Creatio Academy

Marketing components

Version 8.0

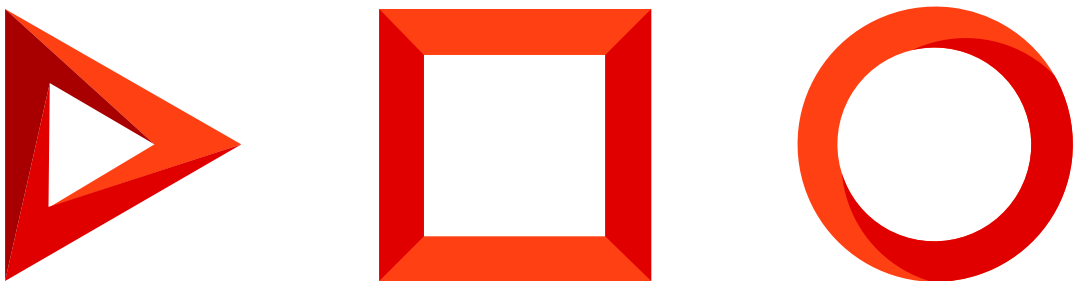


Table of Contents

Marketing campaigns basics

Advanced

Marketing campaign diagrams are created in a visual campaign designer in the [*Campaigns*] section. The campaign diagram consists of campaign elements and transitions (flows).

Once the campaign is launched, the flow-schema of the campaign is created. The campaign elements are converted to a campaign execution chain and the start time is calculated for each element. The flow-schema can be significantly different from the visual campaign diagram in the designer.

Campaign elements can be synchronous and asynchronous.

`Synchronous` elements are executed according to the order specified in the flow-schema. The transition to the subsequent elements is performed once the synchronous element is executed. The execution flow is blocked and waits for the operation to complete.

`Asynchronous` elements wait for the finished execution of certain external systems, resources, asynchronous services, or user reactions (e.g., clicking a link in an email).

Their position in the flow-schema is determined by their element type. The [*Add from folder*] and [*Exit according to folder conditions*] elements are executed first. These elements are used to add or remove participants from the campaign audience. Campaign participants are moving from one element to the other through the flows. If the flow has certain configured conditions, the system filters the participants based on these conditions and determines the execution time of the subsequent element.

The mechanism for planning the next campaign launch

The following is the algorithm for for planning the next campaign launch:

1. The time of the next launch of an element is determined by the configured delay:

   - The “In a day” option is selected. The date and time of the next execution of this element is calculated with the help of the following formula:

     `Date and time of execution = current date and time + N minutes / hour`,

     where `N` is the value of the [*Number of days*] field, populated by the user.

   - The “Few days” option is selected. The next execution of this element is performed with the help of the following formula:

     `Date = current date+ N days,`

     where `N` is the value of the [*Number of days*] field, populated by the user.

     `Execution time = time specified by the user.`

   - The “No, execute after the previous one” option is selected. The next execution of this element is performed at the time of the next launch of the campaign.

2. According to the variant described in paragraph 1, the launch time for each element of the campaign scheme is calculated.

3. Upon comparing all values, the closest launch time selected and set as the campaign launch time.
4. Forming a list of elements, which will be executed upon next launch. The list contains all elements, the launch time of which is the same as the campaign launch time.

Main campaign element classes

JavaScript classes

The base element schema class is `ProcessFlowElementSchema`. The `CampaignBaseCommunicationSchema` is the parent class for all elements in the [*Communications*] group. The `CampaignBaseAudienceSchema` Audience group of elements.

When creating an element in a new group of elements, it is recommended to implement the base schema of the element first, and then inherit each element from it.

Each schema corresponds to the schema of the element properties edit page. The base edit page schema is span class= "filled">BaseCampaignSchemaElementPage. Each new element page extends the base page.

The `CampaignSchemaManager` class manages the schemas of elements available in the system. It inherits the main functionality of the `BaseSchemaManager` class.

C# classes

Simple element classes

`CampaignSchemaElement` – base class. All other elements are inherited from this class.

`SequenceFlowElement` – base class for the [*Sequence flow*] element.

`ConditionSequenceFlowElement` – base class for the [*Condition flow*] element.

`EmailConditionalTransitionElement` – transition element class by response.

`AddCampaignParticipantElement` – add audience (participants) element class.

`ExitFromCampaignElement` – the class of the audience exit element.

`MarketingEmailElement` – the class of the Email element.

Executable element classes

`CampaignProcessFlowElement` – base class. All other executable elements are inherited from this class.

`AddCampaignAudienceElement` – audience element class.

`ExcludeCampaignAudienceElement` – the class of the audience exit element.

`BulkEmailCampaignElement` – the class of the Email element.

Add a custom campaign element

Advanced

Use the Campaign designer to set up marketing campaigns. Using this designer, you can create a campaign diagram that consists of interconnected elements. In addition to default campaign elements you can create custom ones.

Algorithm for adding a custom campaign element

1. Create a new element for the [*Campaign designer*].
2. Create the element's edit page.
3. Expand the [*Campaign designer*] menu with a new element.
4. Create the element's server part.
5. Create executable element for the new campaign element.
6. Add custom logic for processing campaign events.

Example. Create a new campaign element for sending text messages (SMS) for users.

Example implementation algorithm

1. Creating a new element for the [*Campaign designer*]

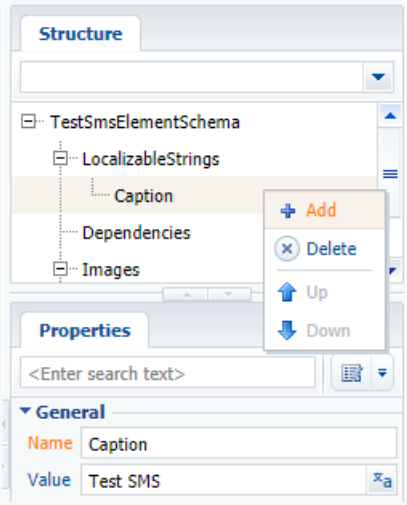
To display the element in the [*Campaign designer*] UI, add a new module schema for the campaign element. The procedure for creating a module schema is covered in the "Create a client schema" article. Set the following properties for the created schema:

- [*Title*] – "Test SMS Element Schema".
- [*Name*] – "TestSmsElementSchema".

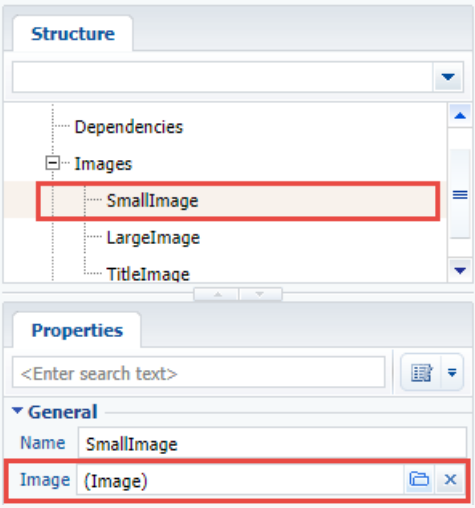
Attention. The schema names in the case below do not contain the `Usr` prefix. You can change the default prefix in the [*Prefix for object name*] (`SchemaNamePrefix`) system setting.

Add a localized string to the schema:

- [*Name*] – "Caption".
- [*Value*] – "Test SMS".



Add images that will represent the campaign element in the [*Campaign designer*]. Use the `SmallImage`, `LargeImage` and `TitleImage` properties to add the images.



In this example we used a scalable vector graphics (SVG) image available here.

