    var oldOwnerId = Entity.GetTypedOldColumnValue<Guid>("OwnerId");
    /* Retrieve the date of the next update. */
    DateTime oldRemindDate = Entity.GetTypedOldColumnValue<DateTime>("NextActualizationD
    /* Compare the old and new owner IDs. */
    bool ownerChanged = !Entity.OwnerId.Equals(oldOwnerId);
    /* Compare the old and new update dates. */
    bool remindDateChanged = !Entity.NextActualizationDate.Equals(oldRemindDate);
    /* Set the GenerateReminding process parameter to true if the owner or date of the r
    GenerateReminding = ownerChanged || remindDateChanged;
}
```

Complete source code of the object's embedded process

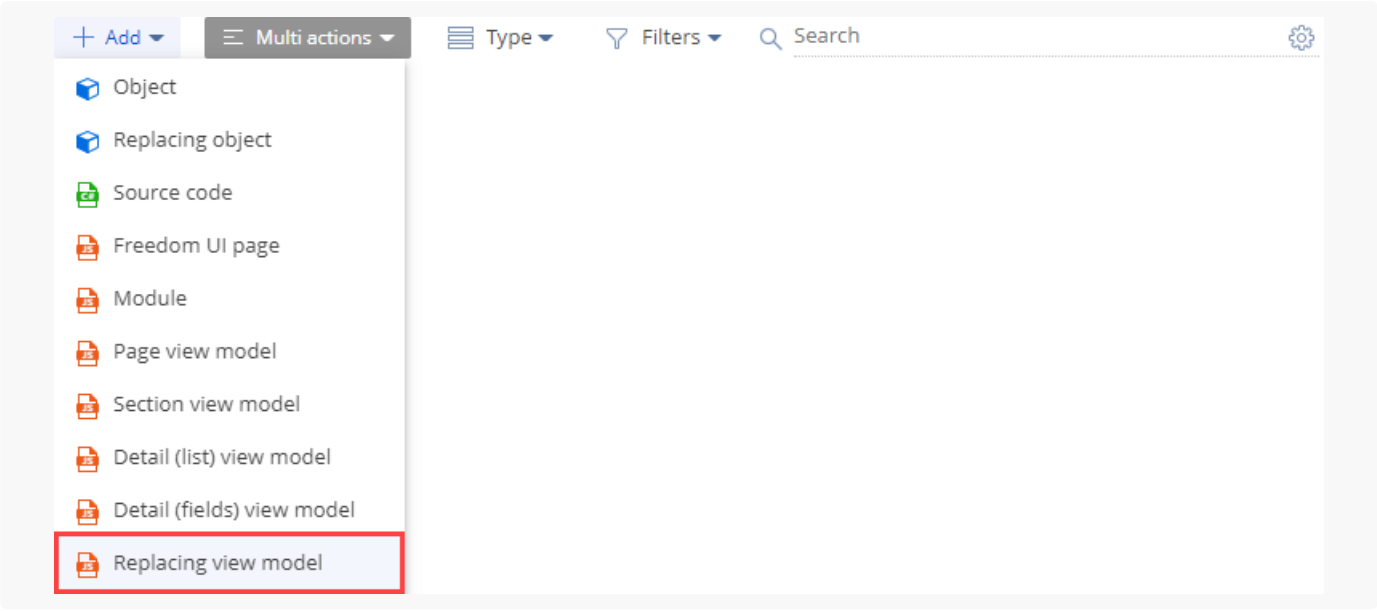
g. Click [*Save*] then [*Publish*] on the Object Designer's toolbar.

h. Click [*Cancel*] on the Object Designer's toolbar.

