

# Компоненты приложения

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

<b>Общие принципы работы интеграции с телефонией</b>	10
Способы интеграции с системой обмена сообщениями в Creatio	10
Взаимодействие коннекторов с Creatio	12
<b>Отчеты MS Word</b>	14
Настроить доступ к разделу Настройка отчетов	14
Создать простой отчет MS Word	15
Создать отчет MS Word с использованием базовых макросов	18
Создать отчет MS Word с использованием пользовательских макросов	19
Привязать отчет MS Word к пакету	20
<b>Создать отчет MS Word (базовые макросы)</b>	20
1. Создать отчет	21
2. Настроить поля отчета	22
3. Настроить шаблон отчета	23
Результат выполнения примера	26
<b>Создать отчет MS Word (пользовательские макросы)</b>	28
1. Создать отчет	28
2. Реализовать пользовательские макросы	29
3. Настроить поля отчета	37
4. Настроить шаблон отчета	39
Результат выполнения примера	43
<b>Базовые макросы</b>	45
Базовые макросы	45
<b>Общие принципы работы с маркетинговыми кампаниями</b>	48
Механизм планирования следующего запуска кампании	48
Основные классы элементов кампании	49
<b>Добавить пользовательский элемент кампании</b>	50
Алгоритм добавления пользовательского элемента кампании	50
1. Создать новый элемент для Дизайнера кампании	50
2. Создать страницу элемента	54
3. Расширить меню Дизайнера кампании новым элементом	58
4. Создать серверную часть элемента кампании	59
5. Создать исполняемый элемент для нового элемента кампании	61
6. Добавить пользовательскую логику для обработки событий кампании	62
Уточнение условий кейса	63
<b>Добавить пользовательский переход для нового элемента кампании</b>	65
Алгоритм добавления пользовательского перехода (стрелки)	66

1. Создать объекты Получатель СМС и Отклики по рассылке	66
2. Создать новую схему для элемента Переход	68
3. Создать страницу свойств элемента Переход	70
4. Создать серверную часть элемента Переход из элемента СМС-рассылка	78
5. Создать исполняемый элемент для перехода из элемента СМС-рассылка	81
6. Создать замещающий модуль CampaignConnectorManager для добавления логики работы перехода	82
7. Подключить замещающий модуль CampaignConnectorManager	83
<b>Общие принципы работы с email-сообщениями</b>	84
Пользовательская фильтрация нежелательных обращений	84
Работа с цепочками email-сообщений	84
Обработчик макроса в шаблоне email-сообщения	88
<b>Добавить свойство для фильтрации email-сообщений</b>	88
Описание кейса	88
Алгоритм реализации кейса	88
<b>Добавить обработчик макроса шаблона email-сообщения</b>	89
Описание примера	89
Исходный код	90
Пример реализации примера	90
<b>Общие принципы работы механизма синхронизации Sync Engine</b>	92
Синхронизация с внешними хранилищами данных	92
Работа с метаданными для синхронизации	95
<b>Класс SyncContext</b>	95
Конструкторы	96
Свойства	96
Методы	96
<b>Класс RemoteProvider</b>	98
Свойства	98
Методы	98
<b>Интерфейс IRemoteItem</b>	99
Методы	100
Атрибут Map	100
<b>Класс LocalProvider</b>	102
Конструкторы	102
Свойства	102
Методы	102
<b>Класс SyncEntity</b>	103
Свойства	103
<b>Класс SyncItemSchema</b>	103
Свойства	103

Методы	104
<b>Класс EntityConfig</b>	<b>104</b>
Свойства	105
<b>Класс DetailEntityConfig</b>	<b>105</b>
Свойства	105
<b>Класс LocalItem</b>	<b>106</b>
Свойства	106
Методы	106
<b>Класс SysSyncMetaData</b>	<b>106</b>
Свойства	107
<b>Класс MetaDataStore</b>	<b>108</b>
Методы	108
<b>Класс ItemMetaData</b>	<b>108</b>
Свойства	108
<b>Интерфейс IReplicaMetadata</b>	<b>109</b>
Свойства	109
Методы	109
<b>Общие принципы работы портала</b>	<b>110</b>
Работа с порталом	110
Схема архитектуры портала	112
Масштабируемость портала	112
Совместимость портала с продуктами Creatio	114
Варианты развертывания портала	114
<b>Web-to-Case</b>	<b>114</b>
Логика автоматического заполнения полей для обращения	115
Рекомендации для выполнения проектных решений	115
<b>Создать Web-to-Case лендинг</b>	<b>116</b>
1. Создать новую запись лендинга в Creatio	116
2. Создать посадочную страницу	117
3. Добавить Web-страницу на сайт	120
Результат выполнения примера	121
<b>Отчеты FastReport</b>	<b>121</b>
Настроить доступ к разделу Настройка отчетов	122
Создать простой отчет FastReport	122
Создать отчет FastReport с изображением	129
Настроить мультиязычие элементов интерфейса отчета FastReport	129
Привязать отчет FastReport к пакету	130
<b>Создать простой отчет FastReport</b>	<b>131</b>
1. Создать отчет	132

2. Указать источники данных	132
3. Реализовать провайдер данных отчета	133
4. Настроить шаблон отчета	137
Результат выполнения примера	140
<b>Создать отчет FastReport с изображением</b>	144
1. Создать отчет	144
2. Указать источники данных	145
3. Реализовать провайдер данных отчета	146
4. Настроить шаблон отчета	149
Результат выполнения примера	153
<b>Синхронизация почты с MS Exchange</b>	156
Реализация интеграции	156
Синхронизируемые данные	157
Логика заполнения участников письма	158
Логика выбора данных для синхронизации	159
<b>Интеграция с Oktell</b>	159
Oktell.js	160
<b>Портал самообслуживания</b>	164
Устройство портала	164
Работа с мастером страниц на портале	164
Настройка портала и порталных пользователей	165
Ограничение доступа к веб-сервисам для пользователей портала	165
<b>Пример использования миксина PortalMessagePublisherExtensions</b>	168
Реализация примера	168
<b>Изменить доступ к сервису для пользователей портала</b>	170
Реализация примера	171
<b>Миксин PortalMessagePublisherExtensions</b>	171
Методы	172
<b>Отправка email-сообщений</b>	172
Алгоритм отправки email-сообщения с существующей учетной записи	173
Алгоритм отправки email-сообщения с явным указанием учетных данных	174
<b>Отправить email-сообщение с существующей учетной записи</b>	177
Алгоритм реализации примера	177
<b>Отправить email-сообщение с явным указанием учетных данных</b>	188
Алгоритм реализации примера	188
<b>Web-To-Object</b>	205
Реализация сервиса Web-to-Object	206
Внешний API сервиса Web-to-Object	206
Интеграция с внешними системами	207

<b>Настроить web-форму для создания пользовательского объекта</b>	207
1. Зарегистрируйте новый тип лендинга	208
2. Добавьте страницу web-формы	209
3. Свяжите новый тип лендинга с созданной страницей записи	211
4. Актуализируйте наполнение скриптами для страницы web-формы	213
5. Создайте и настройте лендинг раздела [Лендинги и web-формы] ([Landing pages and web forms])	215
6. Разверните и настройте посадочную страницу, содержащую web-форму	216
7. Зарегистрируйте пользовательский объект в справочнике	220
<b>Реализовать обработчик для создания сущности с помощью web-формы</b>	222
1. Реализовать пользовательский обработчик	223
2. Зарегистрировать пользовательский обработчик в базе данных	225
Результат выполнения примера	227
<b>Лид</b>	227
Создать пользовательские напоминания и уведомления	227
<b>Создать пользовательское напоминание о дате актуализации продажи в лиде</b>	227
Описание примера	227
Исходный код	228
Алгоритм реализации примера	228
<b>Планирование</b>	237
<b>Изменить логику расчета в разделе [Планирование]</b>	237
1. Скопировать исходный код модуля построения плана	237
2. Создать замещающий модуль построения плана	238
3. Изменить метод открытия страницы плана	239
4. Внести изменения в хранимую процедуру tsp_RecalculateForecastFact	240
Результат выполнения примера	242
<b>Синхронизация с MS Exchange</b>	243
Синхронизация задач с MS Exchange	243
Синхронизация контактов с MS Exchange	246
Синхронизация встреч с MS Exchange	250
<b>Интеграция с Webitel</b>	253
Взаимодействие компонентов	253
Примеры взаимодействия CtiPanel, CtiModel и WebitelCtiProvider	255
Список портов Webitel	255
События Webitel	255
<b>Обращения</b>	256
<b>Добавить новое правило расчета сроков в обращении</b>	256
1. Создать схему объекта	257
2. Добавить в объект колонки	258
2. Создать справочник и заполнить его необходимыми значениями	259

3. Реализовать класс для получения временных параметров	260
4. Подключить новое правило	263
Результат выполнения примера	264
<b>Интеграция с Asterisk</b>	264
Настройка конфигурационного файла сервиса Messaging Service для интеграции Creatio с Asterisk	265
Порты для интеграции Creatio с Asterisk	265
Сервис Terrasoft Messaging Service для интеграции Creatio с Asterisk	265
Пример взаимодействия CtModel, Terrasoft Messaging Service и Asterisk Manager API	266
События Asterisk	268
<b>Сервис машинного обучения</b>	268
Общие принципы работы сервиса машинного обучения	268
Составление запросов на выборку данных для модели машинного обучения	270
Подключение веб-сервиса к функциональности машинного обучения	272
<b>Примеры запросов на выборку данных для модели машинного обучения</b>	274
Использование утилитного класса Terrasoft.Configuration.QueryExtensions	274
<b>Подключить веб-сервис к функциональности машинного обучения</b>	277
Алгоритм реализации примера	277
<b>Реализовать пользовательскую предиктивную модель</b>	286
Описание кейса	286
Алгоритм выполнения кейса	287
<b>Класс IMLModelTrainerJob</b>	290
Методы	290
<b>Класс IMLModelTrainer</b>	291
Методы	291
<b>Класс IMLServiceProxy</b>	291
Методы	291
Свойства типа ClassificationResult	292
<b>Класс IMLEntityPredictor</b>	293
Методы	293
<b>Класс IMLPredictionSaver</b>	293
Методы	293
<b>Справочник MLModel</b>	293
Поля справочника MLModel	294
<b>Сервис обогащения контактов из email</b>	296
Процесс обогащения контакта/контрагента	296
Системные настройки	299
Последовательность идентификации	299
Хэширование информации	300
<b>Планировщик заданий Quartz</b>	301

Рекомендации по настройке планировщика заданий	301
Политики Quartz для обработки не отработавших вовремя заданий	305
<b>Сервис бандлирования статического контента</b>	<b>309</b>
Структура и принцип работы	309
Метрики InfluxDb	312
<b>Развернуть сервис в Docker</b>	<b>313</b>
Системные требования	313
Структура конфигурации сервисов	313
Установка сервисов	313

# Общие принципы работы интеграции с телефонией



Сложный

Creatio предоставляет возможность интеграции с рядом [автоматических телефонных станций](#) (АТС,

[Private Branch Exchange](#), PBX) для управления звонками непосредственно из интерфейса системы.

Функциональность телефонии представлена в интерфейсе в виде CTI-панели ([Computer Telephony Integration](#)), а также в виде раздела [ Звонки ]. Стандартные возможности CTI-панели:

- отображение пользователю входящего звонка с функцией поиска контакта/контрагента по номеру звонящего;
- выполнение звонка из системы в один клик;
- управление звонком в Creatio (ответить, поставить/снять с удержания, завершить, перевести);
- отображение истории звонков для удобного управления связями звонков или возможности перезвонить.

Все звонки, которые были выполнены или приняты с использованием интеграции, сохраняются в разделе [ Звонки ]. В разделе можно посмотреть временные характеристики звонка (дата начала, дата завершения, длительность разговора), а также с чем связан звонок в системе.

Для работы со звонками в приложении необходимо [настроить интеграцию с телефонией](#).

В зависимости от АТС, с которой выполняется интеграция, и особенностей предоставляемого API ([Application Program Interface](#)) используются разные архитектурные механизмы, которые описаны ниже. Также в зависимости от выбранного API может отличаться набор возможностей. Например, функция прослушивания звонков доступна не для всех АТС, а возможность использования Web-телефона доступна только при интеграции с Webitel. Вне зависимости от выбранного механизма интеграции, интерфейс CTI-панели остается одинаковым для всех пользователей.

## Способы интеграции с системой обмена сообщениями в Creatio

Способы интеграции можно разделить на два типа: `first party` и `third party` интеграции.

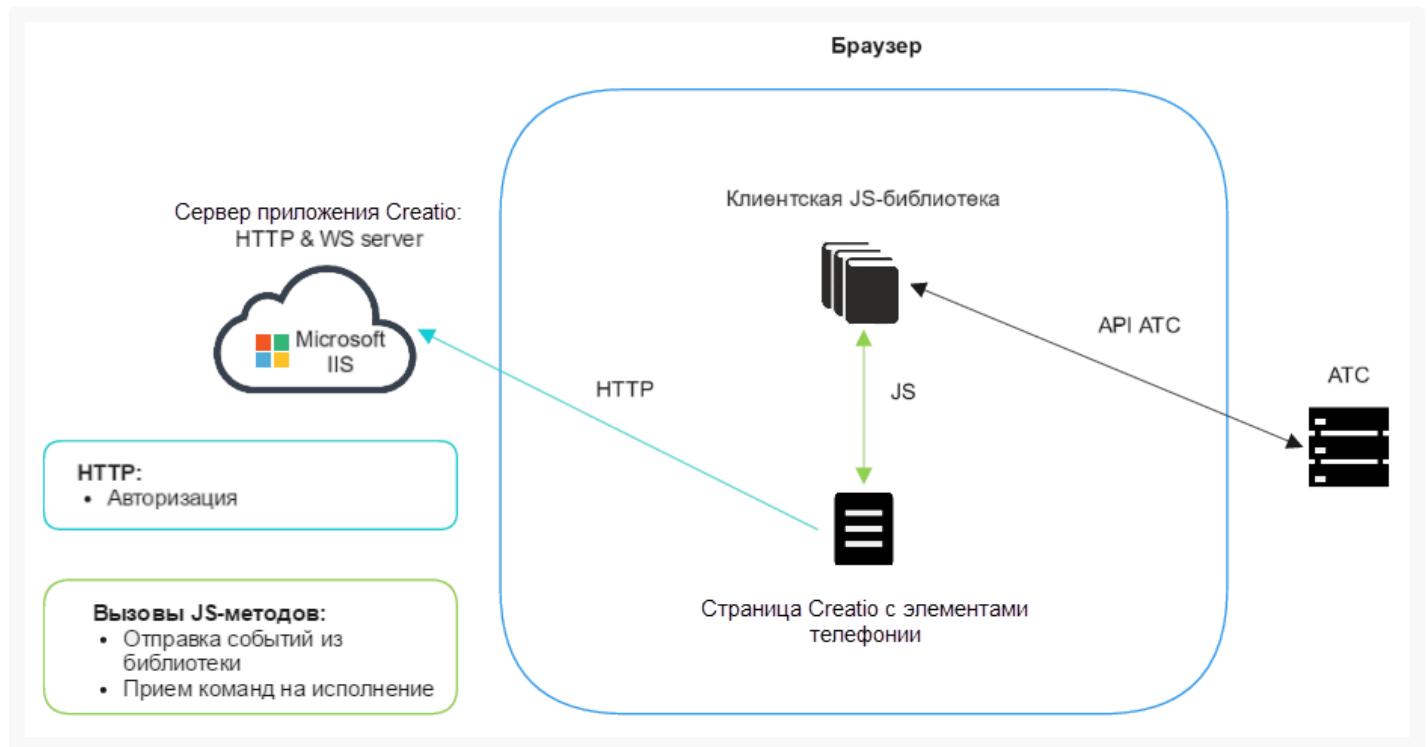
В случае `first party` интеграции для каждого пользователя создается отдельное интеграционное подключение, в рамках которого выполняется обработка событий АТС.

Для `third party` интеграций выполняется одно подключение к серверу АТС и в рамках него выполняется обработка событий АТС для всех пользователей интеграции. В случае `third party` интеграций применяется промежуточное звено `Messaging Service` для распределения информационных потоков разных пользователей.

### JavaScript-адаптер на стороне клиента

При способе интеграции с JavaScript-адаптером на стороне клиента, работа с АТС происходит

непосредственно из браузера. Взаимодействие с АТС и JavaScript-библиотекой, которая, как правило, поставляется производителем АТС, происходит с помощью API АТС. Библиотека отправляет события и принимает команды на исполнение, используя JavaScript. В контексте данной интеграции страница Creatio взаимодействует с сервером приложения для авторизации, используя протокол HTTP(S).



Данный способ интеграции применим к `first party` API телефонии, например, для коннекторов Webitel, Oktell, Finesse. Для коннекторов Webitel и Oktell в качестве протокола соединения используется [WebSocket](#), а для коннектора Finesse используются [long-polling](#) http-запросы.

Преимуществом метода интеграции `first party` является тот факт, что для него не требуется наличие дополнительных узлов, например, Messaging Service. CTI-панель с использованием интеграционной библиотеки выполняет подключения напрямую к API сервера телефонии из браузера на ПК пользователя.

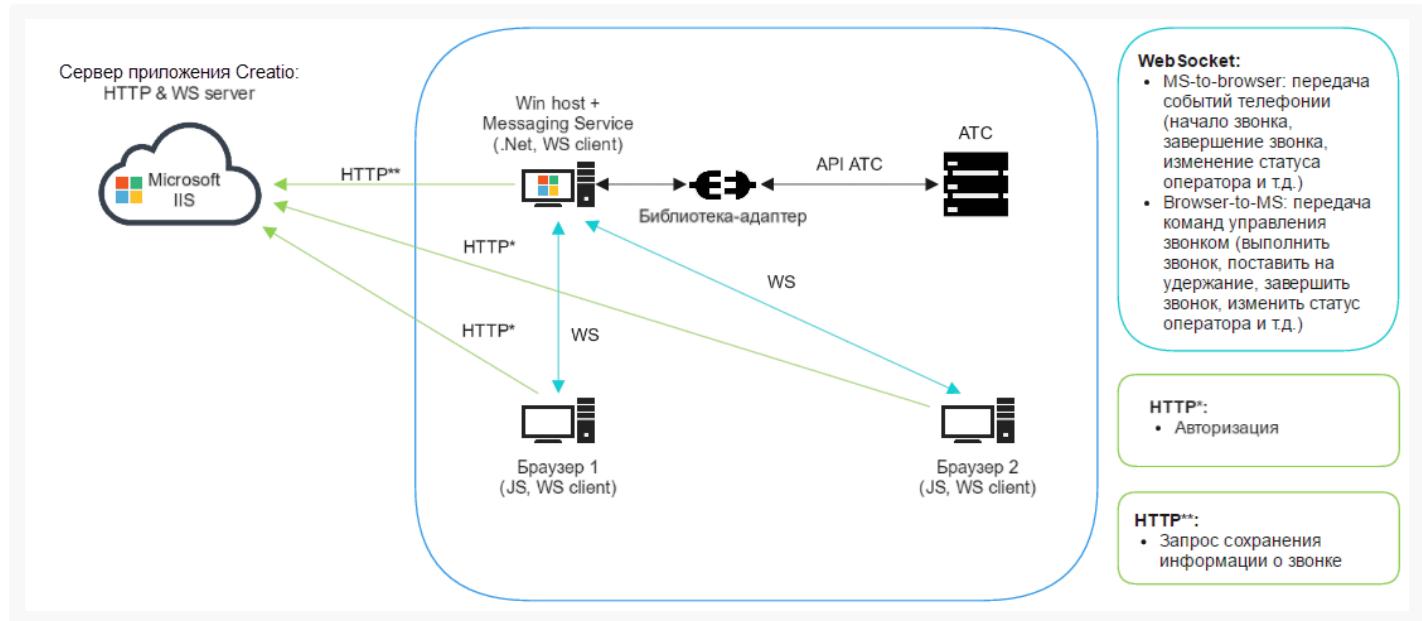
В случае с входящим звонком, сервер телефонии передает событие начала нового звонка и его параметры через WebSocket в библиотеку клиентской интеграции. Библиотека при получении команды нового звонка генерирует событие `RingStarted`, которое передается на страницу приложения.

В случае с исходящим звонком, клиентская часть генерирует команду начала звонка, которая по WebSocket передается на сервер телефонии.

## Terrasoft Messaging Service на серверной стороне

При способе интеграции с Terrasoft Messaging Service (TMS) на серверной стороне все события телефонии проходят через данный сервис, который взаимодействует с АТС посредством библиотеки производителя АТС. Библиотека взаимодействует с АТС посредством API. Также TMS взаимодействует с сервером приложения Creatio для запроса сохранения информации о звонке в базе данных, используя HTTP(S). С клиентским приложением взаимодействие (передача событий и прием команд на исполнение) происходит посредством WebSocket. Как и в случае с интеграцией с JavaScript-адаптером на стороне

клиента, страница Creatio взаимодействует с сервером приложения для авторизации, используя протокол HTTP(S).



Данный способ интеграции применим к `third party` API телефонии (TAPI, TSAPI, New Infinity protocol, WebSocket Oktell). Для данного типа интеграции необходим `Messaging Service` — windows proxy-служба, которая работает с библиотекой-адаптером ATC. `Messaging Service` является универсальным хостером библиотек интеграции с ATC, таких, как Asterisk, Avaya, Callway, Ctios, Infinity, Infra, Tapi. `Messaging Service` при получении клиентских подключений автоматически подключит используемую Creatio библиотеку и инициирует подключение к ATC. Фактически, `Messaging Service` является функциональной "оберткой" для тех коннекторов телефонии, которые не поддерживают клиентскую интеграцию, для того чтобы взаимодействовать с функциональностью телефонии в браузере (генерация и обработка событий, передача данных). Компьютер пользователя производит два вида коммуникации:

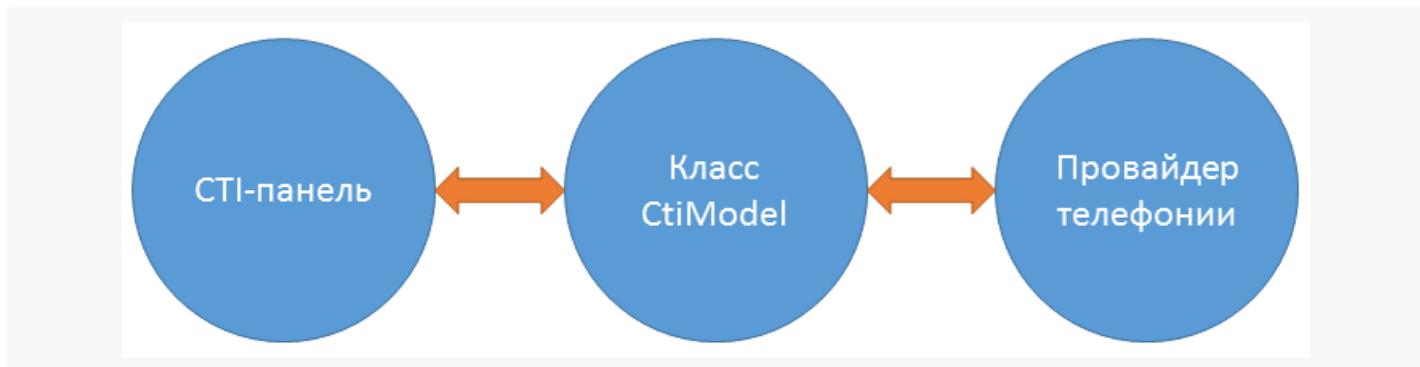
- по протоколу HTTP с сервером приложения Creatio для авторизации и с хостом, на котором установлен `Messaging Service`;
- по WebSocket для непосредственной работы с функциональностью телефонии.

В случае с входящим звонком, ATC передает событие начала нового звонка и его параметры через библиотеку-адаптер на хост с `Messaging Service`. `Messaging Service` при получении команды нового звонка генерирует событие `RingStarted`, которое передается на клиент.

В случае с исходящим звонком, клиентская часть генерирует команду начала звонка, которая по WebSocket передается в `Messaging Service`, который, в свою очередь, генерирует событие исходящего звонка для ATC.

## Взаимодействие коннекторов с Creatio

Все коннекторы взаимодействуют с конфигурацией через класс `CtiModel`. Он получает события от коннектора и обрабатывает их.



Список поддерживаемых событий класса перечислен в таблице.

Список поддерживаемых событий класса CtiModel

Событие	Описание
initialized	Срабатывает при завершении инициализации провайдера.
disconnected	Срабатывает при отключении провайдера.
callStarted	Срабатывает при начале нового вызова.
callFinished	Срабатывает после завершения звонка.
commutationStarted	Срабатывает после установки соединения звонка.
callBusy	Срабатывает при переводе звонка в состояние "занят" (только TAPI).
hold	Срабатывает после постановки звонка на удержание.
unhold	Срабатывает после снятия звонка с удержания.
error	Срабатывает при возникновении ошибки.
lineStateChanged	Срабатывает после изменения набора доступных операций линии либо звонка.
agentStateChanged	Срабатывает после изменения состояния агента.
activeCallSnapshot	Срабатывает при актуализации списка активных звонков.
callSaved	Срабатывает после создания или обновления звонка в базе данных.
rawMessage	Обобщенное событие в провайдере. Срабатывает при любом событии провайдера.
currentCallChanged	Срабатывает при изменении основного звонка. Например, завершается основной звонок при консультации.

callCentreStateChanged	Срабатывает при входе или выходе оператора из режима Call-центр.
callInfoChanged	Срабатывает при изменении данных звонка по идентификатору в БД.
dtmfEntered	Срабатывает, если в линию были отправлены сигналы Dtmf.
webRtcStarted	Срабатывает при старте webRtc сессии.
webRtcVideoStarted	Срабатывает при старте видеопотока webRtc сессии.
webRtcDestroyed	Срабатывает при завершении webRtc сессии.

## Отчеты MS Word



Легкий

**Отчет MS Word (печатная форма)** — документ, который сгенерирован на основе записей разделов Creatio в формате \*.docx. Например, отчеты раздела [ Договоры ] ([ Contracts ]) позволяют распечатать формуляры договоров, а отчеты раздела [ Активности ] ([ Activities ]) можно использовать для распечатки электронных писем, формирования протоколов встреч и т. д. Отчеты MS Word описаны в блоке статей [Отчеты и печатные формы](#).

В зависимости от использования макросов, Creatio позволяет создать различные виды отчетов MS Word.

**Виды** отчетов MS Word, которые позволяет создать Creatio:

- Простой отчет.
- Отчет с использованием базовых макросов.
- Отчет с использованием пользовательских макросов.

## Настроить доступ к разделу [ Настройка отчетов ]

Доступ к разделу [ Настройка отчетов ] ([ Report setup ]) настраивается на уровне системных операций. Если пользователь не имеет доступа к разделу [ Настройка отчетов ] ([ Report setup ]), то отображается стандартное сообщение с указанием операции и недостающих прав. По умолчанию доступ к основным системным операциям имеют только администраторы приложения. Creatio предоставляет возможность настройки доступа к системным операциям для пользователей или групп пользователей. Подробнее читайте в статье [Настроить права доступа на системные операции](#).

Чтобы **настроить доступ к разделу** [ Настройка отчетов ] ([ Report setup ]):

1. Перейдите в дизайнер системы по кнопке . В блоке [ Пользователи и администрирование ] ([ Users and administration ]) перейдите по ссылке [ Права доступа на операции ] ([ Operation permissions ]).
2. Выберите системную операцию [ Доступ к разделу "Настройка отчетов" ] ([ Access to "Report setup" section ], код `CanManageReports` ).
3. На детали [ Доступ к операции ] ([ Operation permission ]) нажмите + и укажите получателя прав.

В результате запись отобразится на детали [ Доступ к операции ] ([ Operation permission ]) в колонке [ Уровень доступа ] ([ Access level ]) со значением [ Да ] ([ Yes ]). Пользователи, которые входят в указанную роль, получат доступ к системной операции [ Доступ к разделу "Настройка отчетов" ] ([ Access to "Report setup" section ], код `CanManageReports` ).

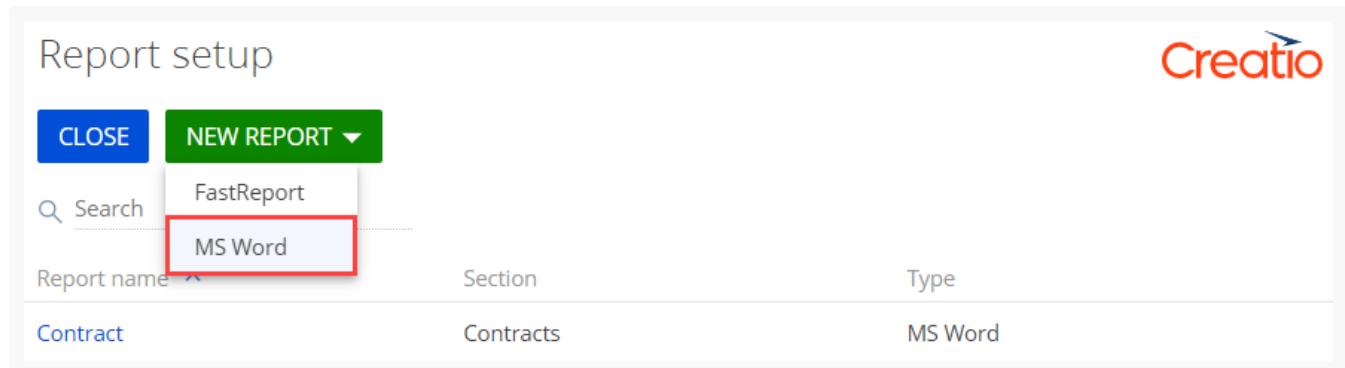
## Создать простой отчет MS Word

- Установите Creatio MS Word Report Designer (выполняется однократно). Для этого воспользуйтесь инструкцией, которая приведена в статье [Установить плагин Creatio для Word](#).

**Плагин MS Word (Creatio MS Word Report Designer)** — программный модуль, который динамически подключается к Creatio. **Назначение** плагина MS Word — формирование шаблона отчета MS Word на основе данных Creatio в формате \*.docx.

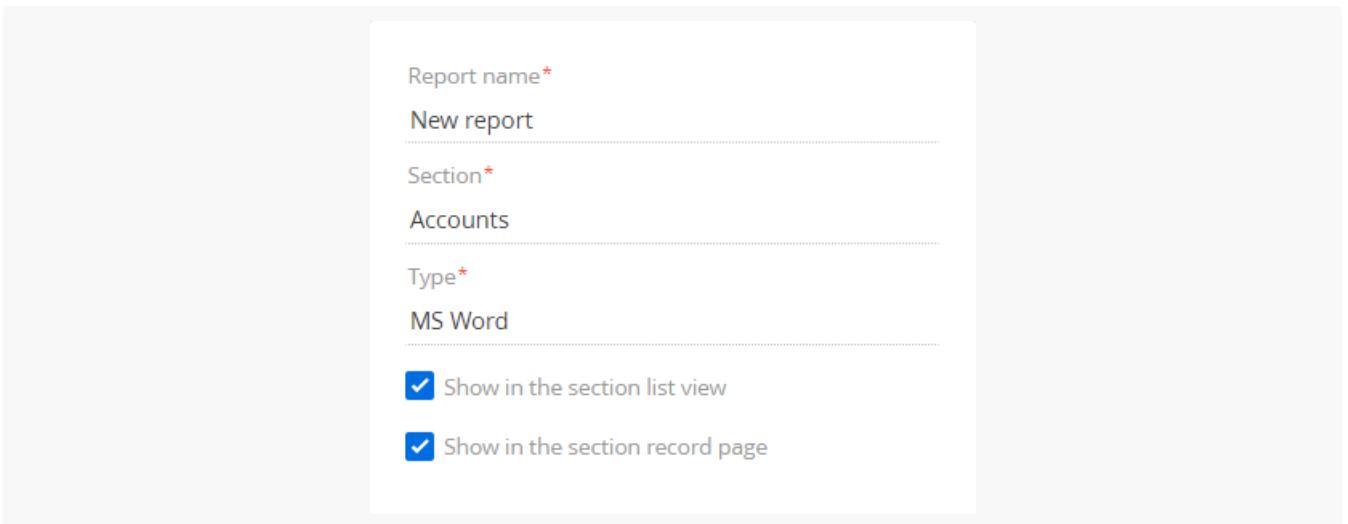
- Создайте **отчет MS Word**.

- Перейдите в дизайнер системы по кнопке . В блоке [ Настройка системы ] ([ System setup ]) перейдите по ссылке [ Настройка отчетов ] ([ Report setup ]).
- Выполните действие [ Добавить отчет ] —>[ MS Word ] ([ New report ]) —>[ MS Word ].



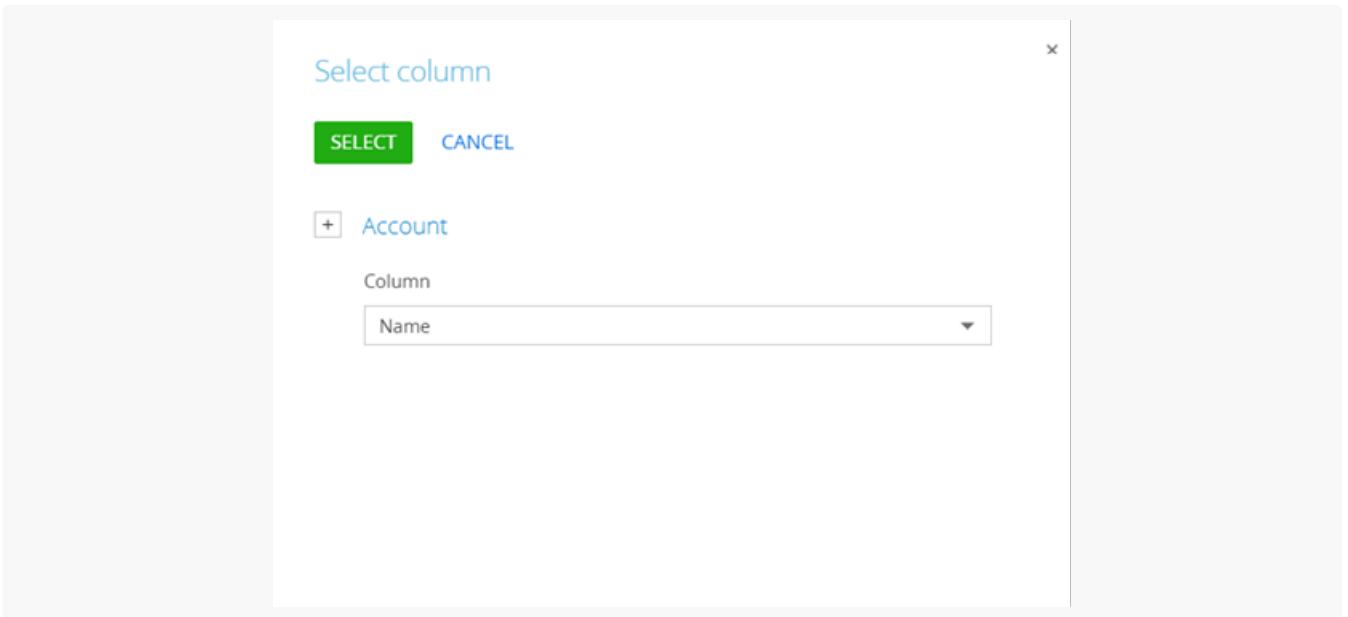
- На панели свойств заполните **свойства отчета**:

- [ Название отчета ] ([ Report name ]) — пользовательское название отчета (обязательное свойство). Отображается на соответствующей панели инструментов, которая зависит от установленных признаков [ Отображать в разделе ] ([ Show in the section list view ]) и [ Отображать на странице записи ] ([ Show in the section record page ]).
- [ Раздел ] ([ Section ]) — выберите раздел, из которого планируется генерировать отчет (обязательное свойство). Например, чтобы отобразить отчет в разделе [ Контрагенты ] ([ Accounts ]), в выпадающем списке выберите соответствующий раздел.
- [ Тип ] ([ Type ]) — тип отчета (обязательное свойство). Заполняется автоматически и недоступно для редактирования.
- Признак [ Отображать в разделе ] ([ Show in the section list view ]) — указывает необходимость отображения отчета в выпадающем меню кнопки [ Печать ] ([ Print ]) панели инструментов раздела.
- Признак [ Отображать на странице записи ] ([ Show in the section record page ]) — указывает необходимость отображения отчета в выпадающем меню кнопки [ Печать ] ([ Print ]) панели инструментов страницы записи.



### 3. Настройте **поля отчета**.

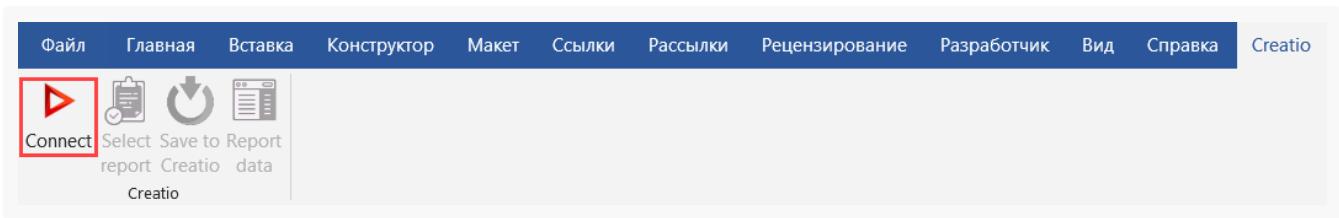
- В блоке [ *Настройте поля отчета* ] ([ *Set up report data* ]) рабочей области страницы настройки отчета нажмите .
- В выпадающем списке [ *Колонка* ] ([ *Column* ]) выберите колонку, которую планируется добавить в отчет. Например, чтобы в отчет добавить колонку [ *Название* ] ([ *Name* ]), в выпадающем списке выберите соответствующую колонку.



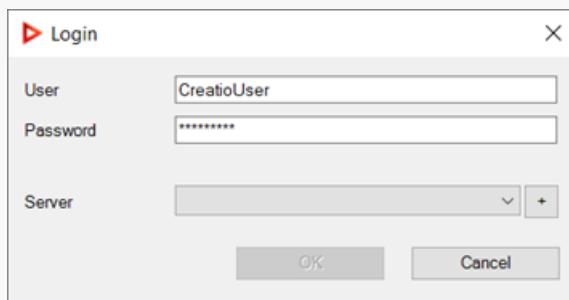
- Нажмите [ *Выбрать* ] ([ *Select* ]).
- На панели инструментов страницы настройки отчета нажмите [ *Применить* ] ([ *Apply* ]).

### 4. Настройте **шаблон отчета**.

- Запустите приложение MS Word.
- На вкладке плагина Creatio панели инструментов нажмите кнопку [ *Connect* ].

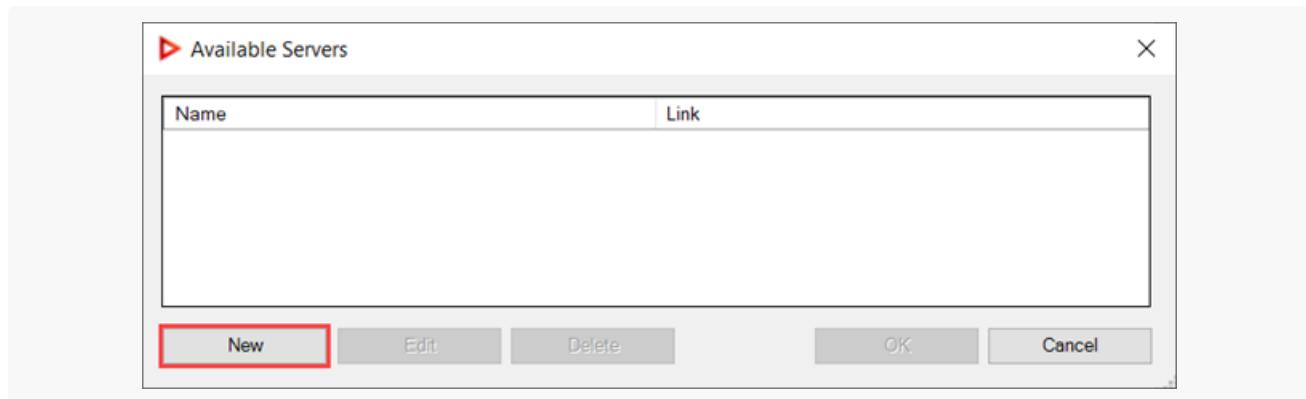


с. Введите логин и пароль пользователя в Creatio.



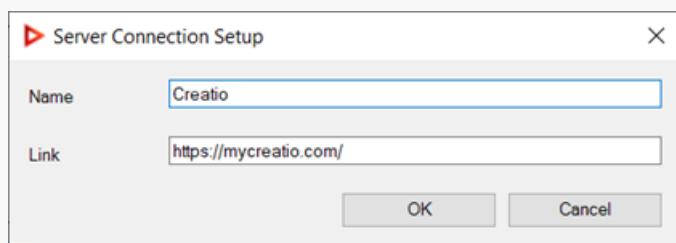
д. Подключите **сервер приложения** Creatio.

- Возле поля [ Server ] нажмите
- В окне [ Available Servers ] нажмите [ New ].



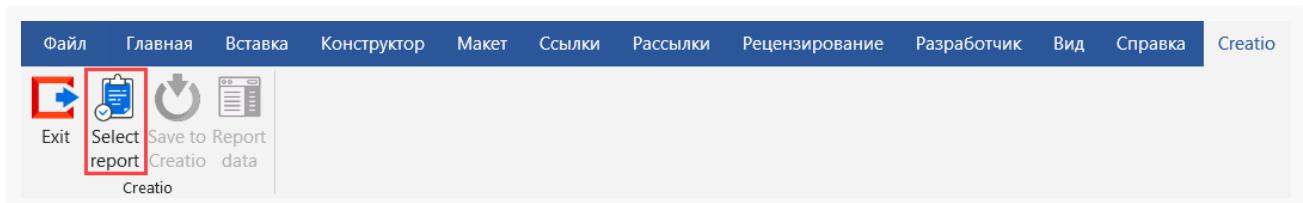
е. Заполните **свойства сервера**:

- [ Name ] — пользовательское имя сервера.
- [ Link ] — адрес сервера приложения Creatio, в котором выполнялась настройка отчета.



ж. В окне [ Server Connection Setup ] нажмите [ OK ] для сохранения параметров подключения сервера Creatio.

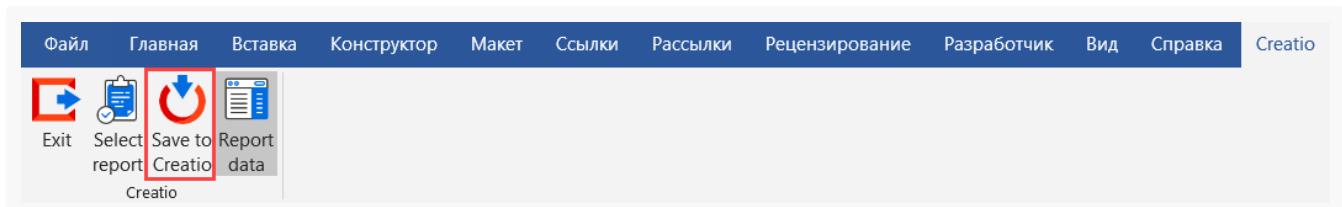
- g. В окне [ Available Servers ] выберите сервер приложения Creatio, в котором выполнялась настройка отчета (опционально), и нажмите [ OK ].
- e. В окне [ Login ] нажмите [ OK ] для подключения к выбранному серверу Creatio.
- f. Выберите **отчет MS Word**, который планируется настроить.
  - a. На вкладке панели инструментов нажмите кнопку [ Select report ].



- b. В окне [ Creatio Word reports ] выберите отчет, шаблон которого планируется настроить.
- c. В окне [ Creatio Word reports ] нажмите [ OK ] для выбора отчета.

В результате открывается панель [ Word report data ] выбранного отчета.

- g. Настройте шаблон отчета. Для этого воспользуйтесь инструкцией, которая приведена в статье [Настроить отчет в плагине Word и загрузить в Creatio](#).
- h. Загрузите настроенный шаблон отчета в Creatio. Для этого на вкладке панели инструментов нажмите кнопку [ Save to Creatio ].



В результате настроенный шаблон отчета загружен в Creatio и отображается на панели свойств.

## Создать отчет MS Word с использованием базовых макросов

1. Выполните шаги 1-3 [алгоритма создания простого отчета MS Word](#).
2. К колонке отчета добавьте **тег макроса**.

