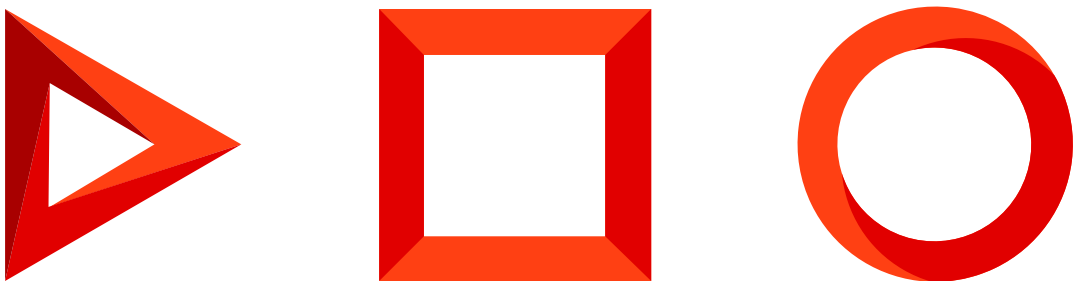
Creatio Academy

Компоненты Marketing

Общие принципы работы с маркетинговыми кампаниями

Версия 8.0



Эта документация предоставляется с ограничениями на использование и защищена законами об интеллектуальной собственности. За исключением случаев, прямо разрешенных в вашем лицензионном соглашении или разрешенных законом, вы не можете использовать, копировать, воспроизводить, переводить, транслировать, изменять, лицензировать, передавать, распространять, демонстрировать, выполнять, публиковать или отображать любую часть в любой форме или посредством любые значения. Обратный инжиниринг, дизассемблирование или декомпиляция этой документации, если это не требуется по закону для взаимодействия, запрещены.

Информация, содержащаяся в данном документе, может быть изменена без предварительного уведомления и не может гарантировать отсутствие ошибок. Если вы обнаружите какие-либо ошибки, сообщите нам о них в письменной форме.

Содержание

Общие принципы работы с маркетинговыми кампаниями

Сложный

Схемы маркетинговых кампаний в Marketing Creatio моделируются при помощи визуального дизайнера во время создания новой кампании. Схема кампании состоит из элементов и условий переходов между ними.

При запуске кампании создается так называемая flow-схема выполнения кампании. Элементы преобразуются в цепочку выполнения кампании и для каждого элемента рассчитывается время запуска. При этом flow-схема может значительно отличаться от визуального представления кампании в дизайнере.

Элементы кампании могут быть синхронными и асинхронными.

Синхронные – элементы, которые выполняются друг за другом в порядке, установленном flow-схемой. Переход к следующим элементам выполняется только после отработки такого элемента. При этом поток выполнения блокируется и ожидает завершения операции.

Асинхронные – элементы, выполнение которых требует ожидания завершения работы некоторых внешних систем, ресурсов, асинхронных сервисов, реакции пользователей или внешних систем (например, переход по ссылке из Email-рассылки).

Тип элементов определяет их положение во flow-схеме выполнения кампании. Первыми выполняются элементы [*Добавление из группы*] и [*Выход из группы*]. Они определяют наполнение аудитории на текущем шаге. По стрелкам аудитория кампании переходит от выполненного элемента к следующему за ним. Если для перехода заданы условия, то с их учетом выполняется фильтрация аудитории и определяется время выполнения следующего элемента.

Механизм планирования следующего запуска кампании

Планирование следующего запуска кампании выполняется по следующему алгоритму:

1. Расчет времени следующего запуска элемента выполняется в зависимости от выбранного варианта перехода по времени:

   - Выбран вариант "`В течение дня`". Дата и время следующего выполнения данного элемента вычисляется по формуле:

     `Дата и время выполнения = текущие дата и время + N минут/часов`,

     где `N` — значение поля [*Количество*], указанное пользователем.

   - Выбран вариант "`Через несколько дней`". Следующее выполнение данного элемента планируется по формулам:

     `Дата = текущая дата+N дней,`

     где `N` - значение поля [*Количество*], указанное пользователем.

     `Время выполнения = указанное пользователем время.`

- Выбран вариант "Без задержки". Следующее выполнение данного элемента планируется на время ближайшего запуска кампании.

2. По варианту, описанному в п.1, рассчитывается время запуска для каждого элемента схемы кампании.
3. После сравнения всех значений выбирается ближайшее время запуска и устанавливается как время следующего запуска кампании.
4. Формирование списка элементов, которые будут выполнены при следующем запуске. В список попадают все элементы, время запуска которых равно времени следующего запуска кампании (см. п. 2, 3).

Основные классы элементов кампании

JavaScript-классы

Базовым классом схемы элемента является класс `ProcessFlowElementSchema`. Класс `CampaignBaseCommunicationSchema` является родительским для всех элементов группы [*Коммуникации*]. Для элементов группы [*Аудитория*] родительским классом является `CampaignBaseAudienceSchema`.

При создании элемента новой группы элементов рекомендуется сначала реализовать базовую схему элемента этой группы, а потом унаследовать каждый элемент от нее.

Каждой схеме соответствует схема страницы свойств элемента. Базовая схема страницы записи – `BaseCampaignSchemaElementPage`. Каждая новая страница элемента расширяет базовую.

Класс `CampaignSchemaManager` управляет схемами элементов, доступных в системе. Он наследует основную функциональность класса `BaseSchemaManager`.

C#-классы

Классы простых элементов

`CampaignSchemaElement` — базовый класс. От него наследуются все другие элементы.

`SequenceFlowElement` — базовый класс для элемента [*Безусловный переход*].

`ConditionSequenceFlowElement` — базовый класс для элемента [*Условный переход*].

`EmailConditionalTransitionElement` — класс элемента перехода по откликам.

`AddCampaignParticipantElement` — класс элемента добавления аудитории (участников кампании).

`ExitFromCampaignElement` — класс элемента выхода из аудитории.

`MarketingEmailElement` — класс элемента Email-рассылки.

Классы исполняемых элементов

`CampaignProcessFlowElement` — базовый класс. От него наследуются все другие исполняемые элементы.

`AddCampaignAudienceElement` — класс элемента аудитории.

`ExcludeCampaignAudienceElement` — класс элемента выхода из аудитории.

`BulkEmailCampaignElement` — класс элемента Email-рассылки.

Добавить пользовательский элемент кампании

 Сложный

Для настройки маркетинговой кампании используется [*Дизайнер кампании*]. С его помощью можно создать визуальную схему кампании, состоящую из связанных предустановленных элементов. Также существует возможность создания пользовательских элементов кампании.

Алгоритм добавления пользовательского элемента кампании

1. Создать новый элемент для [*Дизайнера кампании*].
2. Создать страницу элемента.
3. Расширить меню [*Дизайнера кампании*] новым элементом.
4. Создать серверную часть элемента.
5. Создать исполняемый элемент для нового элемента кампании.
6. Добавить пользовательскую логику для обработки событий кампании.

> **Пример.** Создать новый элемент маркетинговой кампании для отправки СМС сообщений пользователем.