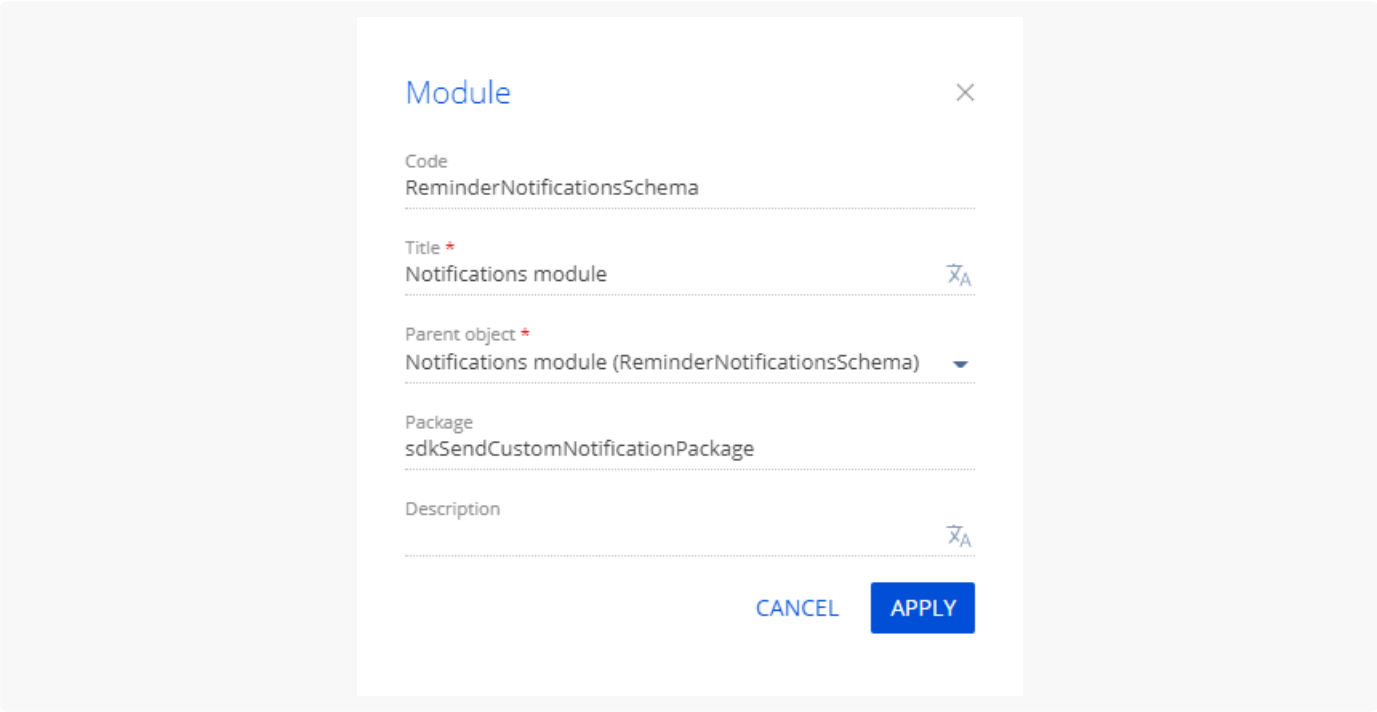
6. Click [*Publish*] to create a corresponding database table.

3. Implement the mechanism that displays the reminder

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



3. Select "`ReminderNotificationsSchema`" in the [*Parent object*] schema property of the Module Designer.



Click [*Apply*] to apply the properties.

4. Implement the **mechanism that displays the reminder**.

   a. Implement the following **methods** in the `methods` property:

- `getIsLeadNotification()`. Determines whether the reminder is connected to the lead.
- `getNotificationSubjectCaption()`. Returns the reminder caption.

d. Add **configuration objects with the settings that determine layouts of the following elements** to the `diff` array of modifications:

- container on the page
- caption container
- image
- date
- reminder text

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

ReminderNotificationsSchema

```
define("ReminderNotificationsSchema", ["ReminderNotificationsSchemaResources"], function() {
    return {
        /* The name of the object schema. */
        entitySchemaName: "Reminding",
        /* The methods of the view model. */
        methods: {
            /* Determine whether the reminder is connected to the lead. */
            getIsLeadNotification: function() {
                return this.get("SchemaName") === "Lead";
            },
            /* Return the reminder caption. */
            getNotificationSubjectCaption: function() {
                var caption = this.get("Description");
                return caption;
            }
        },
        /* Display the reminder. */
        diff: [
            /* The metadata to add the container. */
            {
                /* Add the element to the page. */
                "operation": "insert",
                /* The meta name of the container to add. */
                "name": "NotificationleadItemContainer",
                /* The meta name of the parent container to add the current container. */
                "parentName": "Notification",
                /* Add the container to the parent element's collection of elements. */
                "propertyName": "items",
                /* The properties to pass to the element's constructor. */
                "values": {
                    "itemType": Terrasoft.ViewItemType.CONTAINER,
```

```
                "wrapClass": [
                    "reminder-notification-item-container"
                ],
                /* Display only for leads. */
                "visible": {"bindTo": "getIsLeadNotification"},
                "items": []
            }
        },