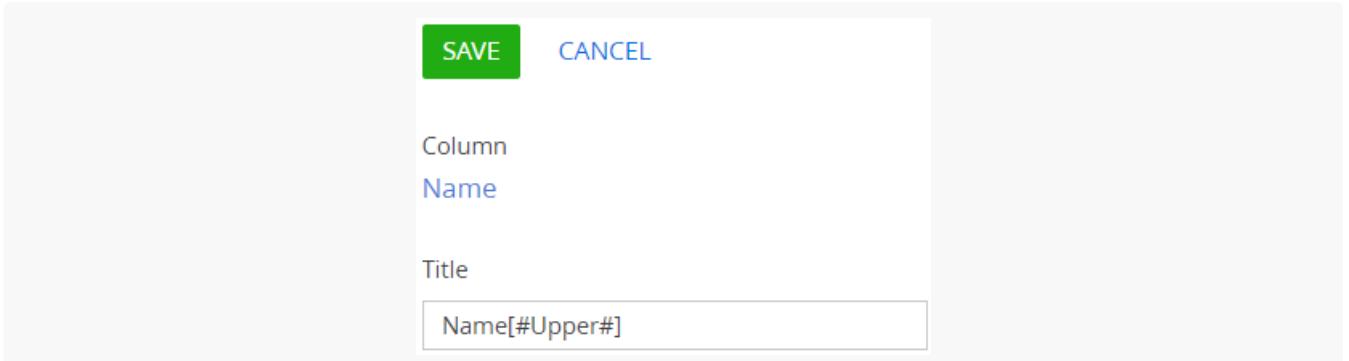
- a. Откройте страницу настройки колонки, к которой планируется добавить тег макроса.

**Способы** открытия страницы настройки колонки:

- В блоке [ Настройте поля отчета ] ([ Set up report data ]) рабочей области страницы настройки отчета дважды кликните по колонке, к которой необходимо добавить тег макроса.
- В блоке [ Настройте поля отчета ] ([ Set up report data ]) рабочей области страницы настройки отчета в строке колонки, к которой необходимо добавить тег макроса, нажмите .

- d. Измените значение поля [ Заголовок ] ([ Title ]) на "НазваниеКолонки[#ИмяМакроса#]".

— тег макроса, который выполняет необходимое действие. Для корректной работы макроса название колонки объекта (поле [ Колонка ] ([ Column ])) должно совпадать с названием колонки с макросом (поле [ Заголовок ] ([ Title ])). Например, для колонки [ Name ] поле [ Заголовок ] ([ Title ]) должно содержать значение `Name[#ИмяМакроса#]` (например, `Name[#Upper#]`, где `#Upper#` — тег макроса, который конвертирует значение строки в верхний регистр).



- e. На панели инструментов страницы настройки колонки нажмите [ Сохранить ] ([ Save ]).
3. Выполните шаг 4 [алгоритма создания простого отчета MS Word](#).

## Создать отчет MS Word с использованием пользовательских макросов

1. Выполните шаг 1 [алгоритма создания простого отчета MS Word](#).
2. Реализуйте **пользовательские макросы**.
  - a. Создайте схему типа [ Исходный код ] ([ Source code ]). Для этого воспользуйтесь инструкцией, которая приведена в статье [Исходный код](#).
  - b. В дизайнере исходного кода реализуйте **класс макроса**.
    - В классе макроса реализуйте интерфейс `IExpressionConverter`. Базовый интерфейс `IExpressionConverter` реализован в схеме `ExpressionConverterHelper` пакета `NUI`.
    - Для макроса добавьте атрибут `ExpressionConverterAttribute` с указанием имени макроса. Этот атрибут позволяет вызвать пользовательский макрос из шаблона отчета.
 Пример атрибута приведен ниже.

### Пример атрибута `CurrentUser`

```
[ExpressionConverterAttribute("CurrentUser")]
```

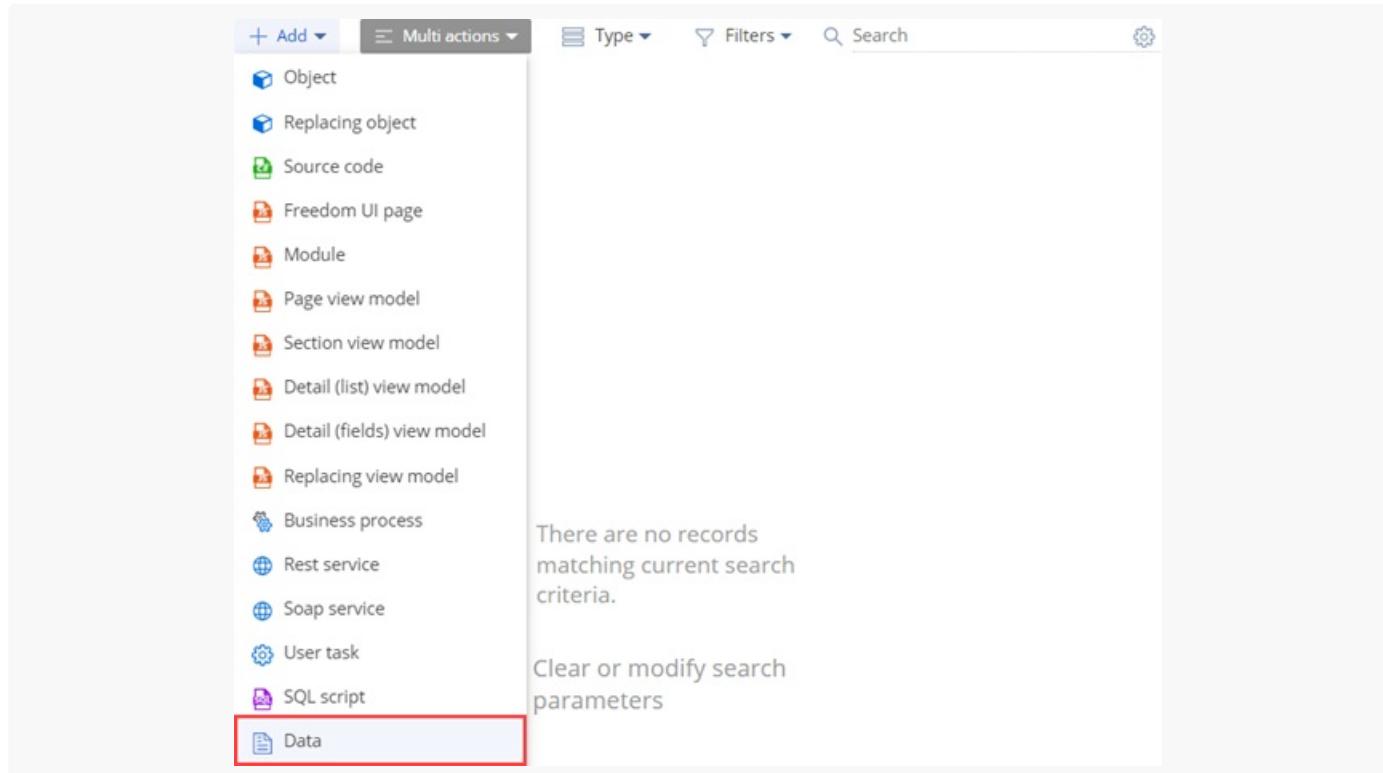
- Реализуйте метод интерфейса `Evaluate(object value, string arguments = "")`. В качестве параметра принимает значение поля из шаблона отчета MS Word. Возвращает значение типа `string`, которое подставляется на место этого поля в сформированном отчете.

3. Выполните шаги 2-4 [алгоритма создания простого отчета MS Word](#). На шаге 2 в список полей отчета

добавьте колонку [ *Id* ], которая является входящим параметром для пользовательского макроса.

## Привязать отчет MS Word к пакету

- Перейдите в раздел [ Конфигурация ] ([ Configuration ]) и выберите пользовательский [пакет](#), в который будет добавлена схема.
- На панели инструментов реестра раздела нажмите [ Добавить ] —> [ Данные ] ([ Add ] —> [ Data ]).



- Выполните привязку данных. Для этого воспользуйтесь инструкцией, которая приведена в статье [Привязать данные к пакету](#).

**Элементы**, привязку которых необходимо выполнить:

- `SysModuleReport_ReportName` — отчет. Подключается по идентификатору отчета (колонка [ *Id* ] таблицы [ `SysModuleReport` ] базы данных).
- `SysModuleReportTable_ReportName` — табличная часть отчета. Подключается по идентификатору отчета (колонка [ *Id* ] таблицы [ `SysModuleReportTable` ] базы данных).

## Создать отчет MS Word (базовые макросы)

Легкий

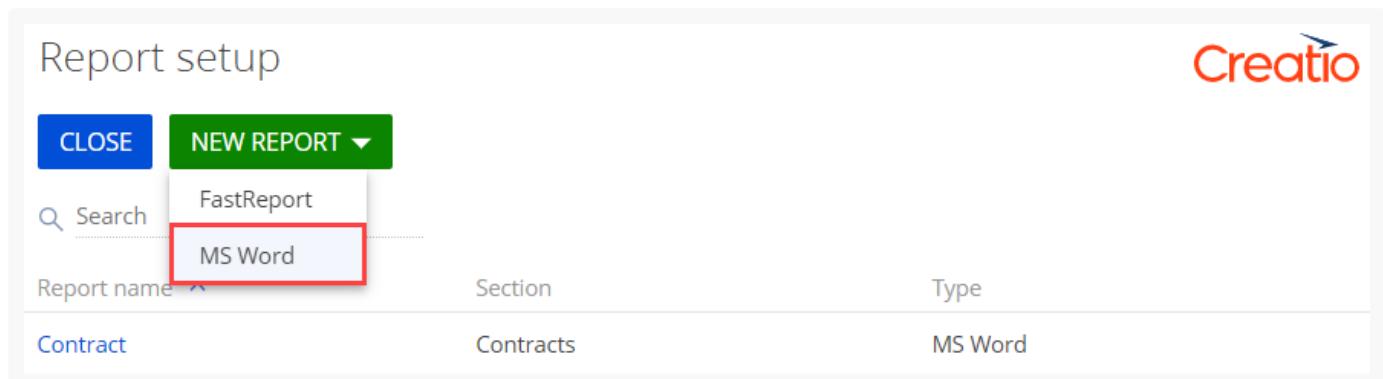
**Пример.** На страницу раздела [ Контрагенты ] ([ Accounts ]) и страницу контрагента добавить отчет [ Информация контрагента ] ([ Account Info ]).

**Поля**, которые содержит отчет [ Информация контрагента ] ([ Account Info ]):

- [ Название ] ([ Name ]) — название контрагента.
- [ Тип ] ([ Type ]) — тип контрагента.
- [ Основной контакт ] ([ Primary contact ]) — основной контакт контрагента.

## 1. Создать отчет

1. Перейдите в дизайнер системы по кнопке . В блоке [ Настройка системы ] ([ System setup ]) перейдите по ссылке [ Настройка отчетов ] ([ Report setup ]).
2. Выполните действие [ Добавить отчет ] —>[ MS Word ] ([ New report ] —>[ MS Word ]).



### 3. На панели свойств заполните **свойства отчета**:

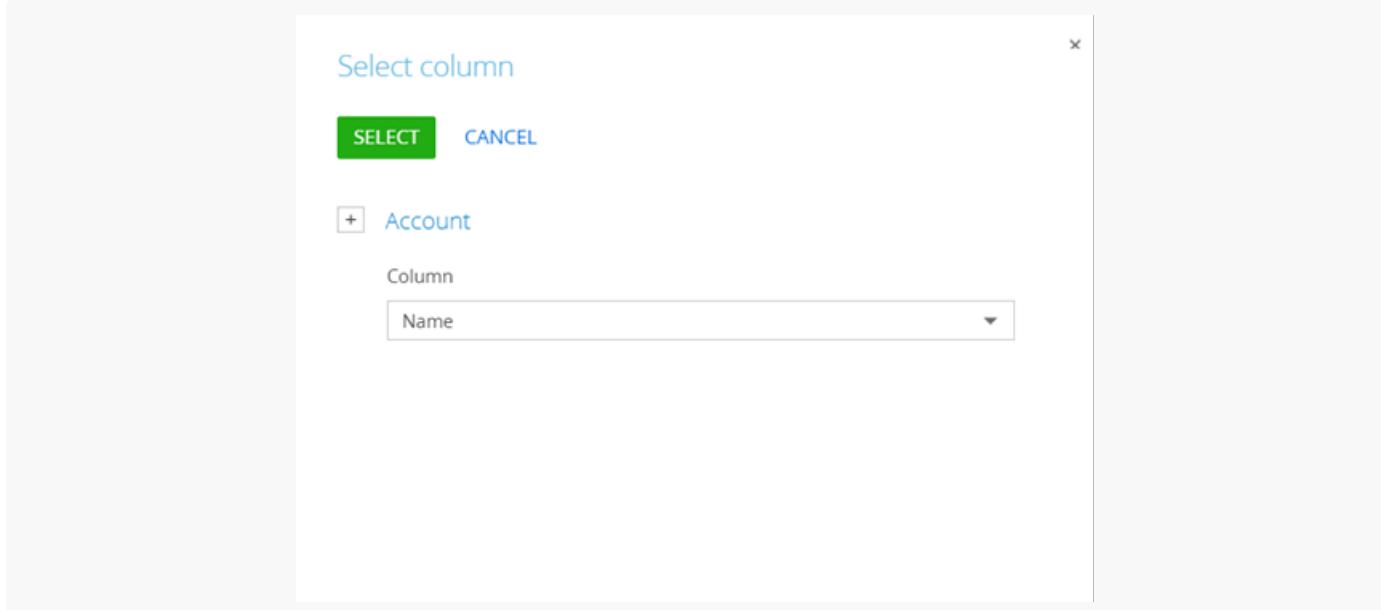
- [ Название отчета ] ([ Report name ]) — "Информация контрагента" ("Account Info").
- [ Раздел ] ([ Section ]) — выберите "Контрагенты" ("Accounts").
- Установите признак [ Отображать в разделе ] ([ Show in the section list view ]).
- Установите признак [ Отображать на странице записи ] ([ Show in the section record page ]).

The screenshot shows the 'Report properties' configuration dialog. It contains the following fields and settings:

- Report name\*: Account Info
- Section\*: Accounts
- Type\*: MS Word
- Show in the section list view
- Show in the section record page

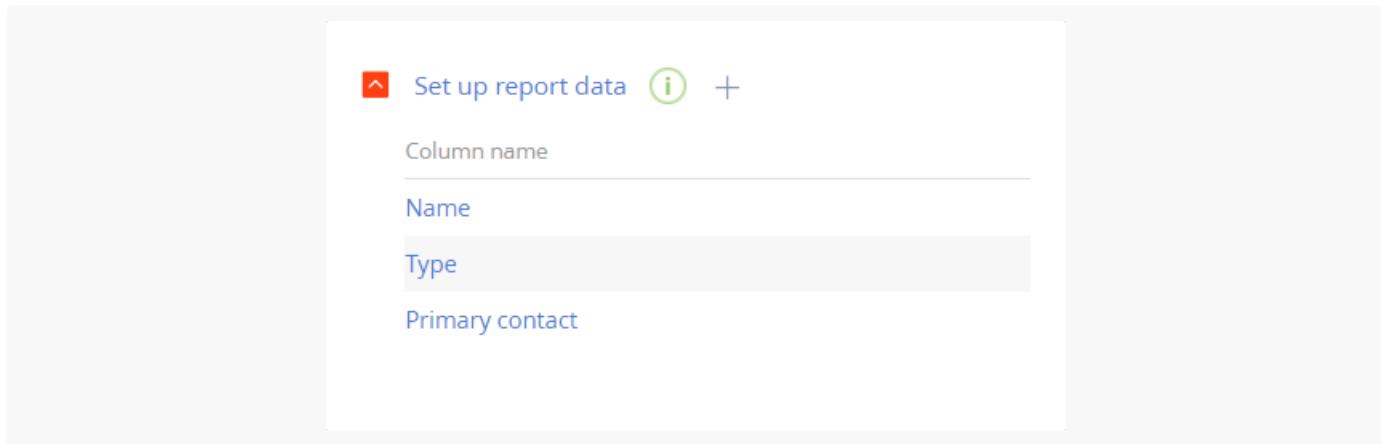
## 2. Настроить поля отчета

1. В блоке [ Настройте поля отчета ] ([ Set up report data ]) рабочей области страницы настройки отчета нажмите .
2. В выпадающем списке [ Колонка ] ([ Column ]) выберите колонку [ Название ] ([ Name ]).



3. Нажмите [ Выбрать ] ([ Select ]). Далее к этой колонке необходимо добавить тег базового макроса.
4. Аналогично добавьте колонки [ Тип ] ([ Type ]) и [ Основной контакт ] ([ Primary contact ]).

В результате в отчет добавлены поля, которые представлены на рисунке ниже.



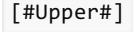
5. К колонке [ Название ] ([ Name ]) добавьте **тег макроса**.

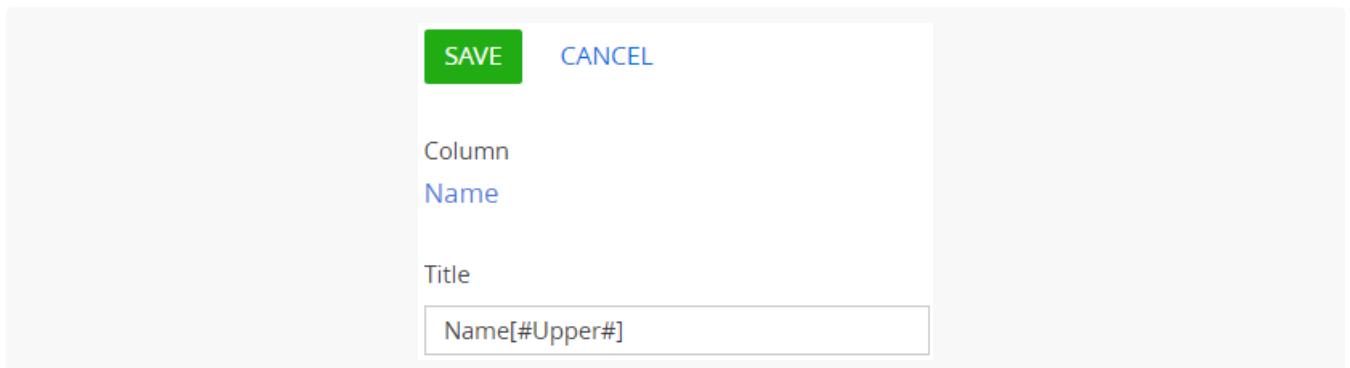
- a. Откройте страницу настройки колонки [ Название ] ([ Name ]).

**Способы** открытия страницы настройки колонки:

- В блоке [ Настройте поля отчета ] ([ Set up report data ]) рабочей области страницы настройки отчета дважды кликните по колонке, к которой необходимо добавить тег макроса.
- В блоке [ Настройте поля отчета ] ([ Set up report data ]) рабочей области страницы настройки

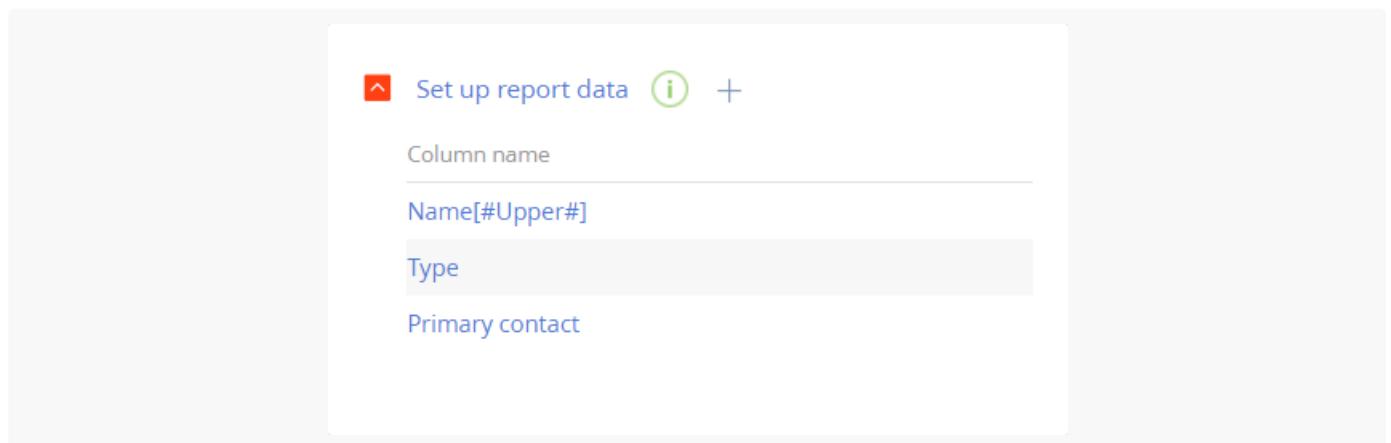
отчета в строке колонки, к которой необходимо добавить тег макроса, нажмите .

- d. Измените значение поля [ Заголовок ] ([ Title ]) на "Название[#Upper#]" ("Name[#Upper#]").  — тег макроса, который конвертирует значение строки в верхний регистр.



- e. На панели инструментов страницы настройки колонки нажмите [ Сохранить ] ([ Save ]).

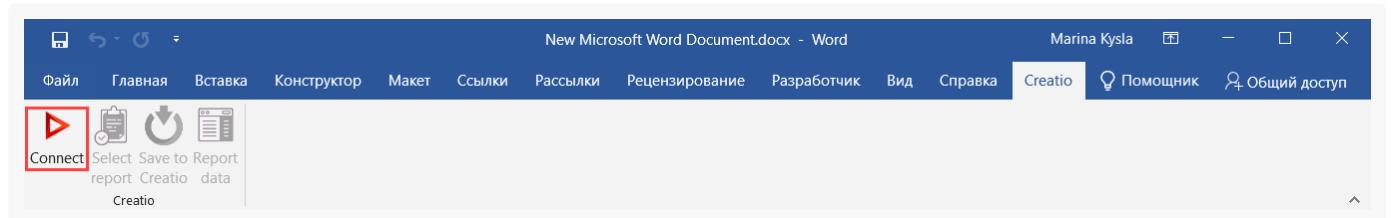
В результате в отчет добавлены поля, которые представлены на рисунке ниже.



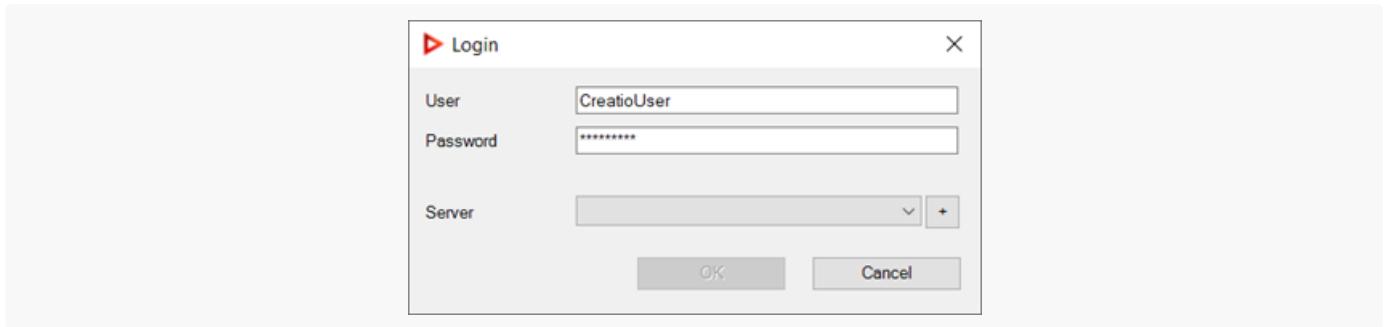
6. На панели инструментов страницы настройки отчета нажмите [ Применить ] ([ Apply ]).

### 3. Настроить шаблон отчета

1. Запустите приложение MS Word.
2. На вкладке панели инструментов Creatio панели инструментов нажмите кнопку [ Connect ].

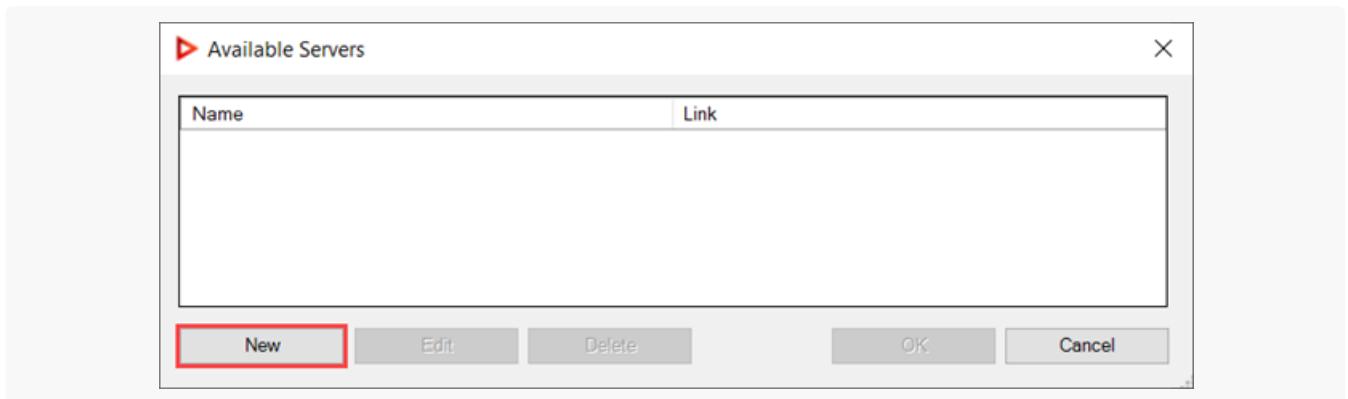


3. Введите логин и пароль пользователя в Creatio.



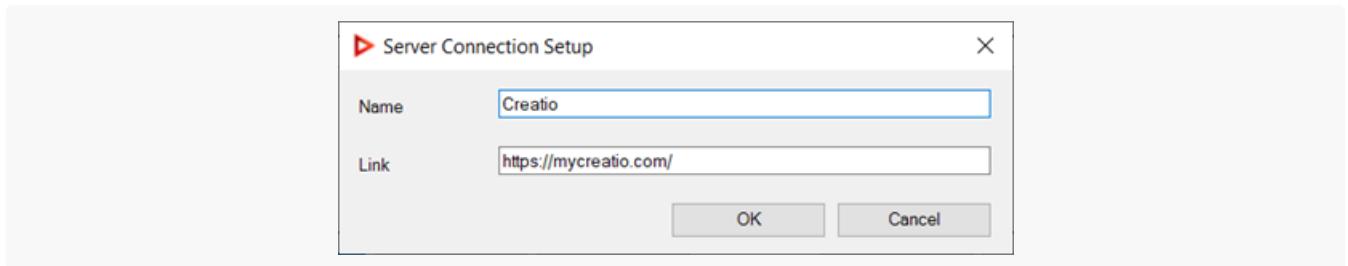
4. Подключите **сервер приложения** Creatio.

- Возле поля [ Server ] нажмите **[ + ]**.
- В окне [ Available Servers ] нажмите [ New ].



c. Заполните **свойства сервера**:

- [ Name ] — пользовательское имя сервера.
- [ Link ] — адрес сервера приложения Creatio, в котором выполнялась настройка отчета.

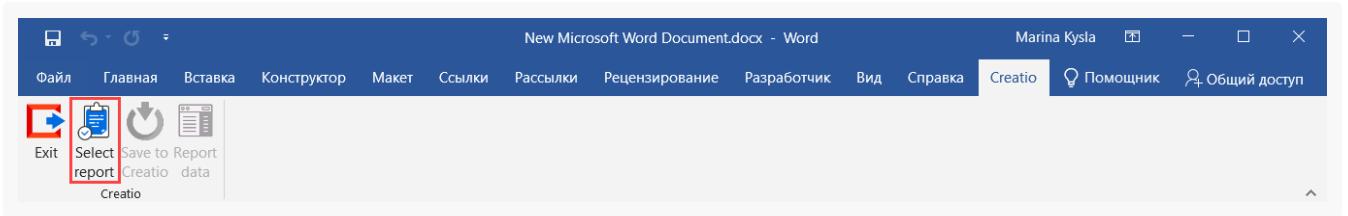


- В окне [ Server Connection Setup ] нажмите [ OK ] для сохранения параметров подключения сервера Creatio.
- В окне [ Available Servers ] выберите сервер приложения Creatio, в котором выполнялась настройка отчета (опционально), и нажмите [ OK ].

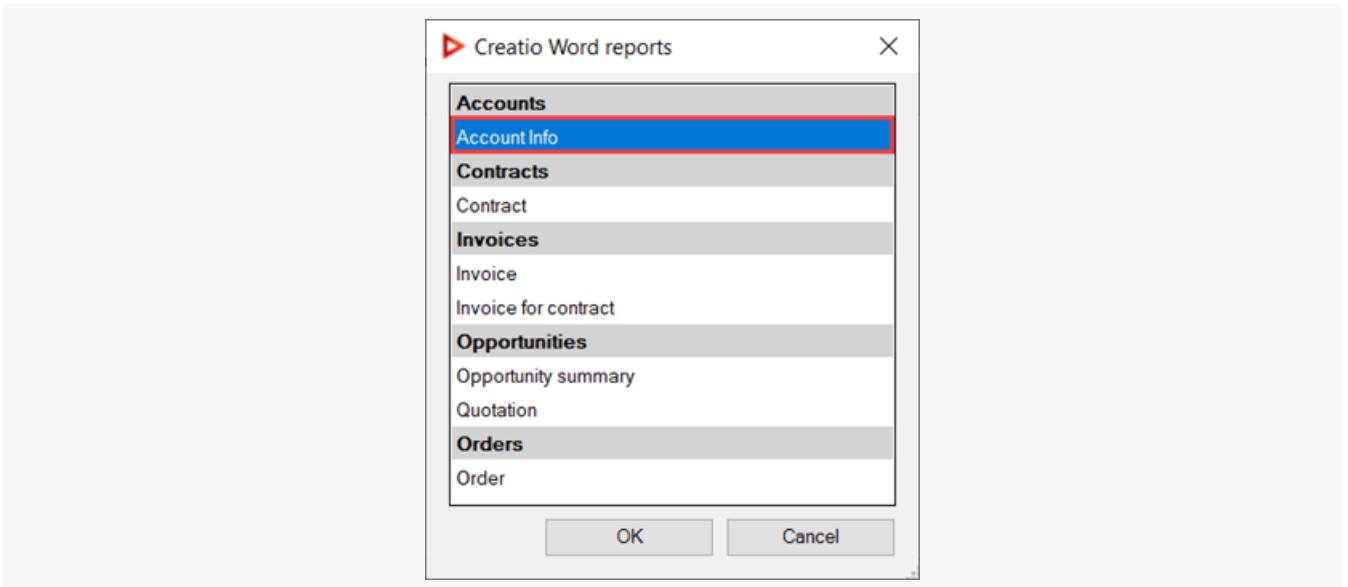
5. В окне [ Login ] нажмите [ OK ] для подключения к выбранному серверу Creatio.

6. Выберите **отчет MS Word**, который планируется настроить.

- На вкладке панели инструментов Creatio нажмите кнопку [ Select report ].

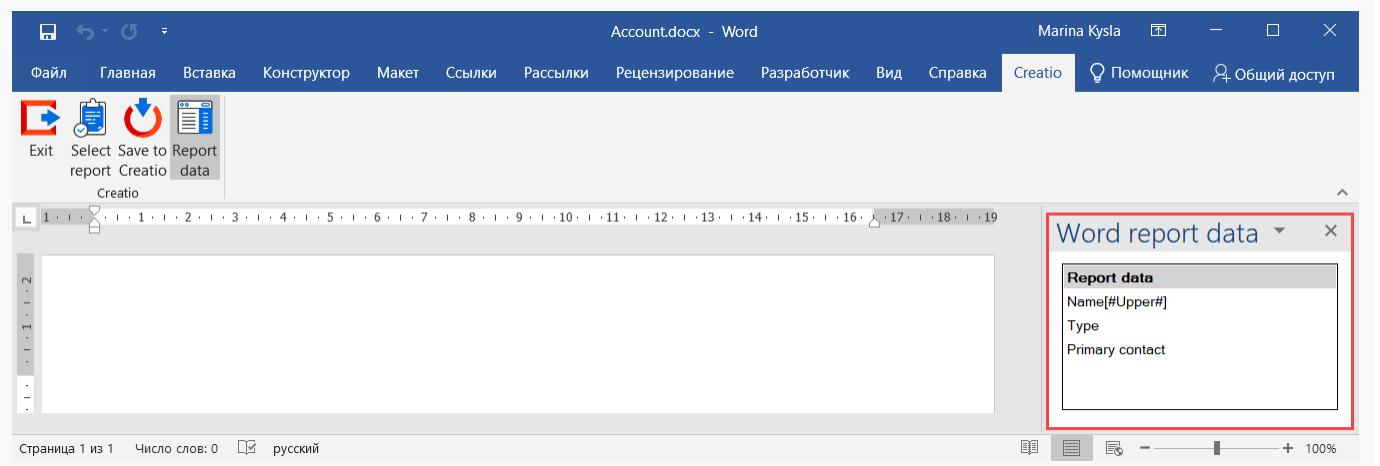


б. В окне [ *Creatio Word reports* ] выберите отчет [ *Информация контрагента* ] ([ *Account Info* ]).



с. В окне [ *Creatio Word reports* ] нажмите [ *OK* ] для выбора отчета.

В результате открыта панель [ *Word report data* ] отчета [ *Информация контрагента* ] ([ *Account Info* ]), который представлен на рисунке ниже.



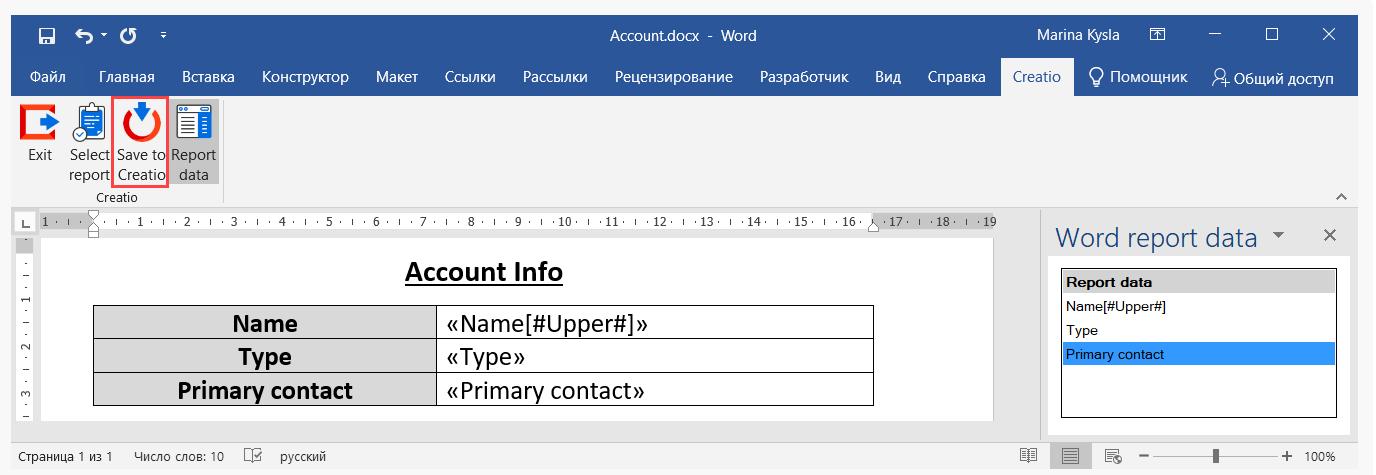
7. Настройте шаблон отчета. Для этого воспользуйтесь инструкцией, которая приведена в статье [Настроить отчет в плагине Word и загрузить в Creatio](#).

Настроенный шаблон отчета [ *Информация контрагента* ] ([ *Account Info* ]) приведен на рисунке ниже.

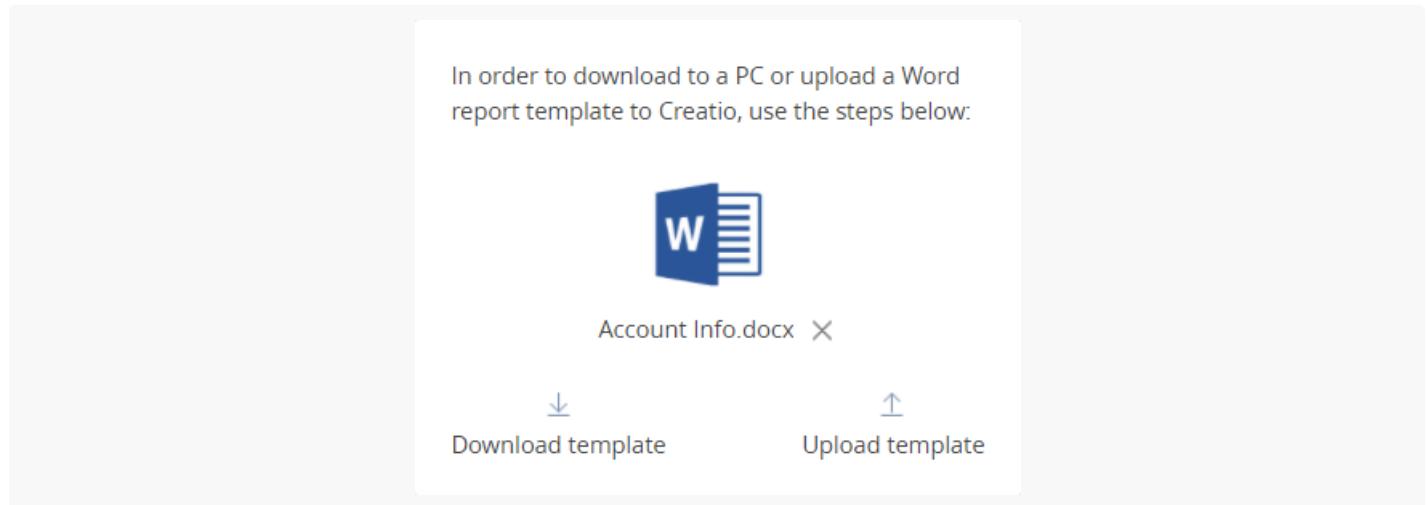
## Account Info

<b>Name</b>	«Name[#Upper#]»
<b>Type</b>	«Type»
<b>Primary contact</b>	«Primary contact»

8. Загрузите настроенный шаблон отчета [ Информация контрагента ] ([ Account Info ]) в Creatio. Для этого на вкладке плагина Creatio панели инструментов нажмите кнопку [ Save to Creatio ].



В результате настроенный шаблон отчета [ Информация контрагента ] ([ Account Info ]) загружен в Creatio и отображается на панели свойств.



## Результат выполнения примера

Чтобы **посмотреть результат выполнения примера**, обновите страницу раздела [ Контрагенты ] ([ Accounts ]).

В результате выполнения примера на страницу раздела [ Контрагенты ] ([ Accounts ]) добавлен отчет [ Информация контрагента ] ([ Account Info ]). Отчет доступен в выпадающем меню кнопки [ Печать ]

([ Print ]) панели инструментов раздела.

The screenshot shows the 'Accounts' module interface. At the top right, there is a search bar with the placeholder 'What can I do for you?' and a red box highlighting the 'PRINT' and 'VIEW' dropdown menus. Below the menu, a sub-menu 'Account Info' is also highlighted with a red box. The main content area displays a contact record for 'Vertigo Systems'. The contact details are as follows:

	<b>Vertigo Systems</b>	Web www.vertigosys.com	Primary phone <a href="#">+44 (20) 3427 1374</a>	Type Customer
	Primary contact Peter Moore	Address 83 Ashton Street	City London	Country United Kingdom

At the bottom of the contact card, there are buttons for 'OPEN', 'COPY', 'DELETE', and 'PRINT'.

Отчет [ Информация контрагента ] ([ Account Info ]) имеет вид, который представлен на рисунке ниже.

The screenshot shows a report titled 'Account Info' with the following data:

Name	VERTIGO SYSTEMS
Type	Customer
Primary contact	Peter Moore

В результате применения макроса название компании в колонке [ Название ] ([ Name ]) отображено в верхнем регистре.

Если в реестре раздела [ Контрагенты ] ([ Accounts ]) не выбрана запись, то отчет [ Информация контрагента ] ([ Account Info ]) не активен в выпадающем меню кнопки [ Печать ] ([ Print ]) панели инструментов.

The screenshot shows the 'Accounts' module interface with a contact record for 'Vertigo Systems' selected. The 'PRINT' and 'VIEW' dropdown menus at the top right are highlighted with a red box. The contact details are identical to the previous screenshot:

	<b>Vertigo Systems</b>	Web www.vertigosys.com	Primary phone <a href="#">+44 (20) 3427 1374</a>	Type Customer
	Primary contact Peter Moore	Address 83 Ashton Street	City London	Country United Kingdom

В результате выполнения примера на страницу контрагента также добавлен отчет [ Информация контрагента ] ([ Account Info ]). Отчет доступен в выпадающем меню кнопки [ Печать ] ([ Print ]) панели инструментов страницы контрагента.

The screenshot shows the Creatio application interface. At the top, there is a search bar with the placeholder "What can I do for you?" and a "CLOSE" button. On the right side, the "Creatio" logo is visible with the version number "8.0.0.5434". Below the search bar, there are "PRINT" and "VIEW" buttons, with "Account Info" highlighted by a red box. The main content area shows a summary for "VERTIGO SYSTEMS" with a progress bar at 95% and an "Enrich data" button. A section titled "NEXT STEPS (0)" contains a message: "You don't have any tasks yet" with a note to "Press F above to add a task". Below this, there are tabs for "ACCOUNT INFO" (which is selected and highlighted in red), "CONTACTS AND STRUCTURE", "MAINTENANCE", "TIMELINE", and "CONNECTED TO". Under the "ACCOUNT INFO" tab, it shows "Also known as" and "Code 109".

# Создать отчет MS Word (пользовательские макросы)

Сложный

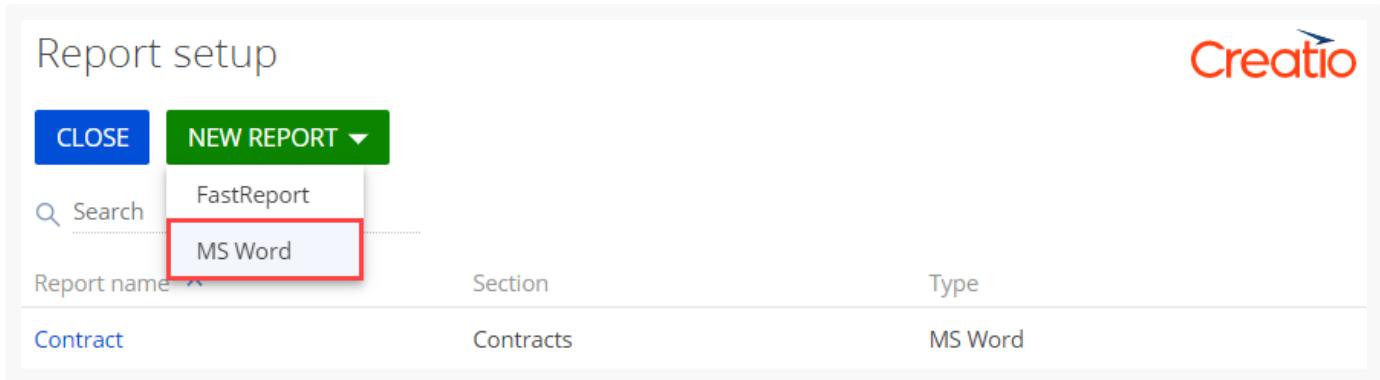
**Пример.** На страницу контрагента добавить отчет [ Информация контрагента ] ([ Account Summary ]).

**Поля**, которые содержит отчет [ Информация контрагента ] ([ Account Info ]):

- [ Название ] ([ Name ]) — название контрагента.
- [ Тип ] ([ Type ]) — тип контрагента.
- [ Основной контакт ] ([ Primary contact ]) — основной контакт контрагента.
- [ Дополнительная информация ] ([ Additional info ]) — дополнительная информация о контрагенте. Для контрагента типа [ Клиент ] ([ Customer ]) отображается годовой оборот, для контрагента типа [ Партнер ] ([ Partner ]) — количество сотрудников.
- Дата формирования отчета.
- Имя сотрудника, который сформировал отчет.

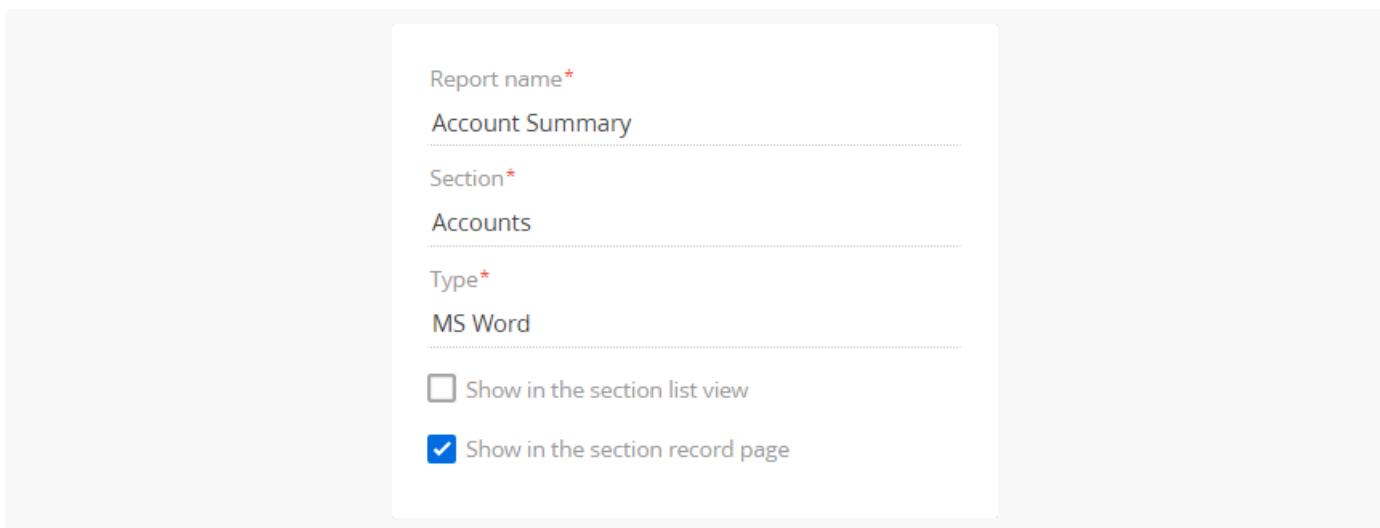
## 1. Создать отчет

1. Перейдите в дизайнер системы по кнопке . В блоке [ Настройка системы ] ([ System setup ]) перейдите по ссылке [ Настройка отчетов ] ([ Report setup ]).
2. Выполните действие [ Добавить отчет ] —>[ MS Word ] ([ New report ]) —>[ MS Word ].