1. Создать новый элемент для [*Дизайнера кампании*]

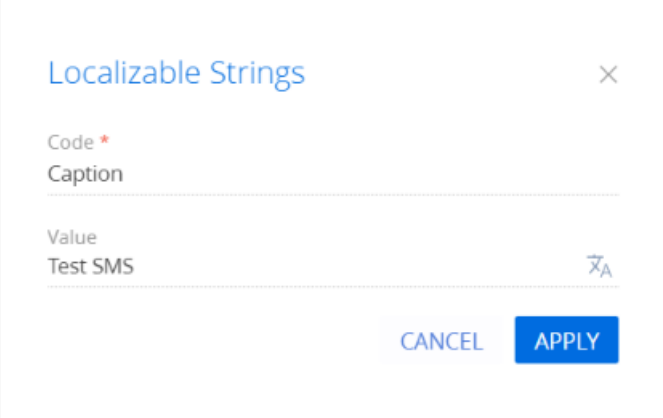
Для отображения элемента в пользовательском интерфейсе [*Дизайнера кампании*] необходимо в пользовательском пакете создать новую схему модуля элемента кампании. Для созданной схемы нужно установить следующие свойства:

- [*Заголовок*] ([*Title*]) — "`Test SMS Element Schema`".
- [*Название*] ([*Name*]) — "`TestSmsElementSchema`".

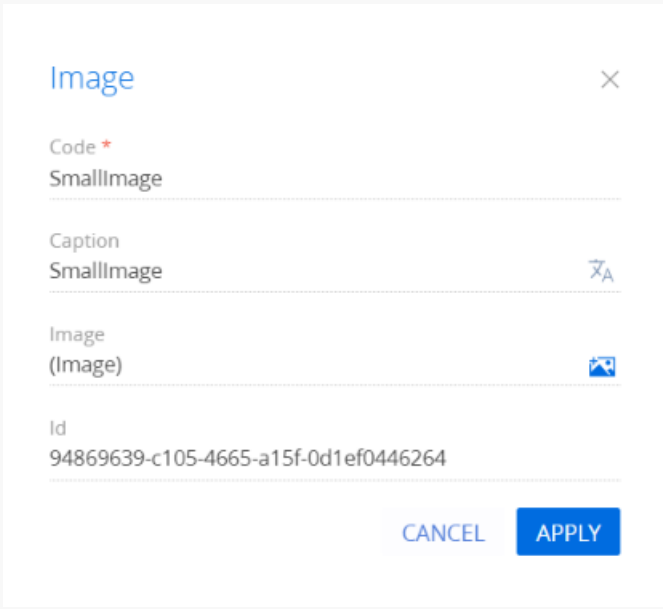
> **Важно.** В этом примере в названиях схем отсутствует префикс `Usr`. Префикс, используемый по умолчанию, можно изменить в системной настройке [*Префикс названия объекта*] (код `SchemaNamePrefix`).

В схему необходимо добавить локализуемую строку со следующими свойствами:

- [*Название*] ([*Name*]) — "`Caption`".
- [*Значение*] ([*Value*]) — "`Test SMS`".



Также в схему необходимо добавить изображения, которые будут отображать элемент кампании в разных режимах в [*Дизайнере кампании*]. Изображения необходимо загрузить в схему, используя свойства `SmallImage`, `LargeImage` и `TitleImage`.



В этом примере используется векторное изображение в формате SVG (Scalable Vector Graphics).

В секцию [*Исходный код*] ([*Source code*]) схемы необходимо добавить исходный код.

`TestSmsElementSchema`

```
define("TestSmsElementSchema", ["TestSmsElementSchemaResources", "CampaignBaseCommunicationSchem
    function(resources) {
        Ext.define("Terrasoft.manager.TestSmsElementSchema", {
            // Родительская схема.
            extend: "Terrasoft.CampaignBaseCommunicationSchema",
            alternateClassName: "Terrasoft.TestSmsElementSchema",
            // Идентификатор менеджера. Должен быть уникальным.
            managerItemUId: "a1226f93-f3e3-4baa-89a6-11f2a9ab2d71",
```

```
            // Подключаемые миксины.
            mixins: {
                campaignElementMixin: "Terrasoft.CampaignElementMixin"
            },
            // Название элемента.
            name: "TestSms",
            // Привязка ресурсов.
            caption: resources.localizableStrings.Caption,
            titleImage: resources.localizableImages.TitleImage,
            largeImage: resources.localizableImages.LargeImage,
            smallImage: resources.localizableImages.SmallImage,
            // Имя схемы карточки записи.
            editPageSchemaName: "TestSmsElementPropertiesPage",
            // Тип элемента.
            elementType: "TestSms",
            // Полное имя класса, соответствующего данной схеме.
            typeName: "Terrasoft.Configuration.TestSmsElement, Terrasoft.Configuration",
            // Переопределение свойств стилей для отображения.
            color: "rgba(249, 160, 27, 1)",
            width: 69,
            height: 55,
            // Настройка специфических свойств элемента.
            smsText: null,
            phoneNumber: null,
            // Определение типов связей, исходящих из элемента.
            getConnectionUserHandles: function() {
                return ["CampaignSequenceFlow", "CampaignConditionalSequenceFlow"];
            },
            // Расширение свойств для сериализации.
            getSerializableProperties: function() {
                var baseSerializableProperties = this.callParent(arguments);
                return Ext.Array.push(baseSerializableProperties, ["smsText", "phoneNumber"]);
            },
            // Настройка отображения иконок на диаграмме кампании.
            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;
    });
```

Особенности:

- Значение свойства `managerItemUId` должно быть уникальным и не должно повторять значение этого свойства у существующих элементов.
- Свойство `typeName` содержит имя C#-класса, соответствующее элементу кампании. Этот класс будет выполнять сохранение и чтение свойств элемента из метаданных схемы.

Методы, которые могут быть переопределены в случае **изменения дефолтного поведения** для элемента:

- `prepareCopy: function() {}` — возвращает копию элемента при копировании (если необходимо сбросить какие-то установленные свойства для элемента). Возвращает экземпляр схемы с необходимыми изменениями.
- `onAfterSave: function() {}` — выполняет специфическую логику для элемента после сохранения. Не возвращает значение.
- `validate: function() {}` — выполняется валидация для элементов (например, проверка наличия входящих стрелок в элемент “Рассылка” и т.п.). Метод возвращает результат валидации в виде коллекции. Если коллекция пуста, значит валидация прошла успешно, иначе в коллекцию добавляется запись с сообщением об ошибке.

После внесения изменений схему необходимо сохранить.

Создание группы элементов

Если для элемента кампании нужно создать новую группу элементов, например, [*Скрипты*] ([*Scripts*]), то исходный код схемы необходимо дополнить кодом.

Создание новой группы элементов

```
// Название новой группы.
group: "Scripts",

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

Также в схему нужно добавить локализуемую строку со следующими свойствами:

- [*Название*] ([*Name*]) — "`ScriptsElementGroupCaption`".
- [*Значение*] ([*Value*]) — "`Scripts`".

После внесения изменений схему необходимо сохранить.

2. Создать страницу элемента

Для отображения и изменения свойств элемента кампании необходимо в пользовательском пакете создать его страницу записи. Для этого нужно создать схему, расширяющую `BaseCampaignSchemaElementPage` (пакет `CampaignDesigner`).

Для созданной схемы требуется установить следующие свойства:

- [*Заголовок*] ([*Title*]) — "`TestSmsElementPropertiesPage`".
- [*Название*] ([*Name*]) — "`TestSmsElementPropertiesPage`".
- [*Родительский объект*] ([*Parent object*]) — "`BaseCampaignSchemaElementPage`".

В созданную схему необходимо добавить локализуемые строки, основные свойства которых приведены в таблице.

Основные свойства локализуемых строк

[*Название*] ([*Name*])	**[*Значение*] ([*Value*])**
PhoneNumberCaption	Телефонный номер отправителя (Sender phone number)
SmsTextCaption	Текст сообщения (Message)
TestSmsText	Какое СМС-сообщение отправить? (Which SMS text to send?)

В секцию [*Исходный код*] ([*Source code*]) схемы необходимо добавить исходный код.

`TestSmsElementPropertiesPage`