        /* The metadata to add the caption container. */
        {
            "operation": "insert",
            "name": "NotificationItemleadTopContainer",
            "parentName": "NotificationleadItemContainer",
            "propertyName": "items",
            "values": {
                "itemType": Terrasoft.ViewItemType.CONTAINER,
                "wrapClass": ["reminder-notification-item-top-container"],
                "items": []
            }
        },
        /* The metadata to add the image. */
        {
            "operation": "insert",
            "name": "NotificationleadImage",
            "parentName": "NotificationItemleadTopContainer",
            "propertyName": "items",
            "values": {
                "itemType": Terrasoft.ViewItemType.BUTTON,
                "className": "Terrasoft.ImageView",
                "imageSrc": {"bindTo": "getNotificationImage"},
                "classes": {"wrapClass": ["reminder-notification-icon-class"]}
            }
        },
        /* The metadata for the date. */
        {
            "operation": "insert",
            "name": "NotificationDate",
            "parentName": "NotificationItemleadTopContainer",
            "propertyName": "items",
            "values": {
                "itemType": Terrasoft.ViewItemType.LABEL,
                "caption": {"bindTo": "getNotificationDate"},
                "classes": {"labelClass": ["subject-text-labelClass"]}
            }
        },
        /* The metadata for the reminder text. */
        {
            "operation": "insert",
            "name": "NotificationleadSubject",
```

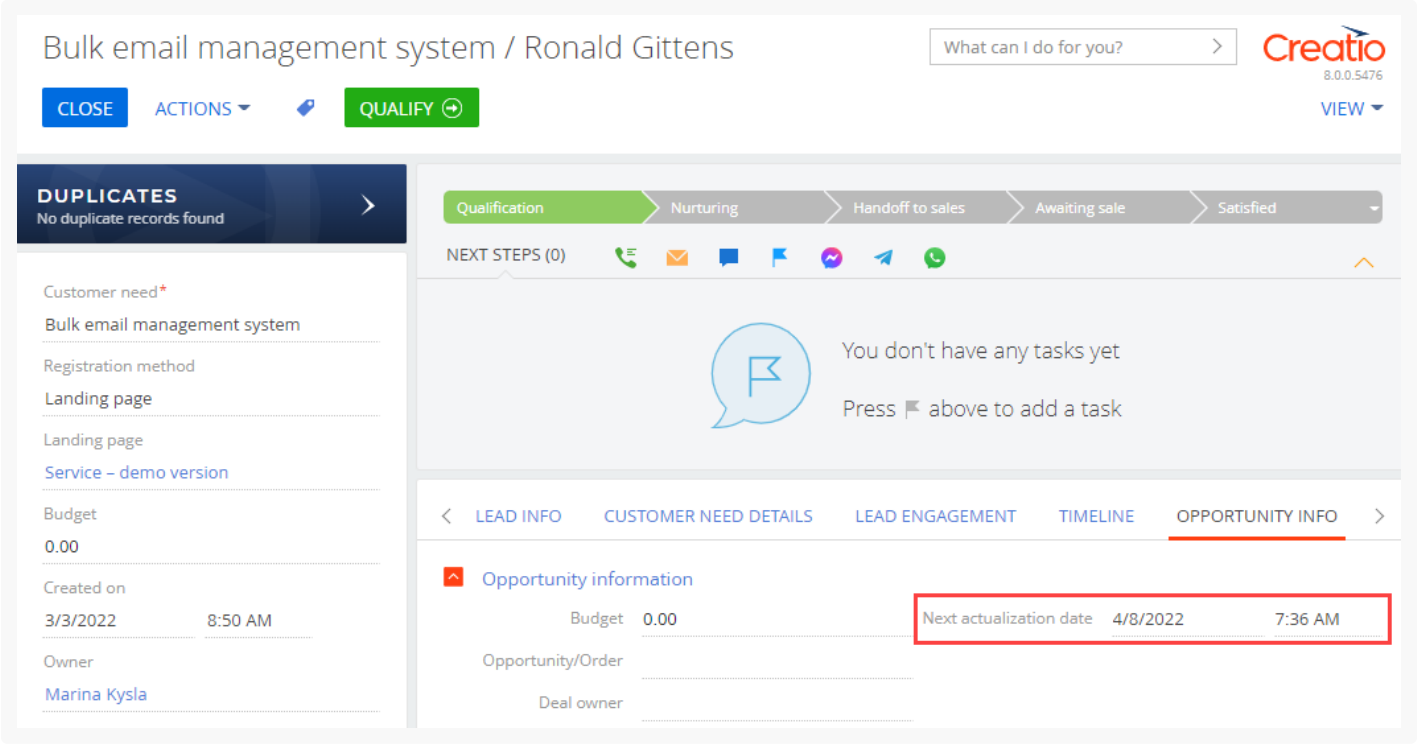
```
                "parentName": "NotificationItemleadTopContainer",
                "propertyName": "items",
                "values": {
                    "itemType": Terrasoft.ViewItemType.LABEL,
                    "caption": {"bindTo": "getNotificationSubjectCaption"},
                    "click": {"bindTo": "onNotificationSubjectClick"},
                    "classes": {"labelClass": ["subject-text-labelClass", "label-link", "labe
                }
            }
        ]
    };
});
```

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

Outcome of the example

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

1. Refresh the [*Leads*] section page.
2. Open the lead page → the [*Opportunity info*] tab.
3. Select the needed date and time of the opportunity update in the [*Next actualization date*] field.
4. Select your user contact in the [*Owner*] field.



As a result, Creatio will display the reminder for the opportunity update date in the lead on the reminder tab of the notification center.