3. На панели свойств заполните **свойства отчета**:

- [ Название отчета ] ([ Report name ]) — "Информация контрагента" ("Account Summary").
- [ Раздел ] ([ Section ]) — выберите "Контрагенты" ("Accounts").
- Установите признак [ Отображать на странице записи ] ([ Show in the section record page ]).



Для применения изменений свойств нажмите [ Применить ] ([ Apply ]).

## 2. Реализовать пользовательские макросы

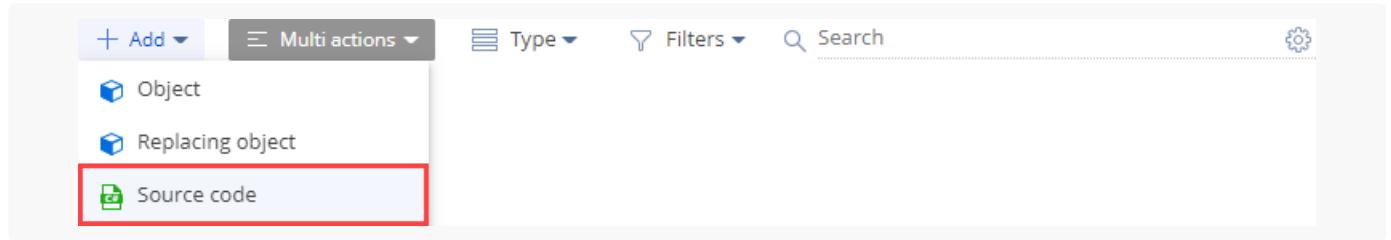
**Макросы**, которые необходимо реализовать:

- Макрос для получения дополнительной информации контрагента, которая зависит от его типа.
- Макрос для получения даты формирования отчета.
- Макрос для получения имени сотрудника, который сформировал отчет.

### 1. Реализовать макрос (дополнительная информация контрагента)

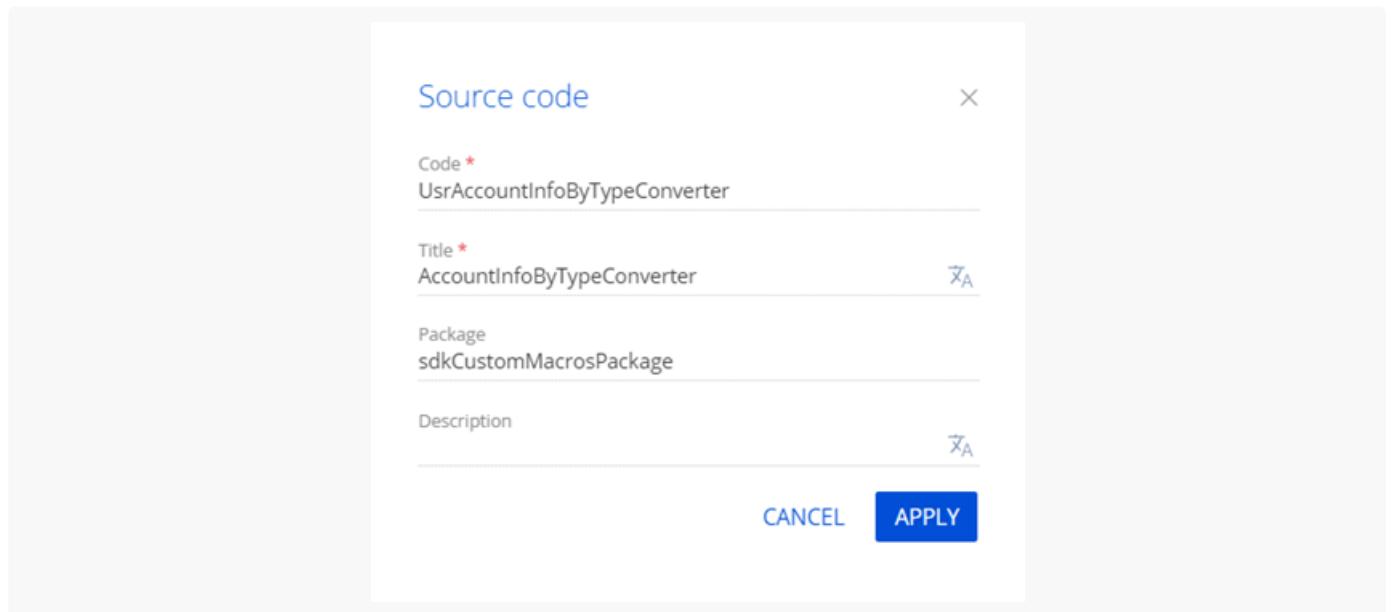
1. Перейдите в раздел [ Конфигурация ] ([ Configuration ]) и выберите пользовательский пакет, в который будет добавлена схема.
2. На панели инструментов реестра раздела нажмите [ Добавить ] —> [ Исходный код ] ([ Add ] —>

[ Source code ]).



3. В дизайнере исходного кода заполните **свойства схемы**:

- [ Код ] ([ Code ]) — "UsrAccountInfoByTypeConverter".
- [ Заголовок ] ([ Title ]) — "AccountInfoByTypeConverter".



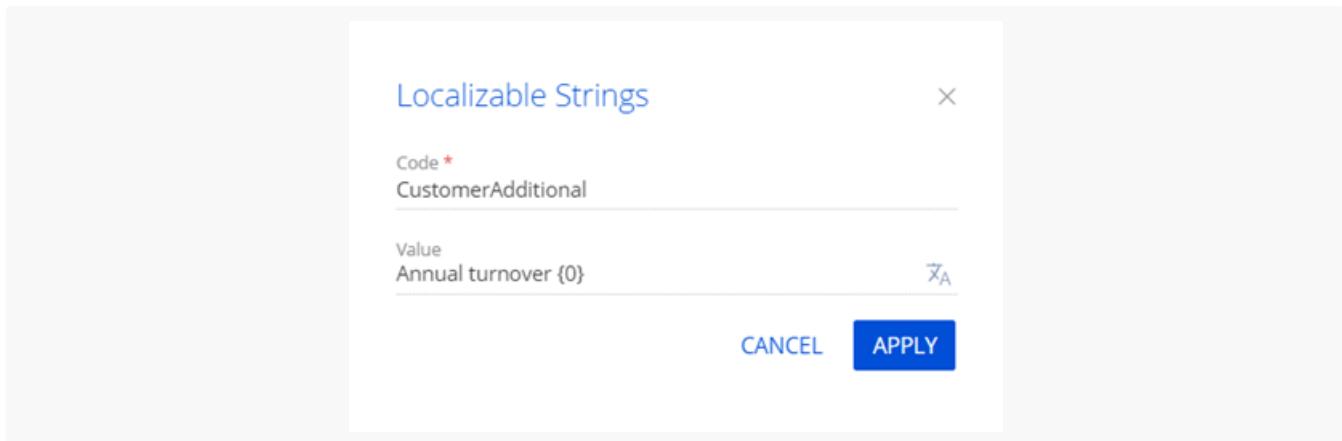
Для применения изменений свойств нажмите [ Применить ] ([ Apply ]).

4. Добавьте **локализуемую строку**, которая содержит тип [ Клиент ] ([ Customer ]) контрагента.

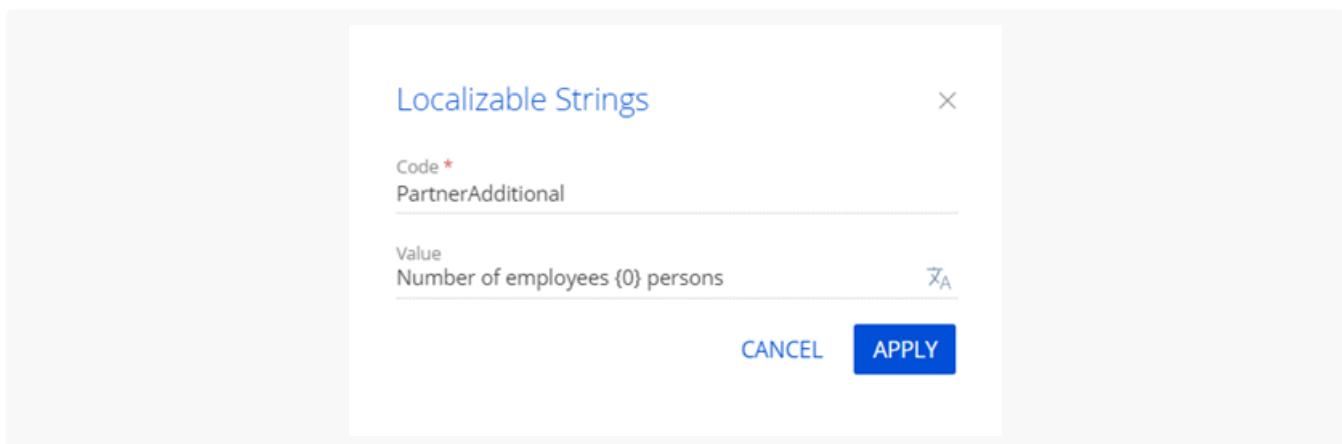
a. В контекстном меню узла [ Локализуемые строки ] ([ Localizable strings ]) нажмите кнопку **+**.

b. Заполните **свойства локализуемой строки**:

- [ Код ] ([ Code ]) — "CustomerAdditional".
- [ Значение ] ([ Value ]) — "Годовой оборот {0}" ("Annual turnover {0}").



- e. Для добавления локализуемой строки нажмите [ Добавить ] ([ Add ]).
5. Добавьте **локализуемую строку**, которая содержит тип [ Партнер ] ([ Partner ]) контрагента.
  - a. В контекстном меню узла [ Локализуемые строки ] ([ Localizable strings ]) нажмите кнопку .
  - b. Заполните **свойства локализуемой строки**:
    - [ Код ] ([ Code ]) — "PartnerAdditional".
    - [ Значение ] ([ Value ]) — "Количество сотрудников {0} чел." ("Number of employees {0} persons").



- e. Для добавления локализуемой строки нажмите [ Добавить ] ([ Add ]).
  6. В дизайнере исходного кода реализуйте класс макроса для получения дополнительной информации контрагента, которая зависит от его типа.
- Исходный код схемы типа [ Исходный код ] ([ Source code ]) представлен ниже.

```
UserAccountInfoByTypeConverter

namespace Terrasoft.Configuration
{
    using System;
    using System.CodeDom.Compiler;
    using System.Collections.Generic;
    using System.Data;
```

```

using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.ServiceModel.Web;
using System.ServiceModel.Activation;
using System.Text;
using System.Text.RegularExpressions;
using System.Web;
using Terrasoft.Common;
using Terrasoft.Core;
using Terrasoft.Core.DB;
using Terrasoft.Core.Entities;
using Terrasoft.Core.Packages;
using Terrasoft.Core.Factories;

/* Атрибут с именем макроса [AccountInfoByType]. */
[ExpressionConverterAttribute("AccountInfoByType")]
/* Класс реализует интерфейс IExpressionConverter. */
class AccountInfoByTypeConverter : IExpressionConverter
{
    private UserConnection _userConnection;
    private string _customerAdditional;
    private string _partnerAdditional;
    /* Вызов значений локализуемых строк. */
    private void SetResources() {
        string sourceCodeName = "UsrAccountInfoByTypeConverter";
        _customerAdditional = new LocalizableString(_userConnection.ResourceStorage, sour
        _partnerAdditional = new LocalizableString(_userConnection.ResourceStorage, sour
    }
    /* Реализация метода Evaluate интерфейса IExpressionConverter. */
    public string Evaluate(object value, string arguments = "")
    {
        try
        {
            _userConnection = (UserConnection)HttpContext.Current.Session["UserConnection"]
            Guid accountId = new Guid(value.ToString());
            return getAccountInfo(accountId);
        }
        catch (Exception err)
        {
            return err.Message;
        }
    }
    /* Метод получения дополнительной информации в зависимости от типа контрагента. В кач
    private string getAccountInfo(Guid accountId)
    {
        SetResources();
        try
        {

```

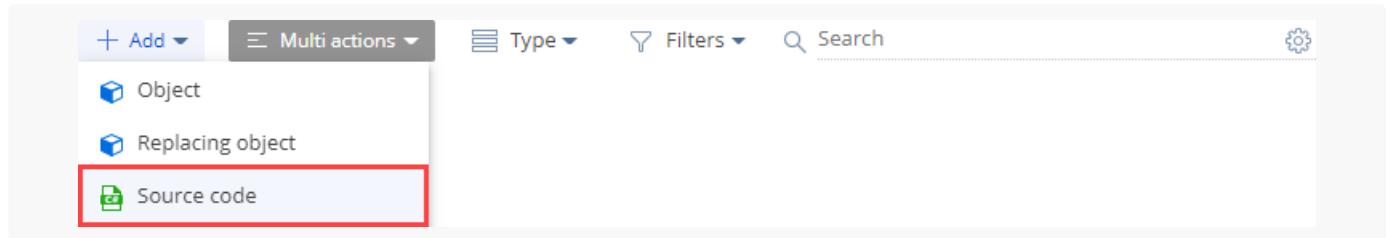
7. На панели инструментов дизайнера исходного кода нажмите [ Опубликовать ] ([ *Publish* ]) для выполнения изменений на уровне базы данных.

## 2. Реализовать макрос (дата формирования отчета)

1. Перейдите в раздел [ Конфигурация ] ([ Configuration ]) и выберите пользовательский [пакет](#), в который

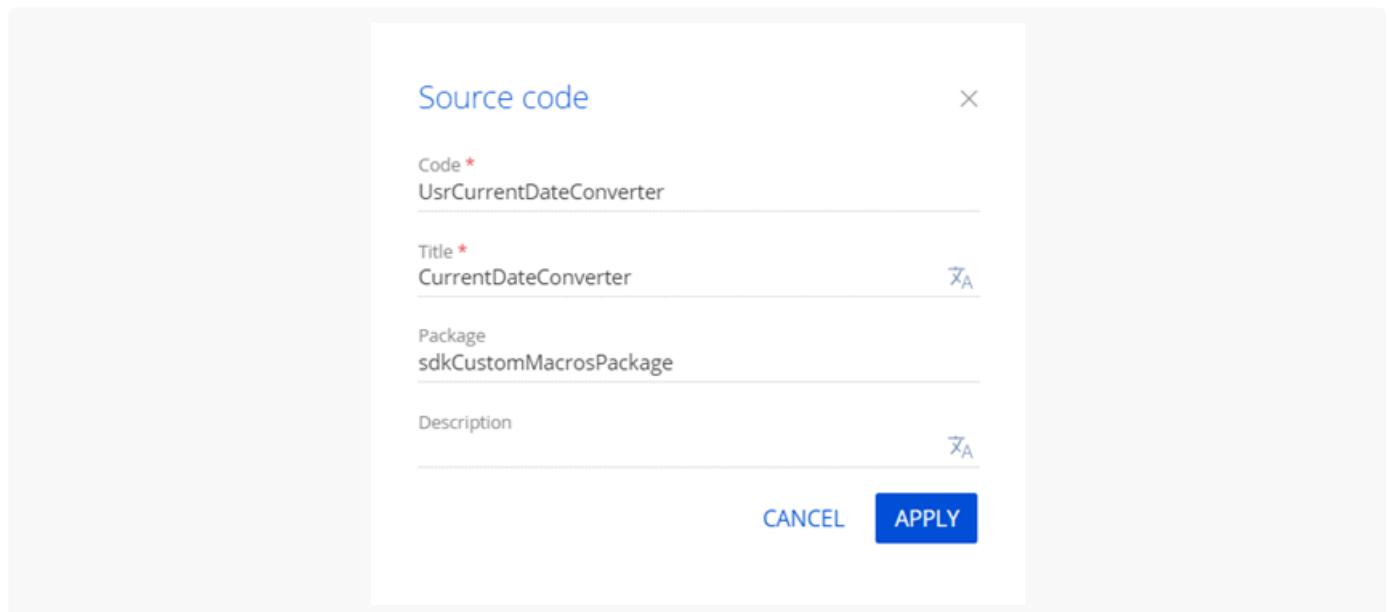
будет добавлена схема.

- На панели инструментов реестра раздела нажмите [ Добавить ] —> [ Исходный код ] ([ Add ] —> [ Source code ]).



- В дизайнере исходного кода заполните **свойства схемы**:

- [ Код ] ([ Code ]) — "UsrCurrentDateConverter".
- [ Заголовок ] ([ Title ]) — "CurrentDateConverter".



Для применения изменений свойств нажмите [ Применить ] ([ Apply ]).

- В дизайнере исходного кода реализуйте класс макроса для получения даты формирования отчета.

Исходный код схемы типа [ Исходный код ] ([ Source code ]) представлен ниже.

```
UserCurrentDateConverter

namespace Terrasoft.Configuration
{
    using System;
    using System.CodeDom.Compiler;
    using System.Collections.Generic;
    using System.Data;
    using System.Linq;
    using System.Runtime.Serialization;
```

```

using System.ServiceModel;
using System.ServiceModel.Web;
using System.ServiceModel.Activation;
using System.Text;
using System.Text.RegularExpressions;
using System.Web;
using Terrasoft.Common;
using Terrasoft.Core;
using Terrasoft.Core.DB;
using Terrasoft.Core.Entities;
using Terrasoft.Core.Packages;
using Terrasoft.Core.Factories;

/* Атрибут с именем макроса [CurrentDate]. */
[ExpressionConverterAttribute("CurrentDate")]
/* Класс реализует интерфейс IExpressionConverter. */
class CurrentDateConverter : IExpressionConverter
{
    private UserConnection _userConnection;

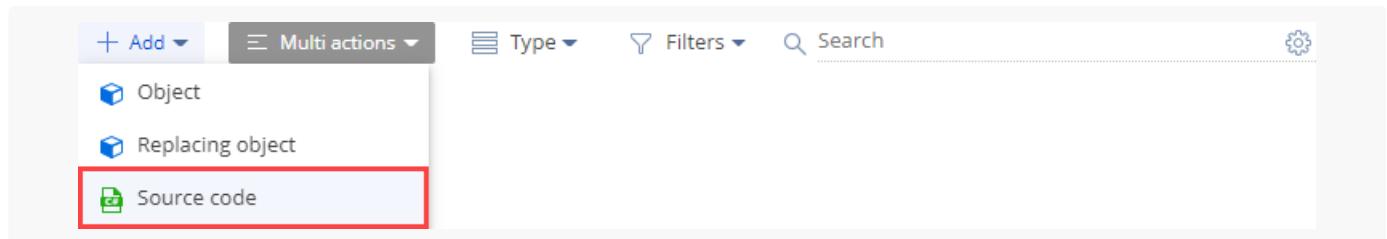
    /* Реализация метода Evaluate интерфейса IExpressionConverter. */
    public string Evaluate(object value, string arguments = "")
    {
        try
        {
            _userConnection = (UserConnection)HttpContext.Current.Session["UserConnection"]
            /* Метод возвращает текущую дату. */
            return _userConnection.CurrentUser.GetCurrentDateTime().Date.ToString("dd MMM
        }
        catch (Exception err)
        {
            return err.Message;
        }
    }
}
}

```

5. На панели инструментов дизайнера исходного кода нажмите [ *Опубликовать* ] ([ *Publish* ]) для выполнения изменений на уровне базы данных.

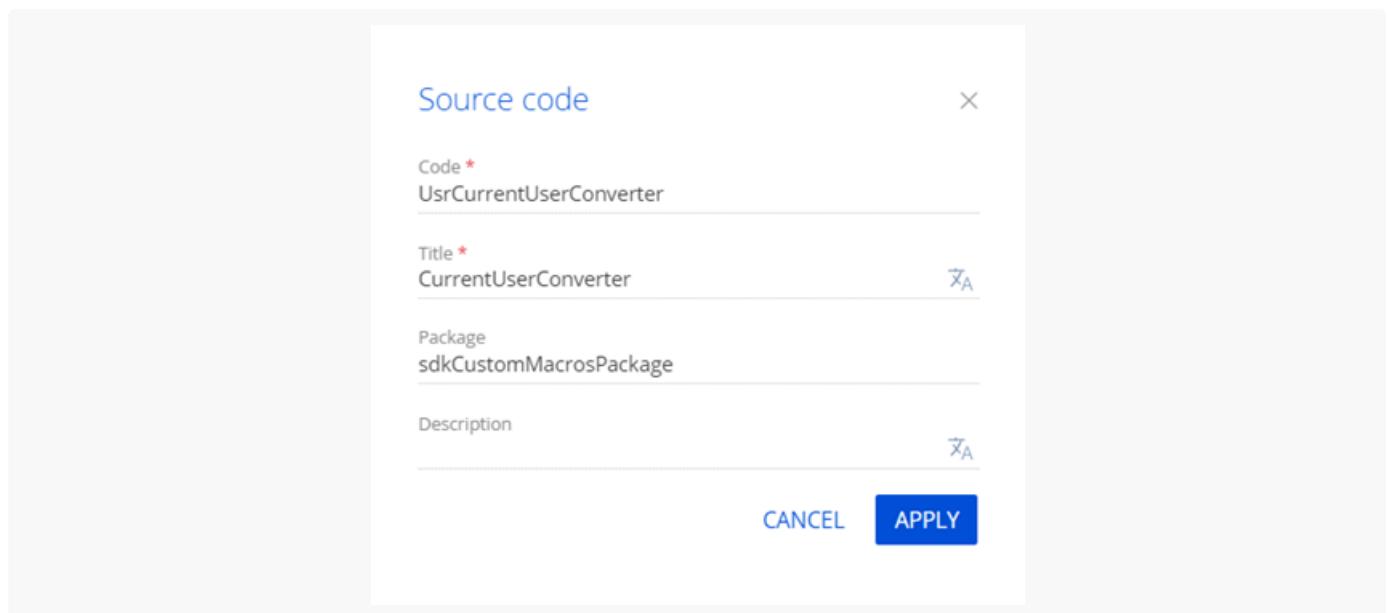
### 3. Реализовать макрос (имя сотрудника, который сформировал отчет)

1. [Перейдите в раздел \[ Конфигурация \]](#) ([ *Configuration* ]) и выберите пользовательский [пакет](#), в который будет добавлена схема.
2. На панели инструментов реестра раздела нажмите [ *Добавить* ] —> [ *Исходный код* ] ([ *Add* ] —> [ *Source code* ]).



3. В дизайнере исходного кода заполните **свойства схемы**:

- [Код] ([Code]) — "UsrCurrentUserConverter".
- [Заголовок] ([Title]) — "CurrentUserConverter".



Для применения изменений свойств нажмите [Применить] ([Apply]).

4. В дизайнере исходного кода реализуйте класс макроса для получения имени сотрудника, который сформировал отчет.

Исходный код схемы типа [Исходный код] ([Source code]) представлен ниже.

```
UsrCurrentUserConverter

namespace Terrasoft.Configuration
{
    using System;
    using System.CodeDom.Compiler;
    using System.Collections.Generic;
    using System.Data;
    using System.Linq;
    using System.Runtime.Serialization;
    using System.ServiceModel;
    using System.ServiceModel.Web;
    using System.ServiceModel.Activation;
```

```

using System.Text;
using System.Text.RegularExpressions;
using System.Web;
using Terrasoft.Common;
using Terrasoft.Core;
using Terrasoft.Core.DB;
using Terrasoft.Core.Entities;
using Terrasoft.Core.Packages;
using Terrasoft.Core.Factories;

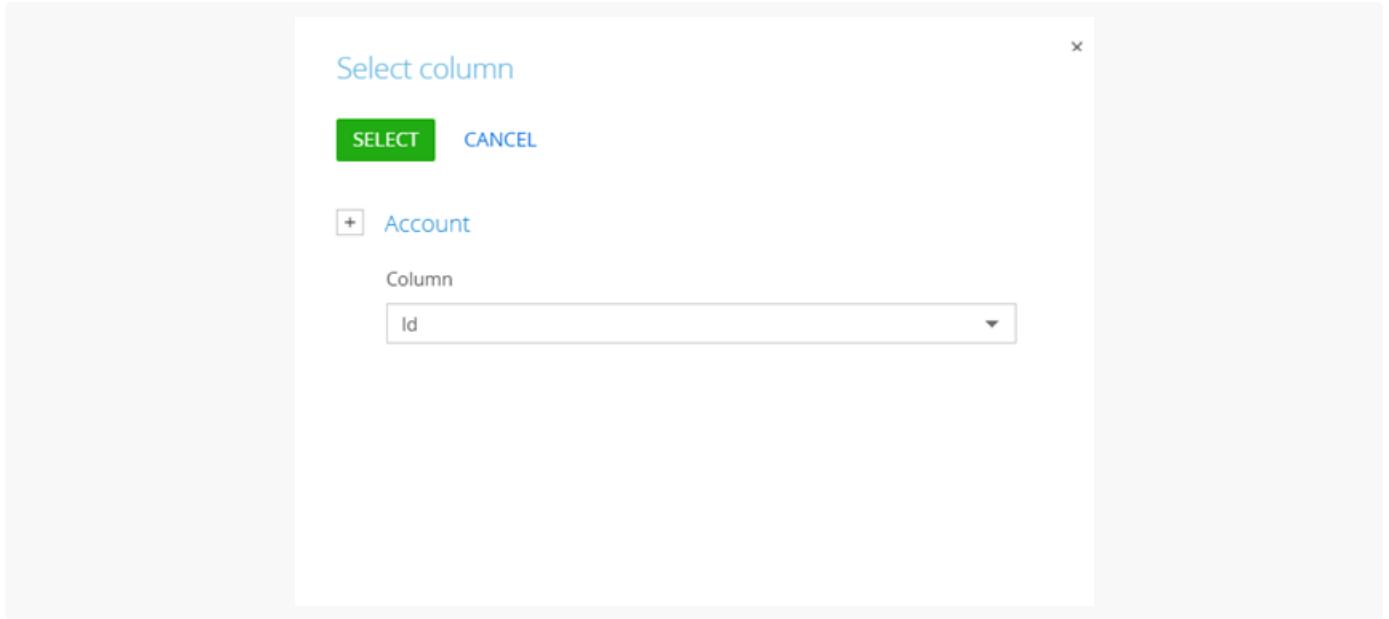
/* Атрибут с именем макроса [CurrentUser]. */
[ExpressionConverterAttribute("CurrentUser")]
/* Класс реализует интерфейс IExpressionConverter. */
class CurrentUserConverter : IExpressionConverter
{
    private UserConnection _userConnection;
    /* Реализация метода Evaluate интерфейса IExpressionConverter. */
    public string Evaluate(object value, string arguments = "")
    {
        try
        {
            _userConnection = (UserConnection)HttpContext.Current.Session["UserConnection"]
            /* Метод возвращает контакт текущего пользователя. */
            return _userConnection.CurrentUser.ContactName;
        }
        catch (Exception err)
        {
            return err.Message;
        }
    }
}
}

```

- На панели инструментов дизайнера исходного кода нажмите [ *Опубликовать* ] ([ *Publish* ]) для выполнения изменений на уровне базы данных.

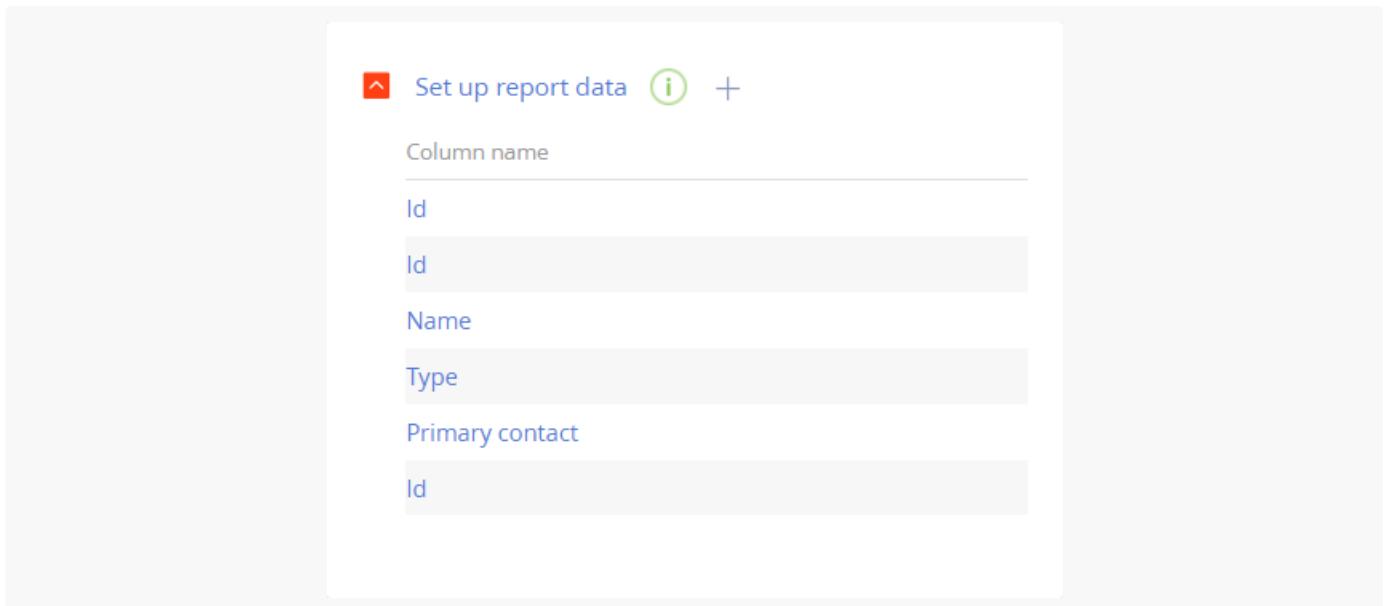
### 3. Настроить поля отчета

- В блоке [ *Настройте поля отчета* ] ([ *Set up report data* ]) рабочей области страницы настройки отчета нажмите .
- В выпадающем списке [ *Колонка* ] ([ *Column* ]) выберите колонку [ *Id* ].



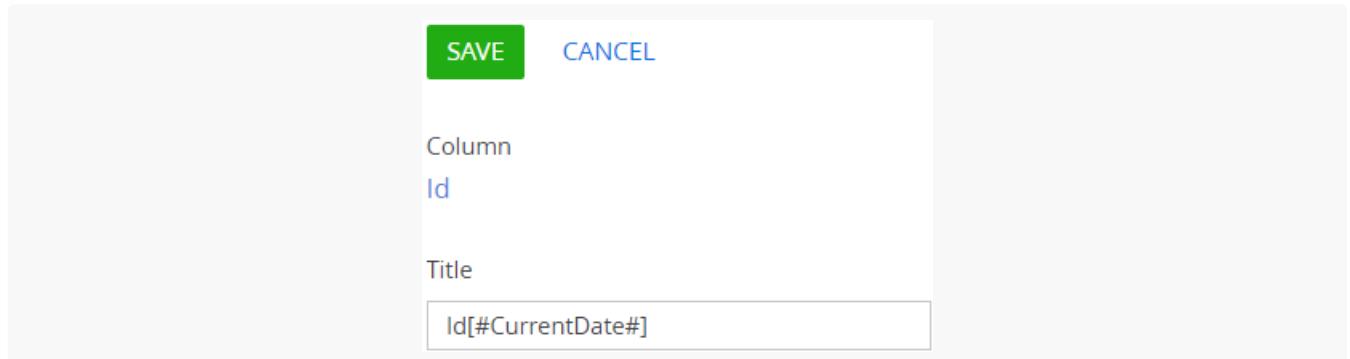
3. Нажмите [ Выбрать ] ([ Select ]). Далее к этой колонке необходимо добавить тег пользовательского макроса для получения даты формирования отчета.
4. Аналогично добавьте колонки [ Id ] (далее к этой колонке необходимо добавить тег пользовательского макроса для получения имени сотрудника, который сформировал отчет), [ Название ] ([ Name ]), [ Тип ] ([ Type ]), [ Основной контакт ] ([ Primary contact ]), [ Id ] (далее к этой колонке необходимо добавить тег пользовательского макроса для получения дополнительной информации контрагента, которая зависит от его типа).

В результате в отчет добавлены поля, которые представлены на рисунке ниже.



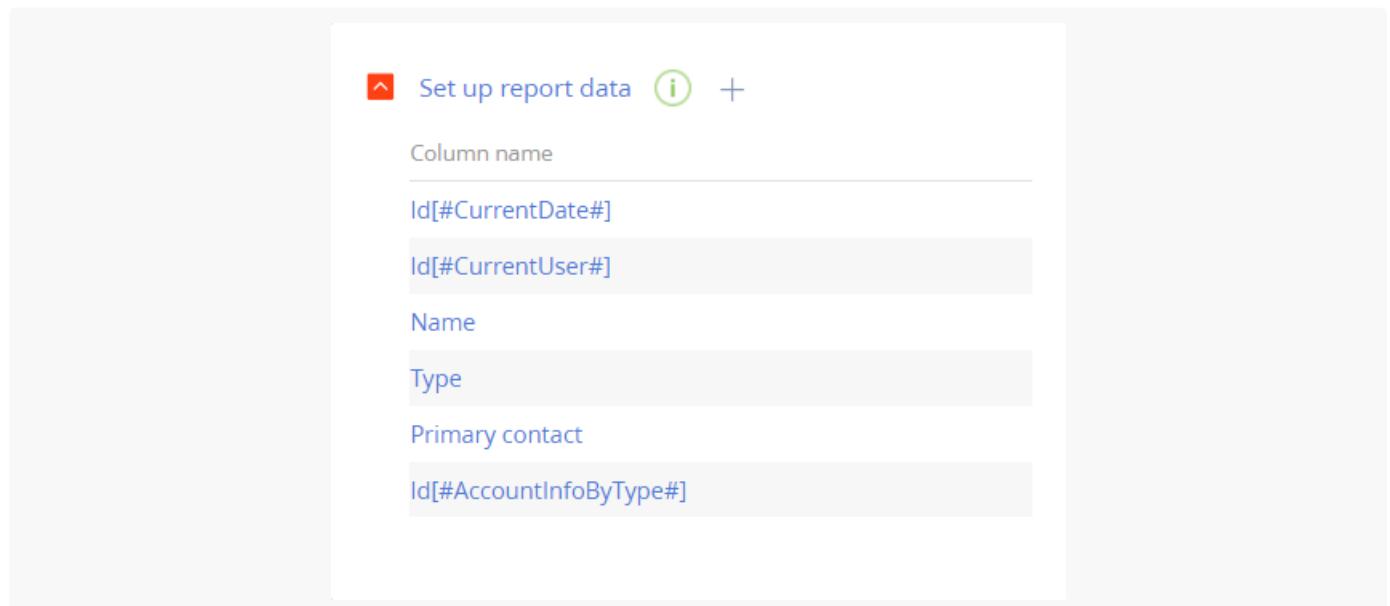
5. К колонке [ Id ] добавьте **тег пользовательского макроса** для получения даты формирования отчета.
    - a. Откройте страницу настройки колонки [ Id ].
- Способы** открытия страницы настройки колонки:

- В блоке [ Настройте поля отчета ] ([ Set up report data ]) рабочей области страницы настройки отчета дважды кликните по колонке, к которой необходимо добавить тег макроса.
  - В блоке [ Настройте поля отчета ] ([ Set up report data ]) рабочей области страницы настройки отчета в строке колонки, к которой необходимо добавить тег макроса, нажмите .
- d. Измените значение поля [ Заголовок ] ([ Title ]) на "Id[#CurrentDate#]". `[#CurrentDate#]` — тег пользовательского макроса, который позволяет получить дату формирования отчета.



- e. На панели инструментов страницы настройки колонки нажмите [ Сохранить ] ([ Save ]).  
f. Аналогично к другим колонкам [ Id ] добавьте тег `[#currentUser#]` пользователяского макроса для получения имени сотрудника, который сформировал отчет, и тег `[#AccountInfoByType#]` пользователяского макроса для получения дополнительной информации контрагента, которая зависит от его типа.

В результате в отчет добавлены поля, которые представлены на рисунке ниже.

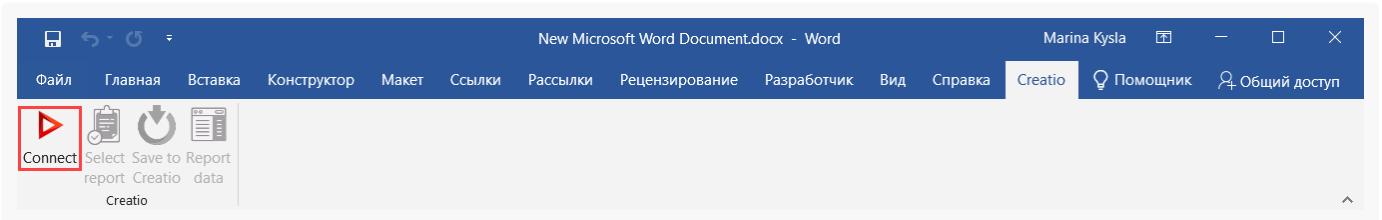


6. На панели инструментов страницы настройки отчета нажмите [ Применить ] ([ Apply ]).

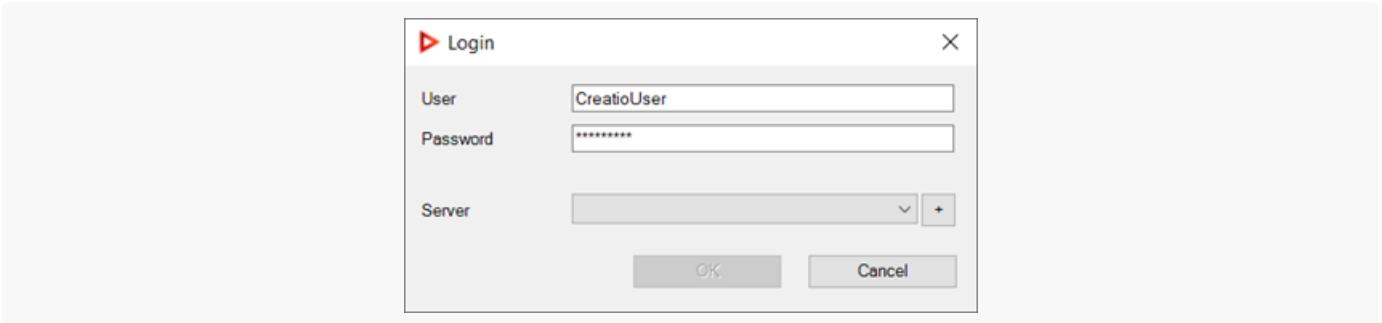
## 4. Настроить шаблон отчета

1. Запустите приложение MS Word.

2. На вкладке панели инструментов Creatio нажмите кнопку [ Connect ].



3. Введите логин и пароль пользователя в Creatio.



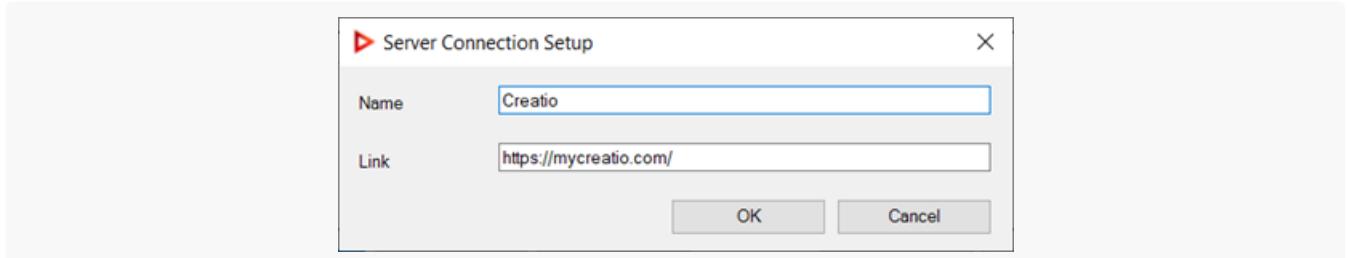
4. Подключите **сервер приложения** Creatio.

- Возле поля [ Server ] нажмите
- В окне [ Available Servers ] нажмите [ New ].

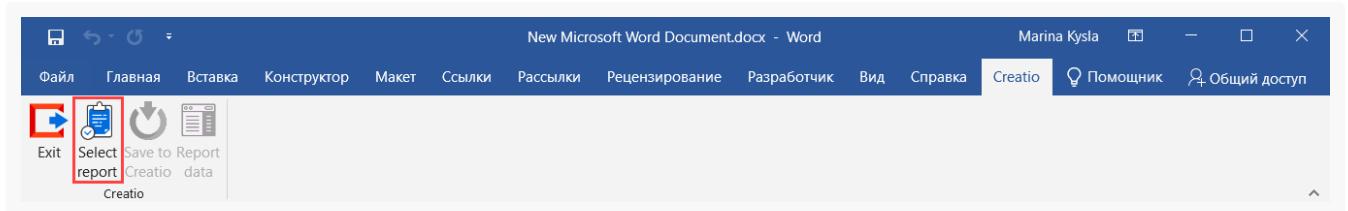


c. Заполните **свойства сервера**:

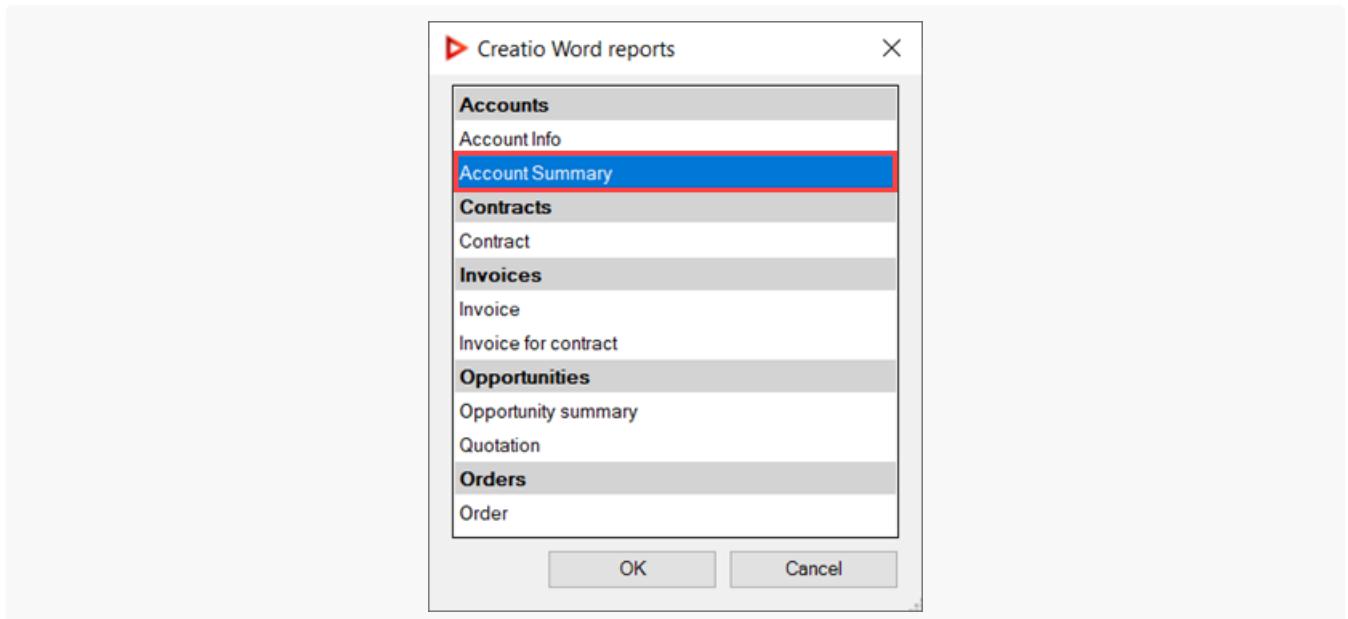
- [ Name ] — пользовательское имя сервера.
- [ Link ] — адрес сервера приложения Creatio, в котором выполнялась настройка отчета.



- f. В окне [ *Server Connection Setup* ] нажмите [ *OK* ] для сохранения параметров подключения сервера *Creatio*.
- g. В окне [ *Available Servers* ] выберите сервер приложения *Creatio*, в котором выполнялась настройка отчета (опционально), и нажмите [ *OK* ].
5. В окне [ *Login* ] нажмите [ *OK* ] для подключения к выбранному серверу *Creatio*.
6. Выберите **отчет MS Word**, который планируется настроить.
  - a. На вкладке панели инструментов нажмите кнопку [ *Select report* ].