```
define("TestSmsElementPropertiesPage", [],
    function() {
        return {
            attributes: {
                // Атрибуты, соответствующие специфическим свойствам схемы элемента.
                "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);
            },
            // Код элемента для генерации ссылки на контекстную справку.
            getContextHelpCode: function() {
                return "CampaignTestSmsElement";
            },
            // Инициализация атрибутов текущими значениями свойств схемы.
            initParameters: function(element) {
                this.callParent(arguments);
                this.set("SmsText", element.smsText);
                this.set("PhoneNumber", element.phoneNumber);
            },
            // Сохранение свойств схемы.
            saveValues: function() {
                this.callParent(arguments);
                var element = this.get("ProcessElement");
                element.smsText = this.getSmsText();
                element.phoneNumber = this.getPhoneNumber();

            },
            // Вычитка текущих значений из атрибутов.
            getPhoneNumber: function() {
                var number = this.get("PhoneNumber");
                return number ? number : "";
            },
            getSmsText: function() {
                var smsText = this.get("SmsText");
                return smsText ? smsText : "";
            }
        },
        diff: [
            // Контейнер для UI
            {
                "operation": "insert",
                "name": "ContentContainer",
                "propertyName": "items",
                "parentName": "EditorsContainer",
                "className": "Terrasoft.GridLayoutEdit",
                "values": {
                    "itemType": Terrasoft.ViewItemType.GRID_LAYOUT,
                    "items": []
                }
            },
            // Главная подпись элемента.
            {
                "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"]
                }
            }
        },
        // Подпись для текстового поля ввода номера отправителя.
        {
            "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"]
                }
            }
        },
        // Текстовое поле для ввода телефонного номера.
        {
            "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" }
            }
        },
        // Подпись для текстового поля ввода текста сообщения.
        {
            "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"
                }
            }
        },
        // Текстовое поле для ввода текста сообщения.
        {
            "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" }
                    }
                }
            ]
        };
    }
);
```

После внесения изменений схему необходимо сохранить.

3. Расширить меню [*Дизайнера кампании*] новым элементом

Чтобы созданный элемент отображался в меню дизайнера кампаний, необходимо расширить базовый менеджер схем элементов кампаний. Для этого нужно в пользовательский пакет добавить схему, расширяющую `CampaignElementSchemaManagerEx` (пакет `CampaignDesigner`).

Для созданной схемы следует установить следующие свойства:

- [*Заголовок*] ([*Title*]) — "`TestSmsCampaignElementSchemaManagerEx`".
- [*Название*] ([*Name*]) — "`CampaignElementSchemaManagerEx`".
- [*Родительский объект*] ([*Parent object*]) — "`CampaignElementSchemaManagerEx`".

В секцию [*Исходный код*] ([*Source code*]) схемы необходимо добавить исходный код.

`CampaignElementSchemaManager`

```
require(["CampaignElementSchemaManager", "TestSmsElementSchema"],
    function() {
        // Добавление новой схемы в список доступных схем элементов в дизайнере кампании.
        var coreElementClassNames = Terrasoft.CampaignElementSchemaManager.coreElementClassNames
        coreElementClassNames.push({
            itemType: "Terrasoft.TestSmsElementSchema"
        });
    });
```

После внесения изменений схему необходимо сохранить.

4. Создать серверную часть элемента кампании

Чтобы реализовать возможность сохранения базовых и пользовательских свойств элемента кампании, для него необходимо создать класс, взаимодействующий с серверной частью приложения. Класс должен быть наследником `CampaignSchemaElement` и переопределять методы `ApplyMetaDataValue()` и `WriteMetaData()`.

Для этого нужно создать схему исходного кода со следующими свойствами:

- [*Заголовок*] ([*Title*]) — "TestSmsElement".
- [*Название*] ([*Name*]) — "TestSmsElement".

В секцию [*Исходный код*] ([*Source code*]) схемы необходимо добавить исходный код.

TestSmsElement

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

    // Описание новых свойств.
    [DesignModeProperty(Name = "PhoneNumber",
        UsageType = DesignModeUsageType.NotVisible, MetaPropertyName = PhoneNumberPropertyName)]
    [DesignModeProperty(Name = "SmsText",
        UsageType = DesignModeUsageType.NotVisible, MetaPropertyName = SmsTextPropertyName)]
    public class TestSmsElement : CampaignSchemaElement
    {
        // Текстовое представление названий свойств.
        private const string PhoneNumberPropertyName = "PhoneNumber";
        private const string SmsTextPropertyName = "SmsText";
        public TestSmsElement() {
            // Указываем тип элемента (CMC - асинхронный элемент).
            ElementType = CampaignSchemaElementType.AsyncTask;
        }
        public TestSmsElement(TestSmsElement source)
                : this(source, null, null) {
        }


        public TestSmsElement(TestSmsElement source, Dictionary<Guid, Guid> dictToRebind,
                Core.Campaign.CampaignSchema parentSchema) : base(source, dictToRebind, parentSc
            // При копировании перенесем значения свойств исходного объекта.
            ElementType = source.ElementType;
            PhoneNumber = source.PhoneNumber;
            SmsText = source.SmsText;
        }


        // Для отображения в логе кампании и т.п.
        protected override Guid Action {
```

```
            get {
                return CampaignConsts.CampaignLogTypeMailing;
            }
        }

        // Новое свойство элемента СМС.
        // Guid генерируем уникальный (для каждого свойства свой).
        [MetaTypeProperty("{A67950E7-FFD7-483D-9E67-3C9A30A733C0}")]
        public string PhoneNumber {
            get;
            set;
        }
        // Новое свойство элемента СМС.
        [MetaTypeProperty("{05F86DF2-B9FB-4487-B7BE-F3955703527C}")]
        public string SmsText {
            get;
            set;
        }
       // При вычитке добавим присвоение значений созданным свойствам.
        protected override void ApplyMetaDataValue(DataReader reader) {
            base.ApplyMetaDataValue(reader);
            switch (reader.CurrentName) {
               // Для PhoneNumber.
                case PhoneNumberPropertyName:
                    PhoneNumber = reader.GetValue<string>();
                    break;
               // Для SmsText.
                case SmsTextPropertyName:
                    SmsText = reader.GetValue<string>();
                    break;
            }
        }

        // При записи данных запишем текущие значения новых свойств.
        public override void WriteMetaData(DataWriter writer) {
            base.WriteMetaData(writer);
            // Для PhoneNumber.
            writer.WriteValue(PhoneNumberPropertyName, PhoneNumber, string.Empty);
            // Для SmsText.
            writer.WriteValue(SmsTextPropertyName, SmsText, string.Empty);
        }
        // Копирует элемент.
        public override object Clone() {
            return new TestSmsElement(this);
        }
        public override object Copy(Dictionary<Guid, Guid> dictToRebind, Core.Campaign.CampaignS
            return new TestSmsElement(this, dictToRebind, parentSchema);
        }
```

```
        // Метод создания исполняемого элемента для текущего элемента схемы (CMC).
        public override ProcessFlowElement CreateProcessFlowElement(UserConnection userConnectio
            var executableElement = new TestSmsCampaignProcessElement {
                UserConnection = userConnection,
                SmsText = SmsText,
                PhoneNumber = PhoneNumber
            };
            InitializeCampaignProcessFlowElement(executableElement);
            return executableElement;
        }
    }
}
```

После внесения изменений схему исходного кода необходимо опубликовать.