Add following source code on the [*Source Code*] section of the schema":

TestSmsElementSchema

```
define("TestSmsElementSchema", ["TestSmsElementSchemaResources", "CampaignBaseCommunicationSchem
    function(resources) {
        Ext.define("Terrasoft.manager.TestSmsElementSchema", {
            // Parent schema.
            extend: "Terrasoft.CampaignBaseCommunicationSchema",
            alternateClassName: "Terrasoft.TestSmsElementSchema",
            // Manager Id. Must be unique.
            managerItemUId: "a1226f93-f3e3-4baa-89a6-11f2a9ab2d71",
            // Plugged mixins.
```

```
            mixins: {
                campaignElementMixin: "Terrasoft.CampaignElementMixin"
            },
            // Element name.
            name: "TestSms",
            // Resource binding.
            caption: resources.localizableStrings.Caption,
            titleImage: resources.localizableImages.TitleImage,
            largeImage: resources.localizableImages.LargeImage,
            smallImage: resources.localizableImages.SmallImage,
            // Schema name of the edit page.
            editPageSchemaName: "TestSmsElementPropertiesPage",
            // Element type.
            elementType: "TestSms",
            // Full name of the class that corresponds to the current schema.
            typeName: "Terrasoft.Configuration.TestSmsElement, Terrasoft.Configuration",
            // Overriding the properties of visual styles.
            color: "rgba(249, 160, 27, 1)",
            width: 69,
            height: 55,
            // Setting up element-specific properties.
            smsText: null,
            phoneNumber: null,
            // Determining the types of the elemen's outbound connections.
            getConnectionUserHandles: function() {
                return ["CampaignSequenceFlow", "CampaignConditionalSequenceFlow"];
            },
            // Expnding the properties for serialization.
            getSerializableProperties: function() {
                var baseSerializableProperties = this.callParent(arguments);
                return Ext.Array.push(baseSerializableProperties, ["smsText", "phoneNumber"]);
            },
            // Setting up the icons that are displayed on the campaign diagram.
            getSmallImage: function() {
                return this.mixins.campaignElementMixin.getImage(this.smallImage);
            },
            getLargeImage: function() {
                return this.mixins.campaignElementMixin.getImage(this.largeImage);
            },
            getTitleImage: function() {
                return this.mixins.campaignElementMixin.getImage(this.titleImage);
            }
        });
        return Terrasoft.TestSmsElementSchema;
    });
```

Specifics:

- The `managerItemUId` property value must be unique for the new element and not repeat the value of the other elements.
- The `typeName` property contains the name of the C# class that corresponds to the campaign element name. This class will be saving and reading the element's properties from the schema metadata.

Save the schema to apply changes.

Adding a group of elements

If a new group of elements, such as [*Scripts*] must be created for the campaign element, the schema source code must be supplemented with the following code:

TestSmsElementSchema

```
// Name of the new group.
group: "Scripts",

constructor: function() {
      if (!Terrasoft.CampaignElementGroups.Items.contains("Scripts")) {
           Terrasoft.CampaignElementGroups.Items.add("Scripts", {
                name: "Scripts",
                caption: resources.localizableStrings.ScriptsElementGroupCaption
           });
      }
      this.callParent(arguments);
}
```

Also, add a localized string with the following properties:

- [*Name*] – "ScriptsElementGroupCaption".
- `Value` – "Scripts".

Save the schema to apply changes.

2. Creating the element's edit page

Create the campaign element's edit page in the custom package to enable the users to view and edit the element's properties. To do this, create a schema that expands `BaseCampaignSchemaElementPage` (`CampaignDesigner` package). The procedure for creating a replacing client schema is covered in the "Create a client schema" article.

Set the following properties for the created schema:

- [*Title*] – "TestSmsElementPropertiesPage".
- [*Name*] – "TestSmsElementPropertiesPage".
- [*Parent object*] – "BaseCampaignSchemaElementPage".

Add localized strings to the created schema with properties given in the table.

Localized string properties

[***Name***]	[***Value***]
PhoneNumberCaption	Sender phone number
SmsTextCaption	Message
TestSmsText	Which text message should be sent? (Which SMS text to send?)

Add following source code on the [*Source Code*] section of the schema":

TestSmsElementPropertiesPage

```
define("TestSmsElementPropertiesPage", [],
    function() {
        return {
            attributes: {
                // Attributes that correspond to specific properties of element schema.
                "PhoneNumber": {
                    "dataValueType": this.Terrasoft.DataValueType.TEXT,
                    "type": this.Terrasoft.ViewModelColumnType.VIRTUAL_COLUMN
                },
                "SmsText": {
                    "dataValueType": this.Terrasoft.DataValueType.TEXT,
                    "type": this.Terrasoft.ViewModelColumnType.VIRTUAL_COLUMN
                }
            },
            methods: {
                init: function() {
                    this.callParent(arguments);
                    this.initAcademyUrl(this.onAcademyUrlInitialized, this);
                },
                // Element code for generating a contextual help link.
                getContextHelpCode: function() {
                    return "CampaignTestSmsElement";
                },
                // Initialization of attributes with the current schema property values.
                initParameters: function(element) {
                    this.callParent(arguments);
                    this.set("SmsText", element.smsText);
                    this.set("PhoneNumber", element.phoneNumber);
                },
                // Saving schema properties.
                saveValues: function() {
                    this.callParent(arguments);
```

```
                var element = this.get("ProcessElement");
                element.smsText = this.getSmsText();
                element.phoneNumber = this.getPhoneNumber();

            },
            // Reading current attribute values.
            getPhoneNumber: function() {
                var number = this.get("PhoneNumber");
                return number ? number : "";
            },
            getSmsText: function() {
                var smsText = this.get("SmsText");
                return smsText ? smsText : "";
            }
        },
        diff: [
            // UI container.
            {
                "operation": "insert",
                "name": "ContentContainer",
                "propertyName": "items",
                "parentName": "EditorsContainer",
                "className": "Terrasoft.GridLayoutEdit",
                "values": {
                    "itemType": Terrasoft.ViewItemType.GRID_LAYOUT,
                    "items": []
                }
            },
            // Element primary signature.
            {
                "operation": "insert",
                "name": "TestSmsLabel",
                "parentName": "ContentContainer",
                "propertyName": "items",
                "values": {
                    "layout": {
                        "column": 0,
                        "row": 0,
                        "colSpan": 24
                    },
                    "itemType": this.Terrasoft.ViewItemType.LABEL,
                    "caption": {
                        "bindTo": "Resources.Strings.TestSmsText"
                    },
                    "classes": {
                        "labelClass": ["t-title-label-proc"]
                    }
                }
            },
```

```
                // Caption for the text field where sender name is entered.
                {
                    "operation": "insert",
                    "name": "PhoneNumberLabel",
                    "parentName": "ContentContainer",
                    "propertyName": "items",
                    "values": {
                        "layout": {
                            "column": 0,
                            "row": 1,
                            "colSpan": 24
                        },
                        "itemType": this.Terrasoft.ViewItemType.LABEL,
                        "caption": {
                            "bindTo": "Resources.Strings.PhoneNumberCaption"
                        },
                        "classes": {
                            "labelClass": ["label-small"]
                        }
                    }
                },
                // Text field for entering phone number.
                {
                    "operation": "insert",
                    "name": "PhoneNumber",
                    "parentName": "ContentContainer",
                    "propertyName": "items",
                    "values": {
                        "labelConfig": {
                            "visible": false
                        },
                        "layout": {
                            "column": 0,
                            "row": 2,
                            "colSpan": 24
                        },
                        "itemType": this.Terrasoft.ViewItemType.TEXT,
                        "classes": {
                            "labelClass": ["feature-item-label"]
                        },
                        "controlConfig": { "tag": "PhoneNumber" }
                    }
                },
                // Caption for text field for entering message text.
                {
                    "operation": "insert",
                    "name": "SmsTextLabel",
                    "parentName": "ContentContainer",
```

```
                    "propertyName": "items",
                    "values": {
                        "layout": {
                            "column": 0,
                            "row": 3,
                            "colSpan": 24
                        },
                        "classes": {
                            "labelClass": ["label-small"]
                        },
                        "itemType": this.Terrasoft.ViewItemType.LABEL,
                        "caption": {
                            "bindTo": "Resources.Strings.SmsTextCaption"
                        }
                    }
                },
                // Text field for entering message text.
                {
                    "operation": "insert",
                    "name": "SmsText",
                    "parentName": "ContentContainer",
                    "propertyName": "items",
                    "values": {
                        "labelConfig": {
                            "visible": false
                        },
                        "layout": {
                            "column": 0,
                            "row": 4,
                            "colSpan": 24
                        },
                        "itemType": this.Terrasoft.ViewItemType.TEXT,
                        "classes": {
                            "labelClass": ["feature-item-label"]
                        },
                        "controlConfig": { "tag": "SmsText" }
                    }
                }
            ]
        };
    }
);
```

Save the schema to apply changes.

3. Expanding the Campaign designer menu with a new element

To display the new element in the Campaign designer menu, expand the campaign element base schema manager. Add a schema that expands `CampaignElementSchemaManagerEx` (the `CampaignDesigner` package) to the custom package. The procedure for creating a replacing client schema is covered in the "Create a client schema" article.

Set the following properties for the created schema:

- [*Title*] – "TestSmsCampaignElementSchemaManagerEx".
- [*Name*] – "CampaignElementSchemaManagerEx".
- [*Parent object*] – "CampaignElementSchemaManagerEx".