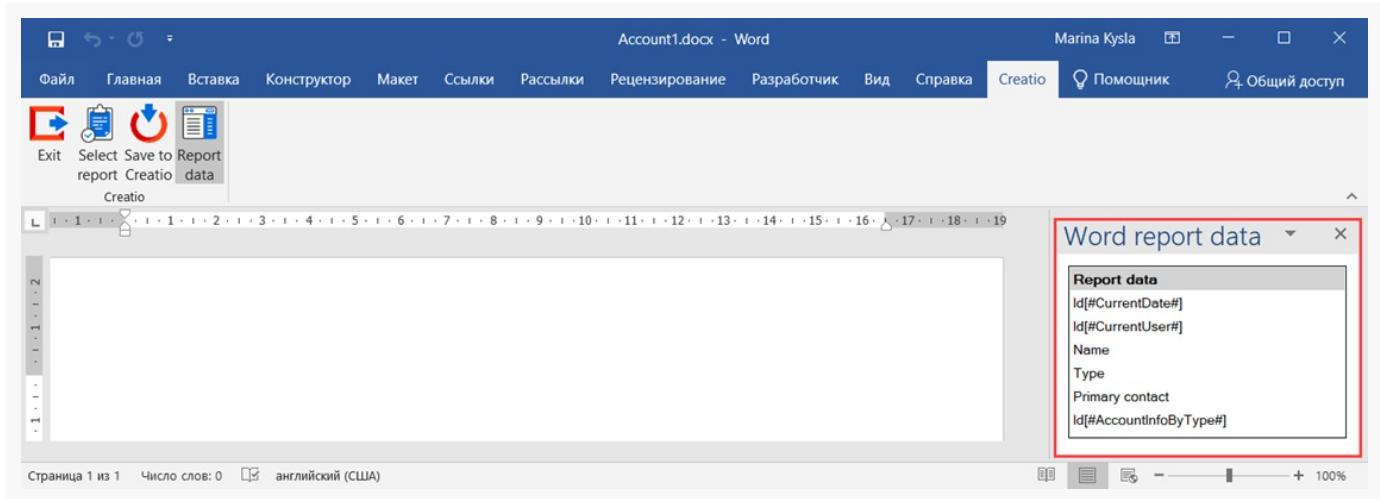


- b. В окне [ *Creatio Word reports* ] выберите отчет [ *Информация контрагента* ] ([ *Account Info* ]).



- c. В окне [ *Creatio Word reports* ] нажмите [ *OK* ] для выбора отчета.

В результате открыта панель [ *Word report data* ] отчета [ *Информация контрагента* ] ([ *Account Info* ]), который представлен на рисунке ниже.



- Настройте шаблон отчета. Для этого воспользуйтесь инструкцией, которая приведена в статье [Настроить отчет в плагине Word и загрузить в Creatio](#).

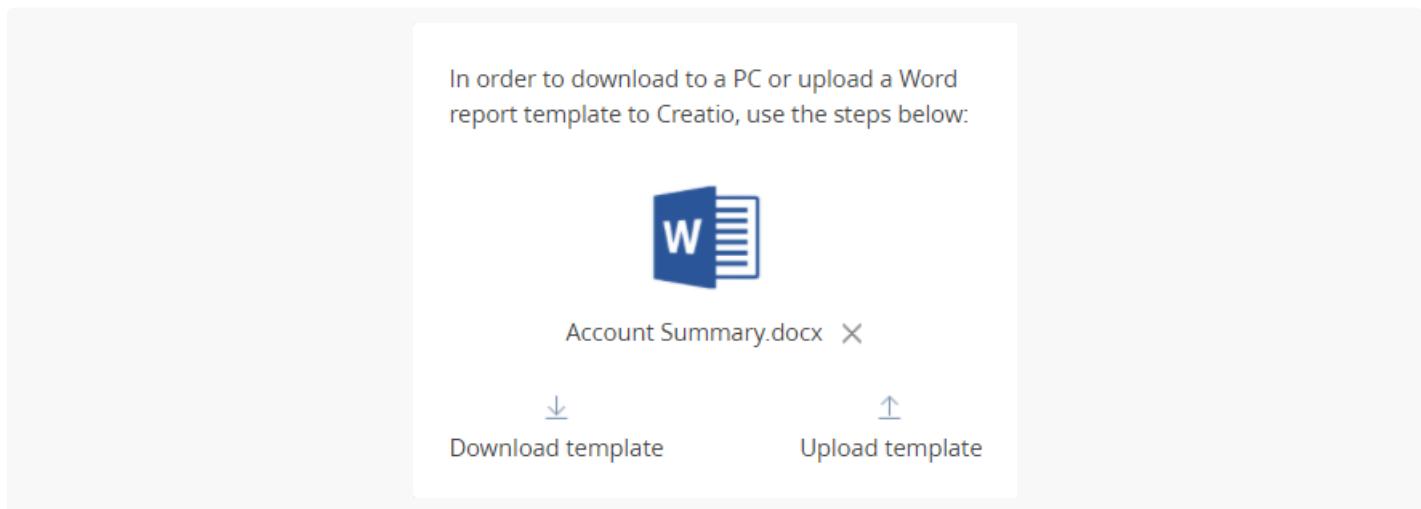
Настроенный шаблон отчета [ *Информация контрагента* ] ([ *Account Summary* ]) приведен на рисунке ниже.

<b><u>Account Summary</u></b>	
<b>Name</b>	«Name»
<b>Type</b>	«Type»
<b>Primary contact</b>	«Primary contact»
<b>Additional info</b>	«Id[#AccountInfoByType#]»
 Date	«Id[#CurrentDate#]»
 Created by	«Id[#CurrentUser#]»

- Загрузите настроенный шаблон отчета [ *Информация контрагента* ] ([ *Account Summary* ]) в Creatio. Для этого на вкладке плагина Creatio панели инструментов нажмите кнопку [ *Save to Creatio* ].

The screenshot shows a Microsoft Word document titled "Account14.docx - Word". The ribbon menu is visible at the top, with "Создание" (Creation) selected. A context menu is open over the "Report" icon in the ribbon, showing options: Exit, Select report, Save to report, Creatio, and Creatio data. The main content area contains a table with four rows labeled "Name", "Type", "Primary contact", and "Additional info". Below the table, two text fields are shown: "Date" followed by the formula "«Id[#CurrentDate#]»" and "Created by" followed by the formula "«Id[#CurrentUser#]»". On the right side, a sidebar titled "Word report data" lists the variables used in the formulas: "Report data", "Id[#CurrentDate#]", "Id[#CurrentUser#]", "Name", "Type", "Primary contact", and "Id[#AccountInfoByType#]". The status bar at the bottom indicates "Страница 1 из 1" (Page 1 of 1), "Число слов: 18", and "Английский (США)".

В результате настроенный шаблон отчета [ Информация контрагента ] ([ Account Summary ]) загружен в Creatio и отображается на панели свойств.



## Результат выполнения примера

Чтобы посмотреть результат выполнения примера:

1. Обновите страницу раздела [ Контрагенты ] ([ Accounts ]).  
2. Откройте страницу контрагента.

В результате выполнения примера на страницу контрагента добавлен отчет [ Информация контрагента ] ([ *Account Summary* ]). Отчет доступен в выпадающем меню кнопки [ Печать ] ([ *Print* ]) панели инструментов раздела.

The screenshot shows the Creatio application interface. At the top, there is a header with the company name "Vertigo Systems", a search bar "What can I do for you?", and a logo for "Creatio 8.0.0.5434". Below the header, there are buttons for "CLOSE", "ACTIONS", and a blue ribbon icon. On the right side, there are "PRINT" and "VIEW" buttons, and a dropdown menu showing "Account Info" and "Account Summary" (which is highlighted with a red box).

In the main content area, there is a section titled "DUPLICATES" with the message "No duplicate records found". To the right, there is a progress bar at 95% completion and an "Enrich data" button. Below this, there is a "NEXT STEPS (0)" section with various icons for communication and tasks.

The main account details are displayed in a card format. It includes a logo for "VERTIGO SYSTEMS", a green progress bar at 95%, and the name "Vertigo Systems". Below the name, there is a field labeled "Name\*" with the value "Vertigo Systems".

At the bottom of the card, there are tabs for "ACCOUNT INFO" (which is selected and highlighted with a red underline), "CONTACTS AND STRUCTURE", "MAINTENANCE", and "TIMELINE". There are also fields for "Also known as" and "Code" (value 109).

Отчет [ Информация контрагента ] ([ Account Summary ]) для контрагента типа [ Клиент ] ([ Customer ]) имеет вид, который представлен на рисунке ниже.

<u><b>Account Summary</b></u>	
<b>Name</b>	Vertigo Systems
<b>Type</b>	Customer
<b>Primary contact</b>	Peter Moore
<b>Additional info</b>	Annual turnover 21 – 30 million

Date	3 Mar 2022
Created by	User

Отчет [ Информация контрагента ] ([ Account Summary ]) для контрагента типа [ Партнер ] ([ Partner ]) имеет вид, который представлен на рисунке ниже.

**Account Summary**

<b>Name</b>	ClearSoft Systems
<b>Type</b>	Partner
<b>Primary contact</b>	William Clarks
<b>Additional info</b>	Number of employees 51-100 persons

Date	3 Mar 2022
Created by	User

## Базовые макросы C#



**Макрос** — инструмент, который позволяет использовать дополнительные возможности получения данных из Creatio для отчета MS Word.

Структура макроса в отчетах MS Word приведена ниже.

### Структура макроса

ИмяКолонки[#ИмяМакроса|Параметр#]

## Базовые макросы

[#Date#]

Конвертирует дату в соответствии с заданным форматом. Формат по умолчанию — dd.MM.yyyy .  
Форматы даты описаны в официальной [документации Microsoft](#). Параметр необязательный.

### Примеры использования макроса [#Date#]

Макрос	Исходное значение	Результат
ИмяКолонки[#Date#]	07/15/2020 11:48:24 AM	15.07.2020
ИмяКолонки[#Date MM/dd/yyyy#]	31/01/2019 08:25:48 AM	01/31/2019

[#Lower#]

Конвертирует значение строки в нижний регистр.

### Пример использования макроса [#Lower#]

Макрос	Исходное значение	Результат
ИмяКолонки[#Lower#]	ПриMeР	пример

[#Upper#]

Конвертирует значение строки в верхний регистр. Параметр необязательный.

### Параметры

FirstChar	В верхний регистр конвертируется только первый символ.
-----------	--

### Примеры использования макроса [#Upper#]

Макрос	Исходное значение	Результат
ИмяКолонки[#Upper#]	пример	ПРИМЕР
ИмяКолонки[#Upper FirstChar#]	пример	Пример

[#NumberDigit#]

Конвертирует дробное число в число с разделителем тысячных разрядов. По умолчанию разделителем является символ пробела. Параметр необязательный. Если дробная часть числа равна нулю, то она не отображается.

### Примеры использования макроса [#NumberDigit#]

Макрос	Исходное значение	Результат
ИмяКолонки[#NumberDigit#]	345566777888.567	345 566 777 888.567
ИмяКолонки[#NumberDigit ,#]	345566777888.567	345,566,777,888.567
	345566777888.000	345,566,777,888

[#NumberRU#]

Конвертирует число в строковое представление. Параметр необязательный. Если параметр не указан, то учитывается только целая часть числа.

## Параметры

Cent	Возвращает дробную часть числа, не конвертируя ее в строковое представление.
Decimal	Возвращает дробную часть числа, конвертируя ее в строковое представление.

## Примеры использования макроса `[#NumberRU#]`

Макрос	Исходное значение	Результат
ИмяКолонки[#NumberRU#]	456	четыреста пятьдесят шесть
	456.78	четыреста пятьдесят шесть
ИмяКолонки[#NumberRU Cent#]	123.45	45
	123	00
ИмяКолонки[#NumberRU Decimal#]	777.77	семьсот семьдесят семь целых семьдесят семь сотых

## `[#Boolean#]`

Конвертирует логическое значение в заданный вид. Параметр обязательный.

## Параметры

CheckBox	Преобразует исходное значение в элемент выбора <input checked="" type="checkbox"/> или <input type="checkbox"/> .
Да, Нет	Преобразует исходное значение в текст <code>да</code> или <code>нет</code> .

## Примеры использования макроса `[#Boolean#]`

Макрос	Исходное значение	Результат
ИмяКолонки[#Boolean CheckBox#]	true	<input checked="" type="checkbox"/>
ИмяКолонки[#Boolean Да, Нет#]	true	Да

# 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* ]

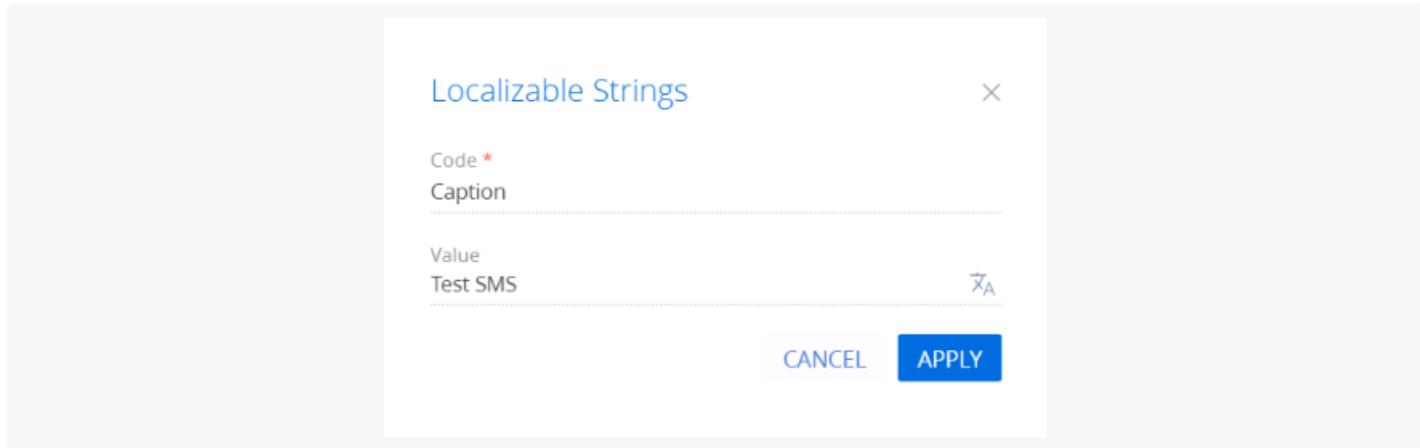
Для отображения элемента в пользовательском интерфейсе [ *Дизайнера кампании* ] необходимо в пользовательском пакете создать новую [схему модуля](#) элемента кампании. Для созданной схемы нужно установить следующие свойства:

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

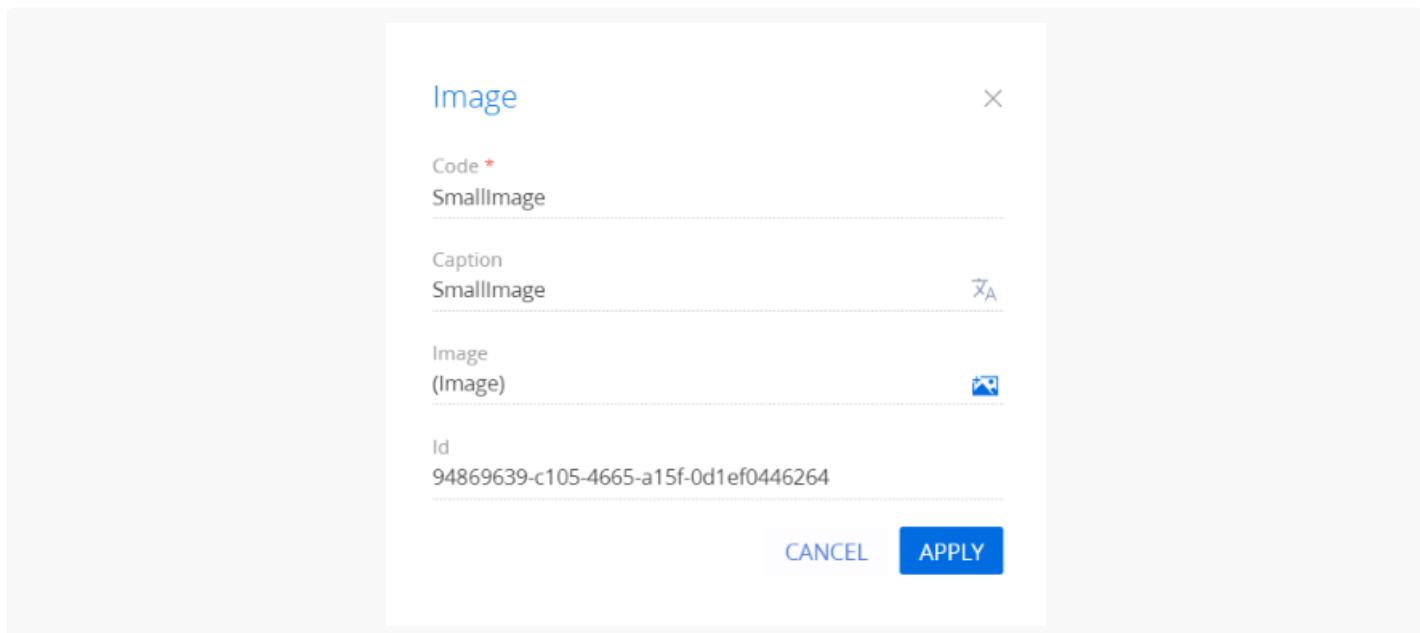
**Важно.** В этом примере в названиях схем отсутствует префикс `usr`. Префикс, используемый по умолчанию, можно изменить в системной настройке [ *Префикс названия объекта* ] (код `SchemaNamePrefix`).

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", "CampaignBaseCommunicationSchemaResources"], 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: [
    // Контеинер 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 кампаний.
    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 (СМС - 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, parentSchema)
        {
            // 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 (СМС).
    public override ProcessFlowElement CreateProcessFlowElement(UserConnection userConnectic
        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 кампаний 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.CampaignParticipantParticipatingStatus)
            .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).

My campaign

SAVE CANCEL

Test SMS

Test SMS 1

Which SMS text to send?

Sender phone number

Message

Test SMS 1

**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 кампаний ]. 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 ])

The screenshot shows the 'Properties' window for the 'TestSmsTarget' object. The left pane, titled 'Structure', shows the object hierarchy: 'TestSmsTarget' contains 'Columns' (which include 'PhoneNumber', 'SmsText', 'Contact', and 'TestSmsResponse', all highlighted with a red box), 'Inherited Columns', and 'Indexes'. The right pane, titled 'Properties', contains the following fields:

- General**: Name (TestSmsTarget), Title (TestSmsTarget), Package (sdkAddingCustomArrow).
- Inheritance**: Parent object (Base object (Base)), Replace parent (unchecked).
- Behavior**: Allow records deactivation (unchecked).
- Access rights**: Operations (unchecked), Records (unchecked).

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

The screenshot shows the 'Structure' and 'Properties' tabs for a custom object named 'TestSmsResponseType'. The 'Structure' tab displays a tree view with 'Columns' expanded, showing 'Inherited Columns' and a new column 'Aa Name' highlighted with a red box. The 'Properties' tab shows the following configuration:

- General**: Name: TestSmsResponseType, Title: TestSmsResponseType, Package: sdkAddingCustomArrow.
- Inheritance**: Parent object: Base object (Base).
- Behavior**: Allow records deactivation:
- Access rights**: Operations: , Records:

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.

The screenshot shows the 'Lookups' screen for the 'Test sms response' object. The list contains the following items:

- Canceled
- Sms delivered
- Error while receiving

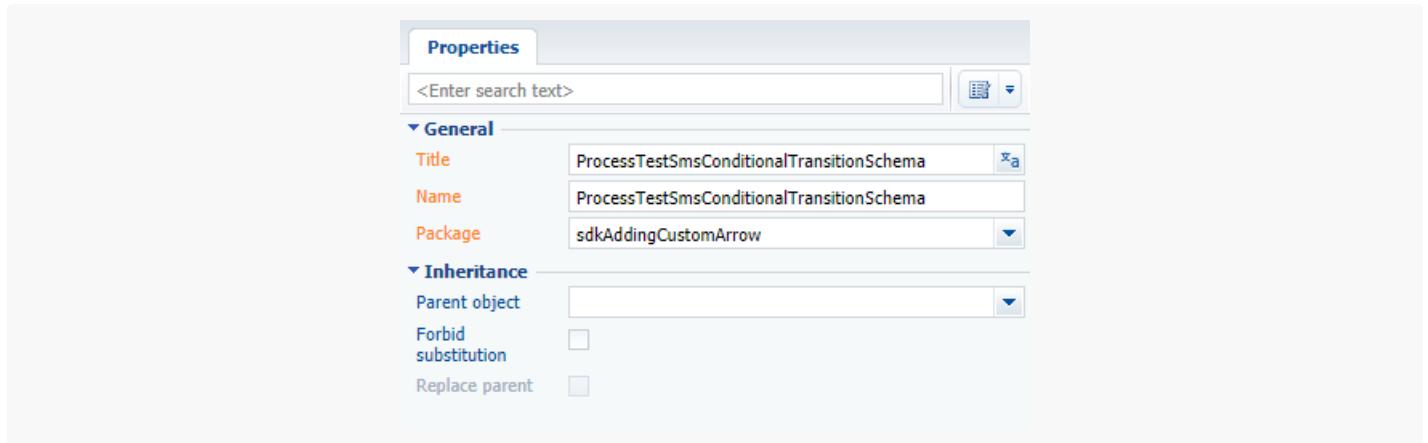
## 2. Создать новую схему для элемента [ Переход ]

Для отображения элемента в пользовательском интерфейсе [ Дизайнер 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.CC
            // Имя элемента стрелки для привязки к элементам кампании.
            connectionUserHandleName: "TestSmsConditionalTransition",
            // Имя карточки свойств стрелки.
            editPageSchemaName: "TestSmsConditionalTransitionPropertiesPage",
            elementType: CampaignEnums.CampaignSchemaElementTypes.CONDITIONAL_TRANSITION,
            // Коллекция откликов по СМС-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.

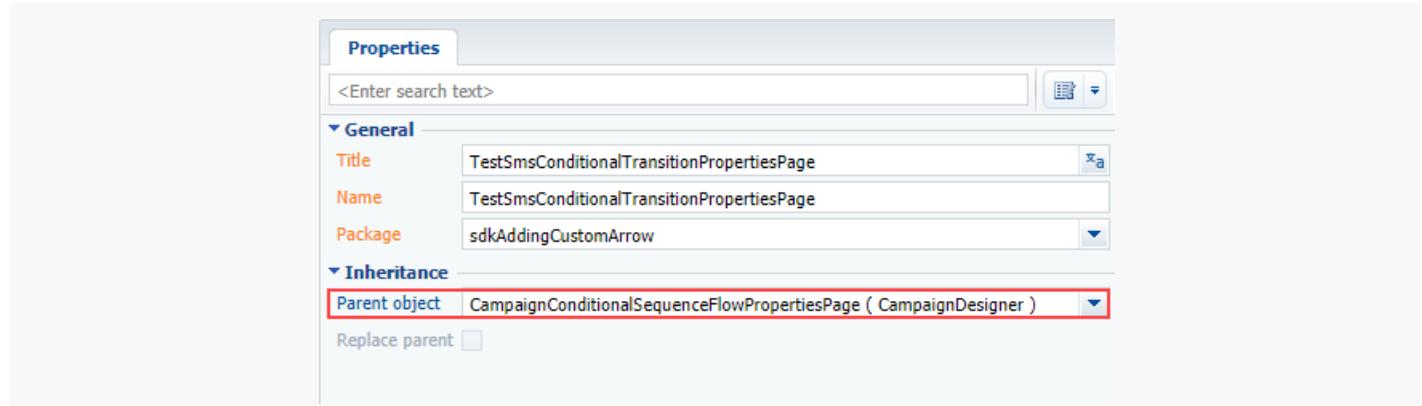
### 3. Создать страницу свойств элемента [ Переход ]

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 троеточие.
    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
    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().getCaption);
    return caption;
}
},
diff: /**SCHEMA_DIFF*/
// Контеинер.
{
    "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 [ СМС-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 = IsResponseBasedStartProperty
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.CampaignSchema parentSchema) {
    return new TestSmsConditionalTransitionElement(this, dictToRebind, parentSchema);
}

// Переопределяет фабричный метод по созданию исполняемого элемента
// Возвращает элемент с типом TestSmsConditionalTransitionFlowElement
public override ProcessFlowElement CreateProcessFlowElement(UserConnection userConnection) {
    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 г
        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.testSmsResponseId : null;
    // Так же переносим и настройку учета откликов
    newElement.isResponseBasedStart = prevElement.isResponseBasedStart ? prevElement.isResponseBasedStart : false;
}
});
```

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.

# Общие принципы работы с email-сообщениями

 Средний

## Пользовательская фильтрация нежелательных обращений

Creatio предоставляет возможность настроить фильтр нежелательных обращений. Пользователи имеют возможность фиксировать список email адресов и доменов, письма из которых могут автоматически помечаться как спам. При этом обращения по ним могут либо не регистрироваться, либо регистрироваться в отмененном состоянии (статус указывается в системной настройке [Состояние нежелательных обращений по умолчанию]).

Также фильтр нежелательных обращений позволяет анализировать заголовок письма на предмет наличия в нем определенных флагов, например `Auto-Submitted`. При наличии таких флагов фильтр позволяет не создавать обращения или создавать обращения в состоянии, указанном в системной настройке.

Для того чтобы анализировать электронный адрес, домен или часть адреса, достаточно просто добавить необходимое значение в справочник [Черный список email адресов и доменов для регистрации обращений]. После добавления значения (тип будет установлен автоматически), во время анализа входящего письма адреса будут помечены как входящие в черный список.

Анализ заголовка письма происходит с использованием справочника [Управление свойствами заголовков email]. Для того чтобы добавить новое свойство для анализа, необходимо выполнить следующие шаги:

- Добавить новый класс, реализующий интерфейс `IHeaderPropertyHandler`. Этот интерфейс содержит в себе единственный метод `Check()`, возвращающий значение типа `bool`. В реализации метода `Check()` необходимо выполнить проверку значения свойства и вернуть результат. Если метод возвращает `true`, обращение будет создано по стандартному механизму, если `false` — результат проверки отрицательный и письмо будет обработано как письмо из черного списка.
- Добавить значение в справочник [Управление свойствами заголовков email]. В колонке `Name` указать имя свойства, анализ которого должен проводиться при регистрации. В колонке `handler` должно содержаться имя класса, который был добавлен в предыдущем пункте.

## Работа с цепочками email-сообщений

Начиная с версии 7.10.0 в приложении добавлен механизм формирования цепочек Email-сообщений. Целью данного решения является возможность улучшить пользовательский интерфейс работы с Email-сообщениями. Также использование этого механизма позволит упростить поиск связей для писем, например, копированием связей предыдущего письма.

Разработанный механизм построения цепочек опирается на данные из заголовков Email-сообщений `In-Reply-To`. В этом заголовке по общепринятым соглашениям должен содержаться идентификатор письма `Message-ID`, на который отвечает текущее письмо. Приложение Creatio получает эти заголовки из

Email-сервиса, с которым настроена синхронизация (IMAP/Exchange).

Условно механизм работы с цепочками можно поделить на две логические части — создание цепочки и заполнение связей активности.

## Создание цепочек email-сообщений

В детали `EmailMessageData` добавлены три поля, отвечающие за цепочки Email-сообщений:

- `MessageId` — строка длиной 500 символов;
- `InReplyTo` — строка длиной 500 символов;
- `ParentMessageId` — справочное поле, ссылающееся на деталь `EmailMessageData`.

При синхронизации писем поля `MessageId` и `InReplyTo` заполняются соответствующими значениями из заголовков письма.

Поле `ParentMessageId` заполняется следующим образом:

1. В таблице `EmailMessageData` выполняется поиск записей, у которых `MessageId` равен текущему `InReplyTo`. Если записи найдены, первая из них устанавливается как текущий `ParentMessageId`.
2. Во всех записях `EmailMessageData`, у которых `InReplyTo` равен текущему `MessageId` и поле `ParentMessageId` не заполнено, полю `ParentMessageId` присваивается значение, равное текущему `Id`.

Таким образом, для каждого письма актуализируются связи с соседними письмами в цепочках.

## Копирование связей предыдущего email-сообщения в цепочке

После добавления письма необходимо распространить связи активности по цепочке. Этот процесс будет рассмотрен на примере поля [ *Обращение* ].

Для текущей записи `EmailMessageData` выполняется поиск такой родительской записи, у которой указана активность с заполненным полем [ *Обращение* ]. Значение поля [ *Обращение* ] из этой активности будет распространяться по цепочке. Начиная с найденной записи `EmailMessageData` выполняется поиск всех дочерних записей `EmailMessageData` и обновляется значение поля [ *Обращение* ] в найденных активностях.

Например, существует цепочка из трех Email-сообщений:

ActivityId	Title	CaseId	EmailMessageId	EmailMessageParentId
28BD6D59-B9D7-4FF9-89F5-FEE1DD003912	Re: relation	NULL	66812FBF-411B-4FE0-94C9-1E70FBEB2D3	F05B529D-C98C-4E26-BE00-21F8721AEF58
DC0A40D4-700A-40EB-B394-90E0376C3B5D	Re: relation	1C6E18E3-48B1-495E-8EF9-ACA35DB9DE0B	F05B529D-C98C-4E26-BE00-21F8721AEF58	E1A0120E-B7C0-4261-9DE0-C63341BF1E0B
D7A9B82C-ED46-437C-980A-B2650D4FF3DA	relation	906909E8-4D64-47FD-AF92-B65B0826AEC3	E1A0120E-B7C0-4261-9DE0-C63341BF1E0B	NULL

Затем в цепочку приходит еще одно Email-сообщение:

ActivityId	Title	CaseId	EmailMessageId	EmailMessageParentId
6623B052-73AD-4AE5-AE61-A6F9BCD930A0	Re: relation	NULL	60C00B01-D0BF-40F6-923E-1830E433AEA1	66812FBF-411B-4FE0-94C9-1E70FBEB2D3
28BD6D59-B9D7-4FF9-89F5-FEE1DD003912	Re: relation	NULL	66812FBF-411B-4FE0-94C9-1E70FBEB2D3	F05B529D-C98C-4E26-BE00-21F8721AEF58
DC0A40D4-700A-40EB-B394-90E0376C3B5D	Re: relation	1C6E18E3-48B1-495E-8EF9-ACA35DB9DE0B	F05B529D-C98C-4E26-BE00-21F8721AEF58	E1A0120E-B7C0-4261-9DE0-C63341BF1E0B
D7A9B82C-ED46-437C-980A-B2650D4FF3DA	relation	906909E8-4D64-47FD-AF92-B65B0826AEC3	E1A0120E-B7C0-4261-9DE0-C63341BF1E0B	NULL

Начиная с записи, у которой `EmailMessageId` = "60C00B01-D0BF-40F6-923E-1830E433AEA1" выполняется поиск записи, у которой колонка `CaseId` заполнена (не содержит `NULL`). Это запись, у которой `EmailMessageId` = "F05B529D-C98C-4E26-BE00-21F8721AEF58", а `CaseId` = "1C6E18E3-48B1-495E-8EF9-ACA35DB9DE0B".

Теперь начиная с записи, у которой `EmailMessageId` = "F05B529D-C98C-4E26-BE00-21F8721AEF58",

необходимо обновить значение поля `CaseId` для дочерних записей по всей цепочке.

<b>ActivityId</b>	<b>Title</b>	<b>CaseId</b>	<b>EmailMessageId</b>	<b>EmailMessageParentId</b>
6623B052-73AD-4AE5-AE61-A6F9BCD930A0	Re: relation	1C6E18E3-48B1-495E-8EF9-ACA35DB9DE0B	60C00B01-D0BF-40F6-923E-1830E433AEA1	66812FBF-411B-4FE0-94C9-1E70FBBEB2D3
28BD6D59-B9D7-4FF9-89F5-FEE1DD003912	Re: relation	1C6E18E3-48B1-495E-8EF9-ACA35DB9DE0B	66812FBF-411B-4FE0-94C9-1E70FBBEB2D3	F05B529D-C98C-4E26-BE00-21F8721AEF58
DC0A40D4-700A-40EB-B394-90E0376C3B5D	Re: relation	1C6E18E3-48B1-495E-8EF9-ACA35DB9DE0B	F05B529D-C98C-4E26-BE00-21F8721AEF58	E1A0120E-B7C0-4261-9DE0-C63341BF1E0B
D7A9B82C-ED46-437C-980A-B2650D4FF3DA	relation	906909E8-4D64-47FD-AF92-B65B0826AEC3	E1A0120E-B7C0-4261-9DE0-C63341BF1E0B	NULL

### **На заметку.**

Сложности данного механизма обусловлены тем, что очень часто почтовый сервер отдает письма не в той последовательности, в которой они были написаны в цепочке.

Самый простой и частый пример — в ходе сессии синхронизации сначала приходит почта из папки "Входящие", а затем — из папки "Исходящие". Если же письма были получены не последовательно, то не будет возможности построить цепочку на момент загрузки каждого письма. Но такая возможность появляется к завершению цикла синхронизации всех писем почтового ящика. Безусловно, логика поиска цепочек может не сработать, если настроена загрузка не всех папок почтового ящика, или цепочка была прервана другими участниками переписки. Тем не менее, в большом количестве случаев цепочку можно построить на момент завершения загрузки всех писем из почтового ящика.

## Рекомендации по добавлению пользовательского процесса поиска всех связей в цепочках после загрузки email-сообщения

Для того чтобы начать работу по новому полученному Email-сообщению после завершения формирования цепочек, необходимо знать следующее:

1. В таблице `EmailMessageData` появляется поле `Id` сессии синхронизации, уникальное для одного запуска процесса синхронизации. Заполняется только для синхронизированных Email-сообщений.
2. Появляется таблица `FinishedSyncSession`, в которую добавляется запись, если в рамках синхронизации

пришло хоть одно Email-сообщение.

3. Процессы, ранее слушавшие сигнал сохранения активности, в новой реализации слушают сигнал добавления (`Inserted`) объекта `FinishedSyncSession`. По значению `Id` сессии синхронизации выбираются Email-сообщения из `EmailMessageData`. Также в `EmailMessageData` существует поле `MailboxSyncSettings`, которое можно использовать для отбора Email-сообщений из сервисного ящика.

## Обработчик макроса в шаблоне email-сообщения

Шаблоны email-сообщений используются, например, при общении со службой поддержки. Шаблоны доступны в справочнике [ *Шаблоны email-сообщений* ] ([ *Email templates* ]). Подробнее об их настройке можно узнать из статьи "[Настройка автоматической отправки email-уведомлений](#)".

Например, в письме, уведомляющем клиента о закрытии его обращения, используется шаблон "Сообщение о закрытии обращения". Для подстановки некоторых значений колонок объектов системы в шаблоны email-сообщений используются преднастроенные макросы, например, обращение к контакту или должность ответственного.

Creatio позволяет реализовать собственную логику заполнения значения, которое возвращает обработчик макросов. При этом во время обработки макроса система выполнит реализованный разработчиком алгоритм, а не базовую логику.

На специализированный обработчик класса указывает тег `@Invoke`. Далее через точку должно быть указано имя класса реализующего интерфейс `IMacrosInvokable`, который включает в себя метод `GetMacrosValue()`. Этот метод должен возвратить строку, которая будет подставлена вместо строки макроса.

Для реализации пользовательского обработчика макроса необходимо:

1. Создать класс, реализующий интерфейс `IMacrosInvokable`.
2. Зарегистрировать макрос в таблице `EmailTemplateMacros`, указав для него `ParentId` (базовый шаблон с тегом `@Invoke`) и `ColumnPath` (имя класса).
3. Добавить макрос в шаблон email-сообщения.

## Добавить свойство для фильтрации email-сообщений



### Описание кейса

Добавить для анализа обращения новое свойство `No-reply`. В случае, если данное свойство присутствует в заголовке письма и его значение отлично от "No", обращение обработать, как обращение из черного списка.

### Алгоритм реализации кейса

## 1. Добавление нового класса, реализующего интерфейс IHeaderPropertyHandler

В пользовательском пакете (например, `Custom`) необходимо добавить новую схему типа "Исходный код".

В этой схеме необходимо определить класс, реализующий интерфейс `IHeaderPropertyHandler`, например:

```
namespace Terrasoft.Configuration
{
    using System;
    public class NoreplyHandler: IHeaderPropertyHandler
    {
        public bool Check(object value) {
            return string.Equals(value.ToString(), "No", StringComparison.OrdinalIgnoreCase);
        }
    }
}
```

После сохранения созданную схему необходимо опубликовать.

Интерфейс `IHeaderPropertyHandler` определен в пакете `JunkFilter`, поэтому он обязательно должен быть добавлен в зависимости пользовательского пакета.

## 2. Добавление значения в справочник

Для корректной работы нового фильтра необходимо добавить в справочник [Управление свойствами заголовков `email`] новое свойство `No-reply`. В качестве класса обработчика (свойство `handler`) необходимо указать созданный на предыдущем шаге класс `NoreplyHandler`.

В результате выполнения перечисленных действий, после получения входящего письма, содержащего в своем заголовке флаг `No-reply` со значением, отличным от "No", возможны следующие варианты:

- обращение не будет создано в случае, если значение системной настройки [Создавать обращения по нежелательным письмам] равно `false`;
- обращение будет создано со статусом, указанным в системной настройке [Состояние нежелательных обращений по умолчанию], если значение системной настройки [Создавать обращения по нежелательным письмам] равно `true`.

# Добавить обработчик макроса шаблона email-сообщения



Сложный

Пример реализован для продуктов линейки Service Creatio.

## Описание примера

Добавить обработчик макроса шаблона email-сообщения, который будет возвращать строку "Test".

## Исходный код

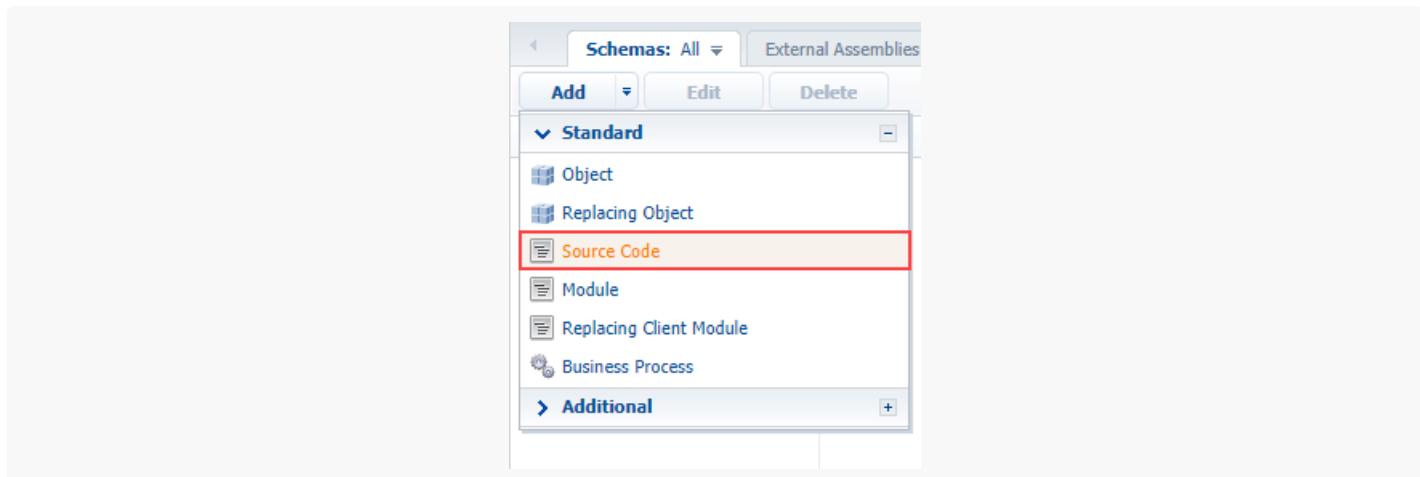
Пакет с реализацией примера можно скачать по [ссылке](#).

## Пример реализации примера

### 1. Создание класса, реализующего интерфейс IMacrosInvokable

Чтобы создать класс, реализующий интерфейс `IMacrosInvokable`, добавьте в пакет разработки схему [ Исходный код ] ([ Source Code ]). Для этого [в разделе \[ Конфигурация \]](#) на вкладке [ Схемы ] ([ Schemas ]) выполните команду меню [ Добавить ] ([ Add ]) — [ Исходный код ] ([ Source Code ]) (рис. 1).

Рис. 1. — Создание новой схемы [ Исходный код ] ([ Source Code ])



Для созданной схемы объекта укажите:

- [Заголовок] ([Title]) — "Генератор текстовой строки" ("Text string generator").
- [Название] ([Name]) — "UsrTestStringGenerator".

Исходный код схемы:

```
namespace Terrasoft.Configuration
{
    using System;
    using Terrasoft.Core;
    // Класс обработчика макроса шаблона Email-сообщения.
    public class UsrTestStringGenerator : IMacrosInvokable
    {
        // Пользовательское соединение.
        public UserConnection UserConnection {
            get;
            set;
        }
}
```

```
        }
        // Метод, возвращающий подставляемое значение.
        public string GetMacrosValue(object arguments) {
            return "Тестовая строка";
        }
    }
}
```

Опубликуйте созданную схему.

## 2. Регистрация макроса в таблице EmailTemplateMacros

Чтобы зарегистрировать макрос в таблице `EmailTemplateMacros`, выполните следующий SQL-запрос:

```
INSERT INTO EmailTemplateMacros(Name, Parentid, ColumnPath)
VALUES (
    'UsrTestStringGenerator',
    (SELECT TOP 1 Id
     FROM EmailTemplateMacros
     WHERE Name = '@Invoke'),
    'Terrasoft.Configuration.UsrTestStringGenerator'
)
```

### 3. Добавление макроса в шаблон email-сообщения

После регистрации макроса его можно использовать в шаблонах email-сообщений. Для этого измените одну или несколько записей справочника [ *Шаблоны email-сообщений* ] ([ *Email templates* ]) (рис. 2).

Рис. 2. — Справочник [Шаблоны email-сообщений]

Lookups

What can I do for you? > Creatio

NEW EMAIL TEMPLATE CLOSE ACTIONS ▾ VIEW ▾

Email templates

Filters/folders ▾

Empty case email template	Subject New message on case #[#Number#]
Case feedback request notification	Subject New message on case #[#Number#]
Case resolution notification	Subject New message on case #[#Number#]
Case assigned to group	Subject Case #[#Number#] "[#Subject#]" assigned to group

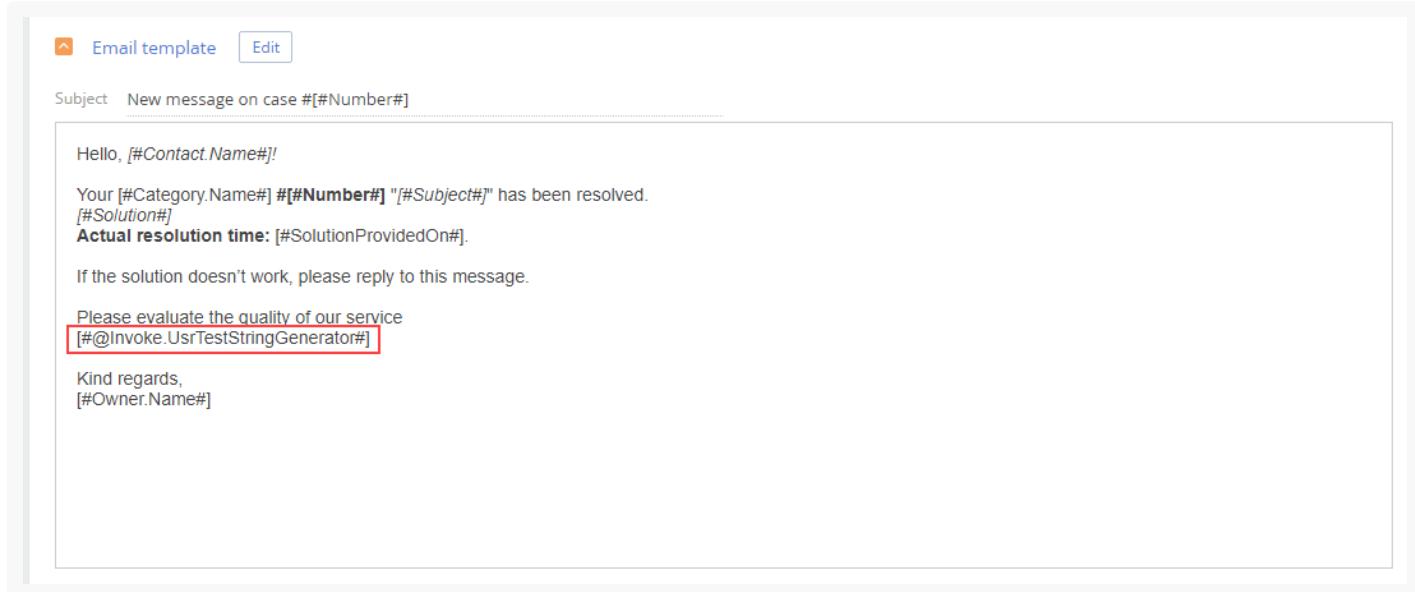
OPEN COPY DELETE

1

Например, для изменения содержимого сообщений, автоматически отправляемых при разрешении

обращения, необходимо изменить запись [ Сообщение о разрешении обращения ] ([ Case resolution notification ]). Если в шаблон добавить макрос [ #@Invoke.UsrTestStringGenerator# ] (рис. 3), то при отправке сообщения пользователю вместо макроса будет подставлено значение "Test".

Рис. 3. — Макрос в шаблоне Email-сообщения



# Общие принципы работы механизма синхронизации Sync Engine

Сложный

## Синхронизация с внешними хранилищами данных

В Creatio реализован механизм синхронизации с внешними хранилищами данных (Sync Engine) который позволяет создавать, изменять и удалять Entity в системе на основании данных из внешних систем и экспортить данные во внешние системы.

Процесс синхронизации осуществляется с помощью класса `SyncAgent`, который реализован в пространстве имен `Terrasoft.Sync` ядра приложения.

## Классы, задействованные в механизме синхронизации

- Агент синхронизации (`SyncAgent`) — класс с одним публичным методом `Synchronize`, который запускает синхронизацию между переданными ему хранилищами данных.
- Контекст синхронизации (`SyncContext`) — класс, представляющий собой агрегацию провайдеров и метаданных для работы `SyncAgent`.
- Хранилище — конкретный репозиторий синхронизируемых данных.
  - Локальное хранилище (`LocalProvider`) — позволяет работать с `LocalItem` в Creatio.
  - Внешнее хранилище (`RemoteProvider`) — внешний сервис или приложение, данные из которого

синхронизируются с Creatio.

- Элемент синхронизации (`SyncItem`) — множество объектов из внешнего и локального хранилища, между которыми устанавливается однозначное соответствие.
  - Элемент синхронизации внешнего хранилища (`RemoteItem`) — представляет набор данных из внешнего хранилища, который синхронизируется атомарно. Может состоять из одной или нескольких сущностей (записей) внешнего хранилища.
  - Элемент синхронизации (`SyncEntity`) — является оберткой над конкретными `Entity`. `SyncEntity` необходим для работы с `Entity` как с синхронизируемым объектом, так и с состоянием и действием, которое с этим `Entity` необходимо произвести (добавить, удалить, изменить).
  - Элемент синхронизации (`LocalItem`) — один или несколько объектов из Creatio, которые синхронизируются с внешним хранилищем как одно целое. Элемент синхронизации из внешнего хранилища, конвертированный в сущности `LocalItem`, в свою очередь, содержит набор экземпляров класса `SyncEntity`.
- Таблица метаданных [`SysSyncMetadata`] — содержит служебную информацию синхронизируемых элементов, по сути является таблицей связки `RemoteItem - LocalItem`. Описание метаданных можно посмотреть в [статье](#).

## Общий алгоритм синхронизации

Перед началом синхронизации необходимо создать экземпляр `SyncAgent` и контекст синхронизации `SyncContext`, после этого актуализировать записи в таблице метаданных данными из Creatio. Для этого нужно вызвать метод `CollectChangesInSyncedEntities` класса, реализующего интерфейс `IReplicaMetadata`.

Общий алгоритм актуализации записей метаданных следующий:

1. Если какая-либо ранее синхронизированная сущность в Creatio была изменена с момента последней синхронизации, значит у соответствующей записи в метаданных изменяется дата модификации, свойству `LocalState` устанавливается значение "Изменен", а в качестве источника модификации устанавливается идентификатор `LocalStore`.
2. Если синхронизированная сущность в Creatio была удалена после последней синхронизации — у соответствующей записи в метаданных устанавливается `LocalState` "Удален".
3. Если для сущности в Creatio нет соответствующей записи в метаданных — она игнорируется.

Далее начинается процесс синхронизации хранилищ:

1. Поочередно запрашиваются все изменения из внешнего хранилища.
2. Для каждого элемента внешнего хранилища необходимо получить метаданные.
  - a. Если метаданные получить не удалось — это новый элемент, который конвертируется в элемент Creatio и сохраняется. Для заполнения объекта синхронизации, в приложении вызывается метод `FillLocalItem` у конкретного экземпляра `RemoteItem`. Также сохраняется запись в метаданных (`Id` внешнего хранилища, `Id` элемента во внешнем хранилище, дата создания и модификации устанавливается текущей, источник создания и модификации — внешнее хранилище).
  - b. Если получены метаданные, значит, этот элемент уже был синхронизирован с Creatio. Необходимо перейти к решению конфликта версий. По умолчанию приоритет имеют последние изменения в

приложении или внешнем хранилище (реализация в `RemoteProvider` ).

- с. Актуализируются метаданные для текущей пары элементов синхронизации.

После перебора всех измененных элементов из внешнего хранилища, в метаданных (в интервале между прошлой и текущей синхронизациями) остались элементы, которые были изменены в Creatio, но не были изменены во внешнем хранилище.

1. Необходимо получить измененные в Creatio элементы в промежутке между прошлой синхронизацией и началом текущей синхронизации.
2. Применяются изменения во внешнем хранилище.
3. Необходимо обновить дату модификации элементов в метаданных (источник изменения Creatio).

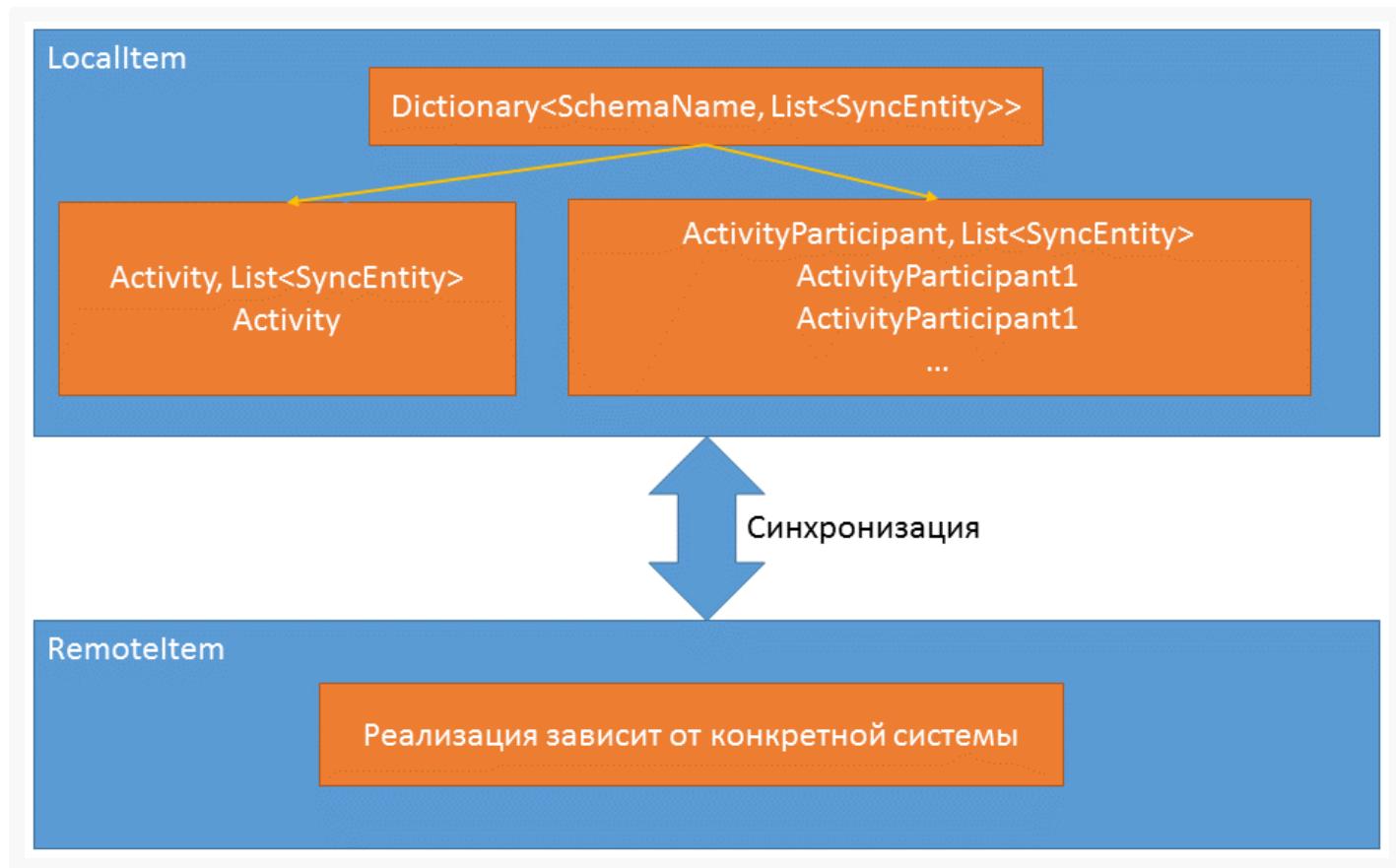
После этого необходимо новые, еще не синхронизированные записи из Creatio добавить во внешнее хранилище, а также добавить метаданные для новых элементов.

## Описание синхронизации

Активность и ее участники синхронизируются в одну задачу Google-календаря. Активность (`Activity`) и ее участники (`SyncEntity`) являются одним элементом синхронизации — `LocalItem`.

`RemoteItem` — это задача Google, которую получили извне Creatio. `LocalItem` — это набор объектов (`SyncEntity`), в которые в итоге конвертируется задача Google.

Схема синхронизации:



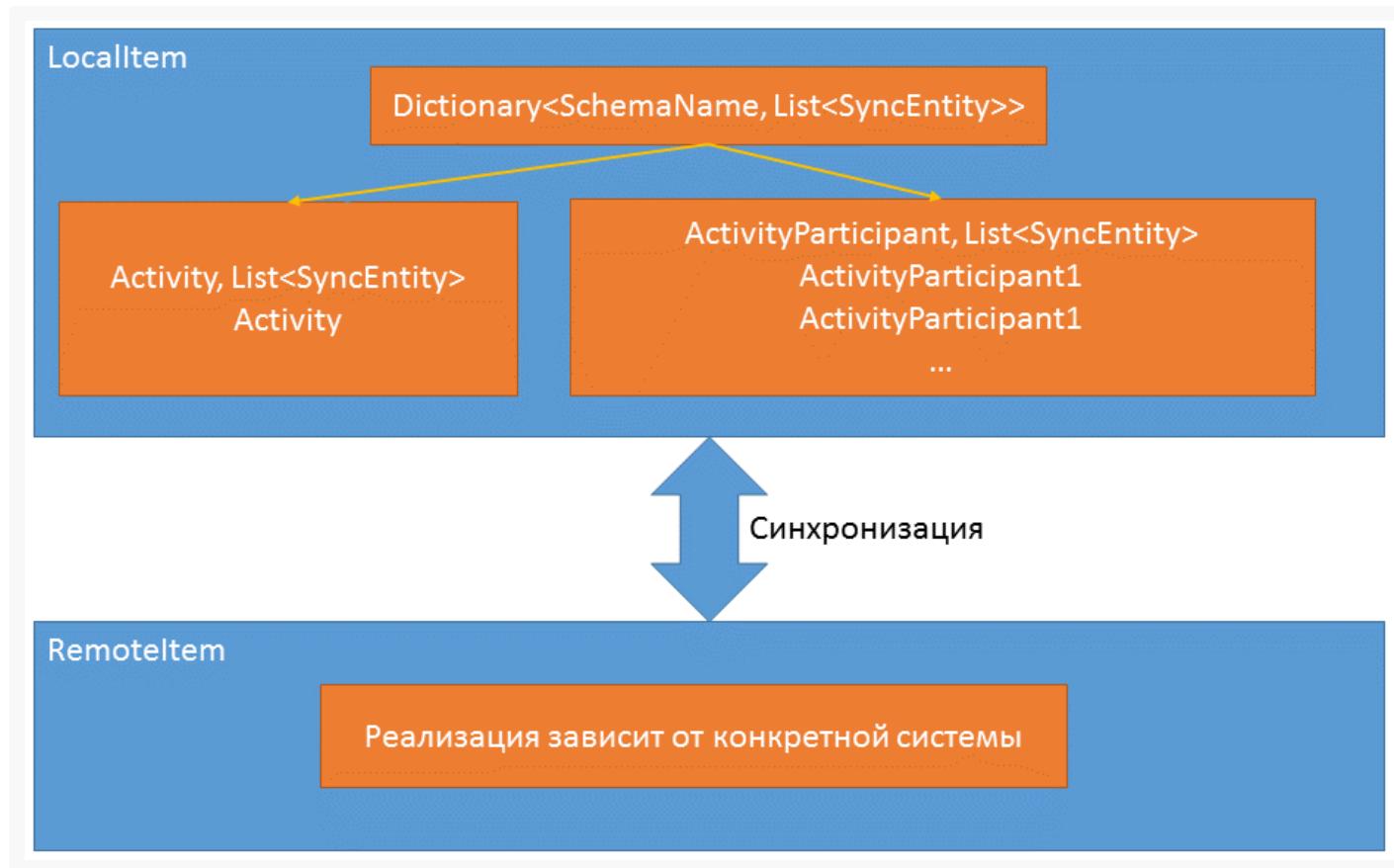
## Работа с метаданными для синхронизации

В метаданных хранится информация только об уже синхронизированных элементах.

Для одного элемента синхронизации может быть несколько записей в таблице метаданных — по одной для каждого объекта приложения, включенного в элемент синхронизации.

Активность и участники — это один элемент синхронизации, но в метаданных будет содержаться одна запись для активности и по одной записи для каждого участника.

На текущий момент только один объект из внешнего хранилища преобразуется в несколько объектов Creatio, как показано на рисунке. Вариант, когда допустимо соответствие многие ко многим (несколько внешних объектов соответствуют нескольким локальным объектам, и эта совокупность является одним элементом синхронизации), не поддерживается.



В системе метаданные для одного элемента синхронизации представлены объектом класса `ItemMetadata` (коллекция элементов `[SysSyncMetaDataTable]`). А управление метаданными осуществляется через класс, реализующий интерфейс `IReplicaMetadata`. Экземпляр класса, реализующий интерфейс `IReplicaMetadata`, создается через класс-фабрику `MetaDataTable` для конкретного хранилища.

## Класс SyncContext C#

Сложный

Пространство имен `Terrasoft.Sync`.

Класс, представляющий собой агрегацию провайдеров и метаданных для работы SyncAgent.

**Note.** Полный перечень методов и свойств класса `SyncContext`, его родительских классов, а также реализуемых им интерфейсов можно найти в документации "[.NET библиотеки классов ядра платформы](#)".

## Конструкторы

---

`SyncContext()`

Создает экземпляр класса.

## Свойства

---

`Logger ISyncLogger`

Объект, который позволяет сохранять сообщения в лог интеграции.

`LocalProvider LocalProvider`

Позволяет работать с `LocalItem`.

`RemoteProvider RemoteProvider`

Внешний сервис или приложение, данные из которого синхронизируются с Creatio.

`ReplicaMetadata IReplicaMetadata`

Реализует работу с метаданными.

`LastSyncVersion DateTime`

Дата и время последней синхронизации в UTC.

`CurrentSyncStartVersion DateTime`

Текущие дата и время синхронизации в UTC. Устанавливается после обновления метаданных.

## Методы

---

```
void LogError(SyncAction operation, SyncDirection direction, System.string format, params System
```

```
void LogError(SyncAction operation, SyncDirection direction, System.string format, System.Except
```

Записывает в журнал сообщение об ошибке.

#### Параметры

operation	Записывает действие для синхронизации объекта.
direction	Направление синхронизации.
format	Формат.
exception	Исключение, вызвавшее ошибку.
args	Формат параметров.

```
void LogInfo(SyncAction operation, SyncDirection direction, System.string format, params System.
```

Записывает в журнал информационное сообщение.

#### Параметры

operation	Записывать действие синхронизации объекта.
direction	Направление синхронизации.
format	Формат.
args	Формат параметров.

```
void SetLookupColumnValue(Entity entity, System.string lookupColumnName, System.string lookupDis
```

```
void SetLookupColumnValue(Entity entity, System.string lookupColumnName, System.string lookupDis
```

Устанавливает значение столбца поиска объекта.

#### Параметры

entity	Экземпляр класса <code>Terrasoft.Core.Entities.Entity</code> .
lookupColumnName	Имя справочного столбца.
lookupDisplayValue	Отображаемое значение справочного столбца.
lookupDisplayColumnName	Отображаемое имя справочного столбца.

# Класс RemoteProvider c#



Внешнее хранилище (`RemoteProvider`) — инкапсулирует работу с данными из внешнего хранилища.

`RemoteProvider` — базовый класс, реализующий работу с внешним хранилищем. По сути, единая точка работы с внешней системой. На данный момент его реализация дает много свободы в реализации внешнего провайдера.

## Свойства

`StoreId` `Guid`

Идентификатор внешнего хранилища, с которым будет происходить синхронизация.

`Version` `DateTime`

Дата и время последней синхронизации.

`SyncItemSchemaCollection` `List<SyncItemSchema>`

Схемы элементов внешнего хранилища.

`RemoteChangesCount` `Int`

Количество элементов, обработанных из внешнего хранилища.

`LocalChangesCount` `Int`

Количество элементов, обработанных из локального хранилища.

## Методы

`KnownTypes()` `IEnumerable<Type>`

Возвращает коллекцию всех типов, реализующих интерфейс `IRemoteItem`. `SyncAgent` строит на их основе экземпляры `SyncItemSchema`, которые описывают сущности, участвующие в синхронизации.

`ApplyChanges(SyncContext context, IRemoteItem synItem)` `Void`

Применяет изменения в элементе внешнего хранилища.

`CommitChanges(SyncContext context) Void`

Вызывается после обработки изменений во внешнем и локальном хранилище. Предназначен для реализации необходимых дополнительных действий для конкретной реализации интеграции.

---

`EnumerateChanges(SyncContext context) IEnumerable<IRemoteItem>`

Возвращает перечисление новых и измененных элементов внешнего хранилища.

---

`LoadSyncItem(SyncItemSchema schema, string id) IRemoteItem`

Заполняет экземпляр `IRemoteItem` данными из внешнего хранилища.

---

`CreateNewSyncItem(SyncItemSchema schema) IRemoteItem`

Создает новый экземпляр `IRemoteItem`.

---

`CollectNewItem(SyncContext context) IEnumerable<LocalItem>`

Возвращает перечисление новых сущностей Creatio, которые будут синхронизированы во внешнее хранилище.

---

`ResolveConflict(IRemoteItem syncItem, ItemMetadata itemMetaData, Guid localStoreId) SyncConflict`

Разрешает конфликт между измененными элементами локального и внешнего хранилища. По умолчанию (реализация в `RemoteProvider`) приоритет имеют изменения в Creatio.

---

`NeedMetaDataActualization() Boolean`

Возвращает признак необходимости актуализации метаданных до начала синхронизации.

---

`GetLocallyModifiedItemsMetadata(SyncContext context, EntitySchemaQuery modifiedItemsEsq) IEnumerable`

Возвращает элементы синхронизации, измененные в локальном хранилище после последней синхронизации.

## Интерфейс IRemoteItem C#



Сложный

Класс, реализующий интерфейс `IRemoteItem`, является неделимой единицей синхронизации и представляет собой один элемент синхронизации из внешнего хранилища данных. Класс одновременно является контейнером для данных, приходящих из внешней системы, и сам знает, как преобразовывать эти данные в сущности `Entity`, и наоборот — из `Entity` наполнить себя. Интерфейс содержит два

метода, `FillLocalItem` и `FillRemoteItem` для конвертации внешнего элемента синхронизации (`RemoteItem`, то есть себя) в `LocalItem` и наоборот.

## Методы

`FillLocalItem(SyncContext context, ref LocalItem localItem) Void`

Заполняет свойства элемента локального хранилища `LocalItem` значениями элемента внешнего хранилища. Используется для применения изменений в локальном хранилище.

`FillRemoteItem(SyncContext context, ref LocalItem localItem) Void`

Заполняет свойства элемента внешнего хранилища значениями из элемента локального хранилища `LocalItem`. Используется для применения изменений во внешнем хранилище.

## Атрибут Map C#

Пространство имен `Terrasoft.Sync`.

Атрибутом `Map` декорируются реализации интерфейса `IRemoteItem`.

Основным параметром атрибута является `SchemaName`. Это название той `EntitySchema`, которая участвует в текущем элементе синхронизации.

### Использование атрибута `Map`

```
[Map("Activity", 0)]
[Map("ActivityParticipant", 1)]
public class GoogleTask: IRemoteItem {
    ...
}
```

В таком объявлении класса задача из календаря Google будет синхронизироваться с активностью и участниками активности из Creatio.

Второй параметр `Order` определяет в каком порядке `Entity` будут сохранены в локальном хранилище. `Activity` указывается первой, т.к. `ActivityParticipant` хранит ссылку на создаваемую активность и при другом порядке будет вызвано исключение.

В большинстве случаев `SyncAgent` может автоматически сформировать запрос по выборке новых элементов синхронизации из Creatio. Для этого необходимо указать основную сущность и способ связи с деталями:

### Пример указания основной сущности и способа связи с деталями

```
[Map("Activity", 0, IsPrimarySchema = true)]
[Map("ActivityParticipant", 1, PrimarySchemaName = "Activity", ForeingKeyColumnName = "Activity")]
```

```
public class GoogleTask: IRemoteItem {
```

```
...
```

В данном случае в БД отправится один запрос на получение новых активностей и по одному запросу на каждую выбранную активность для получения связанных с ней участников. Список свойств атрибута представлен в таблице.

## Свойства

---

**SchemaName** String

Название схемы объекта.

---

**Order** Int

Порядок обработки сущности для элемента синхронизации.

---

**IsPrimarySchema** Boolean

Флаг, указывающий, что данная схема является основной в данном элементе синхронизации. Может быть установлен только у одной схемы.

---

**PrimarySchemaName** String

Имя схемы главного объекта. Не может указываться в паре с **IsPrimarySchema**.

---

**ForeignKeyColumnName** String

Имя колонки для связи детали с главным объектом. Не может указываться в паре с **IsPrimarySchema**.

---

**Direction** SyncDirection

Определяет направление синхронизации для объектов текущего типа. Значение по умолчанию — **DownloadAndUpload**.

Если не содержит флаг **Download** — изменения не будут применять в Creatio.

Если не содержит флаг **Upload** — не будут выбираться новые сущности из Creatio.

---

**FetchColumnNames** String[]

Названия колонок, которые будут загружаться из локального хранилища.

# Класс LocalProvider [c#](#)



Сложный

Пространство имен `Terrasoft.Sync`.

Локальное хранилище (`LocalProvider`) — инкапсулирует работу с данными во внутреннем хранилище (`Creatio`).

`LocalProvider` — базовый класс, реализующий работу с локальным хранилищем. Позволяет работать с `LocalItem` в `Creatio`. Методы данного класса являются неизменными, а их перечень представлен в таблице.

## Конструкторы

---

```
public LocalProvider(UserConnection userConnection)
```

Создает экземпляр класса с помощью `UserConnection`.

## Свойства

---

`StoreId` `Guid`

ID локального хранилища, которое будет синхронизировано.

---

`MaxItemsPerSelect` `int`

Максимальное количество элементов в выбранном количестве.

---

`UserConnection` `UserConnection`

Экземпляр `UserConnection`.

## Методы

---

`AddItemSchemaColumns(EntitySchemaQuery esqForFetching, EntityConfig entityConfig)` `Void`

Добавляет в `EntitySchemaQuery` колонки, указанные в `EntityConfig`.

---

`ApplyChanges(SyncContext context, LocalItem entities)` `Void`

Применяет изменения к каждому `SyncEntity` в `LocalItem`.

---

```
FetchItem(ItemMetadata itemMetaData, SyncItemSchema itemSchema, bool loadAllColumns = false) Loc
```

Загружает коллекцию сущностей, связанных с конкретным элементом синхронизации.

## Класс SyncEntity C#



Сложный

Пространство имен `Terrasoft.Sync`.

Класс `SyncEntity` инкапсулирует экземпляра `Entity` и все необходимые для выполнения действий синхронизации этого экземпляра свойства.

### Свойства

---

`EntitySchemaName String`

Название схемы, для которой создается обертка.

---

`Entity Entity`

`Entity`, для которой создается обертка.

---

`State SyncState`

Последнее действие, произведенное над `entity` (0 — не изменено, 1 — новое, 2 — изменено, 3 — удалено).

## Класс SyncItemSchema C#



Сложный

Пространство имен `Terrasoft.Sync`.

Схема настройки сущности элемента синхронизации.

### Свойства

---

`SyncValueName String`

Название типа сущности.

---

`SyncValueType Type`

Тип сущности.

---

`PrimaryEntityConfig EntityConfig`

Настройка основной сущности элемента синхронизации.

---

`Configs List<EntityConfig>`

Список настроек сущностей элемента синхронизации.

---

`DetailConfigs List<DetailEntityConfig>`

Список настроек сущностей деталей элемента синхронизации.

---

`Direction SyncDirection`

Определяет направление синхронизации для объектов текущего типа. Значение по умолчанию — `DownloadAndUpload`.

Если не содержит флаг `Download` — изменения не будут применяться в Creatio.

Если не содержит флаг `Upload` — не будут выбираться новые сущности из Creatio.

## Методы

---

`CreateSyncItemSchema(Type syncValueType) SyncItemSchema`

Создает настройки сущности элемента синхронизации со всеми настройками связанных сущностей.

---

`Validate(UserConnection userConnection) Void`

Метод проверяет правильность сформированного `EntityConfig`.

Если проверка не прошла — выдается исключение (в случае, если в `EntityConfig` имя схемы указано два раза — генерируется `DuplicateDataException`, если указано имя несуществующей схемы — `InvalidSyncItemSchemaException`).

---

`FetchItem(ItemMetadata itemMetaData, SyncItemSchema itemSchema, bool loadAllColumns = false) LocalCollection<EntityConfig>`

Загружает коллекцию сущностей, связанных с конкретным элементом синхронизации.

## Класс EntityConfig



Сложный

Пространство имен `Terrasoft.Sync`.

Настройка сущностей элемента синхронизации.

## Свойства

---

`SchemaName String`

Название схемы объекта.

`Order Int`

Порядок обработки сущности для элемента синхронизации. Чем меньше значение, тем раньше сущность будет обработана при обработке элемента синхронизации.

`Direction SyncDirection`

Определяет направление синхронизации для объектов текущего типа. Значение по умолчанию — `DownloadAndUpload`.

Если не содержит флаг `Download` — изменения не будут применяться в Creatio.

Если не содержит флаг `Upload` — не будут выбираться новые сущности из Creatio.

`FetchColumnNames String[]`

Названия колонок, которые будут загружаться из локального хранилища. Если при создании экземпляра значение не указано, то будут загружаться все колонки объекта.

## Класс DetailEntityConfig C#

 Сложный

Пространство имен `Terrasoft.Sync`.

Настройка сущностей деталей элемента синхронизации.

## Свойства

---

`PrimarySchemaName String`

Имя схемы основной синхронизируемой сущности Creatio.

`ForeignKeyColumnName String`

Имя колонки для связи детали с главным объектом.

# Класс LocalItem C#



Сложный

Пространство имен `Terrasoft.Sync`.

Представляет собой один или несколько объектов из Creatio, которые синхронизируются с внешним хранилищем как одно целое. Содержит набор экземпляров классов `SyncEntity`.

## Свойства

`Entities` `Dictionary<string, List<SyncEntity>>`

Коллекция `SyncEntity`, которая ставится в соответствие с одним `SyncItem`. Содержит коллекцию пар "ключ-значение", где ключ — имя схемы, а значение — коллекция `SyncEntity` этой схемы.

`Version` `DateTime`

Наибольшее значение даты и времени модификации из всех `Entities` в `LocalItem`.

`Schema` `SyncItemSchema`

Схема настройки сущности элемента синхронизации.

## Методы

`AddOrReplace(string schemaName, SyncEntity syncEntity) Void`

Добавляет новый `SyncEntity` в коллекцию. В случае если `SyncEntity` с таким `EntityId` уже существует, заменяет его.

`Add(UserConnection userConnection, string schemaName) SyncEntity`

Создает и добавляет новый `SyncEntity` в коллекцию.

# Класс SysSyncMetaData C#



Сложный

Для синхронизации используется вспомогательная таблица метаданных `SysSyncMetaData`, которая, по сути, является таблицей связки между внешним `RemoteItem` (элемент синхронизации во внешнем хранилище) и `LocalItem` (элемент синхронизации в Creatio). Каждая строка таблицы представляется в системе экземпляром класса `SysSyncMetaData`.

## Свойства

---

**RemoteId** String

Идентификатор элемента во внешнем хранилище.

---

**SyncSchemaName** String

Название схемы синхронизируемого элемента.

---

**LocalId** Guid

Идентификатор элемента в локальном хранилище.

---

**IsLocalDeleted** Boolean

Указывает, удален ли элемент из локального хранилища со времени последней синхронизации.

Параметр актуализируется перед синхронизацией и при применении изменений в локальном хранилище. На основе его значения при выборке измененных элементов из локального хранилища устанавливается состояние `SyncEntity`. Устарел, оставлен для совместимости. На текущий момент используется `LocalState`.

---

**IsRemoteDeleted** Boolean

Указывает, удален ли элемент из внешнего хранилища со времени последней синхронизации.

Устарел, оставлен для совместимости. На текущий момент используется `RemoteState`.

---

**Version** Date

Дата последнего выполненного изменения элемента (дата модификации).

---

**ModifiedInStoreId** Guid

`Id` хранилища, в котором выполнено последнее изменение.

---

**CreatedInStoreId** Guid

`Id` хранилища, в котором создан элемент синхронизации.

---

**RemoteStoreId** Guid

Идентификатор внешнего хранилища, с которым был синхронизирован элемент.

---

`ExtraParameters String`

Дополнительные параметры для элемента.

`LocalState Int`

Состояние элемента в локальном хранилище (0 — не изменен, 1 — новый, 2 — изменен, 3 — удален).

`RemoteState Int`

Состояние элемента во внешнем хранилище (0 — не изменен, 1 — новый, 2 — изменен, 3 — удален).

## Класс MetaDataStore C#



Сложный

Пространство имен `Terrasoft.Sync`.

Класс-фабрика создает конкретный экземпляр класса, который реализует интерфейс `IReplicaMetadata` для хранилища.

### Методы

`GetReplicaMetadata(Guid localStoreId, Guid remoteStoreId) IreplicaMetadata`

Создает экземпляр класса, реализующий интерфейс `IReplicaMetadata` для конкретного хранилища.

## Класс ItemMetaData C#



Сложный

Пространство имен `Terrasoft.Sync`.

Этот класс является неделимым объектом метаданных синхронизации. Содержит набор метаданных для каждой сущности синхронизации (коллекция элементов `SysSyncMetadata`).

### Свойства

`RemoteId String`

Идентификатор элемента во внешнем хранилище.

`RemoteItemName String`

Наименование элемента во внешнем хранилище.

# Интерфейс IReplicaMetadata C#



Пространство имен `Terrasoft.Sync`.

Класс, реализующий интерфейс `IReplicaMetadata`, инкапсулирует метаданные синхронизации. Реализует работу с объектами `ItemMetadata`.

## Свойства

`RemoteStoreId` `Guid`

Идентификатор внешнего хранилища.

`LocalStoreId` `Guid`

Идентификатор локального хранилища.

## Методы

`FindItemStore (string remoteItemId) ItemMetadata`

Находит и возвращает объект метаданных синхронизации `ItemMetadata` по идентификатору во внешнем хранилище `remoteItemId`.

`UpdateItemMetadata (ItemMetadata oldItemMetaDatas, IRemoteItem remoteItem, LocalItem localItem,`

Обновляет метаданные после синхронизации.

`EnumerateItemsWithChangesInBpm (SyncContext context) IEnumerable<ItemMetadata>`

Возвращает коллекцию объектов `ItemMetadata`, измененных после последней синхронизации в `Creatio` и не обработанных в процессе текущего сеанса синхронизации.

`CollectChangesInSyncedEntities (UserConnection userConnection, string schemaName, DateTime lastS`

Обновляет метаданные для элементов синхронизации, измененных в `Creatio`. Если элемент был изменен в приложении после последней синхронизации `SysMetadata`, колонка `Version` будет заполнена датой модификации элемента. Для актуализации метаданных используется хранимая процедура `ActualizeSysSyncMetaDataTable`.

`CollectNewDetailsForSyncedEntities (UserConnection userConnection, DetailEntityConfig detailEnti`  
 Создает новые записи в таблице `SysSyncMetaDataTable` для деталей элемента синхронизации.

`TryResolveRemoteId (Guid localId, out string remoteId) Boolean`

Возвращает идентификатор элемента во внешнем хранилище `remoteId` из метаданных по уникальному идентификатору элемента синхронизации в `Creatio` `localId`. Если элемент помечен как удаленный, то `remoteId` возвращен не будет и метод возвратит `false`.

`TryResolveExtraParameters (Guid localId, out string extraParameters) Boolean`

Возвращает дополнительные параметры `extraParameters` для элемента синхронизации из метаданных по `localId`. Если `extraParameters` не будут найдены, метод вернет `false`.

## Общие принципы работы портала



### Основы

**Портал Creatio** — это компонент low-code платформы, который представляет собой рабочее место пользователя и предоставляет ограниченный доступ к данным и функциональности основного приложения Creatio для внешних и внутренних пользователей, клиентов и партнеров компании. Интерфейс и [инструменты настройки портала](#) такие же, как в основном приложении Creatio, а разработка портала выполняется аналогично разработке основного приложения.

## Работа с порталом

Портал Creatio предоставляет возможности для решения широкого спектра бизнес-задач. Портал Creatio подходит для решения большого спектра задач, самыми популярными из которых являются:

- **Самообслуживание клиентов, например, в области технической поддержки.** Добавление канала самообслуживания для клиентов и переключение внимания сотрудников службы поддержки на более важные задачи, чем регистрация обращений. Возможность самостоятельного создания запросов клиентами и отслеживания прогресса по их разрешению непосредственно на портале. Предоставление доступа к статьям базы знаний помогает клиентам быстрее находить ответы на свои вопросы. Возможность одновременного обслуживания большого количества клиентов без очередей и потерь продуктивности.
- **Взаимодействие с внутренними и внешними клиентами, например, HR-портал.** Настройка возможности обслуживания внешних сотрудников и подрядчиков, которые не используют приложение Creatio для повседневной работы: создание заявки, передача их на рассмотрение и отслеживание изменения их состояний. Портал может стать источником всех важных документов и регламентов компании, которые находятся в свободном доступе.
- **Взаимодействие с внешними пользователями (клиентами, дилерами, партнерами) на всех этапах процесса продаж компании.** Создание партнерских программ, а затем, совместно с партнерами, обработка лидов и продаж, используя процессы управления лидами и корпоративными

продажами. Фиксация проведения обучающих мероприятий и сертифицированных специалистов партнера.

На портале Creatio пользователи могут получить доступ к разделам, доступным в рамках [лицензий](#) пользователя приложения и связанным с ними данным. Управлять доступом внешних пользователей к бизнес-данным на портале и контролировать сохранность и конфиденциальность важной информации возможно с помощью стандартного механизма [настройки прав доступа](#).

Пользователи портала могут:

- Создавать и изменять записи.
- Добавлять примечания.
- Прикреплять файлы.
- Оставлять сообщения для поддержки.
- Работать по процессам.

Управление доступом пользователей портала к данным системы осуществляется не только через права доступа, но и через справочник [*Список объектов, доступных пользователям портала*] ([*List of objects available for portal users*]). Только те объекты, которые перечислены в справочнике, будут доступны из интерфейса портала.

Наиболее частым сценарием является совместная работа пользователей портала и пользователей системы в одном разделе, но с разным набором полей на страницах. Для портальных пользователей введено ограничение на колонки, которые можно настроить в реестре раздела и колонки для фильтрации (в быстрых фильтрах или настройке динамических групп). При добавлении колонок на страницу записи с помощью мастера эти колонки по умолчанию добавляются справочник [*Список доступных полей объектов на портале*] ([*List of schema fields for portal access*]) и становятся доступными. Если поле не добавлено на страницу, но есть необходимость вывести его в реестр портала, то необходимо вручную добавить его в справочник. Внешний вид реестра и набор отображаемых колонок можно [настроить](#) индивидуально.

Все [пользователи, которые имеют доступ на портал](#), входят в группу организационных ролей "

`Все пользователи портала ("All portal users")`. В системе можно создавать индивидуальных пользователей портала или группировать их в организацию, привязав к определенному контрагенту.

Ограничения доступа к веб-сервисам для пользователя портала описаны в [статье](#).

Портал Creatio доступен в следующих конфигурациях:

- **Портал самообслуживания.** Портал самообслуживания может использоваться как основной канал регистрации обращений, так и в качестве дополнения к уже существующим каналам. Пользователь портала может найти информацию в базе знаний или создать запрос в службу поддержки. Данная конфигурация позволяет работать только с разделами [*Обращения*] ([*Case*]) и [*База знаний*] ([*Knowledge base*]), которые можно кастомизировать с помощью дизайнера системы. Добавление дополнительных разделов на портал самообслуживания не поддерживается.
- **Клиентский портал.** Клиентский портал предназначен для автоматизации процессов, например, предоставления услуг, обработки заявок и запросов на обслуживание. Пользователь портала может инициировать процессы (например, создание заказов, запросов) или участвовать в них (например, утверждать заявки). Такой тип конфигурации портала позволяет настраивать и использовать до трех пользовательских разделов на портале. Пользовательский раздел сначала нужно создать в приложении и затем на его основе создать портальную версию этого раздела. Разделы [*Обращения*] ([*Case*]) и [*База знаний*] ([*Knowledge base*]), также доступны пользователям клиентского портала, а

для пользователей продукта Creatio Bank Customer Journey будет доступен раздел [ Заявки ] ([ Applications ]). Дополнительно можно вывести на портал раздел [ Документы ] ([ Documents ]), если он доступен в основном приложении. Создание раздела на основе лицензируемого объекта из состава базового продукта не считается пользовательским разделом.

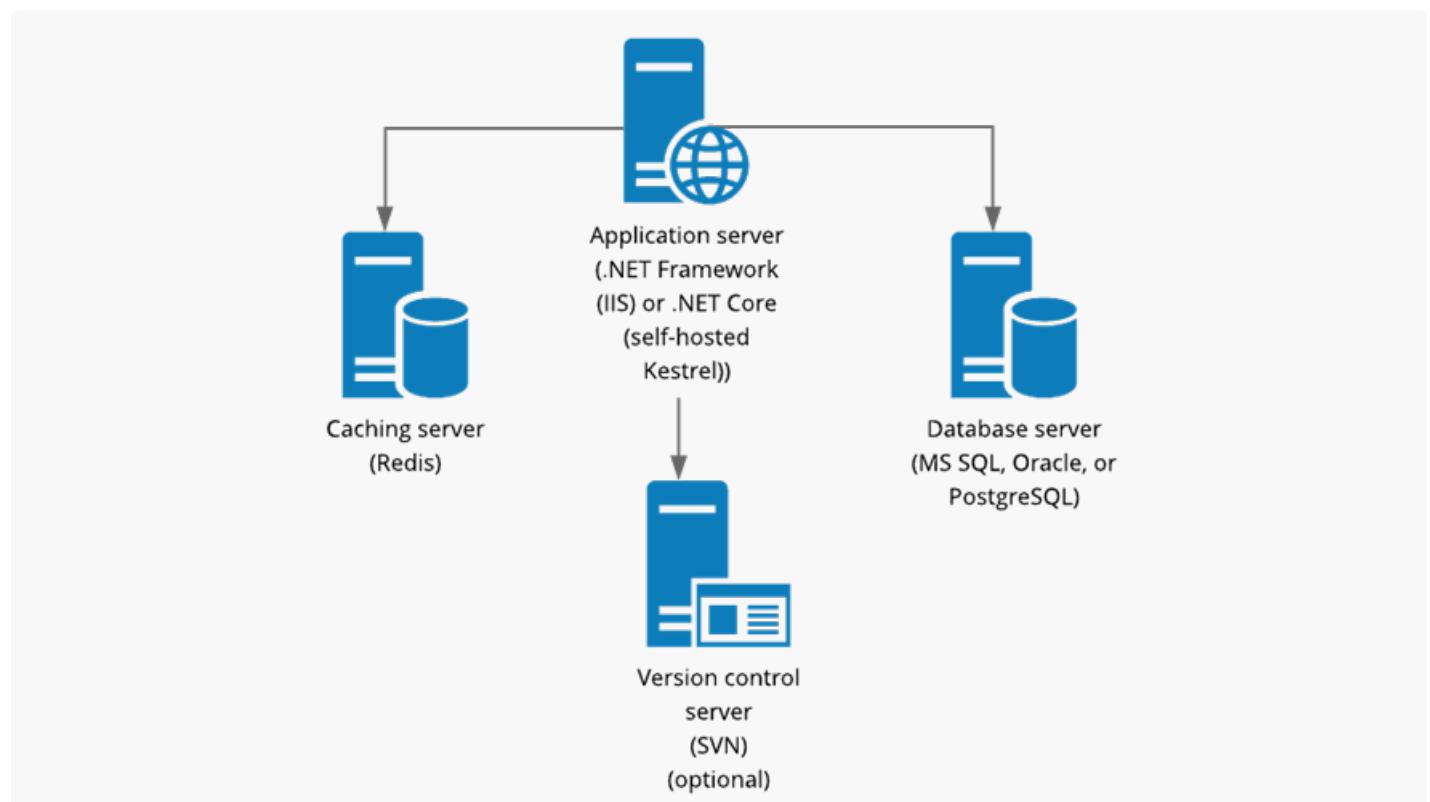
- **[Партнерский портал](#)**. Партнерский портал предназначен для компаний, работающих с клиентами через сеть партнеров. Партнерский портал является единой платформой для коммуникаций и совместной с партнерами работы над лидами и продажами. В данной конфигурации пользователям портала доступны разделы [ Партнерская программа ] ([ Partner program ]), а также [ Лиды ] ([ Leads ]), [ Продажи ] ([ Opportunities ]), [ Маркетинговые активности ] ([ Marketing activities ]) и [ Заказы ] ([ Orders ]).

Конфигурации портала могут использоваться одновременно. Пользователям каждой из конфигураций доступна базовая функциональность портала (главная страница, настройки в профиле пользователя, база знаний, сегментация данных), а перечень разделов, доступных на портале, различается в зависимости от конфигурации портала.

Описание работы с порталом содержится в [статье](#).

## Схема архитектуры портала

Схема архитектуры портала Creatio не отличается от схемы архитектуры [основного приложения](#).

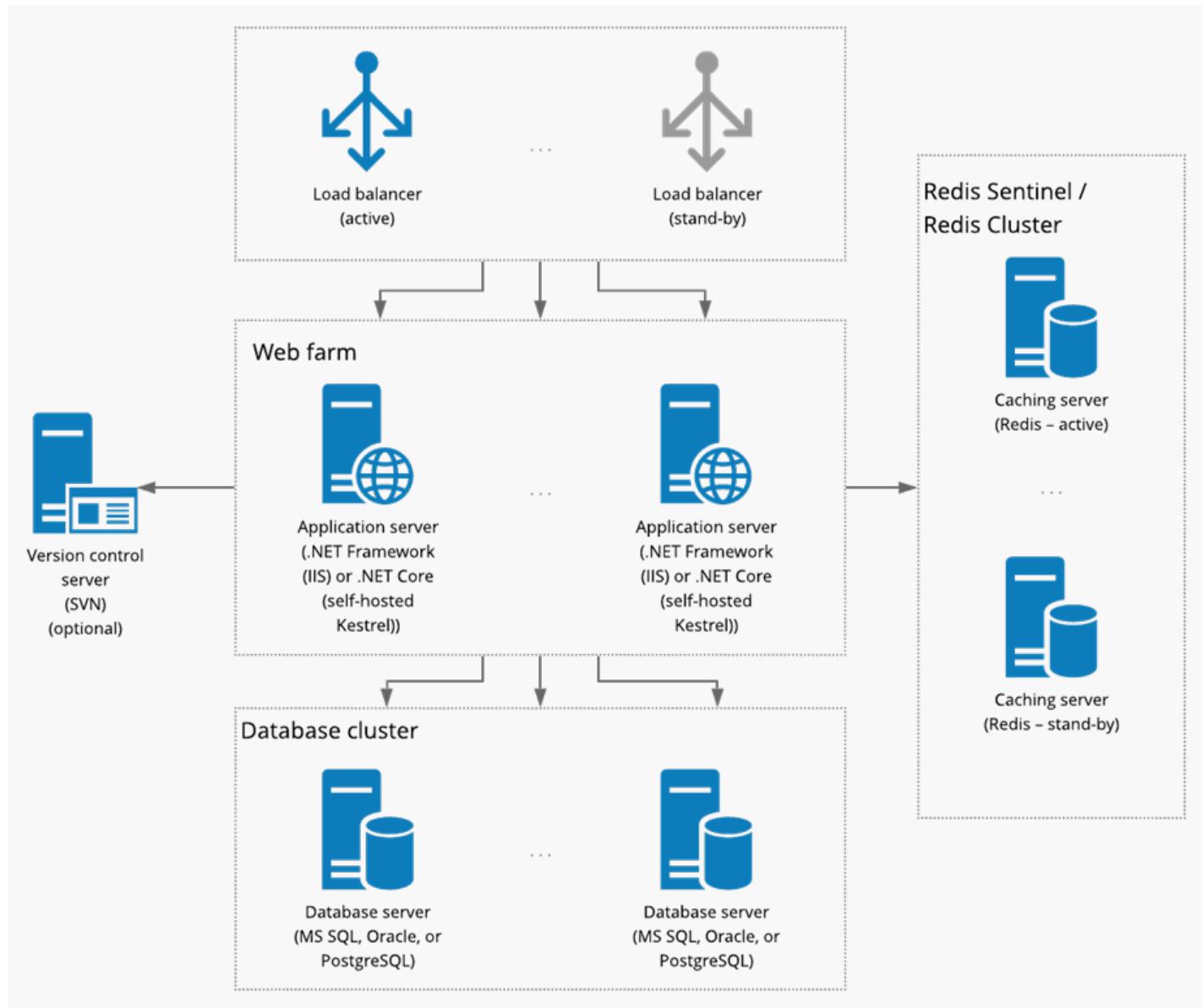


## Масштабируемость портала

В Creatio существует возможность повысить производительность крупных проектов с помощью [горизонтального масштабирования](#). При использовании горизонтального масштабирования схема архитектуры портала включает в себя следующие компоненты:

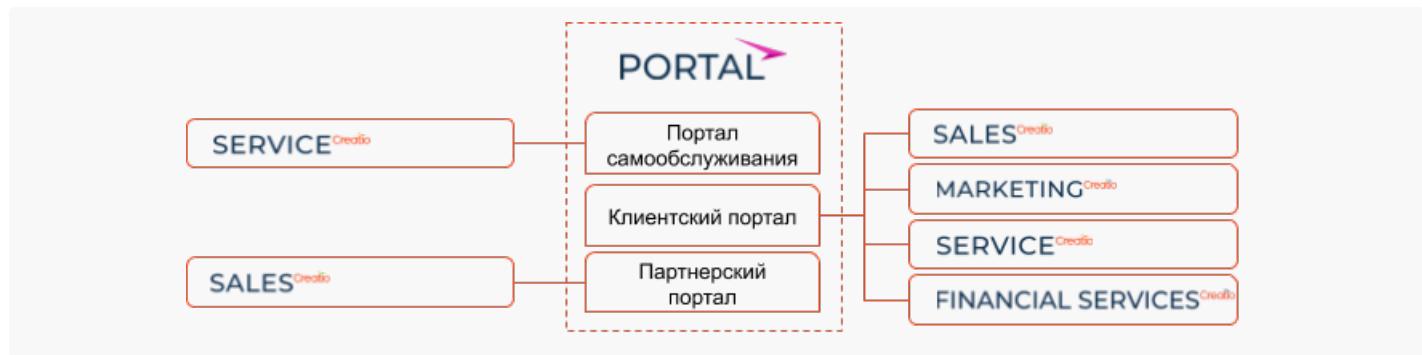
- [Балансирующие нагрузки](#). Балансирующий нагрузки может быть аппаратным или программным. Для работы в отказоустойчивом режиме используется балансировщик HTTP/HTTPS-трафика с поддержкой протокола WebSocket. Описание установки и настройки балансировщика HAProxy содержится в [статье](#).
- [Веб-ферма](#) (несколько серверов приложений).
- [Сервер кэширования Redis или Redis Sentinel / Кластер Redis](#) (несколько серверов кэширования Redis).
- Сервер баз данных или кластер баз данных (несколько серверов баз данных).
- Сервер системы контроля версий (опционально). Описание настройки сервера системы контроля версий содержится в [статье](#).

Схема архитектуры портала Creatio при использовании веб-фермы, кластера Redis и кластера баз данных



Настройка безопасного доступа к порталу, а также типовая схема установки системы с доступом к порталу из внешней сети описана в [статье](#).

## Совместимость портала с продуктами Creatio



## Варианты развертывания портала

Портал является частью платформы Creatio и доступен в базовой конфигурации как в приложениях, развернутых как cloud, так и on-site.

Для обеспечения безопасности данных при установке портала on-site приложение должно быть развернуто с использованием веб-фермы. Настройка веб-фермы описана в [статье](#). Установка приложения с доступом к порталу из внешней сети описана в [статье](#).

## Web-to-Case



Функциональность Web-to-Case реализует возможность создания обращений в Creatio посредством заполнения необходимых полей формы, встроенной на сторонний сайт — лендинга.

Пакет `ProductCore` зависит от пакета `WebForms`, в котором и содержится функциональность Web-to-Case. Это означает, что лендинги могут использоваться во всех продуктах. Преднастроенная "коробочная" функциональность реализована в продуктах `service enterprise`, `customer center`, `marketing` и всех бандлах, включающих эти продукты.

Подробнее о лендингах можно узнать из документации соответствующих продуктов, например, из блока статей "[Раздел \[ Лендинги и web-формы \]](#)" документации Marketing Creatio.

Настройку Web-to-Case можно произвести, воспользовавшись интерфейсом системы, но для внедрения полученного JavaScript-скрипта на сторонний сайт необходимы общие базовые навыки Web-разработчика.

Базовая функциональность Web-to-Case без использования программирования в Creatio (но используя небольшие доработки на стороннем сайте) позволяет настроить следующее:

- Внешний вид формы и ее стили.
- Список дополнительно передаваемых полей.
- Список подстановочных значений по умолчанию для полей, не выведенных в форму.
- Список доменов, из которых будет возможна регистрация обращения по каждому лендингу.
- Адрес перехода, на который будет переадресован пользователь после отправки формы.

- JavaScript-обработчики событий успешной и неуспешной регистрации обращения.
- Дополнительные лендинги, каждый из которых может быть сконфигурирован по-своему, что дает возможность различить обращения, созданные с разных сайтов.

При помощи минимальных проектных доработок можно настроить предварительный обработчик регистрации обращения через Web-to-Case, в котором можно выполнить проверку данных, их коррекцию, создание связанных сущностей и т. п. В базовой поставке Creatio в обработчике регистрации обращения через Web-форму настроено автоматическое создание контакта для регистрируемого обращения.

## Логика автоматического заполнения полей для обращения

При регистрации обращения через Web-форму предлагаются для заполнения следующие поля: [ ФИО ], [ Email ], [ Телефон ], [ Тема обращения ]. Значение поля [ Тема обращения ] будет передаваться в новое обращение.

По сочетанию полей [ ФИО ], [ Email ] и [ Телефон ] в Creatio идентифицируется контакт. Поиск происходит следующим образом:

1. Если существует контакт, у которого поля [ФИО], [Email] и [Телефон] совпадают с заполненными данными формы, то он подставляется в создаваемое обращение.
2. Иначе, если есть контакт, у которого совпадают только поля [ФИО] и [Email], то он подставляется в создаваемое обращение.
3. Иначе, если есть контакт, у которого совпадает только поле [Email], то он подставляется в создаваемое обращение.
4. Иначе создается новый контакт, и у него заполняются поля [ФИО], [Email] и [Телефон]. Созданный контакт подставляется в регистрируемое обращение.

Если по одному правилу найдено несколько контактов, то в качестве контакта обращения будет взят первый найденный. Также автоматически заполнится дата регистрации обращения (колонка `RegisteredOn`) текущей датой и временем.

## Рекомендации для выполнения проектных решений

Если необходима кастомизация функциональности Web-to-Case, проще всего руководствоваться существующим базовым решением как примером.

Для выполнения проектного решения рекомендуется выполнить следующие действия:

1. Создать схему страницы, унаследованную от `CaseGeneratedWebFormPageV2`. Страница не должна быть замещающей.
2. Добавить запись нового типа лендинга в таблицу `LandingType` и локализацию в таблицу `SysLandingTypeLcz`.
3. Зарегистрировать стандартным образом типизированную страницу, созданную на первом шаге (значение типа — новое созданное).
4. Если необходима предварительная обработка данных формы до сохранения записи в базе данных, то

нужно создать класс, реализующий интерфейс `IGeneratedWebFormPreProcessHandler`. Этот класс представляет собой предварительный обработчик регистрации обращения. Также нужно реализовать метод `Execute()`. Этот метод — точка входа в обработчик. В нем реализуются все вспомогательные действия. В качестве примера можно взять схему `WebFormCasePreProcessHandler`.

5. Если необходимо выполнить действия после сохранения обращения в базу данных, то нужно создать класс, реализующий интерфейс `IGeneratedWebFormPostProcessHandler`. Этот класс представляет собой предварительный обработчик регистрации обращения. Далее нужно реализовать метод `Execute()`, в котором выполнить необходимые действия.
6. Если созданы обработчики регистрации обращения, то нужно зарегистрировать их в таблице `WebFormProcessHandlers`. В качестве примера регистрации можно взять уже существующую запись.
7. Отредактировать шаблон скрипта, формирующего конфигурационный JavaScript-объект лендинга, и поместить в локализируемую строку `ScriptTemplate` созданной страницы. Аналогичный скрипт указать для всех используемых локализаций. Пример скрипта можно найти в схеме `CaseGeneratedWebFormPageV2`.
8. Привязать все созданные данные к пакету.

Алгоритм создания Web-to-Case лендинга:

1. Создать новую запись лендинга в Creatio.
2. Создать посадочную страницу, в которой будет содержаться сгенерированный в системе код, связывающий форму лендинга и запись лендинга.
3. Добавить посадочную страницу на сайт.

# Создать Web-to-Case лендинг

 Сложный

**Пример.** Создать Web-to-Case лендинг.

## 1. Создать новую запись лендинга в Creatio

Чтобы создать новую запись лендинга, необходимо в разделе [Лендинги и web-формы] ([Lending pages and web forms]) выполнить действие [Добавить] ([Add]). В открывшейся странице нужно заполнить следующие поля:

- [Название] ([Name]) — заголовок лендинга в Creatio.
- [Домены сайта] ([Website domains]) — URL посадочной страницы.
- [Состояние] ([Status]) — состояние лендинга.
- [Адрес перехода] ([Redirection URL]) — URL страницы, на которую переходит клиент после регистрации на посадочной странице.

The screenshot shows the 'Landing pages and web forms' section of the Creatio interface. A landing page named 'MyLanding' is being configured. The 'Website domains' field contains 'localhost'. The 'Status' field is set to 'Active'. A 'Redirection URL' is specified as <http://bpmonline.com>. On the right, there's a code editor with some JavaScript code related to tracking.

Поскольку из посадочной страницы при создании обращения можно получить только четыре поля ("Subject", "Email", "Name" и "Phone"), то для новой записи лендинга необходимо установить значения по умолчанию.

The screenshot shows the 'Landing pages and web forms' section of the Creatio interface. A landing page named 'MyLanding' is being configured. The 'Website domains' field contains 'localhost'. The 'Status' field is set to 'Active'. A 'Redirection URL' is specified as <http://bpmonline.com>. The 'DEFAULT VALUES' tab is selected in the top navigation bar. A table shows default values for 'Account' (Axiom) and 'Category' (Incident).

Для применения изменений страницу нужно сохранить.

## 2. Создать посадочную страницу

Чтобы создать Web-страницу для лендинга нужно в любом текстовом редакторе при помощи HTML-разметки создать обычную HTML-страницу, содержащую Web-форму.

Для регистрации в Creatio данных, отправляемых через web-форму, в ее код необходимо добавить четыре поля (HTML-элемент `<input>`), определяющие обращение:

- тема обращения;

- Email контакта;
- имя контакта;
- телефон контакта

Для каждого поля нужно указать атрибуты `name` и `id`.

Чтобы при отправке данных формы в Creatio создавался новый объект [Обращение], в HTML-страницу нужно добавить скрипт на языке JavaScript. Исходный код скрипта необходимо скопировать из поля [ШАГ 2. Скопируйте код и настройте в нем соответствие полей] ([STEP 2. Copy the code and configure and map the fields]) страницы редактирования лендинга.

Скрипт необходимо скопировать из уже сохраненного лендинга.

Скрипт содержит конфигурационный объект `config`, в котором определены следующие свойства:

- `fields` — содержит объект со свойствами "Subject", "Email", "Name" и "Phone", значения которых должны совпадать с селекторами атрибутов `id` соответствующих полей формы.
- `landingId` — содержит идентификатор лендинга в базе данных.
- `serviceUrl` — содержит URL службы, по которому будут отправляться данные формы.
- `redirectUrl` — содержит URL адреса перехода, указанного в поле [Адрес перехода] лендинга.
- `onSuccess` — содержит функцию-обработчик успешного создания обращения. Необязательное свойство.
- `onError` — содержит функцию-обработчик ошибки создания обращения. Необязательное свойство.

Конфигурационный объект `config` передается в качестве аргумента функции `createObject()`, которая должна выполняться при отправке формы.

Чтобы функция `createObject()` была вызвана при отправке формы, в тег формы HTML-страницы лендинга необходимо добавить атрибут `onSubmit="createObject(); return false"`.

### Пример полного исходного кода посадочной страницы для регистрации обращений

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <!--ШАГ 2-->
    <!--Эту часть необходимо скопировать из поля ШАГ 2 страницы редактирования лендинга-->
    <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.min.js"></script>
    <script src="https://webtracking-v01.creatio.com/JS/track-cookies.js"></script>
    <script src="https://webtracking-v01.creatio.com/JS/create-object.js"></script>
    <script>
        /**
         * Replace the "css-selector" placeholders in the code below with the element selectors
         * You can use #id or any other CSS selector that will define the input field explicitly
         * Example: "Email": "#MyEmailField".
         * If you don't have a field from the list below placed on your landing, leave the place
         */
    
```

```

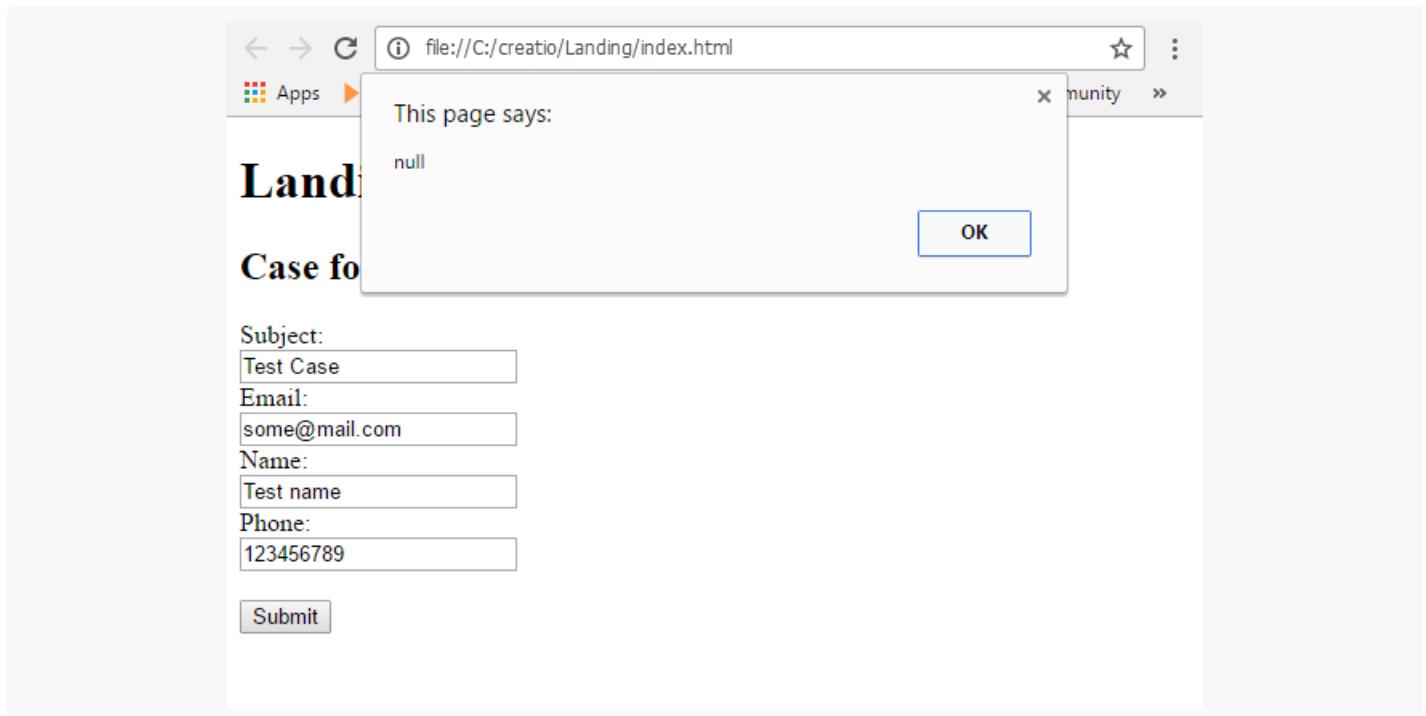
var config = {
    fields: {
        "Subject": "#subject-field", // Case subject
        "Email": "#email-field", // Visitor's email
        "Name": "#name-field", // Visitor's name code
        "Phone": "#phone-field", // Visitor's phone number
    },
    landingId: "8ab71187-0428-4372-b81c-fd05b141a2e7",
    serviceUrl: "http://localhost/creationservice710/0/ServiceModel/GeneratedObjectWebFor",
    redirectUrl: "http://creatio.com",
    onSuccess: function(response) {
        window.alert(response.resultMessage);
    },
    onError: function(response) {
        window.alert(response.resultMessage);
    }
};

/**
 * The function below creates a object from the submitted data.
 * Bind this function call to the "onSubmit" event of the form or any other elements eve
 * Example: <form class="mainForm" name="landingForm" onSubmit="createObject(); return f
 */
function createObject() {
    landing.createObjectFromLanding(config)
}
</script>
<!--ШАГ 2-->
</head>
<body>
<h1>Landing web-page</h1>
<div>
    <h2>Case form</h2>
    <form class="mainForm" name="landingForm" onSubmit="createObject(); return false">
        Subject:<br>
        <input type="text" name="subject" id="subject-field"><br>
        Email:<br>
        <input type="text" name="Email" id="email-field"><br>
        Name:<br>
        <input type="text" name="Name" id="name-field"><br>
        Phone:<br>
        <input type="text" name="Phone" id="phone-field"><br><br>
        <input type="submit" value="Submit">
    </font>
    </form>
</div>
</body>
</html>