5. Создать исполняемый элемент для нового элемента кампании

Чтобы механизм выполнения кампании мог в нужный момент выполнить требуемую логику поведения создаваемого элемента, необходимо создать исполняемый элемент. Это класс, наследник класса `CampaignFlowElement`, в котором ключевой является реализация методов:

- `SingleExecute` — позволит механизму выполнения кампании в нужный момент выполнить требуемую логику поведения элемента.
- `GetAudienceQuery` — определяет обрабатываемую аудиторию для элемента.

Для создания исполняемого элемента необходимо в пользовательском пакете создать схему исходного кода со следующими свойствами:

- [*Заголовок*] ([*Title*]) — "`TestSmsCampaignProcessElement`".
- [*Название*] ([*Name*]) — "`TestSmsCampaignProcessElement`".

В секцию [*Исходный код*] ([*Source code*]) схемы необходимо добавить исходный код.

`TestSmsCampaignProcessElement`

```
namespace Terrasoft.Configuration
{
    using Terrasoft.Core.Campaign;
    using Terrasoft.Core.DB;

    public class TestSmsCampaignProcessElement : CampaignFlowElement
    {
        public TestSmsCampaignProcessElement() {
        }
```

```
        // Свойства, специфические для СМС. Передаются от экземпляра класса TestSmsElement.
        public string PhoneNumber {
            get;
            set;
        }
        public string SmsText {
            get;
            set;
        }
        // Определяет аудиторию, которая должна быть обработана элементом.
        // Если не переопределить этот метод, то по умолчанию базовый класс берет всю аудиторию
        // из текущего шага с признаком "Шаг не выполнен" и статусом "Участвует в кампании".
        protected override Query GetAudienceQuery() =>
            new Select(UserConnection)
                .Column("Id")
                .From(CampaignParticipantTable)
                .Where("CampaignItemId").IsEqual(Column.Parameter(CampaignItemId))
                .And("StatusId").IsEqual(CampaignConstants.CampaignParticipantParticipatingStatu
                .And("StepCompleted").IsEqual(Column.Parameter(0));

        // Реализация метода выполнения элемента.
        // Если не переопределить этот метод, то по умолчанию базовый класс возьмет запрос
        // audienceQuery и для всех кто подпадает под эту выборку проставит "Шаг выполнен"
        // без какой-либо дополнительной логики.

        protected override int SingleExecute(Query audienceQuery) {
            // TODO: Реализовать работу отправки СМС-сообщений.
            //
            // Текущий шаг для аудитории устанавливается как выполненный.
            return SetItemCompleted(audienceQuery as Select);
        }
    }
}
```

После внесения изменений схему исходного кода необходимо опубликовать.

6. Добавить пользовательскую логику для обработки событий кампании

Для реализации пользовательской логики при сохранении, копировании, удалении, запуске и остановке кампании предусмотрен механизм событийных обработчиков. Для его использования достаточно создать публичный запечатанный (`sealed`) класс-обработчик, унаследованный от `CampaignEventHandlerBase`. В нем нужно реализовать один или несколько интерфейсов, описывающих сигнатуры обработчиков конкретных событий. Этот класс не должен быть обобщенным и должен иметь доступный конструктор по-умолчанию.

В текущей версии поддерживаются следующие интерфейсы:

- `IOnCampaignBeforeSave` — содержит метод, который будет вызван перед сохранением кампании.
- `IOnCampaignAfterSave` — содержит метод, который будет вызван после сохранения кампании.
- `IOnCampaignDelete` — содержит метод, который будет вызван перед удалением кампании.
- `IOnCampaignStart` — содержит метод, который будет вызван перед стартом кампании.
- `IOnCampaignStop` — содержит метод, который будет вызван перед остановкой кампании.
- `IOnCampaignValidate` — содержит метод, который будет вызван при валидации кампании.
- `IOnCampaignCopy` — содержит метод, который будет вызван после копирования кампании.

При появлении исключительной ситуации во время обработки события, цепочка вызовов останавливается, а состояние кампании в базе данных откатывается на предыдущее.

> **На заметку.** При реализации интерфейса `IOnCampaignValidate` рекомендуется сохранять ошибки в схему кампании, пользуясь методом `AddValidationInfo(string)`.

Уточнение условий кейса

Для работы созданного элемента кампании для СМС рассылки необходимо работающее соединение с СМС-шлюзом. Предполагается, что проверку соединения, состояния счета и других параметров необходимо осуществлять при валидации кампании, а отправку тестового СМС сообщения — при ее старте.

Для реализации новых условий необходимо в пользовательский пакет добавить схему исходного кода со следующими свойствами:

- [*Заголовок*] ([*Title*]) — "`TestSmsEventHandler`".
- [*Название*] ([*Name*]) — "`TestSmsEventHandler`".

В секцию [*Исходный код*] ([*Source code*]) схемы необходимо добавить исходный код.