Add following source code on the [*Source Code*] section of the schema":

CampaignElementSchemaManager

```
require(["CampaignElementSchemaManager", "TestSmsElementSchema"],
    function() {
        // Adding a new schema to the list of available element schemas in the Campaign designer
        var coreElementClassNames = Terrasoft.CampaignElementSchemaManager.coreElementClassNames
        coreElementClassNames.push({
            itemType: "Terrasoft.TestSmsElementSchema"
        });
    });
```

Save the schema to apply changes.

4. Creating server part of the custom campaign element

To implement saving the campaign element properties, create a class that interacts with the application server part. The class must inherit `CampaignSchemaElement` and override the `ApplyMetaDataValue()` and `WriteMetaData()` methods.

Create a source code schema with the following properties:

- [*Title*] – "TestSmsElement".
- [*Name*] – "TestSmsElement".

For more information on creating source code schemas, please see the Create the [*Source code*] schema article.

Add the following source code on the [*Source Code*] section of the schema":

TestSmsElement

```
namespace Terrasoft.Configuration
{
    using System;
    using Terrasoft.Common;
```

```
using Terrasoft.Core;
using Terrasoft.Core.Campaign;
using Terrasoft.Core.Process;

[DesignModeProperty(Name = "PhoneNumber",
    UsageType = DesignModeUsageType.NotVisible, MetaPropertyName = PhoneNumberPropertyName)]
[DesignModeProperty(Name = "SmsText",
    UsageType = DesignModeUsageType.NotVisible, MetaPropertyName = SmsTextPropertyName)]
public class TestSmsElement : CampaignSchemaElement
{
    private const string PhoneNumberPropertyName = "PhoneNumber";
    private const string SmsTextPropertyName = "SmsText";
    // Default constructor.
    public TestSmsElement() {
        ElementType = CampaignSchemaElementType.AsyncTask;
    }
    // Constructor with parameter.
    public TestSmsElement(TestSmsElement source)
            : base(source) {
        ElementType = CampaignSchemaElementType.AsyncTask;
        PhoneNumber = source.PhoneNumber;
        SmsText = source.SmsText;
    }

    // Instance action Id.
    protected override Guid Action {
        get {
            return CampaignConsts.CampaignLogTypeMailing;
        }
    }

    // Phone number.
    [MetaTypeProperty("{A67950E7-FFD7-483D-9E67-3C9A30A733C0}")]
    public string PhoneNumber {
        get;
        set;
    }
    // Text message.
    [MetaTypeProperty("{05F86DF2-B9FB-4487-B7BE-F3955703527C}")]
    public string SmsText {
        get;
        set;
    }
    // Applies metadata values.
    protected override void ApplyMetaDataValue(DataReader reader) {
        base.ApplyMetaDataValue(reader);
        switch (reader.CurrentName) {
            case PhoneNumberPropertyName:
                PhoneNumber = reader.GetValue<string>();
```

```
                    break;
                case SmsTextPropertyName:
                    SmsText = reader.GetValue<string>();
                    break;
            }
        }

        // Records metadata values.
        public override void WriteMetaData(DataWriter writer) {
            base.WriteMetaData(writer);
            writer.WriteValue(PhoneNumberPropertyName, PhoneNumber, string.Empty);
            writer.WriteValue(SmsTextPropertyName, SmsText, string.Empty);
        }
        // Copies element.
        public override object Clone() {
            return new TestSmsElement(this);
        }

        // Creates a specific ProcessFlowElement instance.
        public override ProcessFlowElement CreateProcessFlowElement(UserConnection userConnectio
            var executableElement = new TestSmsCampaignProcessElement {
                UserConnection = userConnection,
                SmsText = SmsText,
                PhoneNumber = PhoneNumber
            };
            InitializeCampaignProcessFlowElement(executableElement);
            return executableElement;
        }
    }
}
```

Publish the source code schema.

5. Creating executable element for the new campaign element

For the custom campaign element to execute the needed logic, add an executable element. It is a class that inherits the CampaignProcessFlowElement class, where the `SafeExecute()` method is implemented.

To create an executable element, add a source code schema element with the following properties in the custom package:

- [*Title*] – "TestSmsCampaignProcessElement".
- [*Name*] – "TestSmsCampaignProcessElement".

Add following source code on the [*Source Code*] section of the schema":

TestSmsCampaignProcessElement

```
namespace Terrasoft.Configuration
{
    using System;
    using System.Collections.Generic;
    using Terrasoft.Core.Campaign;
    using Terrasoft.Core.DB;
    using Terrasoft.Core.Process;

    public class TestSmsCampaignProcessElement : CampaignProcessFlowElement
    {
        public const string ContactTableName = "Contact";

        public TestSmsCampaignProcessElement(ICampaignAudience campaignAudience) {
            CampaignAudience = campaignAudience;
        }

        public TestSmsCampaignProcessElement() {
        }

        // Audiences for whom to send texts on the current step.
        private ICampaignAudience _campaignAudience;
        private ICampaignAudience CampaignAudience {
            get {
                return _campaignAudience ??
                    (_campaignAudience = new CampaignAudience(UserConnection, CampaignId));
            }
            set {
                _campaignAudience = value;
            }
        }

        // SMS-specific properties. Passed from an instance of the TestSmsElement class.
        public string PhoneNumber {
            get;
            set;
        }
        public string SmsText {
            get;
            set;
        }

        // Implementation of the element execution method
        protected override int SafeExecute(ProcessExecutingContext context) {
            // TODO: Implement sending SMS messages.
            //
            // Current step for audiences is set as completed.
            return CampaignAudience.SetItemCompleted(SchemaElementUId);
        }
```

```
    }
}
```

Publish the source code schema.

6. Adding custom logic for processing campaign events

Use the event handler mechanism to implement custom logic on saving, copying, deleting, running and stopping campaigns. Create a public `sealed` handler class that inherits `CampaignEventHandlerBase`. Implement interfaces that describe specific event handler signatures. This class must not be generic. It must have a constructor available by default.

The following interfaces are supported in the current version:

- `IOnCampaignBeforeSave` – contains method that will be called before saving the campaign.
- `IOnCampaignAfterSave` – contains method that will be called after saving the campaign.
- `IOnCampaignDelete` – contains method that will be called before deleting the campaign.
- `IOnCampaignStart` – contains method that will be called before running the campaign.
- `IOnCampaignStop` – contains method that will be called before stopping the campaign.
- `IOnCampaignValidate` – contains method that will be called on validating the campaign.
- `IOnCampaignCopy` – contains method that will be called after copying the campaign.

If an exception occurs during the event processing, the call chain is stopped, and campaign status is reverted to the previous one in DB.

> **Note.** When implementing the `IOnCampaignValidate` interface, save errors in the campaign schema using the `AddValidationInfo(string)` method.

Additional case conditions

In order for the new custom campaign element to work, SMS gateway connection is required. The connection, account status and other parameters must be checked during campaign validation. The messages must be sent when campaign starts.

To implement these conditions, add a source code schema element with the following properties in the custom package:

- [*Title*] – "TestSmsEventHandler".
- [*Name*] – "TestSmsEventHandler".

Add following source code on the [*Source Code*] section of the schema":