```

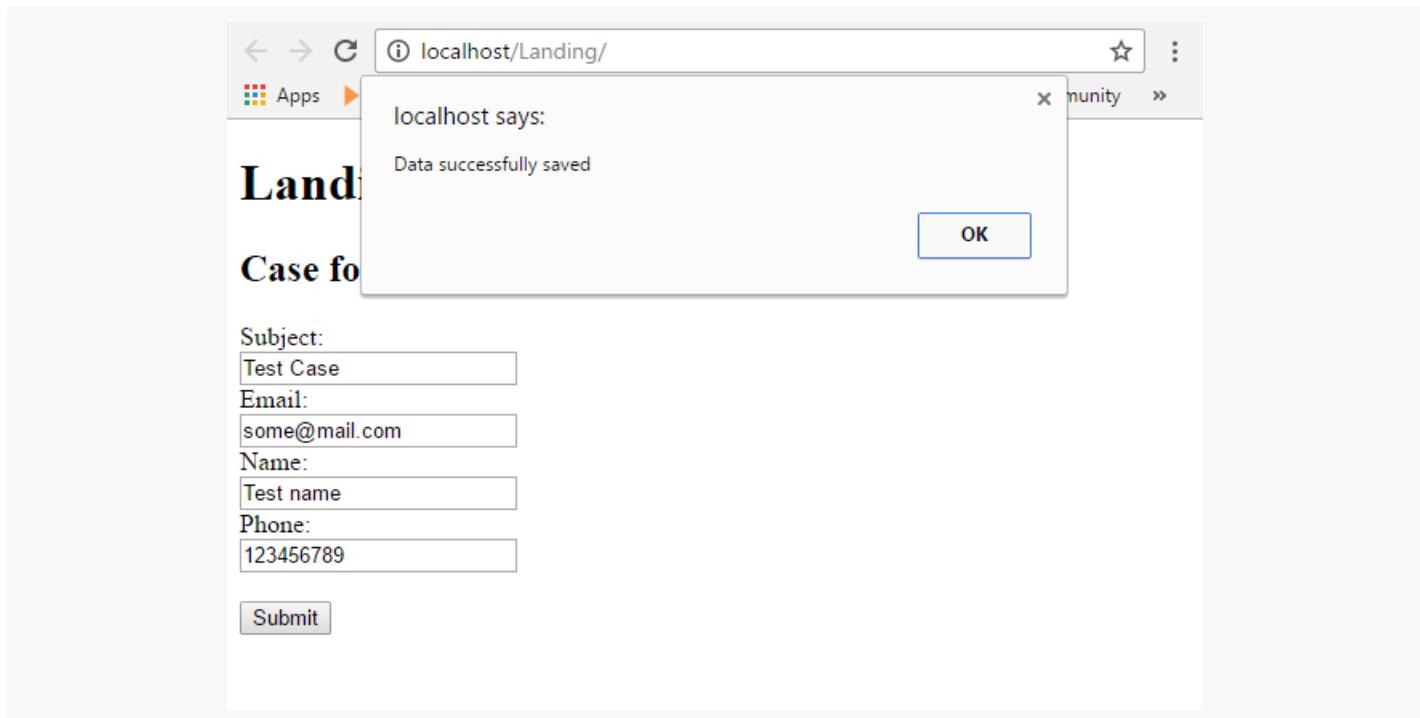
### 3. Добавить Web-страницу на сайт

Обращение с посадочной страницы будет добавлено в Creatio только в том случае, если страница размещена на сайте, имя которого указано в поле [ Домены сайта ] лендинга. Если открыть страницу в браузере локально, то при создании обращения будет выведено пустое сообщение.



Вывод пустого сообщения настроен в методе-обработчике `onError()` конфигурационного объекта.

Если разместить страницу на локальном сервере компьютера, обслуживающий зарезервированное доменное имя `localhost` (как указано в настройке лендинга), то скрипт добавления обращения с Web-страницы лендинга отработает корректно.



## Результат выполнения примера

В результате в системе будет автоматически создано обращение с указанными параметрами.

The screenshot shows a service management application interface. On the left, there is a sidebar with navigation links: Service, Contacts, Accounts, Cases (which is highlighted in orange), Activities, Services, Service agreements, Configuration items, Problems, and Changes. The main area displays a case record for 'Case #SR00000008: Test Case'. The case status is 'New'. The 'History' tab is selected, showing an email from 'John Best' to 'test@gmail.com' dated 5/5/2017 at 10:58 AM via Email. The email body contains a welcome message and information about the registered incident.

What can I do for you? > Creatio

Service

Contacts

Accounts

Cases

Activities

Services

Service agreements

Configuration items

Problems

Changes

Case #SR00000008: Test Case

SAVE CANCEL ACTIONS TAKE IT |

Resolution time

Priority: Medium

Contact: User Test

Account: Axiom

SLA: 4 — Axiom

Category: Incident

Service: Configuration item

Assignees group

New In progress Waiting for re... Resolved Closed

NEXT STEPS (0)

PROCESSING CLOSURE AND FEEDBACK CASE INFORMATION ATTACHMENTS

History

John Best to: test@gmail.com 5/5/2017 at 10:58 AM via Email

Hello, User Test!

According to your request, we have registered the Incident #SR00000008 "Test Case".

Description: Test Case.

Created on: 5/5/2017 7:58:14 AM;

Read more

Draft

# Отчеты FastReport



**Отчет FastReport** — документ, который сгенерирован на основе записей разделов Creatio в формате \*.pdf.

## Настроить доступ к разделу [ Настройка отчетов ]

Доступ к разделу [ Настройка отчетов ] ([ Report setup ]) настраивается на уровне системных операций. Если пользователь не имеет доступа к разделу [ Настройка отчетов ] ([ Report setup ]), то отображается стандартное сообщение с указанием операции и недостающих прав. По умолчанию доступ к основным системным операциям имеют только администраторы приложения. Creatio предоставляет возможность настройки доступа к системным операциям для пользователей или групп пользователей. Подробнее читайте в статье [Настроить права доступа на системные операции](#).

Чтобы **настроить доступ к разделу** [ Настройка отчетов ] ([ Report setup ]):

- Перейдите в дизайнер системы по кнопке . В блоке [ Пользователи и администрирование ] ([ Users and administration ]) перейдите по ссылке [ Права доступа на операции ] ([ Operation permissions ]).
- Выберите системную операцию [ Доступ к разделу "Настройка отчетов" ] ([ Access to "Report setup" section ], код `CanManageReports` ).
- На детали [ Доступ к операции ] ([ Operation permission ]) нажмите и укажите получателя прав.

В результате запись отобразится на детали [ Доступ к операции ] ([ Operation permission ]) в колонке [ Уровень доступа ] ([ Access level ]) со значением [ Да ] ([ Yes ]). Пользователи, которые входят в указанную роль, получат доступ к системной операции [ Доступ к разделу "Настройка отчетов" ] ([ Access to "Report setup" section ], код `CanManageReports` ).

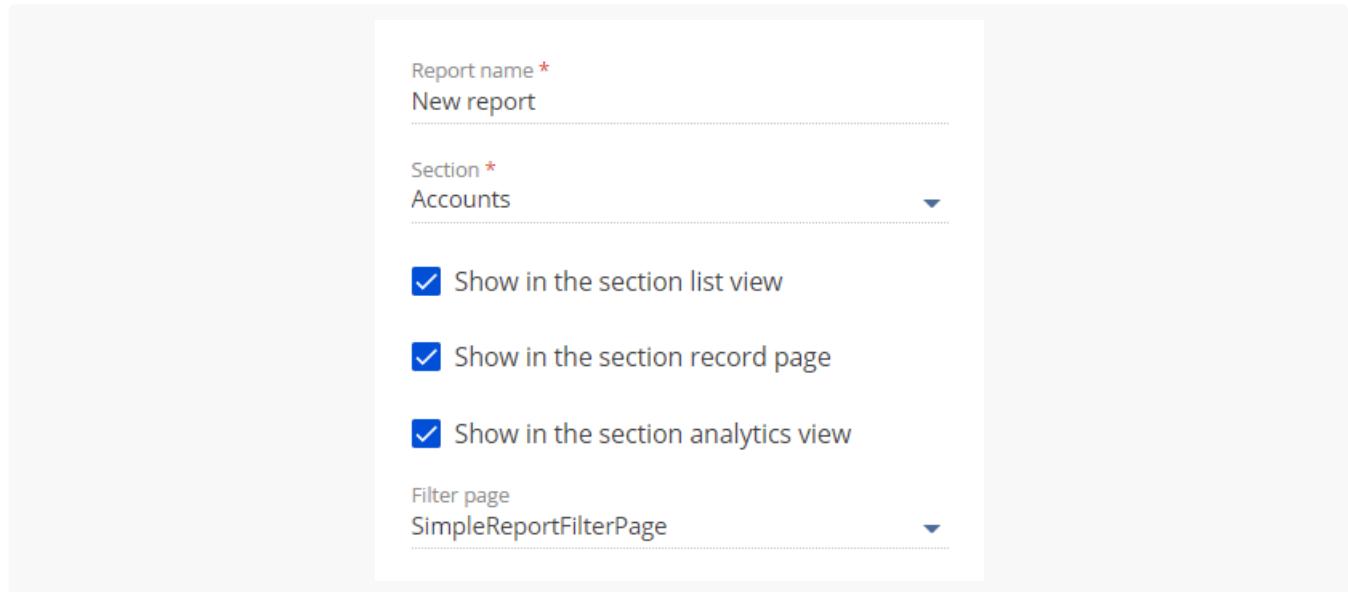
## Создать простой отчет FastReport

1. Создайте **отчет FastReport**.

- Перейдите в дизайнер системы по кнопке . В блоке [ Настройка системы ] ([ System setup ]) перейдите по ссылке [ Настройка отчетов ] ([ Report setup ]).
- Выполните действие [ Добавить отчет ] —>[ FastReport ] ([ New report ] —>[ FastReport ]).

- На панели свойств заполните **свойства отчета**:

- [ Название отчета ] ([ Report title ]) — пользовательское название отчета (обязательное свойство). Отображается на соответствующей панели инструментов, которая зависит от установленных признаков [ Отображать в разделе ] ([ Show in the section list view ]), [ Отображать на странице записи ] ([ Show in the section record page ]), [ Отображать в аналитике раздела ] ([ Show in the section analytics view ]).
- [ Раздел ] ([ Section ]) — выберите раздел, из которого планируется генерировать отчет (обязательное свойство). Например, чтобы отобразить отчет в разделе [ Контрагенты ] ([ Accounts ]), в выпадающем списке выберите соответствующий раздел.
- Признак [ Отображать в разделе ] ([ Show in the section list view ]) — указывает необходимость отображения отчета в выпадающем меню кнопки [ Печать ] ([ Print ]) панели инструментов раздела.
- Признак [ Отображать на странице записи ] ([ Show in the section record page ]) — указывает необходимость отображения отчета в выпадающем меню кнопки [ Печать ] ([ Print ]) панели инструментов страницы записи.
- Признак [ Отображать в аналитике раздела ] ([ Show in the section analytics view ]) — указывает необходимость отображения отчета в выпадающем меню кнопки [ Отчеты ] ([ Reports ]) панели инструментов аналитики раздела. При установленном признаком позволяет использовать пользовательскую страницу дополнительной фильтрации записей для отчета.
- [ Страница фильтрации ] ([ Filter page ]) — пользовательская страница дополнительной фильтрации записей для отчета. Свойство отображается при установленном признаке [ Отображать в аналитике раздела ] ([ Show in the section analytics view ]). Позволяет задать параметры фильтрации записей после выбора отчета в выпадающем меню кнопки [ Отчеты ] ([ Reports ]) аналитики раздела. Можно выбрать стандартную страницу фильтрации [ SimpleReportFilterPage ] или создать пользовательскую страницу фильтрации. Для пользовательской страницы фильтрации в качестве родительского объекта (обязательное свойство [ Родительский объект ] ([ Parent object ]) клиентской схемы) выберите схему `BaseReportFilterPage` пакета `FastReport`.



**Страница фильтрации** [ `SimpleReportFilterPage` ] — клиентская схема, которая содержит реализацию стандартных простых фильтров.

**Параметры** фильтрации, которые позволяет выбрать страница фильтрации `SimpleReportFilterPage`

:

- [ По выделенным записям ] ([ *Selected records* ]).
- [ По отфильтрованным в разделе записям ] ([ *Filtered records in list* ]).
- [ По всем записям без учета фильтрации ] ([ *All records in list* ]).

## 2. Укажите источники данных.

- a. В блоке [ Укажите источники данных для отчета ] ([ *Specify data sources for the report* ]) рабочей области страницы настройки отчета укажите источники данных. Для указания источников данных используйте формат `json`.

**Данные**, которые необходимо указать:

- Перечень объектов.
- Колонки объектов.
- Связи колонок объектов, которые используются для получения данных.
- Локализуемые строки (опционально).

Шаблон, который необходимо использовать для указания источника данных, приведен ниже.

### Шаблон источника данных

```
{
    "ProviderName": "YourProviderName",
    "Schemas": {
        "TableName1": {
            "ColumnName1": {"DataValueType": DataValueType1},
            ...,
            "ColumnNameN": {"DataValueType": DataValueTypeN}
        },
        ...,
        "TableNameN": {
            "ColumnName1": {"DataValueType": DataValueType1},
            ...,
            "ColumnNameN": {"DataValueType": DataValueTypeN}
        },
        "LocalizableStrings": {
            "LocalizedString1": {"DataValueType": 1},
            ...,
            "LocalizedStringN": {"DataValueType": 1}
        }
    }
}
```

`ProviderName` — класс провайдера данных.

`Schemas` — структура таблиц для шаблона отчета.

`TableName1...TableNameN` — имена таблиц базы данных или виртуальных таблиц, колонки которых необходимо добавить в отчет.

`ColumnName1...ColumnNameN` — имена колонок, которые необходимо добавить в отчет.

`DataValueType1...DataValueTypeN` — типы данных соответствующих колонок

`ColumnName1...ColumnNameN`.

`LocalizableStrings` — локализуемые строки.

`LocalizedString1...LocalizedStringN` — имена локализуемых строк отчета.

`"DataValueType": 1` — тип данных `TEXT` соответствующих локализуемых строк  
`LocalizedString1...LocalizedStringN`.

`DataValueType` — параметр, который содержит значение из перечисления

`Terrasoft.core.enums.DataValueType`. Перечисление `Terrasoft.core.enums.DataValueType` описано в [Библиотеке .NET классов](#).

- f. На панели инструментов страницы настройки отчета нажмите [ Применить ] ([ *Apply* ]).

### 3. Реализуйте **провайдер данных отчета**.

**Провайдер данных отчета** — пользовательский класс на языке C#.

a. Создайте схему типа [ Исходный код ] ([ *Source code* ]). Для этого воспользуйтесь инструкцией, которая приведена в статье [Исходный код \(C#\)](#).

b. В дизайнере исходного кода реализуйте **класс провайдера данных**.

- В дизайнере исходного кода добавьте пространство имен `Terrasoft.Configuration` или любое вложенное в него пространство имен.
- Для класса добавьте атрибут `[DefaultBinding]` с необходимыми параметрами. Параметр `Name` совпадает с параметром `YourProviderName`, который был указан в источнике данных.
- Создайте класс, который является наследником `Terrasoft.Configuration.Reporting.FastReport.IFastReportDataSourceDataProvider`.
- Реализуйте **методы класса провайдера данных**.
  - `GetLocalizableStrings(UserConnection)` — метод, который локализует поля отчета.
  - `ExtractFilterFromParameters(UserConnection, Guid, IReadOnlyDictionary)` — метод, который добавляет фильтры интерфейса.
  - `GetData(UserConnection, IReadOnlyDictionary)` — метод, который возвращает значение типа `Task<ReportDataDictionary>`. Содержит реализацию логики получения данных отчета.

### 4. Настройте **шаблон отчета**.

a. Скачайте файл отчета FastReport. Для этого в блоке [ Скачайте файл с источниками данных и сформируйте шаблон отчета ] ([ *Download file with data sources to design a report in the FastReport Designer* ]) рабочей области страницы настройки отчета нажмите кнопку [ Скачать файл ] ([ *Download file* ]).

Download file with data sources to design a report in the FastReport Designer

Open the downloaded file in the FastReport Designer and set up the report layout. The downloaded file will contain the data source structure that you can use in the FastReport Designer.

[DOWNLOAD FILE](#)

В результате скачан файл в формате \*.frx.

b. Откройте **шаблон отчета**.

a. Откройте дизайнер отчетов FastReport.

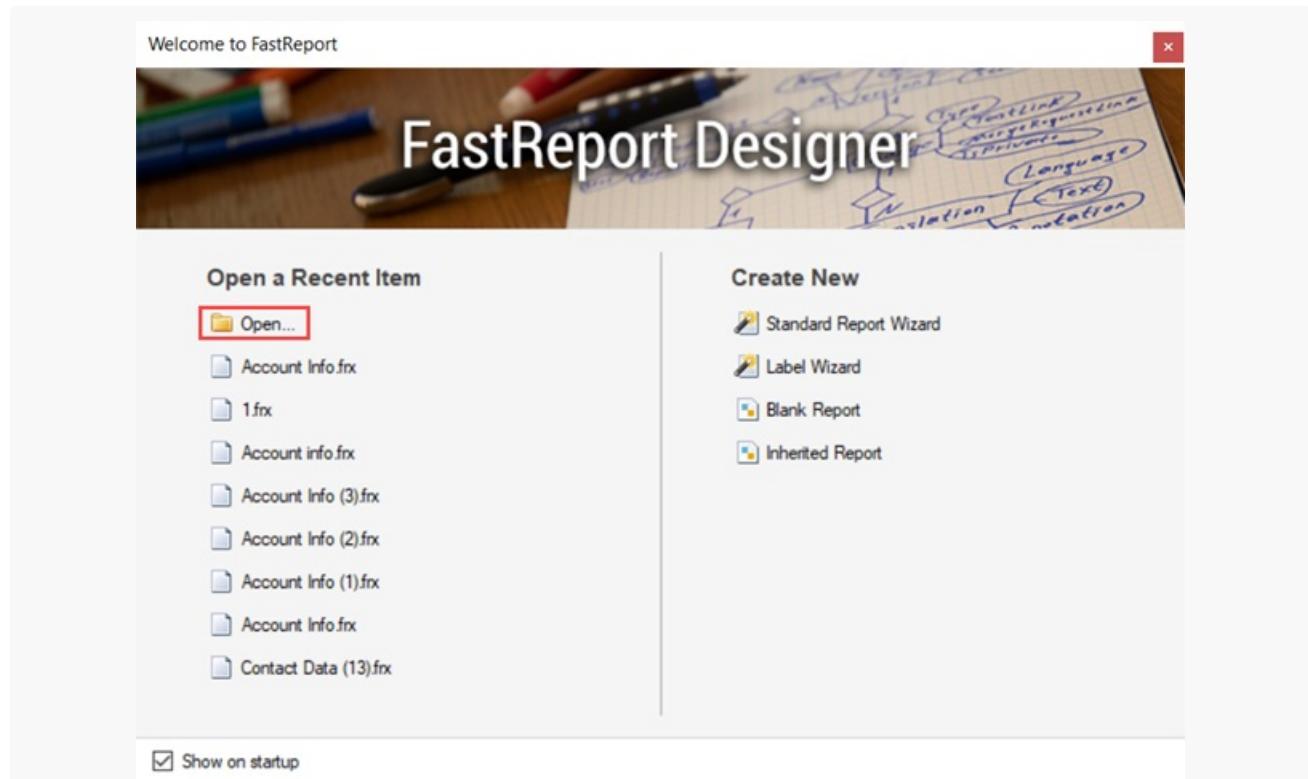
**Компоненты**, которые необходимы для работы дизайнера отчетов FastReport:

- Операционная система Windows.
- 64-разрядный Microsoft .Net Framework 4.7.2.

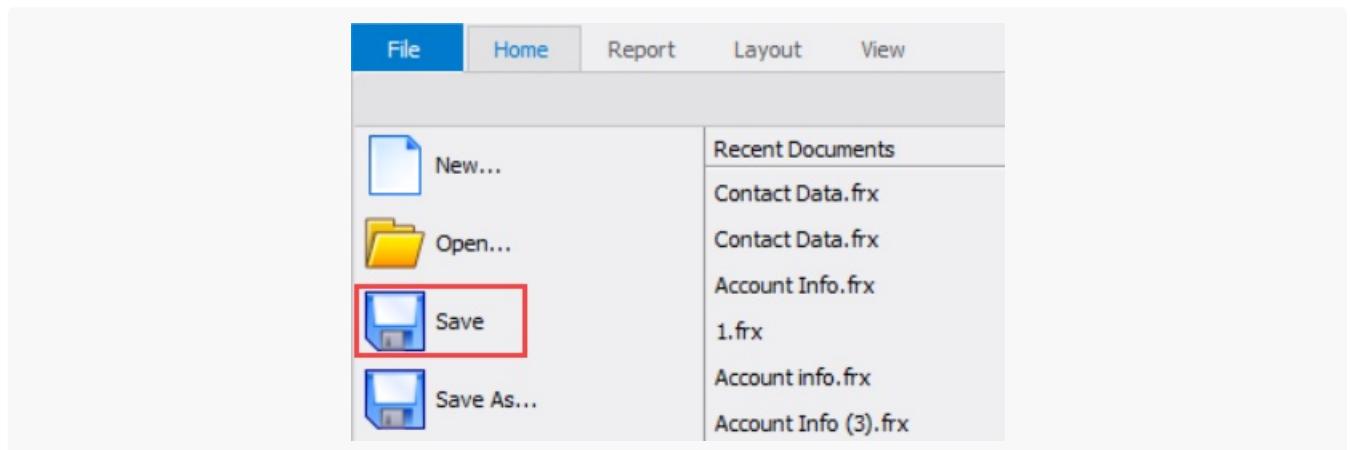
Разархивируйте [\\*.zip-архив](#) и запустите файл `Terrasoft.Reporting.FastReport.Designer.exe`.

Name	Type	Compressed size
FastReport.Bars.dll	Application extension	1,731 KB
FastReport.dll	Application extension	3,101 KB
FastReport.Editor.dll	Application extension	338 KB
<b>Terrasoft.Reporting.FastReport.Designer.exe</b>	Application	3 KB
Terrasoft.Reporting.FastReport.Designer.exe.config	CONFIG File	1 KB

d. Выберите **отчет FastReport**, который планируется настроить. Для этого в блоке [ Open a Recent Item ] окна [ Welcome to FastReport ] нажмите на кнопку [ Open... ].



- е. Перейдите в каталог со скачанным отчетом (обычно это каталог `Downloads`), выберите предварительно скачанный \*.frx-файл и нажмите на кнопку [ *Open* ].  
В шаблоне сохраняется структура источников данных, которая реализована на странице настройки отчета.
- с. Настройте шаблон отчета. Выполнение настройки шаблона описано в официальной [документации FastReport](#).  
Поскольку дизайнер отчетов FastReport является сторонним приложением, функция предпросмотра настроенного отчета недоступна.
- д. Сохраните шаблон отчета. Для этого в меню [ *File* ] панели инструментов нажмите на кнопку [ *Save* ].



- е. Загрузите настроенный шаблон отчета в Creatio.
- а. В блоке [ *Загрузите настроенный шаблон отчета в Creatio* ] ([ *Import a file with the report template* ]) рабочей области страницы настройки отчета нажмите на кнопку [ *Загрузить шаблон* ] ([ *Upload template* ]).

#### Import a file with the report template

After you import the .frx file, the report will appear in the "Print" menu of the section or on the record page as per the report settings.

**UPLOAD FILE**

- б. Перейдите в каталог с настроенным шаблоном отчета (обычно это каталог `Downloads`), выберите файл с отчетом и нажмите на кнопку [ *Open* ].

В результате настроенный шаблон отчета загружен в Creatio.

**Способы** генерации отчета FastReport, которые зависят от настроек на странице настройки отчета:

- Выпадающее меню кнопки [ *Печать* ] ([ *Print* ]) панели инструментов раздела.
- Выпадающее меню кнопки [ *Отчеты* ] ([ *Reports* ]) панели инструментов аналитики раздела.
- Кнопка [ *Печать* ] ([ *Print* ]) активной записи реестра раздела.
- Выпадающее меню кнопки [ *Печать* ] ([ *Print* ]) панели инструментов страницы записи.

Отчет генерируется в формате \*.pdf.

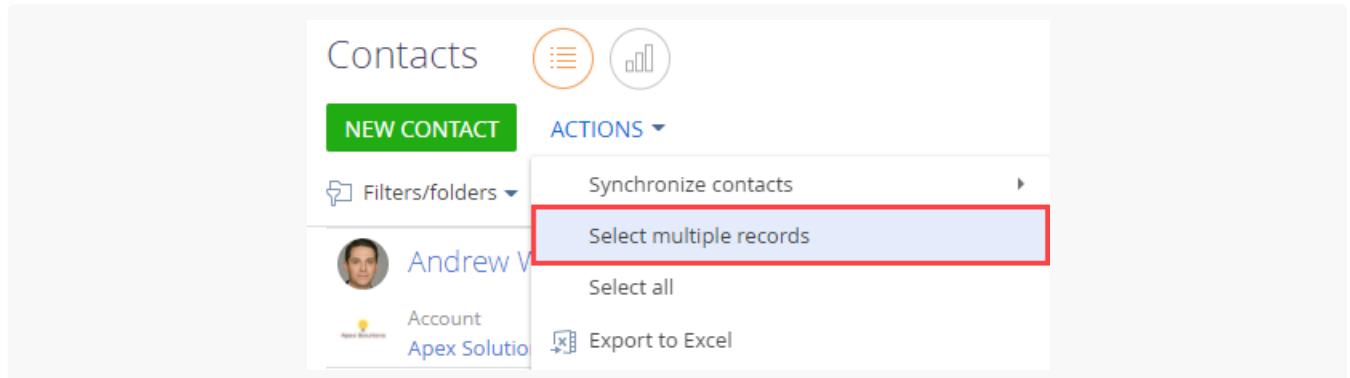
Если в реестре раздела не выбрана запись, то отчет не активен.

Чтобы **сгенерировать отчет FastReport по нескольким записям раздела**:

1. Обновите страницу соответствующего раздела.
2. Выберите несколько записей раздела.

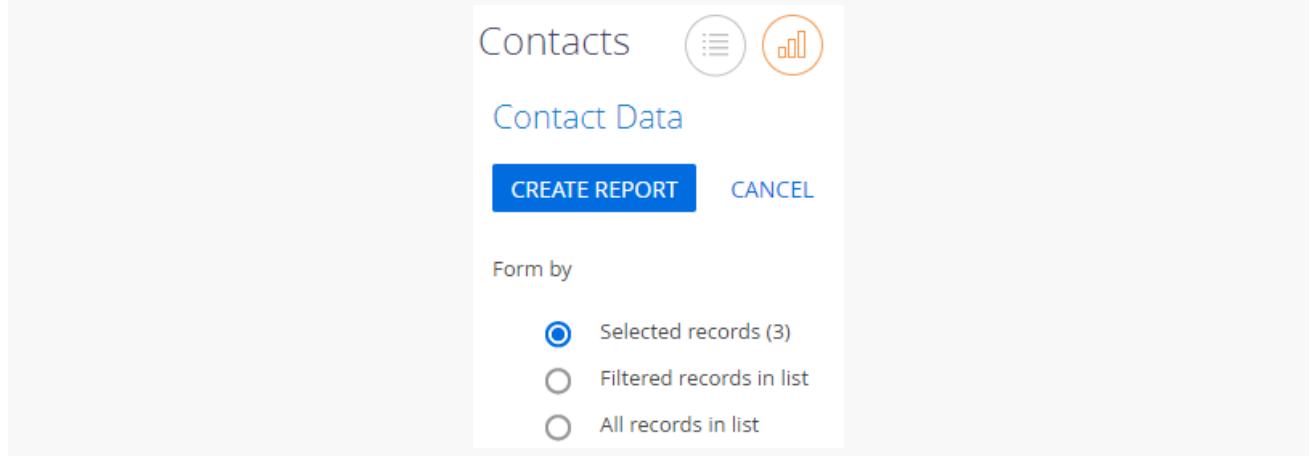
**Способы** выбора нескольких записей раздела:

- **В реестре раздела** (на странице настройки отчета установлен признак [ Отображать в разделе ] ([ *Show in the section list view* ])). Для этого на панели инструментов раздела выполните действие [ Действия ] —> [ Выбрать несколько записей ] ([ *Actions* ] —> [ *Select multiple records* ]).