`TestSmsEventHandler`

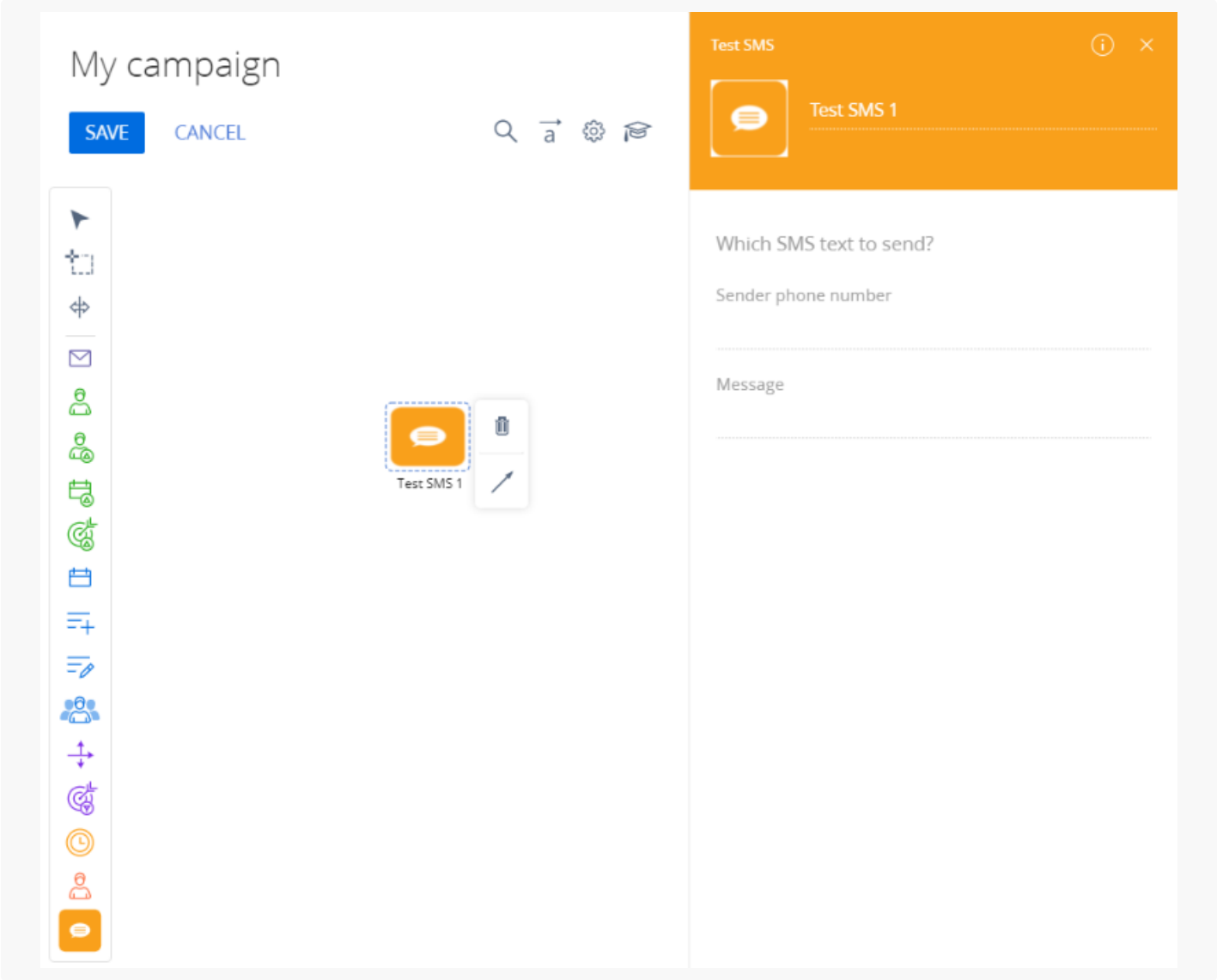
```
namespace Terrasoft.Configuration
{
    using System;
    using Terrasoft.Core.Campaign.EventHandler;

    public sealed class TestSmsEventHandler : CampaignEventHandlerBase, IOnCampaignValidate, IOn
    {
        // Реализация обработчика события старта кампании.
        public void OnStart() {
            // TODO: Логика отправки тестовой СМС...
            //
        }
        // Реализация обработчика события валидации кампании.
        public void OnValidate() {
```

```
            try {
                // TODO: Логика валидации соединения с СМС-шлюзом...
                //
            } catch (Exception ex) {
                // При наличии ошибки, добавляем ее в схему кампании.
                CampaignSchema.AddValidationInfo(ex.Message);
            }
        }
    }
}
```

После внесения изменений необходимо опубликовать схему. Затем нужно выполнить полную компиляцию приложения и очистить кэш.

В результате выполнения кейса, в меню элементов кампании будет добавлен новый элемент [*TestSMS*] (1), который можно добавить на диаграмму кампании (2). При выборе добавленного элемента будет отображена страница свойств этого элемента (3).



Важно. При сохранении кампании возможно возникновение ошибки "`Parameter 'type' cannot be null`". Она связана с тем, что после компиляции не была обновлена библиотека конфигурации и созданные типы в нее не попали.

Для устранения ошибки необходимо перекомпилировать проект и очистить все возможные хранилища с закэшированными данными. Возможно, нужно будет очистить пул приложения или перезапустить сайт в IIS.

Добавить пользовательский переход для нового элемента кампании

 Сложный

Для настройки маркетинговой кампании используется [*Дизайнер кампании*]. С его помощью можно создать визуальную схему кампании, состоящую из связанных предустановленных элементов. Также существует возможность создания пользовательских элементов кампании.

Алгоритм добавления пользовательского перехода (стрелки)

1. Создание новой схемы для элемента [*Переход*].
2. Создание страницы свойств элемента [*Переход*].
3. Создание серверной части элемента [*Переход*].
4. Создание исполняемого элемента для перехода.
5. Создание замещающего модуля `CampaignConnectorManager` для добавления логики работы перехода.
6. Подключение замещающего модуля `CampaignConnectorManager`.

Пример. Необходимо создать пользовательский переход (стрелку) из нового элемента кампании для отправки СМС сообщений пользователем, в котором добавить возможность выбора условия отклика по рассылке. В карточке настройки пользователь может выбрать опцию не учитывать отклики, либо учитывать с перечнем возможных откликов по рассылке. Если не выбран ни один отклик, переводить для любого значения отклика.

Важно. В этом примере в названиях схем отсутствует префикс `Usr`. Префикс, используемый по умолчанию, можно изменить в системной настройке [*Префикс названия объекта*] (код `SchemaNamePrefix`).

Важно. Перед выполнением примера добавьте пользовательский элемент кампании.

1. Создать объекты [*Получатель СМС*] и [*Отклики по рассылке*]

Для полноценной работы примера в пакете разработки создайте схемы объектов [*Получатель СМС*] ([*TestSmsTarget*]) и [*Отклики по рассылке*] ([*TestSmsResponseType*]).

В схему [*Отклики по рассылке*] ([*TestSmsResponseType*]) добавьте колонку со следующими свойствами:

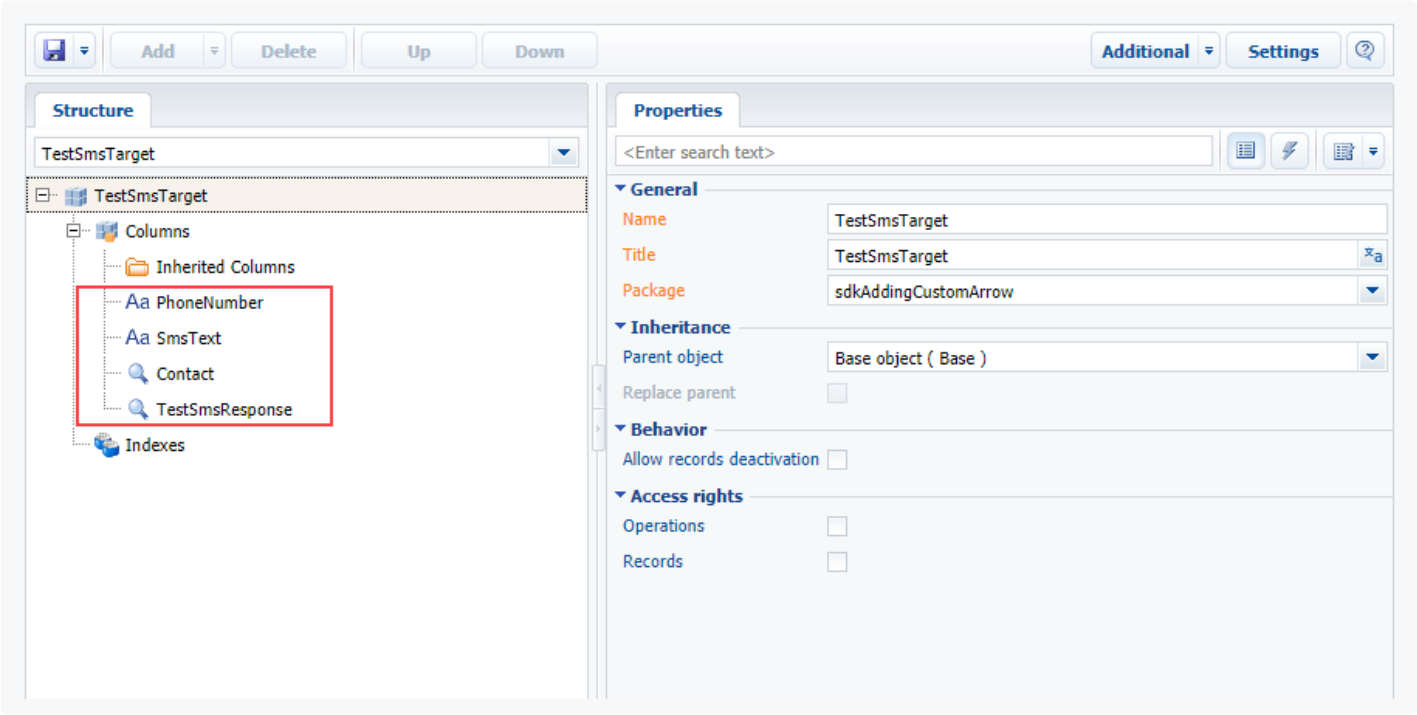
- [*Заголовок*] ([*Title*]) — "Название" ("Name");
- [*Название*] ([*Name*]) — "Name";
- [*Тип данных*] ([*Data type*]) — "Строка 50 символов" ("Text (50 characters)").