TestSmsEventHandler

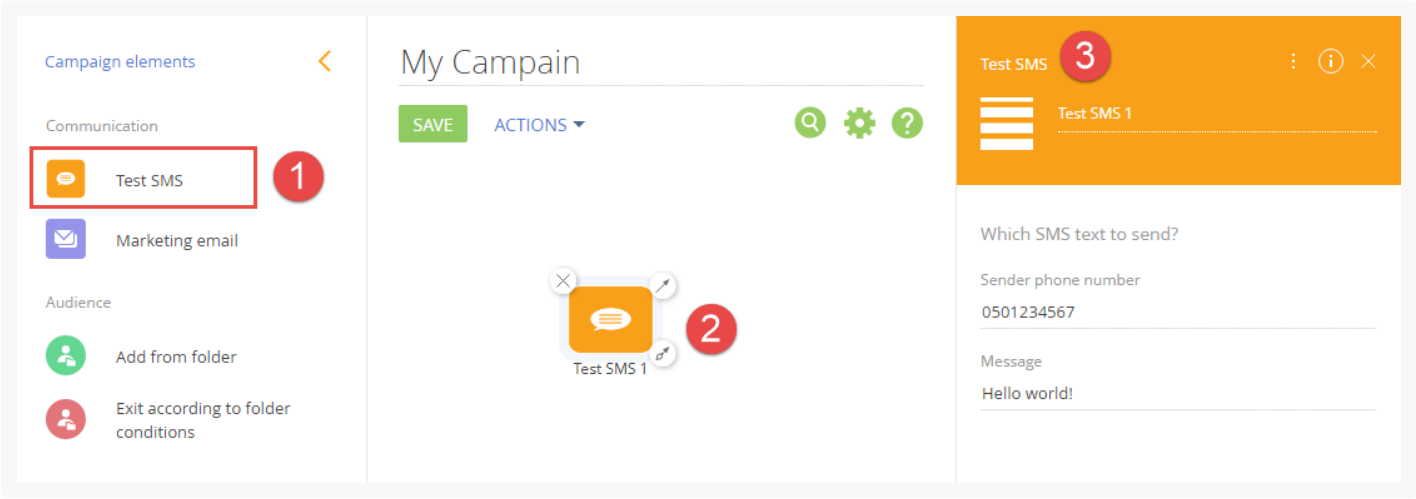
```
namespace Terrasoft.Configuration
```

```
{
    using System;
    using Terrasoft.Core.Campaign.EventHandler;

    public sealed class TestSmsEventHandler : CampaignEventHandlerBase, IOnCampaignValidate, IOr
    {
        // Implementing handler for the campaign start event.
        public void OnStart() {
            // TODO: Text SMS message sending logic...
            //
        }
        // Implementing event handler for campaign validation.
        public void OnValidate() {
            try {
                // TODO: SMS gateway connection validation logic...
                //
            } catch (Exception ex) {
                // If errors are found, add information to the campaign schema.
                CampaignSchema.AddValidationInfo(ex.Message);
            }
        }
    }
}
```

After making the changes, publish the schema. Compile the application and clear the cache.

As a result, a new [*TestSMS*] element will be added in the campaign element menu (1) that the users can add to the campaign diagram (2). When an added element is selected, its edit page will be displayed (3).



Attention. When saving the campaign, the “Parameter ‘type’ cannot be null” may occur. The error indicates that the configuration library was not updated after the compilation and therefor does not contain the new types.

Recompile the project and clear all possible storages with cached data. You may also need to clear the

application pool and restart the website in IIS on the application server.

Configure campaign elements for working with triggers

Advanced

Starting from version 7.12.4, a new [*Triggered adding*] campaign element has been added to Creatio. The way the element works with campaign audience has been changed: as soon as the trigger is run, the synchronous campaign element is launched.

Thus, new and existing campaign elements must be configured for working with this element.

To include the custom element into the synchronous fragment and make sure it works correctly both for the scheduled execution and triggers, make the following changes on the server side:

- Specify the additional `CampaignSchemaElementType.Sessioned` type for the element.
- The executed element for the campaign custom element (the inheritor of the `CampaignProcessFlowElement` class) must work using the `CampaignAudience` base property. Perform all operations (specifying the audience, selecting the [*Step complete*] checkbox) via the object in the `CampaignAudience` property of the `ICampaignAudience` type.

Example. Configure the marketing campaign element for sending SMS messages to users. Learn how to create this element in the "Add a custom campaign element" article.

Example implementation algorithm

1. Modifying the class that interacts with the server side of the application

Attention. Perform steps 1-6 from the "Add a custom campaign element" article before you implement the case.

Change the `TestSmsElement` source code schema class (the inheritor of the `CampaignSchemaElement` class). Specify additional `ElementType` — `CampaignSchemaElementType.Sessioned` in the class constructor.

TestSmsElement

```
public TestSmsElement() {
    // TestSmsElement is asynchronous and session element that can be triggered.
    ElementType = CampaignSchemaElementType.AsyncTask | CampaignSchemaElementType.Sessioned;
}
```

2. Modifying the executed element for the new campaign element

Modify the `TestSmsCampaignProcessElement` source code schema class (the inheritor of the `CampaignProcessFlowElement` class). Add the audience reading operation to the `SafeExecute()` method implementation via the object in the `CampaignAudience` property of the `ICampaignAudience` type.

TestSmsCampaignProcessElement

```
// Method implementation of the element performance.
    protected override int SafeExecute(ProcessExecutingContext context) {
        // TODO: Implement sending SMS-messages.
        //
        //
        // Receive the audience that is available for processing by the element at the moment..
        var audienceSelect = CampaignAudience.GetItemAudienceSelect(CampaignItemId);
        //
        // Specify the current audience step as [Complete].
        return CampaignAudience.SetItemCompleted(SchemaElementUId);
    }
```

Add a custom flow to a new campaign element

Advanced

Use [*Campaign designer*] to set up your marketing campaigns. You can create a visual campaign diagram that would consist of interconnected pre-configured elements. You can also add custom campaign elements.

Algorithm of adding a custom flow (an arrow)

1. Create a new schema for the [*Flow*] element.
2. Create the edit page for the [*Flow*] element properties.
3. Create the server part for the [*Flow*] element.
4. Create the executed element for the flow transition.
5. Create the `CampaignConnectorManager` replacing module for adding the flow operation logic.
6. Connect the `CampaignConnectorManager` replacing module.

Example. Create a custom flow (an arrow) from the new campaign element for sending SMS-messages to a user and add a possibility to select the bulk sms response condition. On the setup page, a user can select an option to either ignore the responses or take them into consideration based on the list of possible bulk sms response types. If no response type is selected, the flow is enabled for any response.

Attention. In this case, the `Usr` prefix is not used in schema names. You can change the prefix used by default in the [*Prefix for object name*] system setting (the `SchemaNamePrefix` code).

Attention. Perform steps 1-6 from the "Add a custom campaign element" article before you implement the case.

Example implementation algorithm

1. Creating the [*TestSmsTarget*] and [*TestSmsResponseType*] objects

Create schemas for the [*TestSmsTarget*] and [*TestSmsResponseType*] objects in the development package.

Learn more about creating object schemas in the "Create the entity schema" article.

Add a column with the following properties to the [*TestSmsResponseType*] schema:

- [*Title*] – "Name"
- [*Name*] – "Name"
- [*Data type*] – "Text (50 characters)"

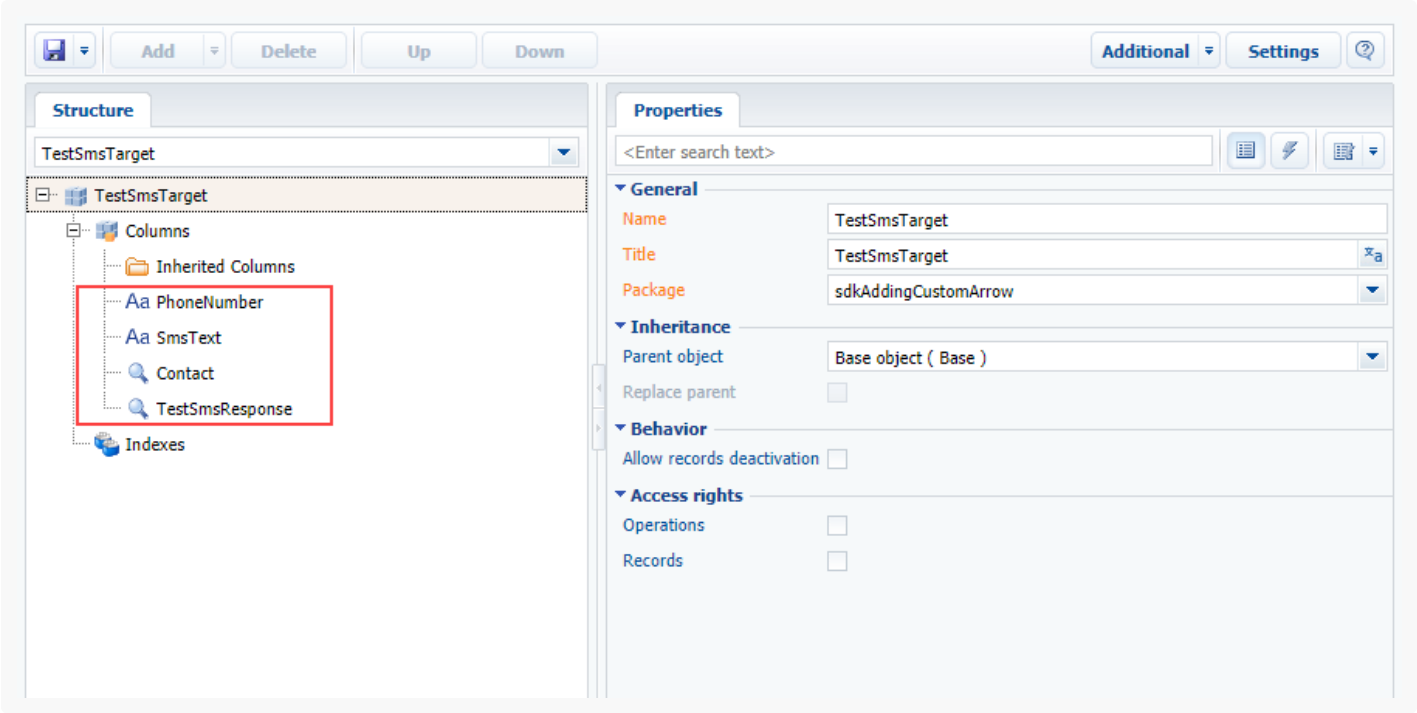
Add columns with the following properties to the [*TestSmsTarget*] schema:

Primary column properties of the [*TestSmsTarget*] object schema

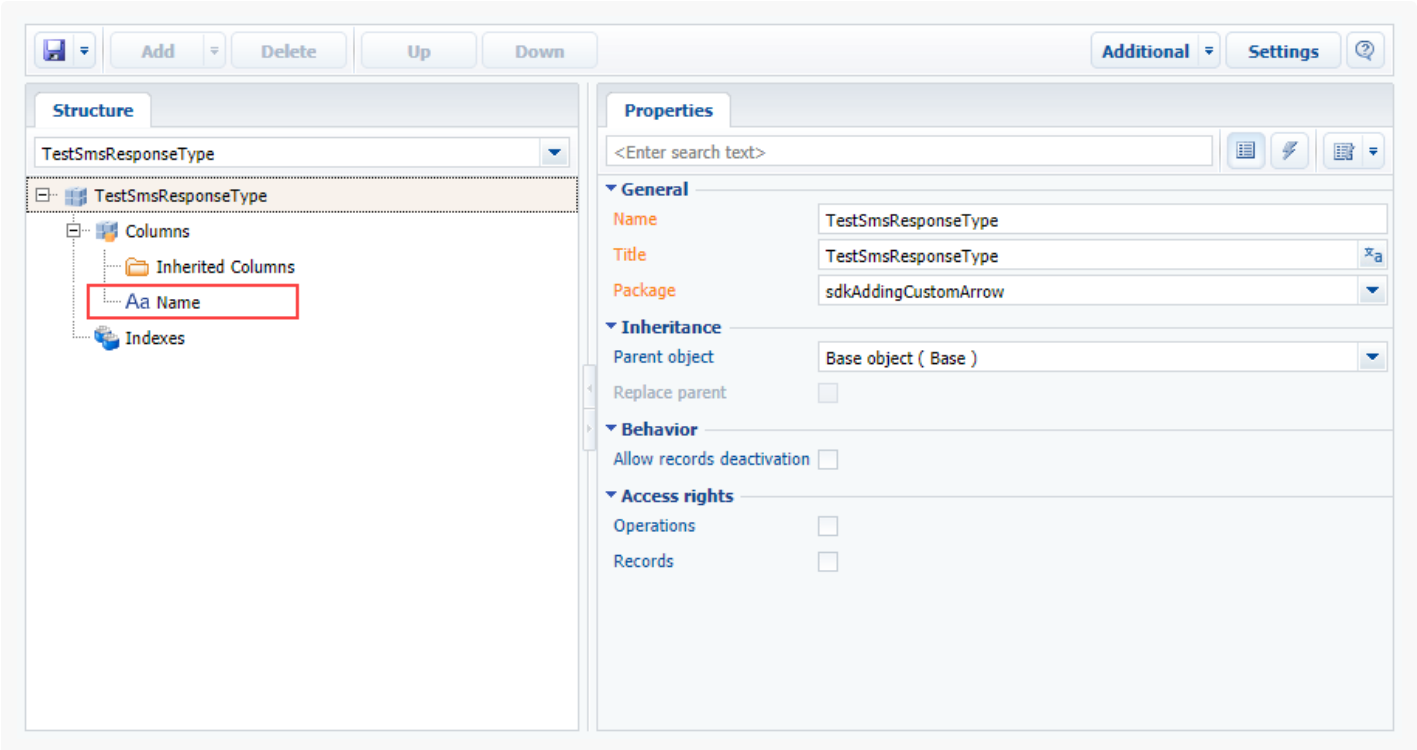
[*Title*]	**[*Name*]**	**[*Data type*]**
Phone number	PhoneNumber	"Text (50 characters)"
SMS text	SmsText	"Text (50 characters)"
Contact	Contact	"Lookup" – "Contact"
Test SMS response	TestSmsResponse	"Lookup" – "TestSmsResponseType"

Learn more about adding object schema columns in the "Create the entity schema" article.

Columns and properties of the [*TestSmsTarget*] object schema

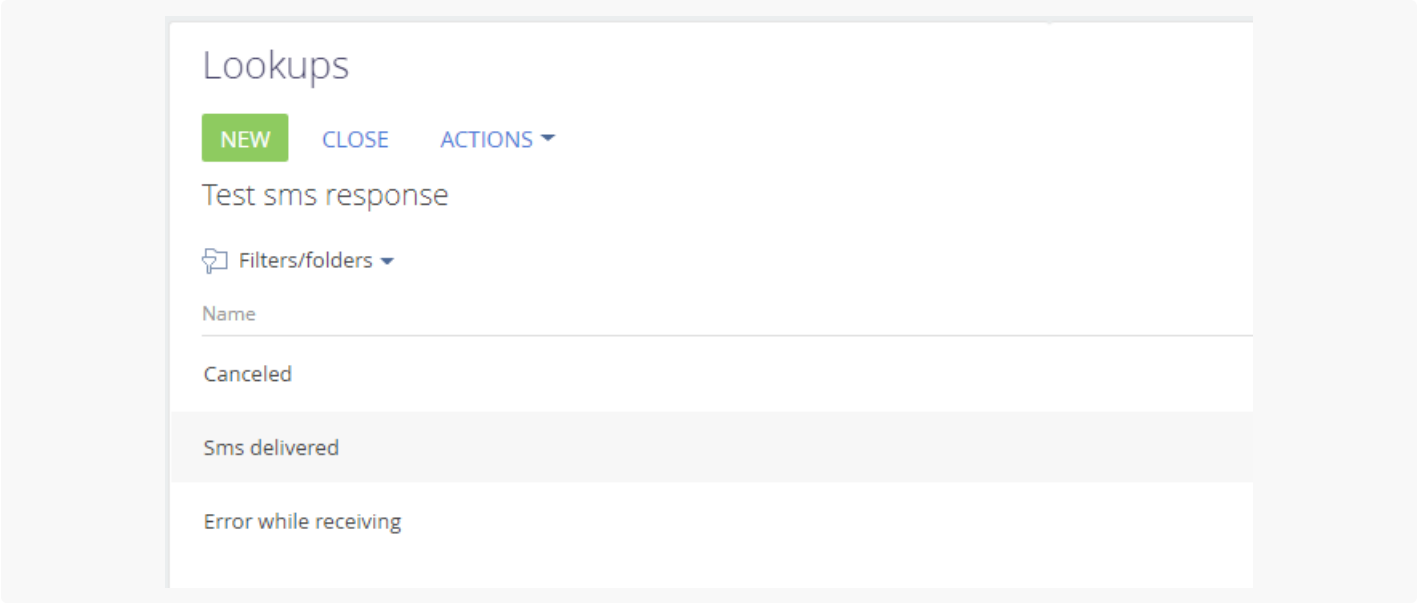


Columns and properties of the [*TestSmsResponseType*] object schema



Save and publish the objects.

Create a lookup for the TestSmsResponseType object and populate it with the “Sms delivered”, “Canceled”, “Error while receiving”, etc. values. The response identifiers (Ids) will be used in the edit page code of the condition flow properties from the bulk SMS.