- **В аналитике раздела** (на странице настройки отчета установлен признак [ Отображать в аналитике раздела ] ([ *Show in the section analytics view* ])).
  - На панели инструментов раздела выполните действие [ Действия ] —> [ Выбрать несколько записей ] ([ *Actions* ] —> [ *Select multiple records* ]).
  - Нажмите на кнопку .
  - В выпадающем меню кнопки [ Отчеты ] ([ *Reports* ]) панели инструментов раздела выберите отчет, который необходимо сгенерировать.

В результате откроется страница фильтрации.

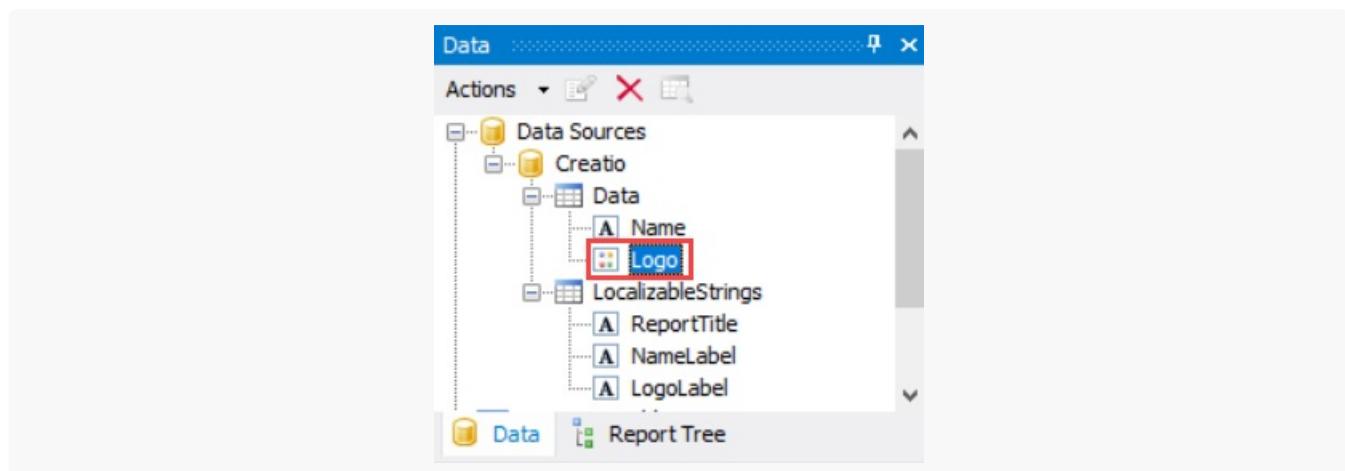


- Отфильтруйте записи (опционально). Для этого на странице фильтрации выберите опцию [ По

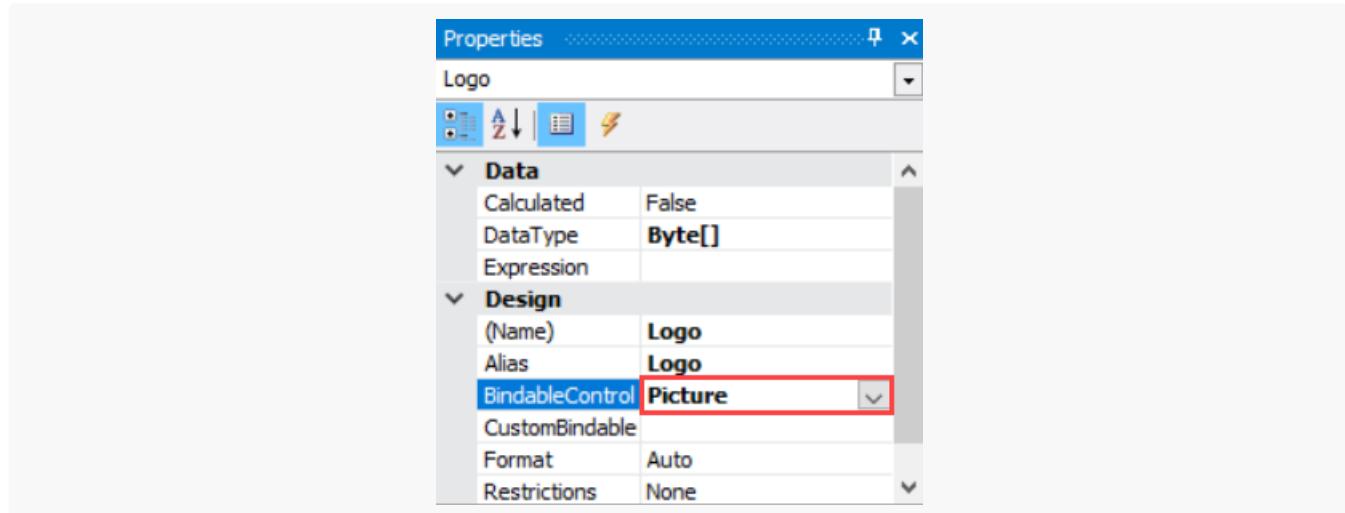
- отфильтрованным в разделе записям ] ([ Filtered records in list ]).*
- Для скачивания отчета нажмите на кнопку [ Создать отчет ] ([ Create report ]).

## Создать отчет FastReport с изображением

1. Выполните шаги 1-4 [алгоритма создания простого отчета FastReport](#).
2. На шаге 4 настройте **отображение изображения в шаблоне отчета**.
  - a. На вкладке [ Data ] панели свойств [ Data ] выполните [ Data Sources ] —> [ Creatio ] —> [ Data ] и кликните по элементу, который является изображением. Например, элемент [ Logo ], как представлено на рисунке ниже.



- b. В группе [ Design ] панели свойств [ Properties ] элемента, который является изображением, в выпадающем списке свойства [ BindableControl ] выберите значение "Picture".



- c. Добавьте элемент, который является изображением, в рабочую область дизайнера отчетов.

## Настроить мультиязычие элементов интерфейса отчета FastReport

Чтобы **настроить мультиязычие элементов интерфейса отчета FastReport**, используйте раздел [ Переводы ] ([ *Translations* ]) дизайнера системы. Работа с разделом [ Переводы ] ([ *Translations* ]) описана в статье [Перевести элементы интерфейса в разделе \[ Переводы \]](#).

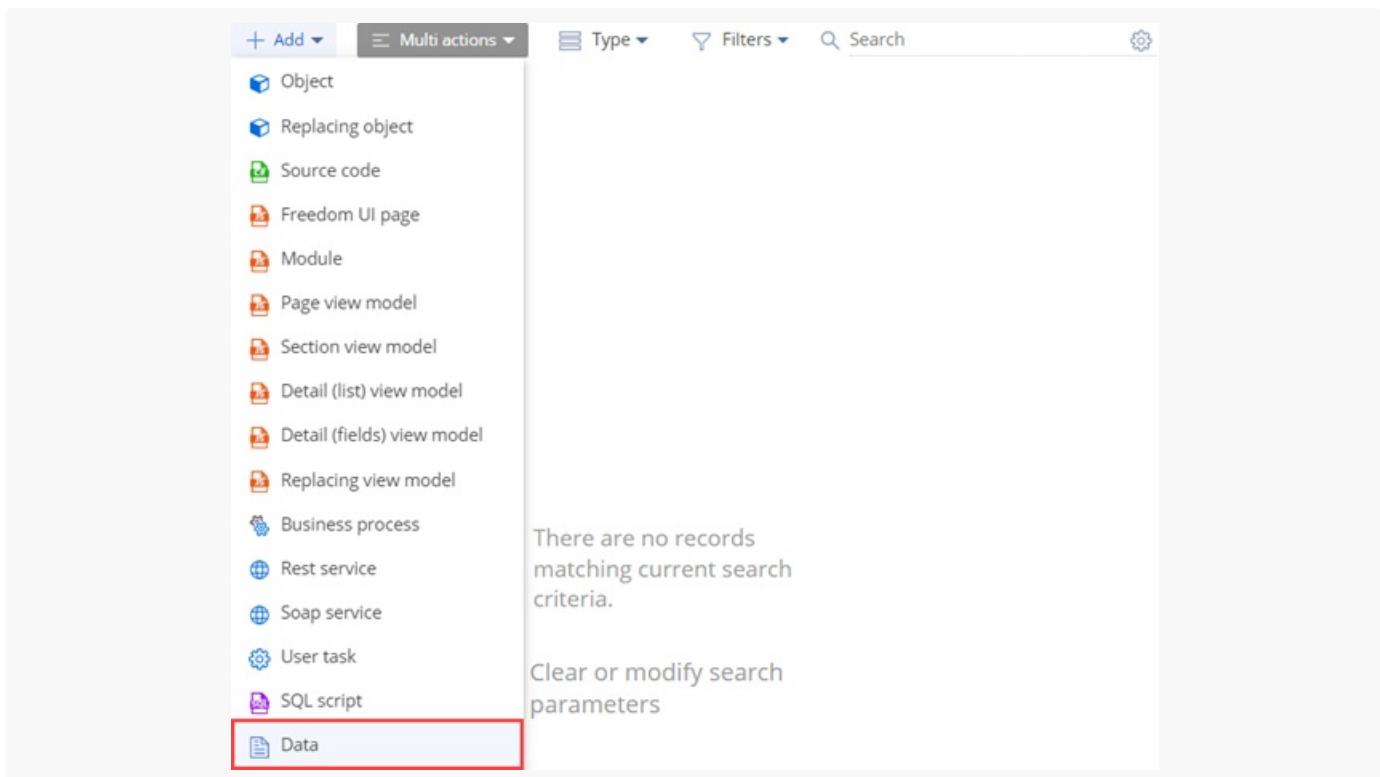
**Особенности** поиска элементов интерфейса отчета FastReport в разделе [ Переводы ] ([ *Translations* ]):

- Для поиска ранее локализуемых строк отчета используйте шаблон ключа Configuration:НазваниеСхемы (например, Configuration:UsrContactDataSourceCode ).
- Для поиска полей отчета используйте шаблон ключа Configuration:НазваниеСхемы:НазваниеПоля.Value (например, Configuration:UsrContactDataSourceCode:LocalizableStrings.ReportTitle.Value ).
- Для поиска отображаемого названия отчета при установленных на странице настройки отчета признаках [ Отображать в разделе ] ([ *Show in the section list view* ]) и [ Отображать на странице записи ] ([ *Show in the section record page* ]) используйте шаблон ключа Configuration:НазваниеСхемы:Caption (например, Configuration:UsrContactDataSourceCode:Caption ).
- Для поиска отображаемого названия отчета при установленном на странице настройки отчета признаком [ Отображать в аналитике раздела ] ([ *Show in the section analytics view* ]) используйте шаблон ключа Data:SysModuleAnalyticsReport.Caption:ИдентификаторПоля (например, Data:SysModuleAnalyticsReport.Caption:d52e8b78-772b-77ee-3394-bdb3616d859a ).

Если на странице настройки отчета установлен хотя бы один из признаков [ Отображать в разделе ] ([ *Show in the section list view* ]), [ Отображать на странице записи ] или [ Отображать в аналитике раздела ] ([ *Show in the section analytics view* ]), то обязательно выполняется перевод отображаемого названия отчета.

## Привязать отчет FastReport к пакету

1. [Перейдите в раздел \[ Конфигурация \]](#) ([ *Configuration* ]) и выберите пользовательский [пакет](#), в который будет добавлена схема.
2. На панели инструментов реестра раздела нажмите [ Добавить ] —> [ Данные ] ([ *Add* ] —> [ *Data* ]).



3. Выполните привязку данных. Для этого воспользуйтесь инструкцией, которая приведена в статье [Привязать данные к пакету](#).

**Элементы**, привязку которых необходимо выполнить:

- `FastReportTemplate_ReportName` — шаблон отчета. Подключается по идентификатору отчета (колонка `[Id]` таблицы `[FastReportTemplate]` базы данных).
- `FastReportDataSource_ReportName` — источник данных отчета. Подключается по идентификатору отчета (колонка `[Id]` таблицы `[FastReportDataSource]` базы данных).
- `SysModuleReport_ReportName` — отчет. Подключается по идентификатору отчета (колонка `[Id]` таблицы `[SysModuleReport]` базы данных).

## Создать простой отчет FastReport

Средний

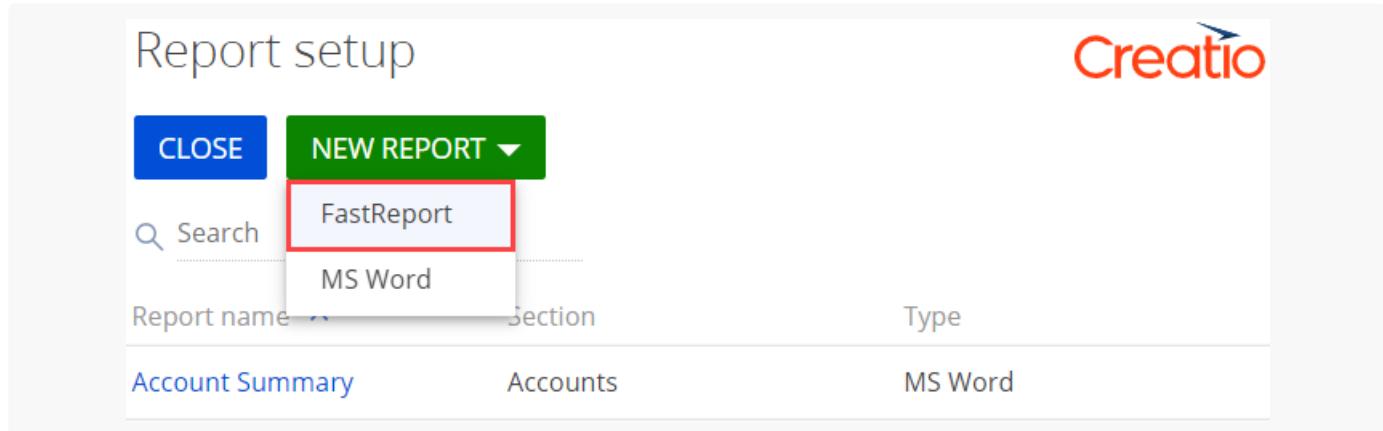
**Пример.** На страницу раздела [ Контакты ] ([ *Contacts* ]) и страницу контакта добавить отчет [ Информация контакта ] ([ *Contact Data* ]).

**Поля**, которые содержит отчет [ Информация контакта ] ([ *Contact Data* ]):

- [ ФИО ] ([ *Full name* ]) — имя контакта.
- [ Дата рождения ] ([ *Birthday* ]) — дата рождения контакта.
- [ Пол ] ([ *Gender* ]) — пол контакта.
- [ Контрагент ] ([ *Account* ]) — контрагент, который связан с контактом.

## 1. Создать отчет

- Перейдите в дизайнер системы по кнопке . В блоке [ Настройка системы ] ([ System setup ]) перейдите по ссылке [ Настройка отчетов ] ([ Report setup ]).
- Выполните действие [ Добавить отчет ] —>[ FastReport ] ([ New report ] —>[ FastReport ]).



- На панели свойств заполните **свойства отчета**:

- [ Название отчета ] ([ Report title ]) — "Информация контакта" ("Contact Data").
- [ Раздел ] ([ Section ]) — выберите "Контакты" ("Contacts").
- Установите признак [ Отображать в разделе ] ([ Show in the section list view ]).
- Установите признак [ Отображать на странице записи ] ([ Show in the section record page ]).
- Установите признак [ Отображать в аналитике раздела ] ([ Show in the section analytics view ]).
- [ Страница фильтрации ] ([ Filter page ]) — выберите "SimpleReportFilterPage".

Report name *	Contact Info
Section *	Contacts
<input checked="" type="checkbox"/> Show in the section list view <input checked="" type="checkbox"/> Show in the section record page <input checked="" type="checkbox"/> Show in the section analytics view	
Filter page	SimpleReportFilterPage

## 2. Указать источники данных

1. В блоке [ Укажите источники данных для отчета ] ([ *Specify data sources for the report* ]) рабочей области страницы настройки отчета укажите источники данных.

**Данные**, которые необходимо указать:

- Перечень объектов.
- Колонки объектов.
- Связи колонок объектов, которые используются для получения данных.
- Локализуемые строки (опционально).

Исходный код источника данных `ContactDataProvider` приведен ниже.

```
ContactDataProvider

{
    "ProviderName": "ContactDataProvider",
    "Schemas": {
        "ContactData": {
            "Full name": {"ValueType": 1},
            "Birthday": {"ValueType": 1},
            "Gender": {"ValueType": 1},
            "Account": {"ValueType": 1}
        },
        "LocalizableStrings": {
            "ReportTitle": {"ValueType": 1},
            "FullNameLabel": {"ValueType": 1},
            "BirthdayLabel": {"ValueType": 1},
            "GenderLabel": {"ValueType": 1},
            "AccountLabel": {"ValueType": 1}
        }
    }
}
```

`ProviderName` — класс провайдера данных.

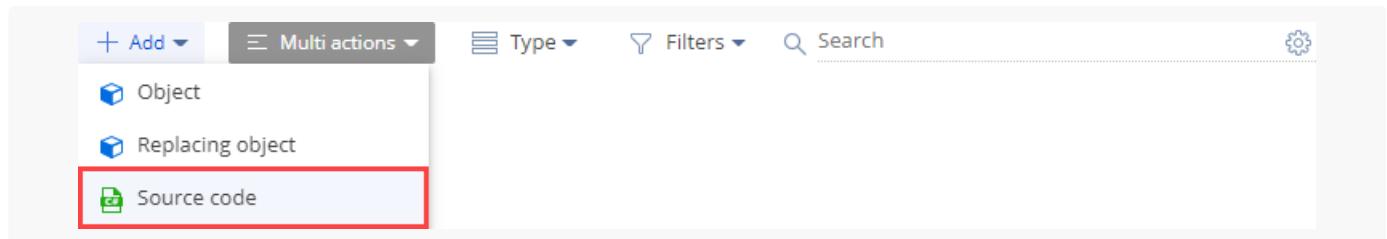
`ValueType` — типы данных соответствующих колонок. `1` — тип данных `TEXT`. Перечисление `Terrasoft.core.enums.DataValueType` описано в [Библиотеке .NET классов](#).

`LocalizableStrings` — локализуемые строки.

2. На панели инструментов страницы настройки отчета нажмите [ *Применить* ] ([ *Apply* ]).

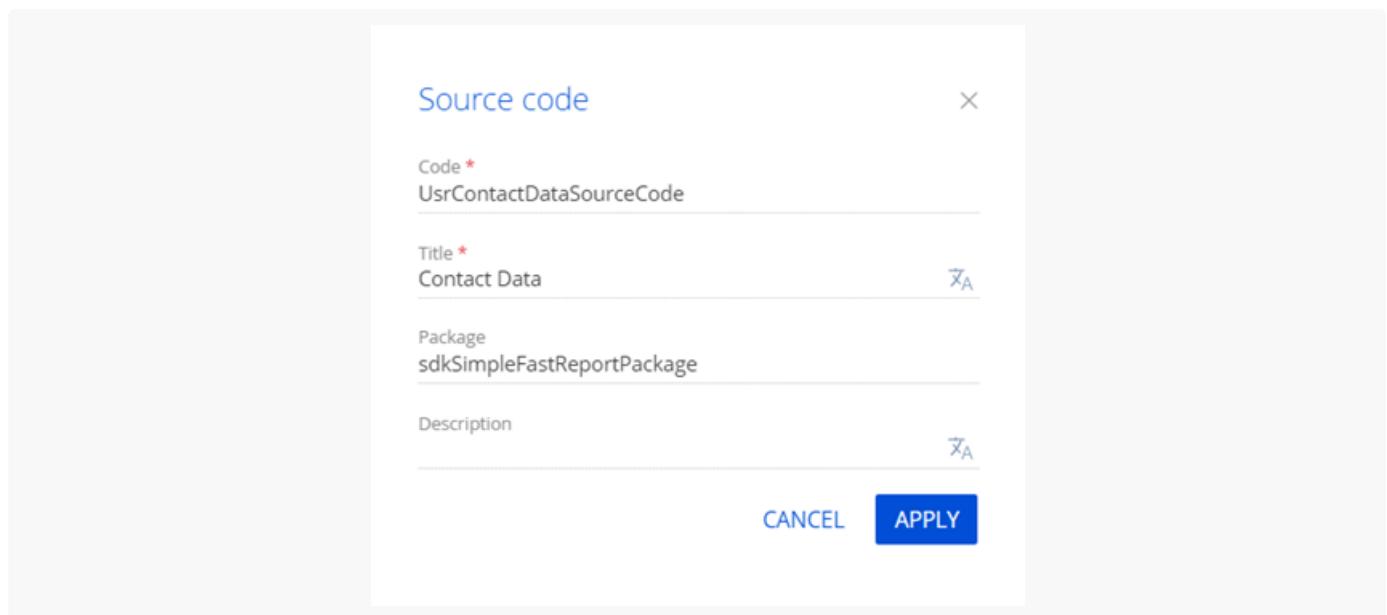
### 3. Реализовать провайдер данных отчета

1. [Перейдите в раздел \[ Конфигурация \]](#) ([ *Configuration* ]) и выберите пользовательский [пакет](#), в который будет добавлена схема.
2. На панели инструментов реестра раздела нажмите [ *Добавить* ] —> [ *Исходный код* ] ([ *Add* ] —> [ *Source code* ]).



3. В дизайнере исходного кода заполните **свойства схемы**:

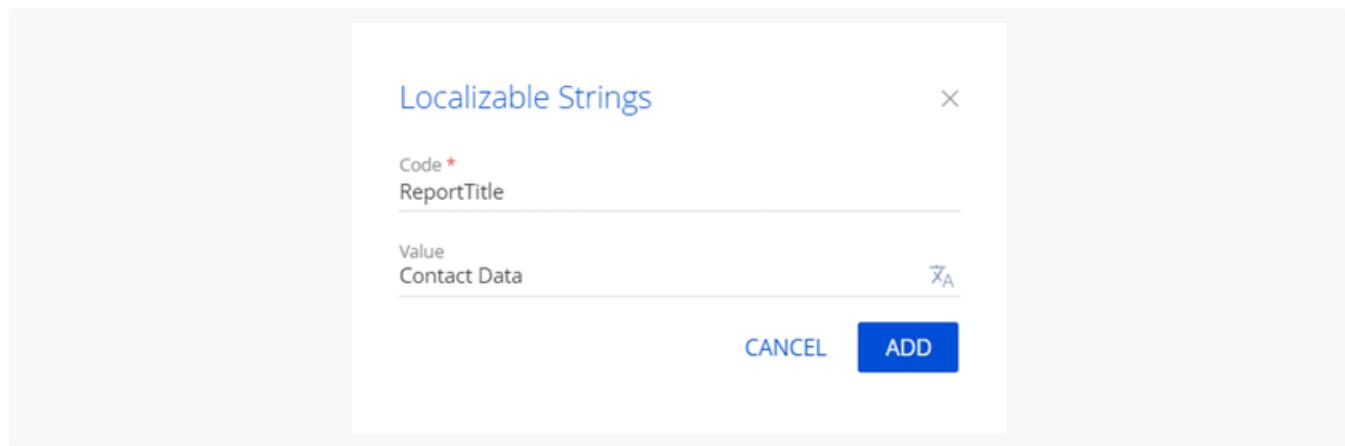
- [ Код ] ([ Code ]) — "UsrContactDataSourceCode".
- [ Заголовок ] ([ Title ]) — "Информация контакта" ("Contact Data").



Для применения изменений свойств нажмите [ Применить ] ([ Apply ]).

4. Добавьте **локализуемую строку**, которая содержит заголовок отчета.

- a. В контекстном меню узла [ Локализуемые строки ] ([ Localizable strings ]) нажмите кнопку **+**.
- b. Заполните **свойства локализуемой строки**:
  - [ Код ] ([ Code ]) — "ReportTitle".
  - [ Значение ] ([ Value ]) — "Информация контакта" ("Contact Data").



- e. Для добавления локализуемой строки нажмите [ Добавить ] ([ Add ]).
5. Аналогично добавьте локализуемые строки, которые соответствуют полям отчета.

Значения локализуемых строк, которые необходимо добавить, приведены в таблице ниже.

Значения локализуемых строк

[ Код ] ([ Code ])	[ Значение ] ([ Value ])
"FullNameLabel"	"ФИО" ("Full name")
"BirthdayLabel"	"Дата рождения" ("Birthday")
"GenderLabel"	"Пол" ("Gender")
"AccountLabel"	"Контрагент" ("Account")

6. В дизайнере исходного кода реализуйте логику работы класса провайдера данных для отчета.

Исходный код схемы типа [ Исходный код ] ([ Source code ]) представлен ниже.

#### UsrContactDataSourceCode

```
namespace Terrasoft.Configuration
{
    using System;
    using System.Collections.Generic;
    using System.Collections.ObjectModel;
    using System.Linq;
    using System.Threading.Tasks;
    using Terrasoft.Common;
    using Terrasoft.Configuration.Reporting.FastReport;
    using Terrasoft.Core;
    using Terrasoft.Core.Entities;
    using Terrasoft.Core.Factories;
```

```

using Terrasoft.Nui.ServiceModel.Extensions;
using EntitySchema = Terrasoft.Core.Entities.EntitySchema;
using EntitySchemaColumn = Terrasoft.Core.Entities.EntitySchemaColumn;

/* Название класса провайдера данных для отчета, логику которого необходимо реализовать.
[DefaultBinding(typeof(IFastReportDataSourceDataProvider), Name = "ContactDataProvider")]
public class ContactDataProvider : IFastReportDataSourceDataProvider
{

    private Guid _entitySchemaUID = new Guid("16BE3651-8FE2-4159-8DD0-A803D4683DD3");
    /* Название схемы с исходным кодом. */
    private readonly string _resourceManagerName = "UsrContactDataSourceCode";
    private readonly string[] _localizableStringNames = new[] {
        "ReportTitle",
        "FullNameLabel",
        "BirthdayLabel",
        "GenderLabel",
        "AccountLabel"
    };

    /* Заполнение колонок в отчете. */
    private IEnumerable<IReadOnlyDictionary<string, object>> GetContactData(
        UserConnection userConnection,
        Guid entitySchemaUID,
        IEntitySchemaQueryFilterItem filter) {
        /* Получить схему объекта. */
        var entitySchema = userConnection.EntitySchemaManager.GetInstanceById(entitySchemaUID);
        /* Создание объекта класса EntitySchemaQuery. */
        EntitySchemaQuery query = new EntitySchemaQuery(entitySchema);
        /* Добавление колонок в запрос. */
        query.AddColumn("Name");
        query.AddColumn("BirthDate");
        var gender = query.AddColumn("Gender.Name");
        var account = query.AddColumn("Account.Name");
        /* Добавление созданного фильтра. */
        query.Filters.Add(filter);
        /* Получить коллекцию контактов. */
        var contacts = query.GetEntityCollection(userConnection);
        var contactsCollection = new Collection<Dictionary<string, object>>();
        /* Заполнение колонок в отчете. */
        foreach (var entity in contacts)
        {
            contactsCollection.Add(new Dictionary<string, object> {
                ["Full name"] = entity.GetTypedColumnValue<string>("Name"),
                ["Birthday"] = entity.GetTypedColumnValue<string>("BirthDate"),
                ["Gender"] = entity.GetTypedColumnValue<string>(gender.Name),
                ["Account"] = entity.GetTypedColumnValue<string>(account.Name)
            });
        }
    }
}

```

```
        return contactsCollection;
    }

/* Локализация заголовков отчета. */
private IEnumerable<IReadOnlyDictionary<string, object>> GetLocalizableStrings(UserCo
    var localizableStrings = _localizableStringNames.ToDictionary(
        x => x,
        x => (object)(new LocalizableString(userConnection.ResourceStorage, _resource
    return new[] { localizableStrings });
}

/* Добавление фильтров интерфейса. */
private IEntitySchemaQueryFilterItem ExtractFilterFromParameters(UserConnection userC
    var managerItem = userConnection.EntitySchemaManager.GetItemById(entitySchemaUID
    return parameters.ExtractEsqFilterFromReportParameters(userConnection, managerIt
}

/* Получить данные. */
public Task<ReportDataDictionary> GetData(UserConnection userConnection, IReadOnlyDic
    var filter = ExtractFilterFromParameters(userConnection, _entitySchemaUID, paramete
    var result = new ReportDataDictionary {
        /* Заполнить колонки в отчете. */
        ["ContactData"] = GetContactData(userConnection, _entitySchemaUID, filter
    };
    return Task.FromResult(result);
}
}
```

7. На панели инструментов дизайнера исходного кода нажмите [ Опубликовать ] ([ *Publish* ]) для выполнения изменений на уровне базы данных.

#### 4. Настроить шаблон отчета

1. Скачайте файл отчета FastReport. Для этого в блоке [ Скачайте файл с источниками данных и сформируйте шаблон отчета ] ([ Download file with data sources to design a report in the FastReport Designer ]) рабочей области страницы настройки отчета нажмите кнопку [ Скачать файл ] ([ Download file ]).

[Download file with data sources to design a report in the FastReport Designer](#)

Open the downloaded file in the FastReport Designer and set up the report layout. The downloaded file will contain the data source structure that you can use in the FastReport Designer.

[DOWNLOAD FILE](#)

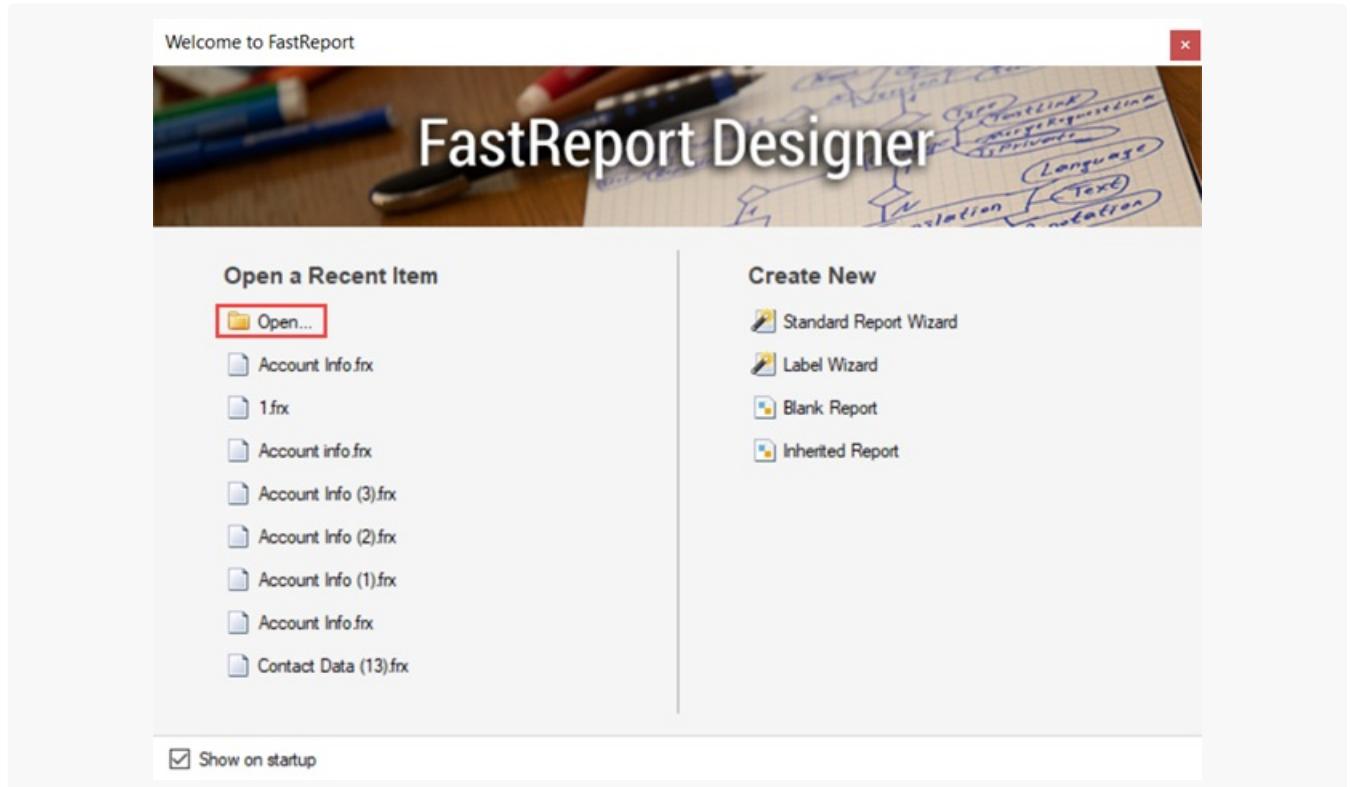
В результате скачан файл ContactData.frx .

- ## 2. Откройте **шаблон отчета**.

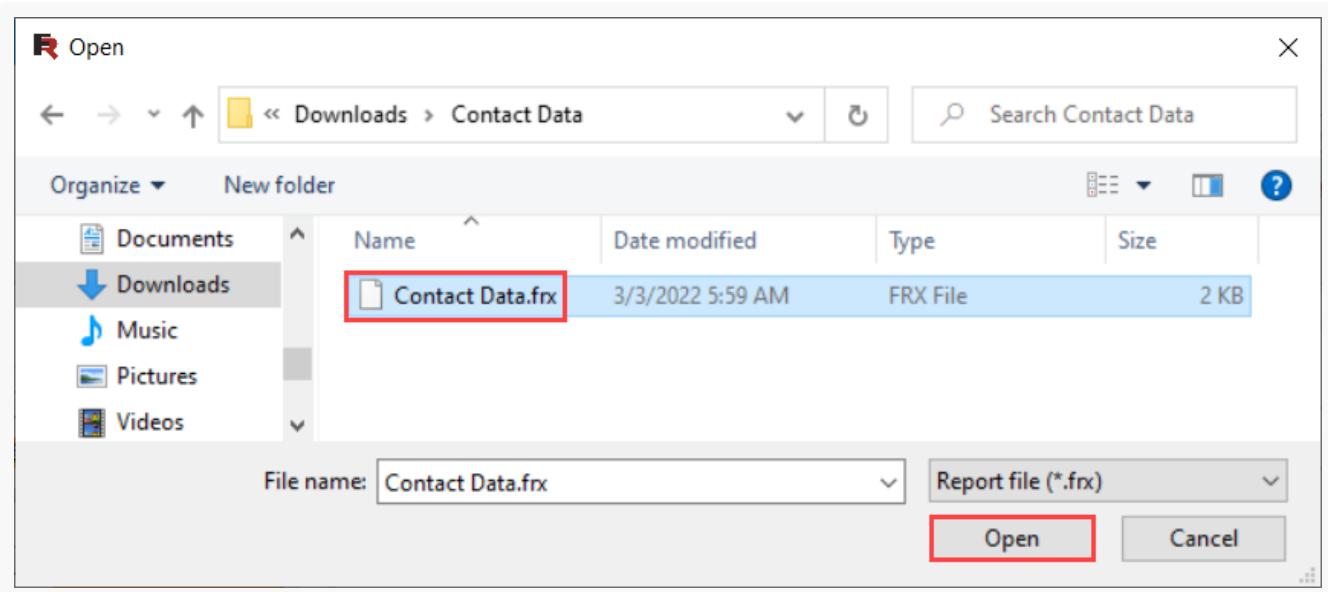
- a. Откройте дизайнер отчетов FastReport. Для этого разархивируйте [\\*.zip-архив](#) и запустите файл `Terrasoft.Reporting.FastReport.Designer.exe`.

Name	Type	Compressed size
FastReport.Bars.dll	Application extension	1,731 KB
FastReport.dll	Application extension	3,101 KB
FastReport.Editor.dll	Application extension	338 KB
Terrasoft.Reporting.FastReport.Designer.exe	Application	3 KB
Terrasoft.Reporting.FastReport.Designer.exe.config	CONFIG File	1 KB

- b. Выберите **отчет FastReport**, который планируется настроить. Для этого в блоке [ *Open a Recent Item* ] окна [ *Welcome to FastReport* ] нажмите на кнопку [ *Open...* ].



- c. Перейдите в каталог со скачанным отчетом (обычно это каталог `Downloads`), выберите предварительно скачанный файл `ContactData.frx` и нажмите на кнопку [ *Open* ].

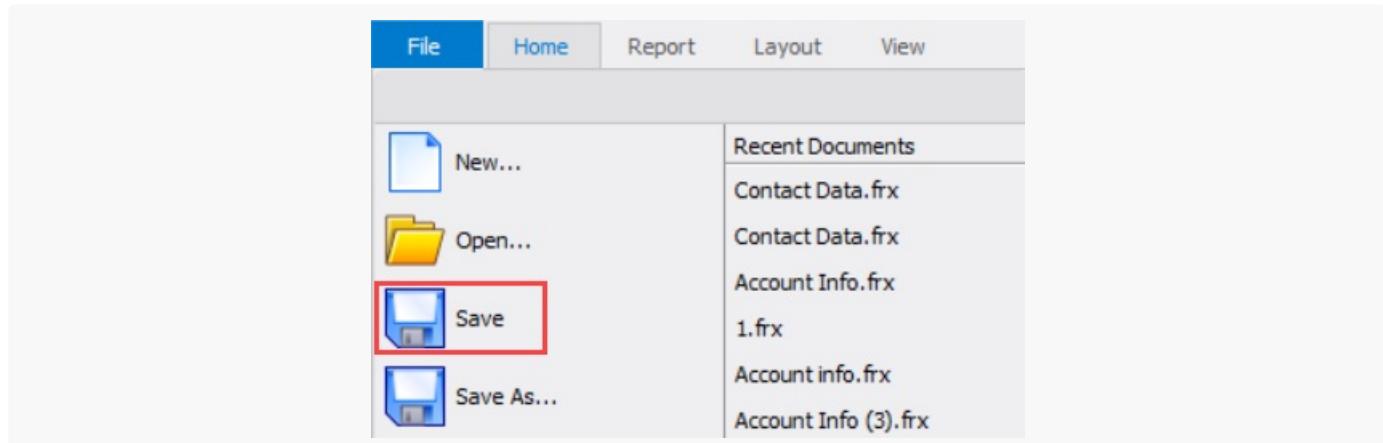


3. Настройте шаблон отчета.

Настроенный шаблон отчета [ Информация контакта ] ([ Contact Data ]) приведен на рисунке ниже.

<u>[LocalizableStrings.ReportTitle]</u>			
<u>[LocalizableString.s.FullNameLabel]</u>	<u>[LocalizableStrings.BirthDayLabel]</u>	<u>[LocalizableStrings.GenderLabel]</u>	<u>[LocalizableStrings.AccountLabel]</u>
<u>[ContactData.FullName]</u>	<u>[ContactData.Birthday]</u>	<u>[ContactData.Gender]</u>	<u>[ContactData.Account]</u>

4. Сохраните шаблон отчета. Для этого в меню [ File ] панели инструментов нажмите на кнопку [ Save ].



5. Загрузите настроенный шаблон отчета в Creatio.

- a. В блоке [ Загрузите настроенный шаблон отчета в Creatio ] ([ Import a file with the report template ]) рабочей области страницы настройки отчета нажмите на кнопку [ Загрузить шаблон ] ([ Upload

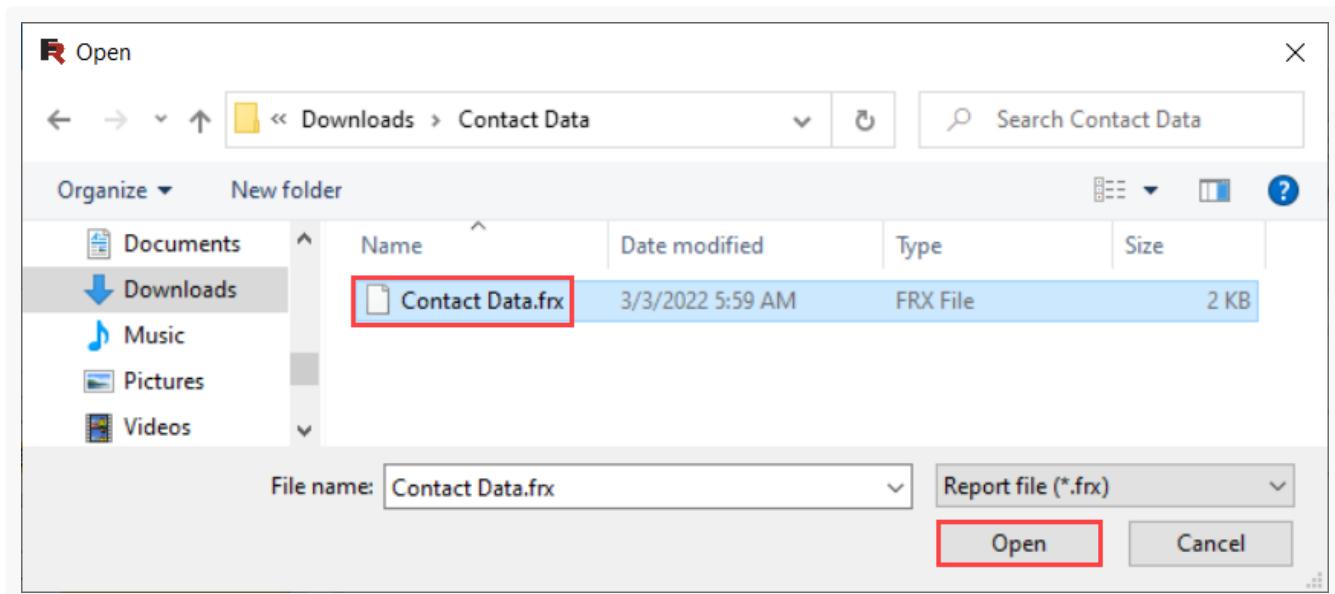
*template ]).*

**Import a file with the report template**

After you import the .frx file, the report will appear in the "Print" menu of the section or on the record page as per the report settings.

**UPLOAD FILE**

- b. Перейдите в каталог с настроенным шаблоном отчета (обычно это каталог `Downloads`), выберите файл с отчетом и нажмите на кнопку [ *Open* ].



В результате настроенный шаблон отчета [ *Информация контакта* ] ([ *Contact Data* ]) загружен в Creatio. Для переноса изменений между рабочими средами, привяжите отчет к пакету. Для этого воспользуйтесь инструкцией, которая приведена в статье [Отчеты FastReport](#).

## Результат выполнения примера

Чтобы **посмотреть результат выполнения примера**, обновите страницу раздела [ *Контакты* ] ([ *Contacts* ]).

В результате выполнения примера на страницу раздела [ *Контакты* ] ([ *Contacts* ]) добавлен отчет [ *Информация контакта* ] ([ *Contact Data* ]). Отчет доступен в выпадающем меню кнопки [ *Печать* ] ([ *Print* ]) панели инструментов раздела.

Contacts

NEW CONTACT ACTIONS ▾

Filters/folders Tag

Andrew Wayne

Job title: CEO  
Business phone: +44 141 429 1595

Account: Apex Solutions Email: a.wayne@apex.co.uk  
Mobile phone: +44 141 258 9878

OPEN COPY DELETE PRINT

Отчет [ Информация контакта ] ([ Contact Data ]) имеет вид, который представлен на рисунке ниже.

<u>Contact Data</u>			
Full name	Birthday	Gender	Account
Andrew Wayne	12/8/1971 12:00:00 AM	Male	Apex Solutions

Если в реестре раздела [ Контакты ] ([ Contacts ]) не выбрана запись, то отчет [ Информация контакта ] ([ Contact Data ]) не активен в выпадающем меню кнопки [ Печать ] ([ Print ]) панели инструментов.