В схему [*Получатель СМС*] ([*TestSmsTarget*]) добавьте колонки со следующими свойствами:

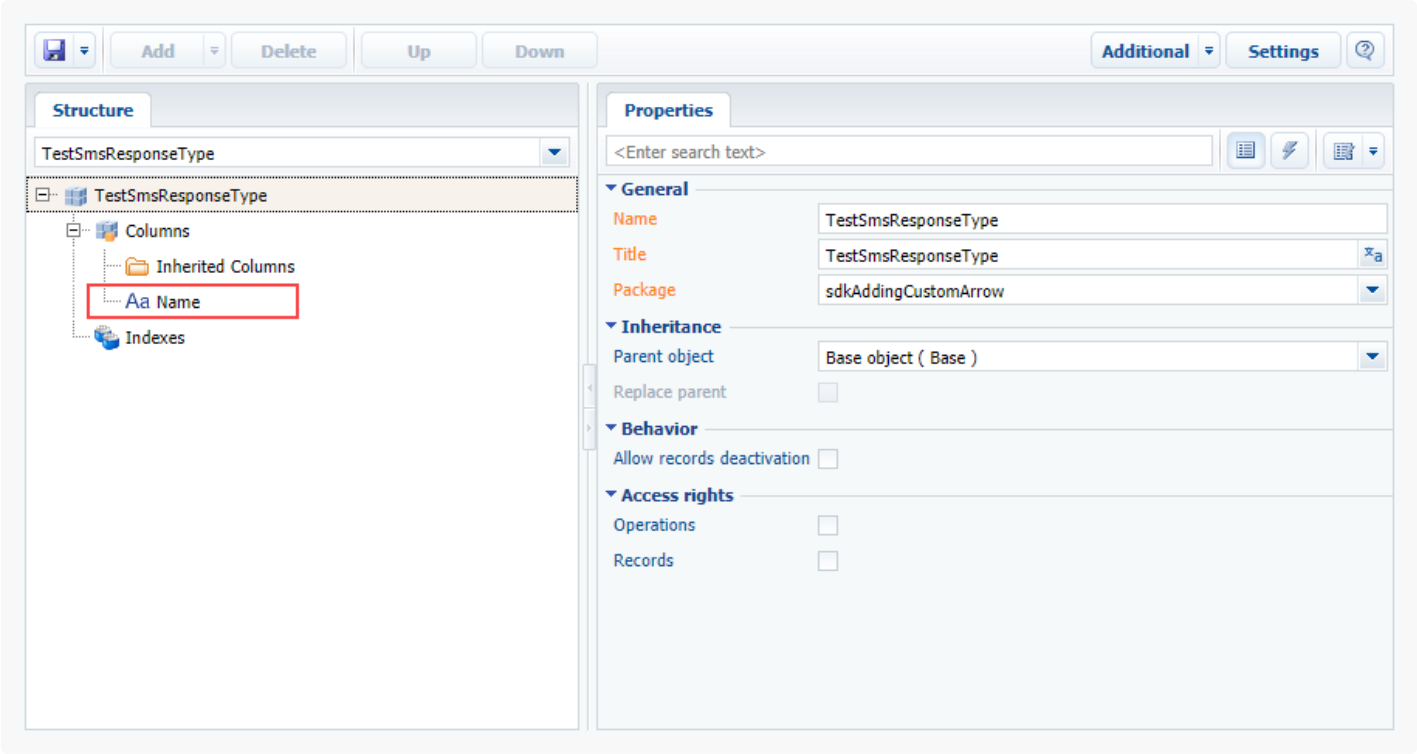
Основные свойства колонок схемы объекта [*Получатель СМС*] ([*TestSmsTarget*])

[*Заголовок*] ([*Title*])	**[*Название*] ([*Name*])**	**[*Тип данных*] ([*Data type*])**
Phone number	PhoneNumber	"Строка 50 символов" ("Text (50 characters)")
SMS text	SmsText	"Строка 50 символов" ("Text (50 characters)")
Contact	Contact	"Справочник" ("Lookup") — "Contact"
Test SMS response	TestSmsResponse	"Справочник" ("Lookup") — "TestSmsResponseType"

Свойства и колонки схемы объекта [*Получатель СМС*] ([*TestSmsTarget*])

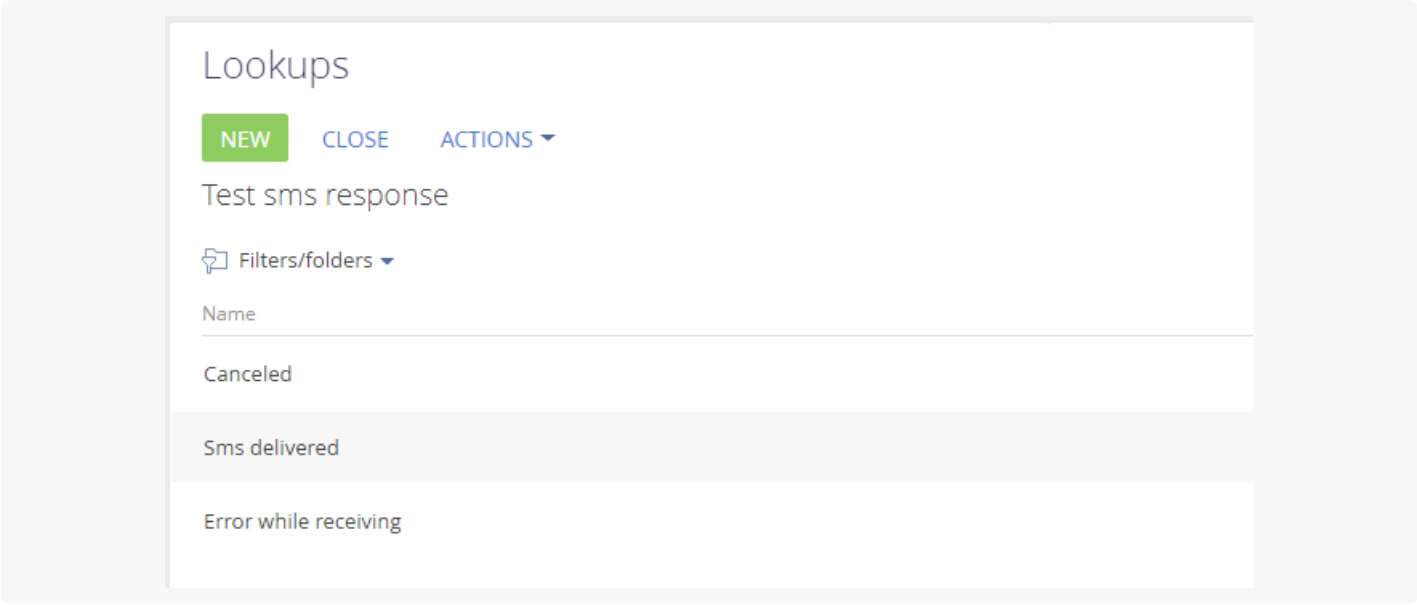


Свойства и колонки схемы объекта [*Отклики по рассылке*] ([*TestSmsResponseType*])



Сохраните и опубликуйте объекты.