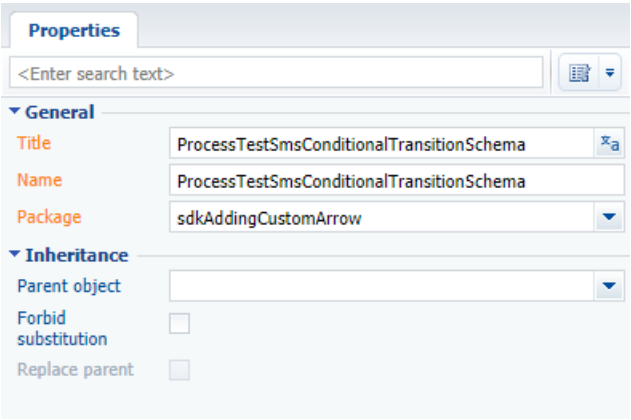


2. Creating a new schema for the flow element

To display the element in the [*Campaign designer*] user interface, create a new module schema in the development package. The procedure for creating a module schema is covered in the "Create a client schema" article. Set the following properties for the created schema:

- [*Title*] – "ProcessTestSmsConditionalTransitionSchema".
- [*Name*] – "ProcessTestSmsConditionalTransitionSchema".

Attention. The schema name for the arrow must contain the "Process" prefix



Add the following source code to the [*Source code*] section of the schema:

ProcessTestSmsConditionalTransitionSchema

```
define("ProcessTestSmsConditionalTransitionSchema", ["CampaignEnums",
    "ProcessTestSmsConditionalTransitionSchemaResources",
    "ProcessCampaignConditionalSequenceFlowSchema"],
    function(CampaignEnums) {
```

```
        Ext.define("Terrasoft.manager.ProcessTestSmsConditionalTransitionSchema", {
            extend: "Terrasoft.ProcessCampaignConditionalSequenceFlowSchema",
            alternateClassName: "Terrasoft.ProcessTestSmsConditionalTransitionSchema",
            managerItemUId: "4b5e70b0-a631-458e-ab22-856ddc913444",
            mixins: {
                parametrizedProcessSchemaElement: "Terrasoft.ParametrizedProcessSchemaElement"
            },
            // The full type name of the connected arrow element.
            typeName: "Terrasoft.Configuration.TestSmsConditionalTransitionElement, Terrasoft.Co
            // The name of the arrow element for connecting to campaign elements.
            connectionUserHandleName: "TestSmsConditionalTransition",
            // The name of the arrow property setup page.
            editPageSchemaName: "TestSmsConditionalTransitionPropertiesPage",
            elementType: CampaignEnums.CampaignSchemaElementTypes.CONDITIONAL_TRANSITION,
            // Bulk sms response collection.
            testSmsResponseId: null,
            // The checkbox that takes into consideration the response condition for contact tra
            isResponseBasedStart: false,
            getSerializableProperties: function() {
                var baseSerializableProperties = this.callParent(arguments);
                // Properties for serialization and transfer to the server part when saving.
                Ext.Array.push(baseSerializableProperties, ["testSmsResponseId", "isResponseBase
                return baseSerializableProperties;
            }
        });
        return Terrasoft.ProcessTestSmsConditionalTransitionSchema;
    });
```

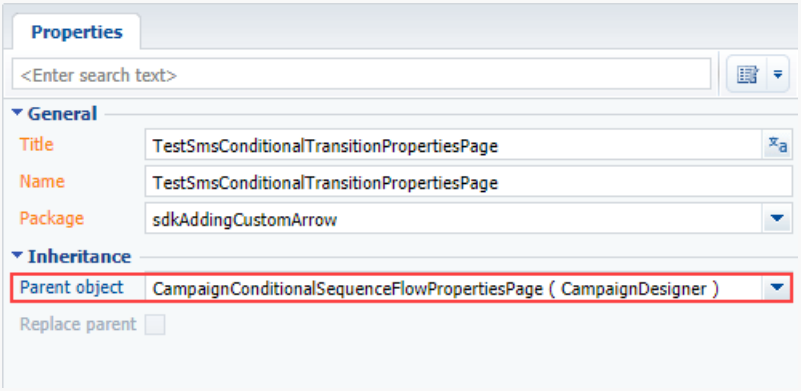
Save the created schema.

3. Creating the edit page of the flow element properties

To display and modify campaign element properties, create its edit page in the development package. Create the schema replacing `CampaignConditionalSequenceFlowPropertiesPage` (the `CampaignDesigner` package). The procedure for creating a replacing schema is covered in the “Create a client schema” article.

Set the following properties for the created schema:

- [*Title*] – “TestSmsConditionalTransitionPropertiesPage”.
- [*Name*] – “TestSmsConditionalTransitionPropertiesPage”.
- [*Parent object*] – “CampaignConditionalSequenceFlowPropertiesPage”.



Add localizable strings, whose properties are listed in table to the created schema.

Primary properties of the localizable strings

[***Name***]	[***Value***]
`ReactionModeCaption`	What is the result of the {0} step?
`ReactionModeDefault`	Transfer participants regardless of their response
`ReactionModeWithCondition`	Set up responses for transferring participants
`IsTestSmsDelivered`	Test SMS delivered
`IsErrorWhileReceiving`	Error while receiving

Add the following source code to the [*Source code*] section of the schema:

TestSmsConditionalTransitionPropertiesPage

```
define("TestSmsConditionalTransitionPropertiesPage", ["BusinessRuleModule"],
    function(BusinessRuleModule) {
        return {
            messages: {},
            attributes: {
                "ReactionModeEnum": {
                    dataValueType: this.Terrasoft.DataValueType.CUSTOM_OBJECT,
                    type: this.Terrasoft.ViewModelColumnType.VIRTUAL_COLUMN,
                    value: {
                        Default: {
                            value: "0",
                            captionName: "Resources.Strings.ReactionModeDefault"
                        },
                        WithCondition: {
                            value: "1",
```

```
                        captionName: "Resources.Strings.ReactionModeWithCondition"
                    }
                }
            },
            "ReactionMode": {
                "dataValueType": this.Terrasoft.DataValueType.LOOKUP,
                "type": this.Terrasoft.ViewModelColumnType.VIRTUAL_COLUMN,
                "isRequired": true
            },
            "IsTestSmsDelivered": {
                "dataValueType": this.Terrasoft.DataValueType.BOOLEAN,
                "type": this.Terrasoft.ViewModelColumnType.VIRTUAL_COLUMN
            },
            "IsErrorWhileReceiving": {
                "dataValueType": this.Terrasoft.DataValueType.BOOLEAN,
                "type": this.Terrasoft.ViewModelColumnType.VIRTUAL_COLUMN
            }
        },

        rules:
        {
            "ReactionConditionDecision": {
                "BindReactionConditionDecisionRequiredToReactionMode": {
                    "ruleType": BusinessRuleModule.enums.RuleType.BINDPARAMETER,
                    "property": BusinessRuleModule.enums.Property.REQUIRED,
                    "conditions": [{
                        "leftExpression": {
                            "type": BusinessRuleModule.enums.ValueType.ATTRIBUTE,
                            "attribute": "ReactionMode"
                        },
                        "comparisonType": this.Terrasoft.ComparisonType.EQUAL,
                        "rightExpression": {
                            "type": BusinessRuleModule.enums.ValueType.CONSTANT,
                            "value": "1"
                        }
                    }]
                }
            }
        },
        methods: {
            // Ensures sms-response correspondence (based on the TestSmsResponseType lookup)
            // We assume that Creatio already contains the TestSmsResponseType lookup
            // with the TestSmsDelivered and ErrorWhileReceiving records.
            getResponseConfig: function() {
                return {
                    "IsTestSmsDelivered": "F2FC75B3-58C3-49A6-B2F2-353262068145",
                    "IsErrorWhileReceiving": "37B9F9D5-E897-4B7B-A65E-3B3799A18D72"
                };
            },
```

```
subscribeEvents: function() {
    this.callParent(arguments);
    // Connecting the handler to the event of changing the ReactionMode attribut
    this.on("change:ReactionMode", this.onReactionModeLookupChanged, this);
},

// Event handler-method of changing the ReactionMode attribute.
onReactionModeLookupChanged: function() {
    var reactionModeEnum = this.get("ReactionModeEnum");
    var reactionMode = this.get("ReactionMode");
    var decisionModeEnabled = (reactionMode && reactionMode.value === reactionMo
    if (!decisionModeEnabled) {
        this.set("ReactionConditionDecision", null);
    }
},

// Initiates the viewModel properties to display the page when opening.
initParameters: function(element) {
    this.callParent(arguments);
    var isResponseBasedStart = element.isResponseBasedStart;
    this.initReactionMode(isResponseBasedStart);
    this.initTestSmsResponses(element.testSmsResponseId);
},

// Auxiliary method cutting the line to the specified length and adding allipsis
cutString: function(strValue, strLength) {
    var ellipsis = Ext.String.ellipsis(strValue.substring(strLength), 0);
    return strValue.substring(0, strLength) + ellipsis;
},

// Sets the status value to "Sms delivered".
initIsTestSmsDelivered: function(value) {
    if (value === undefined) {
        value = this.get("IsTestSmsDelivered");
    }
    this.set("IsTestSmsDelivered", value);
},

// Sets the status value to "Error while receiving".
initIsErrorWhileReceiving: function(value) {
    if (value === undefined) {
        var isErrorWhileReceiving = this.get("IsErrorWhileReceiving");
        value = isErrorWhileReceiving;
    }
    this.set("IsErrorWhileReceiving", value);
},
```

```
// Initiates the selected responses when opening the page.
initTestSmsResponses: function(responseIdsJson) {
    if (!responseIdsJson) {
        return;
    }
    var responseIds = JSON.parse(responseIdsJson);
    var config = this.getResponseConfig();
    Terrasoft.each(config, function(propValue, propName) {
        if (responseIds.indexOf(propValue) > -1) {
            this.set(propName, true);
        }
    }, this);
},

initReactionMode: function(value) {
    var isDefault = !value;
    this.setLookupValue(isDefault, "ReactionMode", "WithCondition", this);
},

// Auxiliary method extracting the identifier array from the incoming JSON param
getIds: function(idsJson) {
    if (idsJson) {
        try {
            var ids = JSON.parse(idsJson);
            if (this.Ext.isArray(ids)) {
                return ids;
            }
        } catch (error) {
            return [];
        }
    }
    return [];
},

onPrepareReactionModeList: function(filter, list) {
    this.prepareList("ReactionModeEnum", list, this);
},

// Saves the response values and the setting of adding the response condition.
saveValues: function() {
    this.callParent(arguments);
    var element = this.get("ProcessElement");
    var isResponseBasedStart = this.getIsReactionModeWithConditions();
    element.isResponseBasedStart = isResponseBasedStart;
    element.testSmsResponseId = this.getTestSmsResponseId(isResponseBasedStart);
},

// Receives the serialized Ids of the selected responses.
getTestSmsResponseId: function(isResponseActive) {
```

```
                var responseIds = [];
                if (isResponseActive) {
                    var config = this.getResponseConfig();
                    Terrasoft.each(config, function(propValue, propName) {
                        var attrValue = this.get(propName);
                        if (attrValue && propValue) {
                            responseIds.push(propValue);
                        }
                    }, this);
                }
                return JSON.stringify(responseIds);
            },

            getLookupValue: function(parameterName) {
                var value = this.get(parameterName);
                return value ? value.value : null;
            },

            getContextHelpCode: function() {
                return "CampaignConditionalSequenceFlow";
            },

            getIsReactionModeWithConditions: function() {
                return this.isLookupValueEqual("ReactionMode", "1", this);
            },

            getSourceElement: function() {
                var flowElement = this.get("ProcessElement");
                if (flowElement) {
                    return flowElement.findSourceElement();
                }
                return null;
            },
            // Adds the name of the element that the arrow is generated from to the text.
            getQuestionCaption: function() {
                var caption = this.get("Resources.Strings.ReactionModeCaption");
                caption = this.Ext.String.format(caption, this.getSourceElement().getCaptior
                return caption;
            }
        },
        diff: /**SCHEMA_DIFF*/[
            // Container.
            {
                "operation": "insert",
                "name": "ReactionContainer",
                "propertyName": "items",
                "parentName": "ContentContainer",
                "className": "Terrasoft.GridLayoutEdit",
```

```
                    "values":
                    {
                        "layout":
                        {
                            "column": 0,
                            "row": 2,
                            "colSpan": 24
                        },
                        "itemType": this.Terrasoft.ViewItemType.GRID_LAYOUT,
                        "items": []
                    }
                },
                // Title.
                {
                    "operation": "insert",
                    "name": "ReactionModeLabel",
                    "parentName": "ReactionContainer",
                    "propertyName": "items",
                    "values":
                    {
                        "layout":
                        {
                            "column": 0,
                            "row": 0,
                            "colSpan": 24
                        },
                        "itemType": this.Terrasoft.ViewItemType.LABEL,
                        "caption":
                        {
                            "bindTo": "getQuestionCaption"
                        },
                        "classes":
                        {
                            "labelClass": ["t-title-label-proc"]
                        }
                    }
                },
                // List.
                {
                    "operation": "insert",
                    "name": "ReactionMode",
                    "parentName": "ReactionContainer",
                    "propertyName": "items",
                    "values":
                    {
                        "contentType": this.Terrasoft.ContentType.ENUM,
                        "controlConfig":
                        {
                            "prepareList":
```

```
                    {
                        "bindTo": "onPrepareReactionModeList"
                    }
                },
                "isRequired": true,
                "layout":
                {
                    "column": 0,
                    "row": 1,
                    "colSpan": 24
                },
                "labelConfig":
                {
                    "visible": false
                },
                "wrapClass": ["no-caption-control"]
            }
        },
        // List element.
        {
            "operation": "insert",
            "parentName": "ReactionContainer",
            "propertyName": "items",
            "name": "IsTestSmsDelivered",
            "values":
            {
                "wrapClass": ["t-checkbox-control"],
                "visible":
                {
                    "bindTo": "ReactionMode",
                    "bindConfig":
                    {
                        converter: "getIsReactionModeWithConditions"
                    }
                },
                "caption":
                {
                    "bindTo": "Resources.Strings.IsTestSmsDelivered"
                },
                "layout":
                {
                    "column": 0,
                    "row": 2,
                    "colSpan": 22
                }
            }
        },
        // List element.
```

```
                {
                    "operation": "insert",
                    "parentName": "ReactionContainer",
                    "propertyName": "items",
                    "name": "IsErrorWhileReceiving",
                    "values":
                    {
                        "wrapClass": ["t-checkbox-control"],
                        "visible":
                        {
                            "bindTo": "ReactionMode",
                            "bindConfig":
                            {
                                converter: "getIsReactionModeWithConditions"
                            }
                        },
                        "caption":
                        {
                            "bindTo": "Resources.Strings.IsErrorWhileReceiving"
                        },
                        "layout":
                        {
                            "column": 0,
                            "row": 3,
                            "colSpan": 22
                        }
                    }
                }
            ]/**SCHEMA_DIFF*/
        };
    }
);
```

Save the created schema.

4. Creating the server part of the flow element from the [*Bulk SMS*] element

To implement saving the base and user campaign element properties, create a class interacting with the server part of the application. The class should inherit `CampaignSchemaElement` and override the `ApplyMetaDataValue()` and `WriteMetaData()` methods.

Create the source code schema with the following properties:

- [*Title*] – “TestSmsConditionalTransitionElement”;
- [*Name*] – “TestSmsConditionalTransitionElement”.

Creating the source code schema is covered in the “Create the [*Source code*] schema” article.

Add the following source code to the [*Source code*] section of the schema:

TestSmsConditionalTransitionElement

```
namespace Terrasoft.Configuration
{
    using System;
    using System.Collections.Generic;
    using System.Collections.ObjectModel;
    using System.Globalization;
    using System.Linq;
    using Newtonsoft.Json;
    using Terrasoft.Common;
    using Terrasoft.Core;
    using Terrasoft.Core.Campaign;
    using Terrasoft.Core.DB;
    using Terrasoft.Core.Process;

    [DesignModeProperty(Name = "TestSmsResponseId",
        UsageType = DesignModeUsageType.NotVisible, MetaPropertyName = TestSmsResponseIdProperty
    [DesignModeProperty(Name = "IsResponseBasedStart",
        UsageType = DesignModeUsageType.Advanced, MetaPropertyName = IsResponseBasedStartPropert
    public class TestSmsConditionalTransitionElement : ConditionalSequenceFlowElement
    {

        private const string TestSmsResponseIdPropertyName = "TestSmsResponseId";
        private const string IsResponseBasedStartPropertyName = "IsResponseBasedStart";

        public TestSmsConditionalTransitionElement() {}

        public TestSmsConditionalTransitionElement(TestSmsConditionalTransitionElement source)
            : this(source, null, null) {}

        public TestSmsConditionalTransitionElement(TestSmsConditionalTransitionElement source,
            Dictionary<Guid, Guid> dictToRebind, Core.Campaign.CampaignSchema parentSchema)
            : base(source, dictToRebind, parentSchema) {
            IsResponseBasedStart = source.IsResponseBasedStart;
            _testSmsResponseIdJson = JsonConvert.SerializeObject(source.TestSmsResponseId);
        }

        private string _testSmsResponseIdJson;

        private IEnumerable<Guid> Responses {
            get {
                return TestSmsResponseId;
            }
        }
```

```
        [MetaTypeProperty("{DC597899-B831-458A-A58E-FB43B1E266AC}")]
        public IEnumerable<Guid> TestSmsResponseId {
            get {
                return !string.IsNullOrWhiteSpace(_testSmsResponseIdJson)
                    ? JsonConvert.DeserializeObject<IEnumerable<Guid>>(_testSmsResponseIdJson)
                    : Enumerable.Empty<Guid>();
            }
        }

        [MetaTypeProperty("{3FFA4EA0-62CC-49A8-91FF-4096AEC561F6}",
    IsExtraProperty = true, IsUserProperty = true)]
        public virtual bool IsResponseBasedStart {
            get;
            set;
        }

        protected override void ApplyMetaDataValue(DataReader reader) {
            base.ApplyMetaDataValue(reader);
            switch (reader.CurrentName) {
                case TestSmsResponseIdPropertyName:
                    _testSmsResponseIdJson = reader.GetValue<string>();
                    break;
                case IsResponseBasedStartPropertyName:
                    IsResponseBasedStart = reader.GetBoolValue();
                    break;
                default:
                    break;
            }
        }

        public override void WriteMetaData(DataWriter writer) {
            base.WriteMetaData(writer);
            writer.WriteValue(IsResponseBasedStartPropertyName, IsResponseBasedStart, false);
            writer.WriteValue(TestSmsResponseIdPropertyName, _testSmsResponseIdJson, null);
        }

        public override object Clone() {
            return new TestSmsConditionalTransitionElement(this);
        }

        public override object Copy(Dictionary<Guid, Guid> dictToRebind, Core.Campaign.CampaignS
            return new TestSmsConditionalTransitionElement(this, dictToRebind, parentSchema);
        }

        // Overrides the factory method for creating the executed element
        // Returns the element with the  TestSmsConditionalTransitionFlowElement type
        public override ProcessFlowElement CreateProcessFlowElement(UserConnection userConnectic
            var sourceElement = SourceRef as TestSmsElement;
            var executableElement = new TestSmsConditionalTransitionFlowElement {
```

```
                UserConnection = userConnection,
                TestSmsResponses = TestSmsResponseId,
                PhoneNumber = sourceElement.PhoneNumber,
                SmsText = sourceElement.SmsText
            };
            InitializeCampaignProcessFlowElement(executableElement);
            InitializeCampaignTransitionFlowElement(executableElement);
            InitializeConditionalTransitionFlowElement(executableElement);
            return executableElement;
        }
    }
}
```

Save and publish the created schema.

5. Creating the executed element for the flow from the [*Bulk SMS*] element

To add a functionality that would take into consideration the response type as per the sent bulk sms, create the executed element. It is a class, the inheritor of the `ConditionalTransitionFlowElement` class.

To create the executed element, add the source code schema with the following properties in the development package:

- [*Title*] – "TestSmsConditionalTransitionElement".
- [*Name*] – "TestSmsConditionalTransitionElement".

Add the following source code to the [*Source code*] section of the schema:

TestSmsConditionalTransitionFlowElement

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

    public class TestSmsConditionalTransitionFlowElement : ConditionalTransitionFlowElement
    {

        public string SmsText { get; set; }

        public string PhoneNumber { get; set; }

        public IEnumerable<Guid> TestSmsResponses { get; set; }
```

```
        private void ExtendWithResponses() {
            TransitionQuery.CheckArgumentNull("TransitionQuery");
            if (TestSmsResponses.Any()) {
                Query responseSelect = GetSelectByParticipantResponses();
                TransitionQuery.And("ContactId").In(responseSelect);
            }
        }

        private Query GetSelectByParticipantResponses() {
            var responseSelect = new Select(UserConnection)
                .Column("ContactId")
                .From("TestSmsTarget")
                .Where("SmsText").IsEqual(Column.Parameter(SmsText))
                    .And("PhoneNumber").IsEqual(Column.Parameter(PhoneNumber))
                    .And("TestSmsResponseId")
                        .In(Column.Parameters(TestSmsResponses)) as Select;
            responseSelect.SpecifyNoLockHints(true);
            return responseSelect;
        }

        protected override void CreateQuery() {
            base.CreateQuery();
            ExtendWithResponses();
        }
    }
}
```

Save and publish the created schema.

6. Creating the CampaignConnectorManager replacing module for adding the flow operation logic

To add specific logic of the flow operation upon changing the source (i.e. the element that the outgoing arrow is generated from), create a new module schema replacing the `CampaignConnectorManager` module in the development package. The procedure for creating a module schema is covered in the "Create a client schema" article. Set the following properties for the created schema:

- [*Title*] – "TestSmsCampaignConnectorManager".
- [*Name*] – "TestSmsCampaignConnectorManager".

Add the following source code to the [*Source code*] section of the schema:

TestSmsCampaignConnectorManager

```
define("TestSmsCampaignConnectorManager", [], function() {

    Ext.define("Terrasoft.TestSmsCampaignConnectorManager", {
```

```
        // Specify replacing of the CampaignConnectorManager module
        override: "Terrasoft.CampaignConnectorManager",

        // Add mapping of the name of arrow source campaign element – arrow type (full name)
        initMappingCollection: function() {
            this.callParent(arguments);
            this.connectorTypesMappingCollection.addIfNotExists("TestSmsElementSchema",
                "Terrasoft.ProcessTestSmsConditionalTransitionSchema");
        },

        // Virtual method for reload
        // Arrow processing logic before its sudstitute by an arrow with a new type.
        additionalBeforeChange: function(prevTransition, sourceItem, targetItem) {
            // additional logic here
        },

        // Virtual method for reload
        // Populating specific fields of the created arrow based on the previous arrow.
        fillAdditionalProperties: function(prevElement, newElement) {
            if (newElement.getTypeInfo().typeName === "ProcessTestSmsConditionalTransitionSchema
                // Copy the configured responses if the previous arrow is of the same type
                newElement.testSmsResponseId = prevElement.testSmsResponseId ? prevElement.testS
                // Copy the configuration of response setup
                newElement.isResponseBasedStart = prevElement.isResponseBasedStart
                    ? prevElement.isResponseBasedStart
                    : false;
            }
        }
    });
});
```

Save the created schema.

7. Connecting the `CampaignConnectorManager` replacing module

To connect the module created on the previous step, create a replacing client module and specify `BootstrapModulesV2` from the `NUI` package as the parent object. The procedure of creating a replacing client module is covered in the “Create a client schema“ article.

Add the following source code to the [*Source code*] section of the schema:

BootstrapModulesV2

```
// Set the previously created TestSmsCampaignConnectorManager module as a dependency
define("BootstrapModulesV2", ["TestSmsCampaignConnectorManager"], function() {});
```

Save the created schema.

Note. During an actual implementation of this case, we recommend creating a separate [*Bulk SMS*] object schema. The [*TestSmsElement*] and [*TestSmsConditionalTransitionElement*] objects will contain the [*Id*] of this object and not the `SmsText`, `PhoneNumber` ... fields. The `TestSmsCampaignProcessElement` executed element in the `Execute()` method must contain the logic of adding contacts to the bulk sms audience. A separate mechanism (or several mechanisms) must perform sending of the bulk sms and afterwards record the participants' responses. Based on these responses, the arrow will transfer the campaign audience to the following campaign step.

Lead

Advanced

Create custom reminders and notifications

Starting with version 7.12.0, reminder and notification sending mechanics has been reworked in Creatio.

Previously, to send a custom notification, you would have to:

- Creates a class that implements `INotificationProvider` interface or an inherited abstract `BaseNotificationProvider` class.
- Add logic for selecting custom notifications by Creatio.
- Register a class in the `NotificationProvider` table.

The notifications were sent once a minute, calling all classes from the `NotificationProvider` table.

Starting with version 7.12.0, it is sufficient to create a notification or reminder with the needed parameters. After this, the application will either send the notification immediately, or display a reminder at the specified time.

To set up custom notifications:

1. Create a [*Source code*] schema in the custom package and define a class for generating the notification text and pop-up window. The class must implement the `IRemindingTextFormer` interface (declared in the `IRemindingTextFormer` schema of the `Base` package).

2. Replace the needed object (such as [*Lead*]) or specify notification sending logic in it.

3. Replace the reminder tab schema `ReminderNotificationsSchema` for displaying notifications for the needed object.