Создайте справочник для объекта [*TestSmsResponseType*] и наполните его значениями "Доставлено", "Отменено", "Ошибка доставки" ("Sms delivered", "Canceled", "Error while receiving") и проч. Идентификаторы ([*Id*]) откликов будут использованы в коде страницы свойств условного перехода из СМС-рассылки.



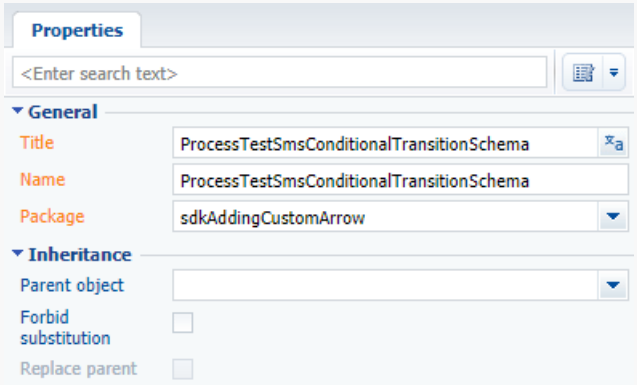
2. Создать новую схему для элемента [*Переход*]

Для отображения элемента в пользовательском интерфейсе [*Дизайнера кампании*] необходимо в

пакете разработки создать новую схему модуля. Для созданной схемы нужно установить следующие свойства:

- [*Заголовок*] ([*Title*]) — "ProcessTestSmsConditionalTransitionSchema".
- [*Название*] ([*Name*]) — "ProcessTestSmsConditionalTransitionSchema".

Важно. Название схемы для стрелки должно начинаться с префикса "`Process`".



В секцию [*Исходный код*] ([*Source code*]) схемы добавьте исходный код.

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"
            },
            // Полное имя типа связанного элемента стрелки.
            typeName: "Terrasoft.Configuration.TestSmsConditionalTransitionElement, Terrasoft.Co
            // Имя элемента стрелки для привязки к элементам кампании.
            connectionUserHandleName: "TestSmsConditionalTransition",
            // Имя карточки свойств стрелки.
            editPageSchemaName: "TestSmsConditionalTransitionPropertiesPage",
            elementType: CampaignEnums.CampaignSchemaElementTypes.CONDITIONAL_TRANSITION,
            // Коллекция откликов по CMC-рассылке.
            testSmsResponseId: null,
            // Признак, который учитывает условие откликов при переводе контактов.
            isResponseBasedStart: false,
```

```
            getSerializableProperties: function() {
                var baseSerializableProperties = this.callParent(arguments);
                // Свойства для сериализации и передачи в серверную часть при сохранении.
                Ext.Array.push(baseSerializableProperties, ["testSmsResponseId", "isResponseBase
                return baseSerializableProperties;
            }
        });
        return Terrasoft.ProcessTestSmsConditionalTransitionSchema;
    });
```

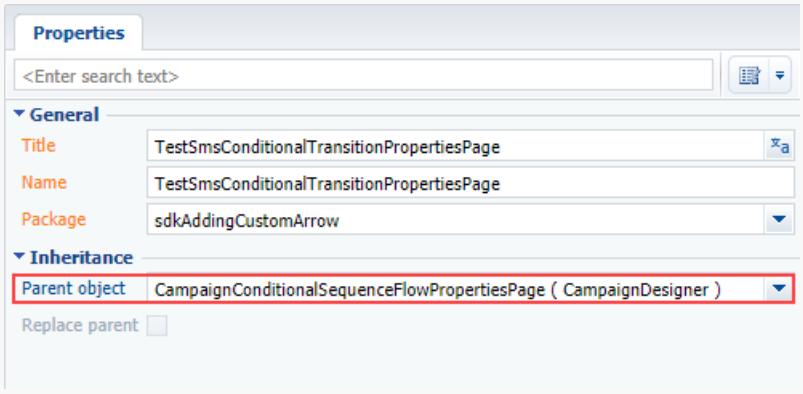
Сохраните созданную схему.

3. Создать страницу свойств элемента [*Переход*]

Для отображения и изменения свойств элемента кампании необходимо в пакете разработки создать его страницу записи. Для этого нужно создать схему, расширяющую `CampaignConditionalSequenceFlowPropertiesPage` (пакет `CampaignDesigner`).

Для созданной схемы требуется установить следующие свойства:

- [*Заголовок*] ([*Title*]) — "`TestSmsConditionalTransitionPropertiesPage`".
- [*Название*] ([*Name*]) — "`TestSmsConditionalTransitionPropertiesPage`".
- [*Родительский объект*] ([*Parent object*]) — "`CampaignConditionalSequenceFlowPropertiesPage`".



В созданную схему добавьте локализуемые строки, основные свойства которых приведены в таблице.

Основные свойства локализуемых строк

[*Название*] ([*Name*])	[*Значение*] ([*Value*])
ReactionModeCaption	Результат {0} шага? (What is the result of the {0} step?)
ReactionModeDefault	Передача участников независимо от их ответа (Transfer participants regardless of their response)
ReactionModeWithCondition	Настройка ответов для передачи участников (Set up responses for transferring participants)
IsTestSmsDelivered	Тестовое сообщение доставлено (Test SMS delivered)
IsErrorWhileReceiving	Ошибка при получении (Error while receiving)

В секцию [*Исходный код*] ([*Source code*]) схемы добавьте исходный код.

`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: {
        // Формирует соответствие смс-откликов (на основании справочника TestSmsResponse
        // Предполагается, что в системе есть справочник TestSmsResponseType
        // с записями TestSmsDelivered и ErrorWhileReceiving.
        getResponseConfig: function() {
            return {
                "IsTestSmsDelivered": "83389cd3-2dff-489b-aaa9-f3dd196777e1",
                "IsErrorWhileReceiving": "69f29da1-eb09-44f6-8e0c-697dd05d8ccc"
            };
        },

        subscribeEvents: function() {
            this.callParent(arguments);
            // Привязка обработчика к событию изменения значения атрибута ReactionMode
            this.on("change:ReactionMode", this.onReactionModeLookupChanged, this);
        },

        // Метод-обработчик события изменения атрибута ReactionMode.
        onReactionModeLookupChanged: function() {
            var reactionModeEnum = this.get("ReactionModeEnum");
            var reactionMode = this.get("ReactionMode");
            var decisionModeEnabled = (reactionMode && reactionMode.value === reactionMo
```

```
        if (!decisionModeEnabled) {
            this.set("ReactionConditionDecision", null);
        }
    },

    // Инициализирует свойства viewModel для отображения карточки при открытии.
    initParameters: function(element) {
        this.callParent(arguments);
        var isResponseBasedStart = element.isResponseBasedStart;
        this.initReactionMode(isResponseBasedStart);
        this.initTestSmsResponses(element.testSmsResponseId);
    },

    // Вспомогательный метод, обрезающий строку до указанной длины и добавляющий тро
    cutString: function(strValue, strLength) {
        var ellipsis = Ext.String.ellipsis(strValue.substring(strLength), 0);
        return strValue.substring(0, strLength) + ellipsis;
    },

    // Устанавливает значение статуса "СМС доставлено".
    initIsTestSmsDelivered: function(value) {
        if (value === undefined) {
            value = this.get("IsTestSmsDelivered");
        }
        this.set("IsTestSmsDelivered", value);
    },

    // Устанавливает значение статуса "Ошибка при получении".
    initIsErrorWhileReceiving: function(value) {
        if (value === undefined) {
            var isErrorWhileReceiving = this.get("IsErrorWhileReceiving");
            value = isErrorWhileReceiving;
        }
        this.set("IsErrorWhileReceiving", value);
    },

    // Инициализирует выбранные отклики при открытии карточки.
    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);
    },

    // Вспомогательный метод, извлекающий массив идентификаторов из входящего JSON г
    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);
    },

    // Сохраняет значения откликов и настройку, добавлять условия откликов или нет.
    saveValues: function() {
        this.callParent(arguments);
        var element = this.get("ProcessElement");
        var isResponseBasedStart = this.getIsReactionModeWithConditions();
        element.isResponseBasedStart = isResponseBasedStart;
        element.testSmsResponseId = this.getTestSmsResponseId(isResponseBasedStart);
    },

    // Получает сериализированные id выбранных откликов.
    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;
        },
        // Подставляет в текст название элемента, из которого выходит стрелка.
        getQuestionCaption: function() {
            var caption = this.get("Resources.Strings.ReactionModeCaption");
            caption = this.Ext.String.format(caption, this.getSourceElement().getCaptior
            return caption;
        }
    },
    diff: /**SCHEMA_DIFF*/[
        // Контейнер.
        {
            "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": []
            }
        },
```

```
                // Заголовок.
                {
                    "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"]
                        }
                    }
                },
                // Список.
                {
                    "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"]
            }
        },
        // Элемент списка.
        {
            "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
                }
            }
        },
        // Элемент списка.
        {
            "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*/
        };
    }
);
```

Сохраните созданную схему.

4. Создать серверную часть элемента [*Переход*] из элемента [*CMC-рассылка*]

Чтобы реализовать возможность сохранения базовых и пользовательских свойств элемента кампании, для него необходимо создать класс, взаимодействующий с серверной частью приложения. Класс должен быть наследником `CampaignSchemaElement` и переопределять методы `ApplyMetaDataValue()` и `WriteMetaData()`.

Для этого создайте схему исходного кода со следующими свойствами:

- [*Заголовок*] ([*Title*]) — "`TestSmsConditionalTransitionElement`".
- [*Название*] ([*Name*]) — "`TestSmsConditionalTransitionElement`".

В секцию [*Исходный код*] ([*Source code*]) схемы необходимо добавить исходный код.

`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);
    }

    // Переопределяет фабричный метод по созданию исполняемого элемента
    // Возвращает элемент с типом TestSmsConditionalTransitionFlowElement
    public override ProcessFlowElement CreateProcessFlowElement(UserConnection userConnectio
        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);
        InitializeAudienceSchemaInfo(executableElement, userConnection);
        return executableElement;
    }
```

```
    }
}
```

Сохраните и опубликуйте созданную схему.

5. Создать исполняемый элемент для перехода из элемента [*СМС-рассылка*]

Чтобы добавить функциональность, которая выполнит учет условия откликов по отправленой смс-рассылке, создайте исполняемый элемент. Это класс, наследник класса `ConditionalTransitionFlowElement`.

Для создания исполняемого элемента в пакете разработки создайте схему исходного кода со следующими свойствами:

- [*Заголовок*] ([*Title*]) — "`TestSmsConditionalTransitionFlowElement`".
- [*Название*] ([*Name*]) — "`TestSmsConditionalTransitionFlowElement`".

В секцию [*Исходный код*] ([*Source code*]) схемы необходимо добавить исходный код.

`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();
        }
    }
}
```

Сохраните и опубликуйте созданную схему.

6. Создать замещающий модуль CampaignConnectorManager для добавления логики работы перехода

Для добавления специфической логики работы перехода при изменении источника (элемента, из которого выходит стрелка) в пакете разработки создайте новую схему модуля замещающего модуля `CampaignConnectorManager`. Для созданной схемы нужно установить следующие свойства:

- [*Заголовок*] ([*Title*]) — `"TestSmsCampaignConnectorManager"`.
- [*Название*] ([*Name*]) — `"TestSmsCampaignConnectorManager"`.

В секцию [*Исходный код*] ([*Source code*]) схемы необходимо добавить исходный код.

`TestSmsCampaignConnectorManager`

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

    Ext.define("Terrasoft.TestSmsCampaignConnectorManager", {
        // Указываем, что замещаем модуль CampaignConnectorManager
        override: "Terrasoft.CampaignConnectorManager",

        // Добавляем маппинг название схема-элемента источника для стрелки - тип стрелки (full r
        initMappingCollection: function() {
            this.callParent(arguments);
            this.connectorTypesMappingCollection.addIfNotExists("TestSmsElementSchema",
                "Terrasoft.ProcessTestSmsConditionalTransitionSchema");
        },
```

```
        // Виртуальный метод для перегрузки
        // Логика для процессинга стрелки перед ее подменой стрелкой с новым типом.
        additionalBeforeChange: function(prevTransition, sourceItem, targetItem) {
            // additional logic here
        },

        // Виртуальный метод для перегрузки
        // Заполнение специфических полей созданной стрелки на основе предыдущей стрелки.
        fillAdditionalProperties: function(prevElement, newElement) {
            if (newElement.getTypeInfo().typeName === "ProcessTestSmsConditionalTransitionSchema
                // Переносим настроенные отклики, если предыдущая стрелка такого же типа
                newElement.testSmsResponseId = prevElement.testSmsResponseId ? prevElement.testS
                // Так же переносим и настройку учета откликов
                newElement.isResponseBasedStart = prevElement.isResponseBasedStart ? prevElement
            }
        }
    });
});
```

Сохраните созданную схему.

7. Подключить замещающий модуль CampaignConnectorManager

Для подключения модуля, созданного на предыдущем шаге создайте замещающий клиентский модуль, в котором в качестве родительского объекта укажите `BootstrapModulesV2` из пакета `NUI`.

В секцию [*Исходный код*] ([*Source code*]) схемы необходимо добавить исходный код.

`BootstrapModulesV2`

```
// Ставим в зависимости созданный ранее модуль TestSmsCampaignConnectorManager
define("BootstrapModulesV2", ["TestSmsCampaignConnectorManager","ProcessTestSmsConditionalTransi
```

Сохраните созданную схему.

На заметку. В реальном примере желательно создать отдельную схему объекта [*CMC-рассылка*]. Объекты [*TestSmsElement*] и [*TestSmsConditionalTransitionElement*] будут содержать [*Id*] этого объекта, а не поля [*SmsText*], [*PhoneNumber*]... Исполняемый элемент `TestSmsCampaignProcessElement` в методе `Execute()` должен содержать логику по добавлению контактов в аудиторию рассылки. Отдельный механизм (или несколько механизмов) должны выполнить отправку рассылки, а затем зафиксировать отклики участников. На основании этих откликов и будет выполнена работа стрелки по перемещению аудитории кампании на следующий шаг.