Contacts

NEW CONTACT ACTIONS ▾

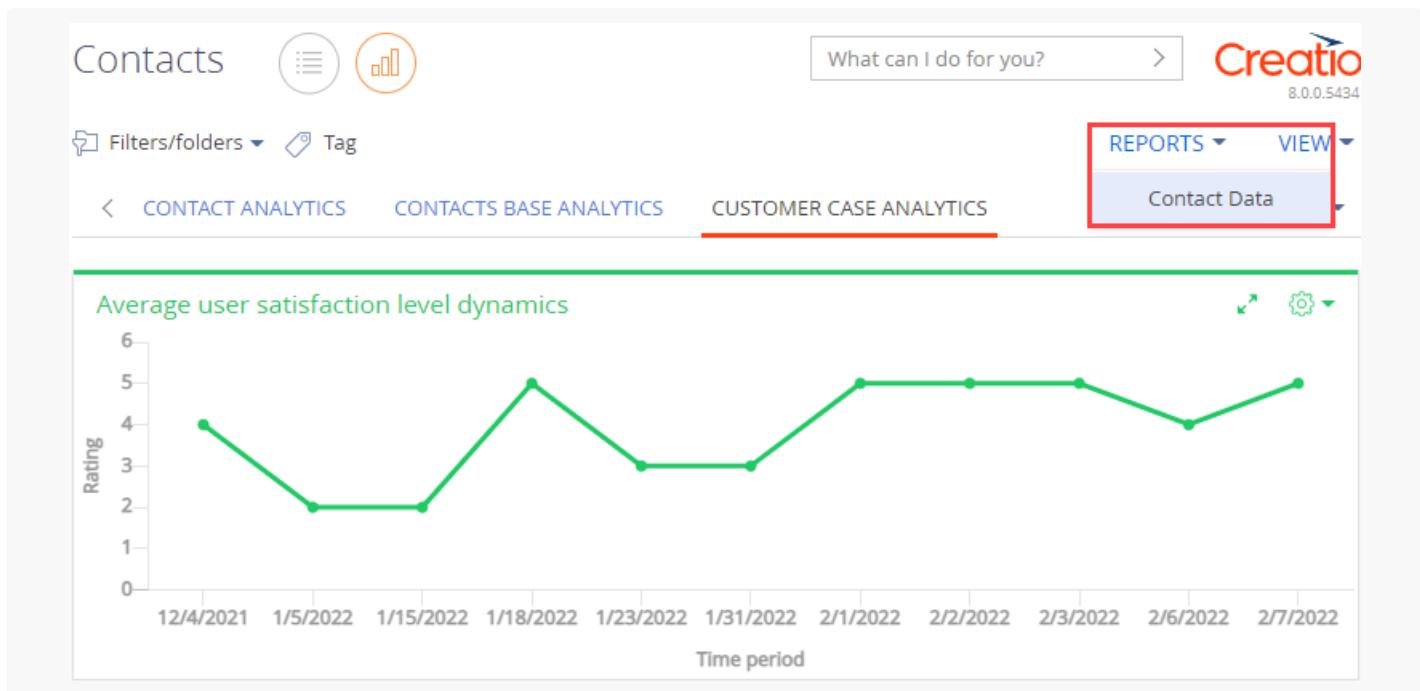
Filters/folders Tag

Andrew Wayne

Job title: CEO  
Business phone: +44 141 429 1595

Account: Apex Solutions Email: a.wayne@apex.co.uk  
Mobile phone: +44 141 258 9878

В результате выполнения примера в аналитику раздела [ Контакты ] ([ Contacts ]) также добавлен отчет [ Информация контакта ] ([ Contact Data ]). Отчет доступен в выпадающем меню кнопки [ Отчеты ] ([ Reports ]) панели инструментов раздела.



В результате выполнения примера на страницу контакта также добавлен отчет [ Информация контакта ] ([ Contact Data ]). Отчет доступен в выпадающем меню кнопки [ Печать ] ([ Print ]) панели инструментов страницы контакта.

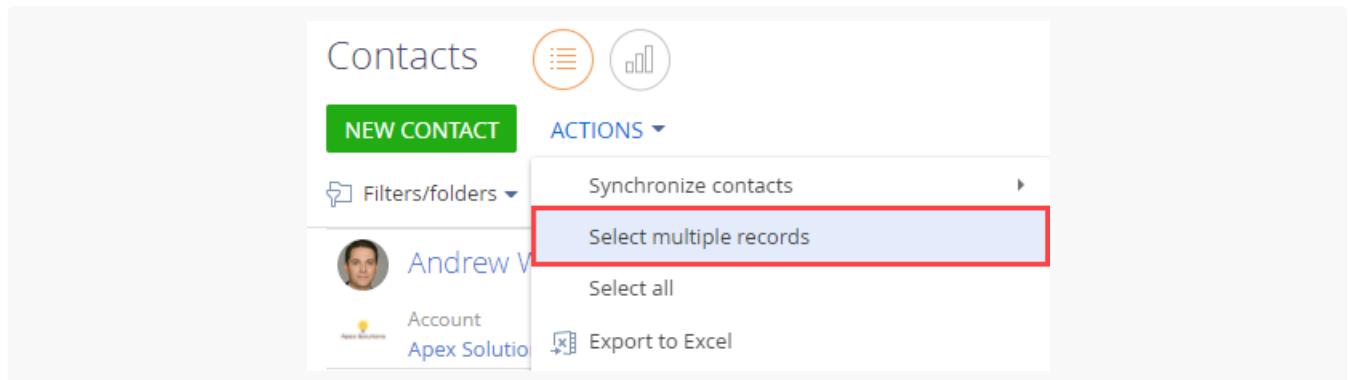
Чтобы **посмотреть результат выполнения примера по нескольким контактам**:

1. Обновите страницу раздела [ Контакты ] ([ Contacts ]).
2. Выберите несколько записей раздела.

**Способы** выбора нескольких записей раздела:

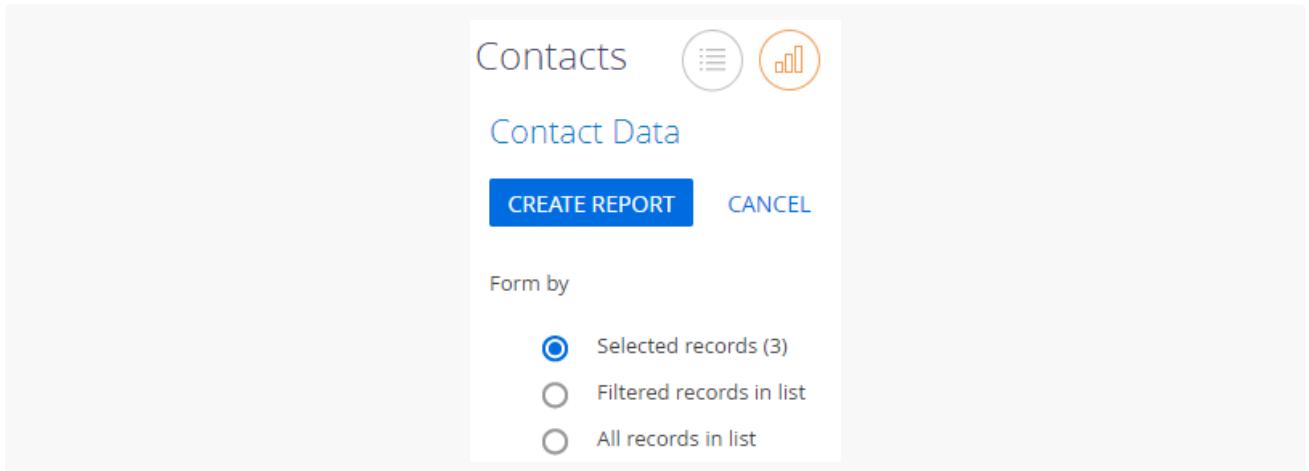
- **В реестре раздела** (на странице настройки отчета установлен признак [ Отображать в разделе ]

([ Show in the section list view ])). Для этого на панели инструментов раздела выполните действие [ Действия ] —> [ Выбрать несколько записей ] ([ Actions ] —> [ Select multiple records ]).



- В **аналитике раздела** (на странице настройки отчета установлен признак [ Отображать в аналитике раздела ] ([ Show in the section analytics view ])).
- На панели инструментов раздела выполните действие [ Действия ] —> [ Выбрать несколько записей ] ([ Actions ] —> [ Select multiple records ]).
- Нажмите на кнопку .
- В выпадающем меню кнопки [ Отчеты ] ([ Reports ]) панели инструментов раздела выберите отчет [ Информация контакта ] ([ Contact Data ]).

В результате откроется страница фильтрации.



- Отфильтруйте записи (опционально). Для этого на странице фильтрации выберите опцию [ По отфильтрованным в разделе записям ] ([ Filtered records in list ]).
- Для скачивания отчета нажмите на кнопку [ Создать отчет ] ([ Create report ]).

Отчет [ Информация контакта ] ([ Contact Data ]) по нескольким записям имеет вид, который представлен на рисунке ниже.

<u>Contact Data</u>			
Full name	Birthday	Gender	Account
Alice Phillips	10/12/1982 12:00:00 AM	Female	Axiom
Andrew Wayne	12/8/1971 12:00:00 AM	Male	Apex Solutions
Alexander Wilson	5/5/1984 12:00:00 AM	Male	Alpha Business

## Создать отчет FastReport с изображением

 Средний

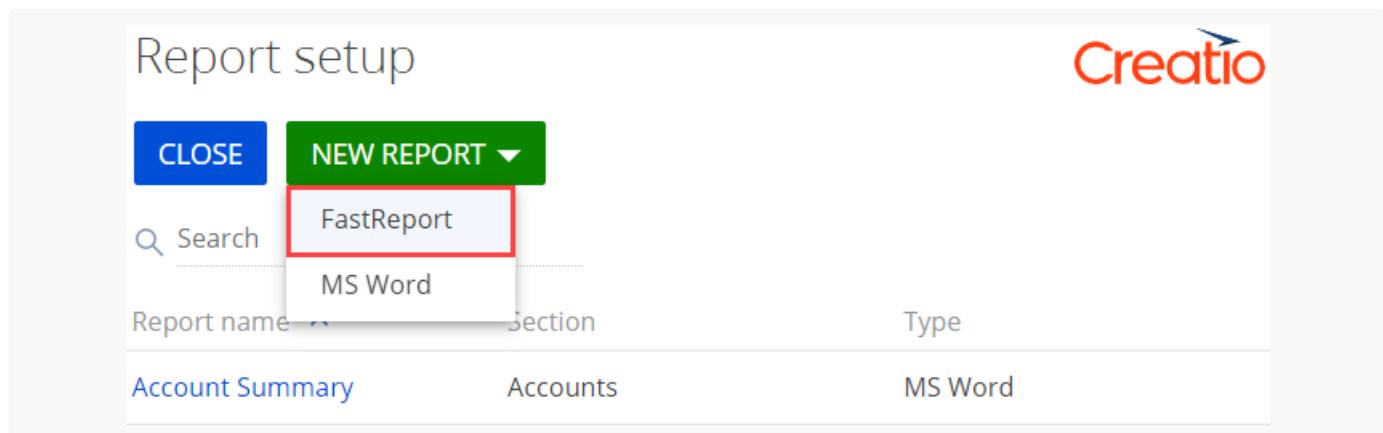
**Пример.** На страницу раздела [ Контрагенты ] ([ Accounts ]) и страницу контрагента добавить отчет [ Информация контрагента ] ([ Account Info ]).

**Поля**, которые содержит отчет [ Информация контрагента ] ([ Account Info ]):

- [ Название контрагента ] ([ Name ]) — название контрагента.
- [ Логотип ] ([ Logo ]) — логотип контрагента.

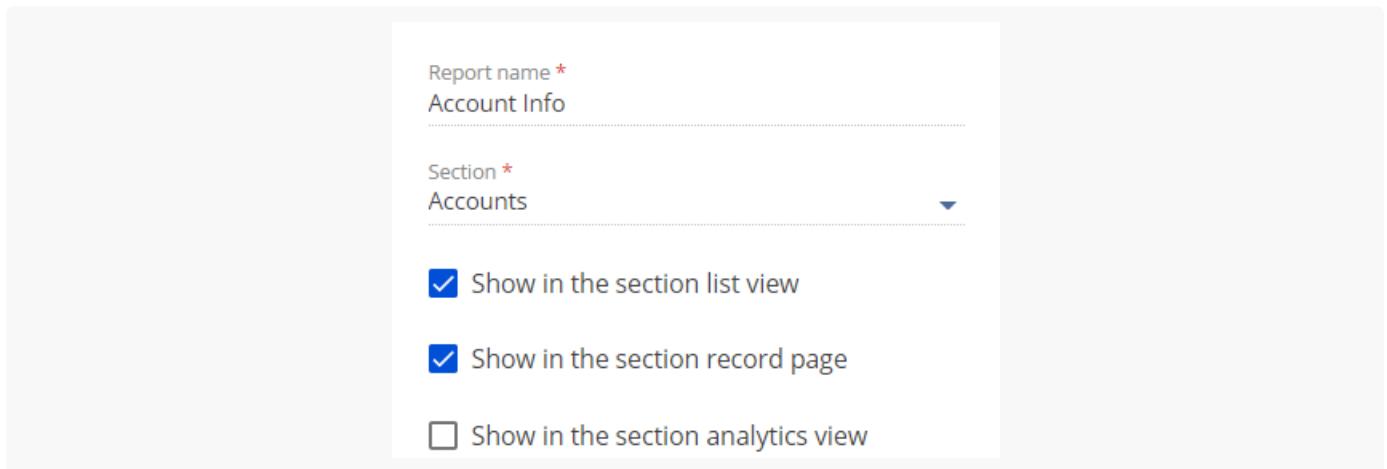
### 1. Создать отчет

1. Перейдите в дизайнер системы по кнопке  . В блоке [ Настройка системы ] ([ System setup ]) перейдите по ссылке [ Настройка отчетов ] ([ Report setup ]).
2. Выполните действие [ Добавить отчет ] —>[ FastReport ] ([ New report ]) —>[ FastReport ]).



### 3. На панели свойств заполните **свойства отчета**:

- [ Название отчета ] ([ Report title ]) — "Информация контрагента" ("Account Info").
- [ Раздел ] ([ Section ]) — выберите "Контрагенты" ("Accounts").
- Установите признак [ Отображать в разделе ] ([ Show in the section list view ]).
- Установите признак [ Отображать на странице записи ] ([ Show in the section record page ]).



## 2. Указать источники данных

### 1. В блоке [ Укажите источники данных для отчета ] ([ Specify data sources for the report ]) рабочей области страницы настройки отчета укажите источники данных.

**Данные**, которые необходимо указать:

- Перечень объектов.
- Колонки объектов.
- Связи колонок объектов, которые используются для получения данных.
- Локализуемые строки (опционально).

Исходный код источника данных `AccountInfoProvider` приведен ниже.

```
AccountInfoProvider

{
    "ProviderName": "AccountInfoProvider",
    "Schemas": {
        "Data": {
            "Name": {"DataProviderType": 1},
            "Logo": {"DataProviderType": 14}
        },
        "LocalizableStrings": {
            "ReportTitle": {"DataProviderType": 1},
            "NameLabel": {"DataProviderType": 1},
        }
    }
}
```

```

        "LogoLabel": {"DataValueType": 1}
    }
}
}

```

`ProviderName` — класс провайдера данных.

`DataValueType` — типы данных соответствующих колонок. Перечисление

`Terrasoft.core.enums.DataValueType` описано в [Библиотеке .NET классов](#).

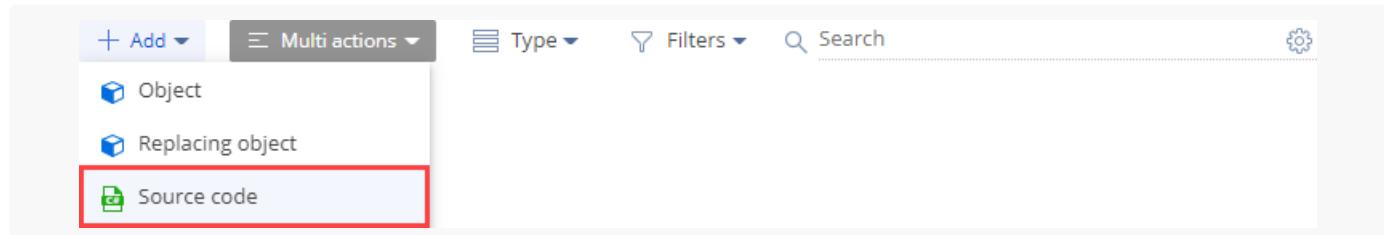
- `1` — тип данных `TEXT`.
- `14` — тип данных `IMAGE`.

`LocalizableStrings` — локализуемые строки.

2. На панели инструментов страницы настройки отчета нажмите [ Применить ] ([ *Apply* ]).

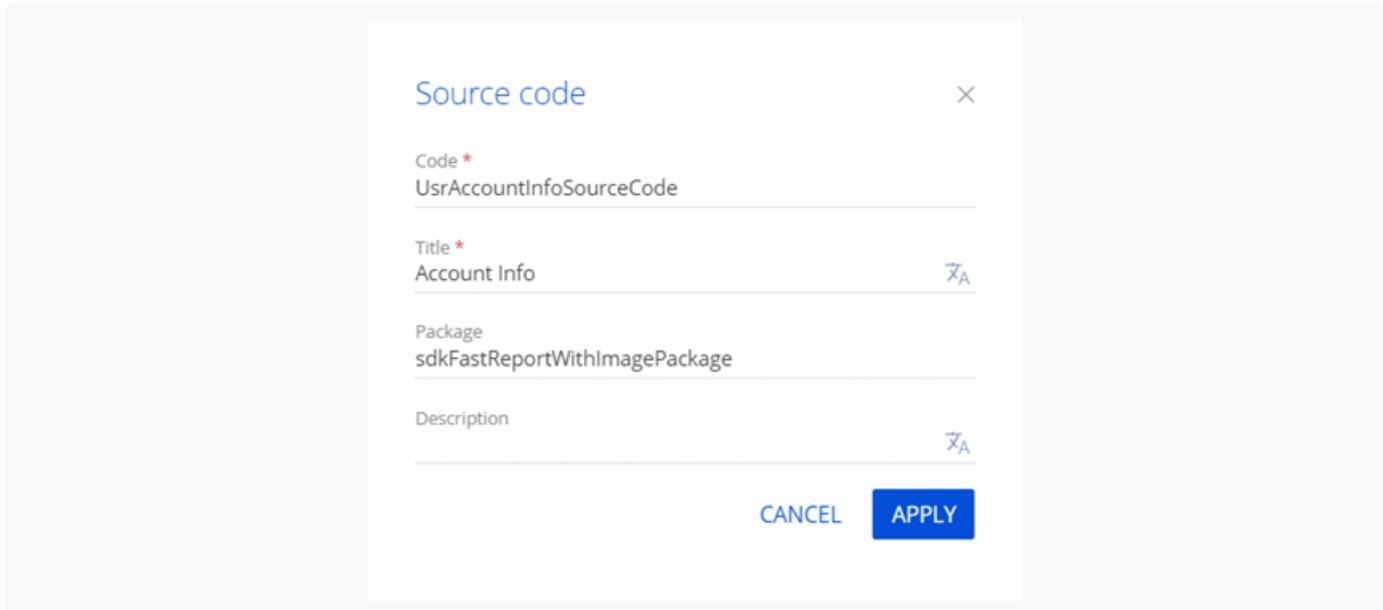
### 3. Реализовать провайдер данных отчета

1. [Перейдите в раздел \[ Конфигурация \]](#) ([ *Configuration* ]) и выберите пользовательский [пакет](#), в который будет добавлена схема.
2. На панели инструментов реестра раздела нажмите [ Добавить ] —> [ Исходный код ] ([ *Add* ] —> [ *Source code* ]).



3. В дизайнере исходного кода заполните **свойства схемы**:

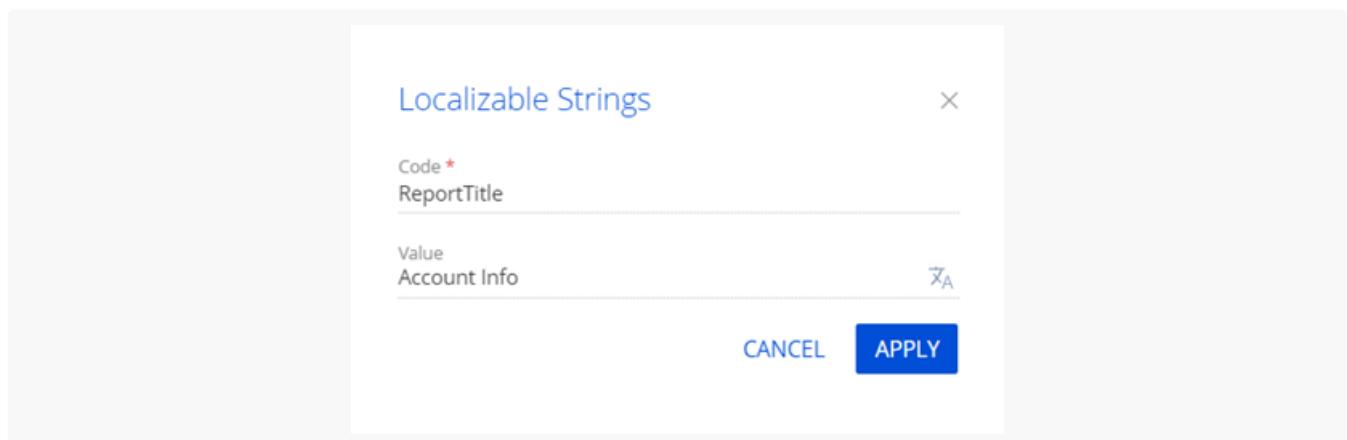
- [ Код ] ([ *Code* ]) — "UsrAccountInfoSourceCode".
- [ Заголовок ] ([ *Title* ]) — "Информация контрагента" ("Account Info").



Для применения изменений свойств нажмите [ Применить ] ([ *Apply* ]).

4. Добавьте **локализуемую строку**, которая содержит заголовок отчета.

- В контекстном меню узла [ *Локализуемые строки* ] ([ *Localizable strings* ]) нажмите кнопку +.
- Заполните **свойства локализуемой строки**:
  - [ *Код* ] ([ *Code* ]) — "ReportTitle".
  - [ *Значение* ] ([ *Value* ]) — "Информация контрагента" ("Account Info").



- Для добавления локализуемой строки нажмите [ *Добавить* ] ([ *Add* ]).

5. Аналогично добавьте локализуемые строки, которые соответствуют полям отчета.

Значения локализуемых строк, которые необходимо добавить, приведены в таблице ниже.

Значения локализуемых строк

[ Код ] ([ Code ])	[ Значение ] ([ Value ])
"NameLabel"	"Название контрагента" ("Name")
"LogoLabel"	"Логотип" ("Logo")

6. В дизайнере исходного кода реализуйте логику работы класса провайдера данных для отчета.

Исходный код схемы типа [ Исходный код ] ([ Source code ]) представлен ниже.

#### UsrAccountInfoSourceCode

```
namespace Terrasoft.Configuration
{
    using System;
    using System.Collections.Generic;
    using System.Linq;
    using System.Threading.Tasks;
    using Terrasoft.Common;
    using Terrasoft.Configuration.Reporting.FastReport;
    using Terrasoft.Core;
    using Terrasoft.Core.Entities;
    using Terrasoft.Core.Factories;

    /* Название класса провайдера данных для отчета, логику которого необходимо реализовать.
    [DefaultBinding(typeof(IFastReportDataSourceDataProvider), Name = "AccountInfoProvider")]
    public class AccountInfoProvider : IFastReportDataSourceDataProvider
    {
        /* Название схемы с исходным кодом. */
        private readonly string _resourceManagerName = "UsrAccountInfoSourceCode";
        private readonly string[] _localizableStringNames = new[] {
            "ReportTitle",
            "NameLabel",
            "LogoLabel"
        };

        /* Заполнение колонок в отчете. */
        private IEnumerable<IReadOnlyDictionary<string, object>> GetData(UserConnection userC
            var esq = new EntitySchemaQuery(userConnection.EntitySchemaManager, "Account");
            /* Добавление колонок в запрос. */
            var nameColumn = esq.AddColumn("Name").OrderByDesc();
            var logoColumn = esq.AddColumn("AccountLogo.Data");
            /* Добавление созданного фильтра. */
            esq.Filters.Add(filter);
            return esq.GetEntityCollection(userConnection)
    }
}
```

```

        .Select(x => new Dictionary<string, object> {
            ["Name"] = x.GetTypedColumnValue<string>(nameColumn.Name),
            ["Logo"] = x.GetStreamValue(logoColumn.Name)?.ToArray()
        });
    }

    /* Локализация заголовков отчета. */
    private IEnumerable<IReadOnlyDictionary<string, object>> GetLocalizableStrings(UserConnection userConnection)
    {
        var localizableStrings = _localizableStringNames.ToDictionary(
            x => x,
            x => (object)(new LocalizableString(userConnection.ResourceStorage, _resourceStorage[x].Text)));
        return new[] { localizableStrings };
    }

    /* Добавление фильтров интерфейса. */
    private IEntitySchemaQueryFilterItem ExtractFilterFromParameters(UserConnection userConnection, EntitySchemaQueryFilterItem parameters)
    {
        return parameters.ExtractEsqFilterFromReportParameters(userConnection, "Account");
    }

    /* Получить данные. */
    public Task<ReportDataDictionary> GetData(UserConnection userConnection, IReadOnlyDictionary<string, object> parameters)
    {
        var filter = ExtractFilterFromParameters(userConnection, parameters);
        var result = new ReportDataDictionary {
            /* Заполнить колонки в отчете. */
            ["Data"] = GetData(userConnection, filter),
            ["LocalizableStrings"] = GetLocalizableStrings(userConnection)
        };
        return Task.FromResult(result);
    }
}
}

```

- На панели инструментов дизайнера исходного кода нажмите [ *Опубликовать* ] ([ *Publish* ]) для выполнения изменений на уровне базы данных.

## 4. Настроить шаблон отчета

- Скачайте файл отчета FastReport. Для этого в блоке [ *Скачайте файл с источниками данных и сформируйте шаблон отчета* ] ([ *Download file with data sources to design a report in the FastReport Designer* ]) рабочей области страницы настройки отчета нажмите кнопку [ *Скачать файл* ] ([ *Download file* ]).

[Download file with data sources to design a report in the FastReport Designer](#)

Open the downloaded file in the FastReport Designer and set up the report layout. The downloaded file will contain the data source structure that you can use in the FastReport Designer.

[DOWNLOAD FILE](#)

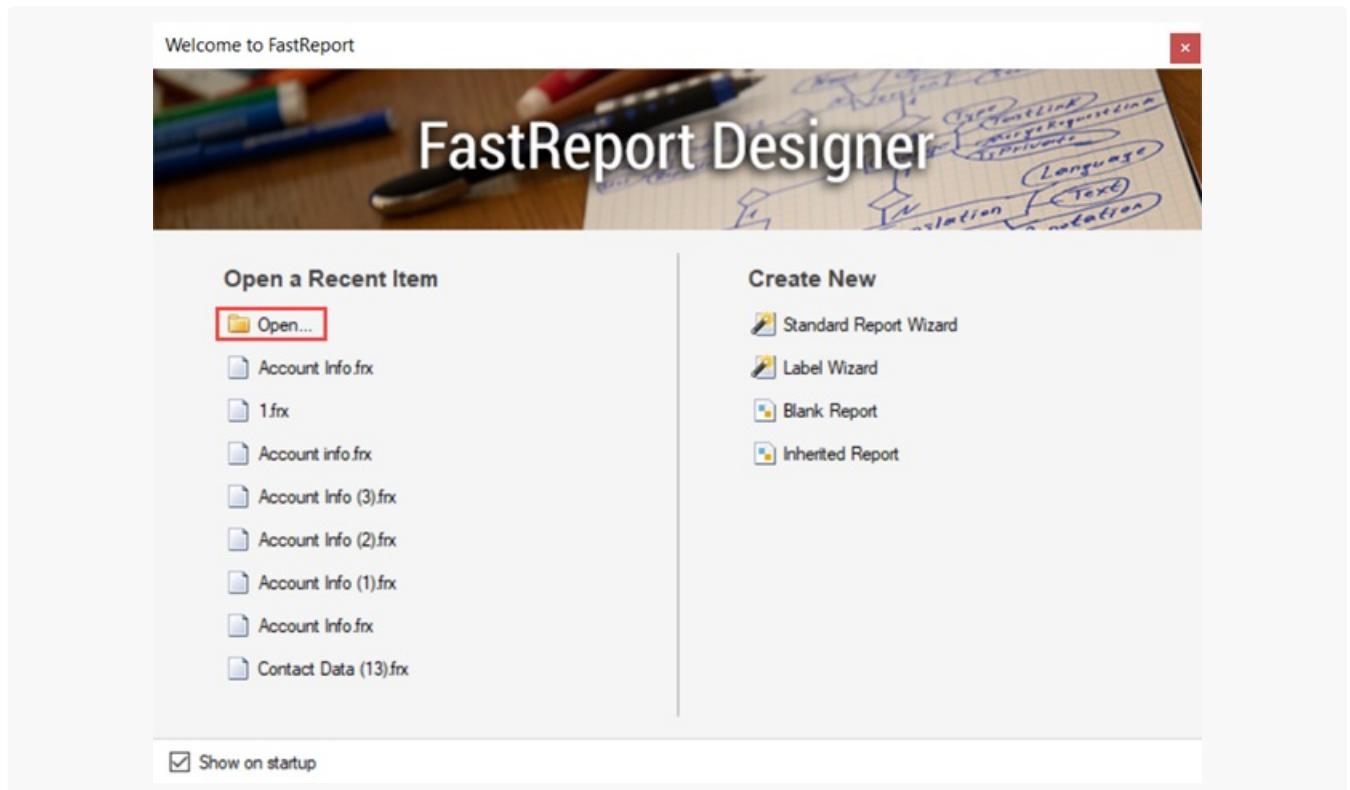
В результате скачан файл `AccountInfo.frx`.

## 2. Откройте шаблон отчета.

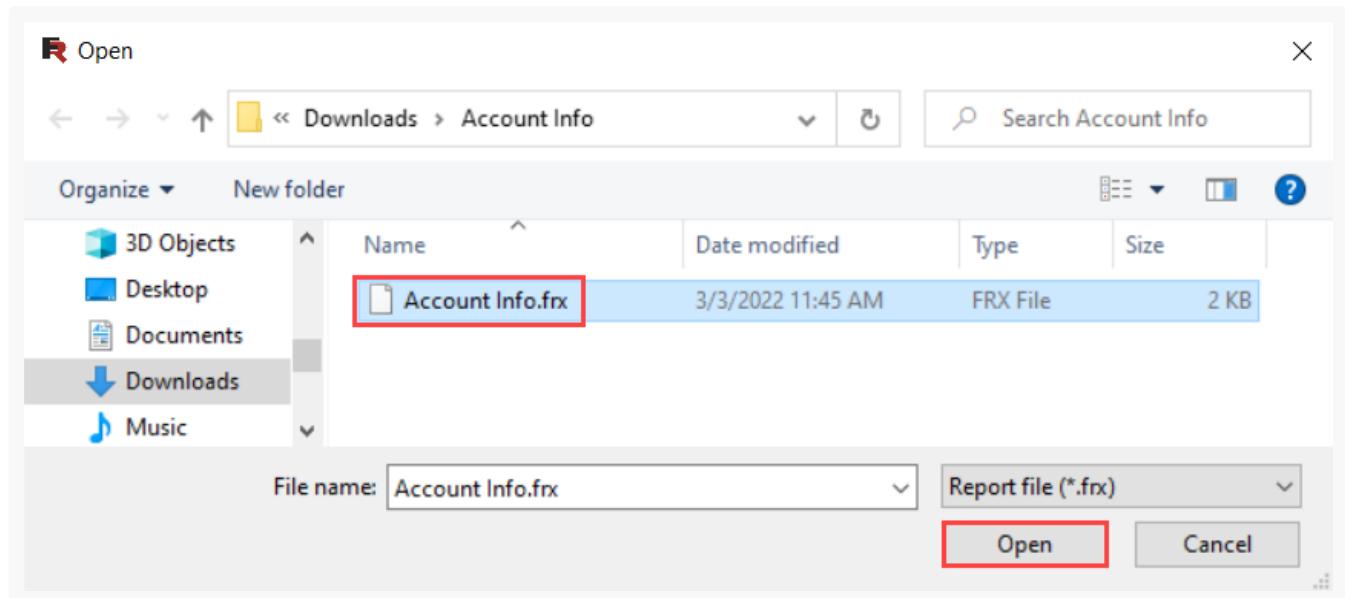
- a. Откройте дизайнер отчетов FastReport. Для этого разархивируйте [\\*.zip-архив](#) и запустите файл `Terrasoft.Reporting.FastReport.Designer.exe`.

Name	Type	Compressed size
<code>FastReport.Bars.dll</code>	Application extension	1,731 KB
<code>FastReport.dll</code>	Application extension	3,101 KB
<code>FastReport.Editor.dll</code>	Application extension	338 KB
<code>Terrasoft.Reporting.FastReport.Designer.exe</code>	Application	3 KB
<code>Terrasoft.Reporting.FastReport.Designer.exe.config</code>	CONFIG File	1 KB

- b. Выберите **отчет FastReport**, который планируется настроить. Для этого в блоке [ *Open a Recent Item* ] окна [ *Welcome to FastReport* ] нажмите на кнопку [ *Open...* ].



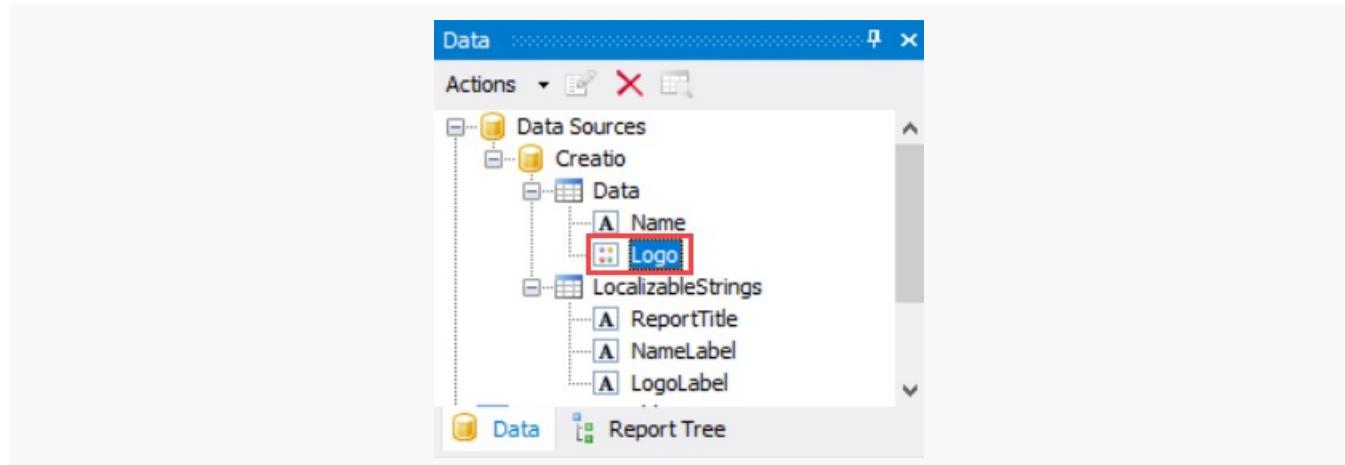
- c. Перейдите в каталог со скачанным отчетом (обычно это каталог `Downloads`), выберите предварительно скачанный файл `AccountInfo.frx` и нажмите на кнопку [ *Open* ].



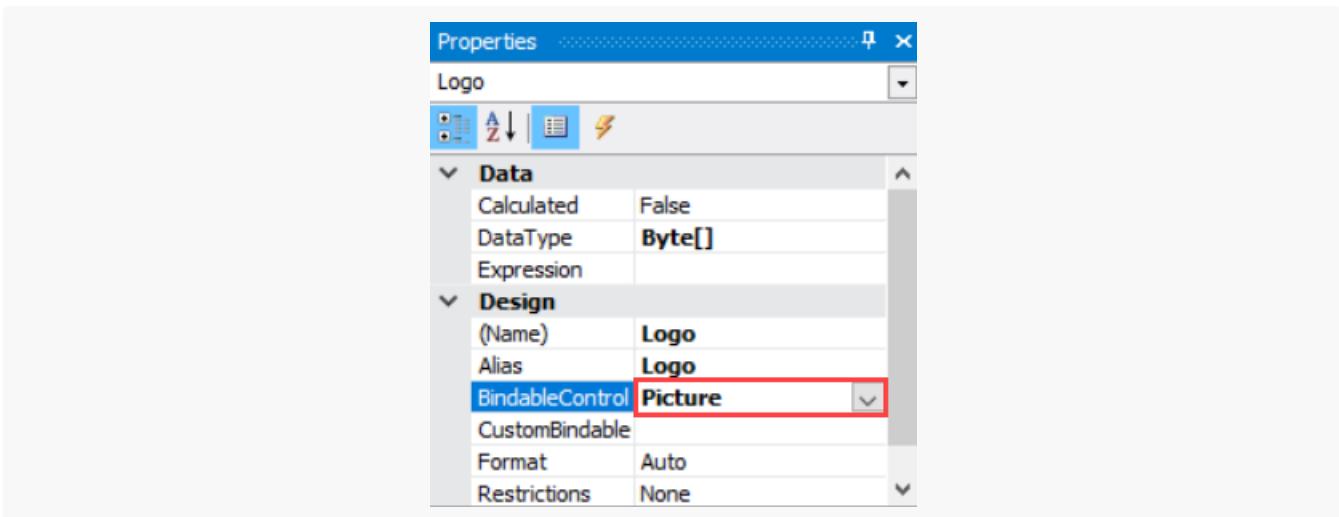
### 3. Настройте шаблон отчета.

Чтобы **изображение отобразилось** в сформированном отчете:

- На вкладке [ Data ] панели свойств [ Data ] выполните [ Data Sources ] —> [ Creatio ] —> [ Data ] и кликните по элементу [ Logo ].

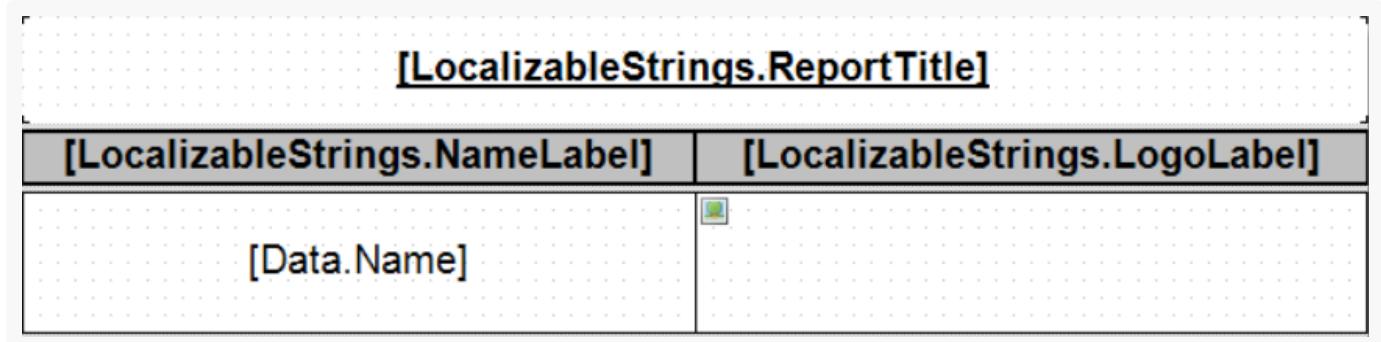


- В группе [ Design ] панели свойств [ Properties ] элемента [ Logo ] в выпадающем списке свойства [ BindableControl ] выберите значение "Picture".

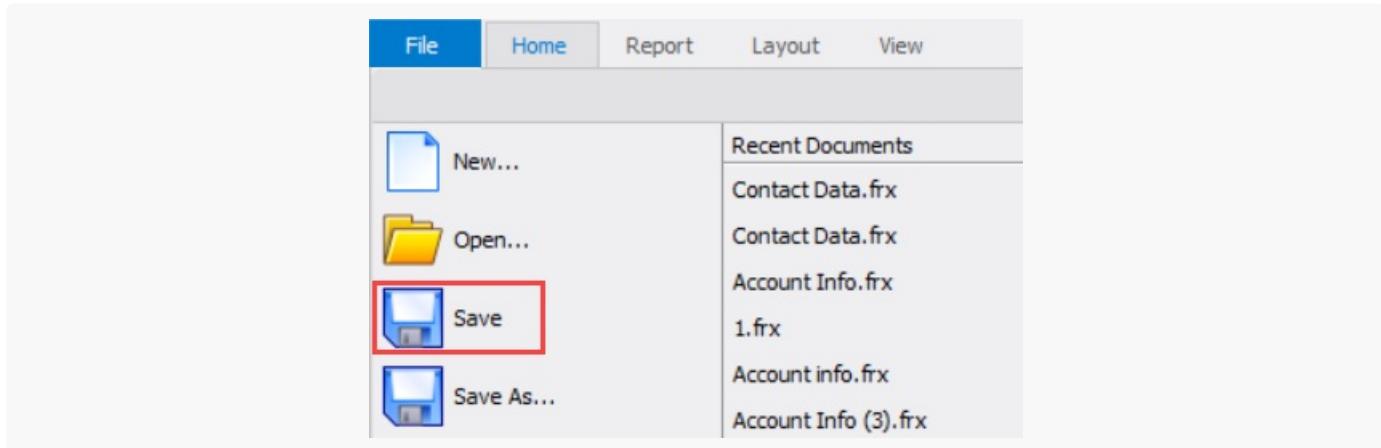


с. Добавьте элемент [ Logo ] в рабочую область дизайнера отчетов.

Настроенный шаблон отчета [ Информация контрагента ] ([ Account Info ]) приведен на рисунке ниже.



4. Сохраните шаблон отчета. Для этого в меню [ File ] панели инструментов нажмите на кнопку [ Save ].



5. Загрузите настроенный шаблон отчета в Creatio.

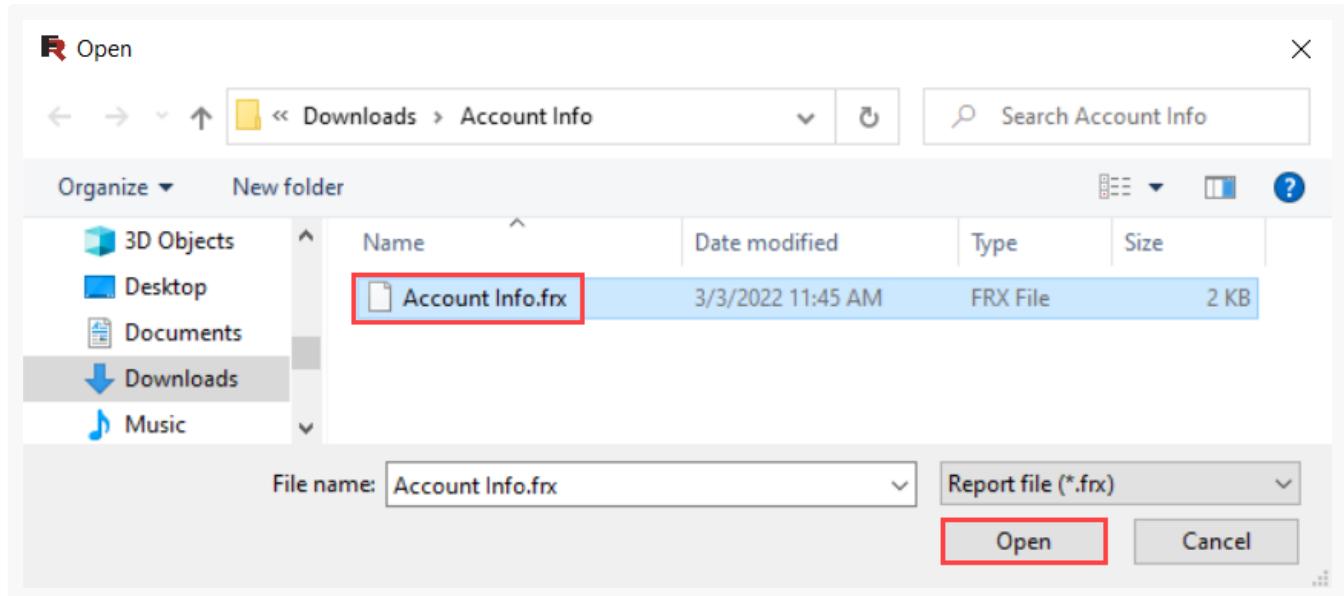
а. В блоке [ Загрузите настроенный шаблон отчета в Creatio ] ([ Import a file with the report template ]) рабочей области страницы настройки отчета нажмите на кнопку [ Загрузить шаблон ] ([ Upload template ]).

## Import a file with the report template

After you import the .frx file, the report will appear in the "Print" menu of the section or on the record page as per the report settings.

**UPLOAD FILE**

- Перейдите в каталог с настроенным шаблоном отчета (обычно это каталог `Downloads`), выберите файл с отчетом и нажмите на кнопку [ *Open* ].



В результате настроенный шаблон отчета [ *Информация контрагента* ] ([ *Account Info* ]) загружен в Creatio.

Для переноса изменений между рабочими средами, привяжите отчет к пакету. Для этого воспользуйтесь инструкцией, которая приведена в статье [Отчеты FastReport](#).

## Результат выполнения примера

Чтобы **посмотреть результат выполнения примера**, обновите страницу раздела [ *Контрагенты* ] ([ *Accounts* ]).

В результате выполнения примера на страницу раздела [ *Контрагенты* ] ([ *Accounts* ]) добавлен отчет [ *Информация контрагента* ] ([ *Account Info* ]). Отчет доступен в выпадающем меню кнопки [ *Печать* ] ([ *Print* ]) панели инструментов раздела.

Accounts

NEW ACCOUNT ACTIONS ▾

Filters/folders Tag

**Vertigo Systems**

Primary contact Peter Moore

Web www.vertigosys.com Primary phone +44 (20) 3427 1374 Type Customer

Address 83 Ashton Street City London Country United Kingdom

OPEN COPY DELETE

PRINT ▾ VIEW ▾

Account Info

Отчет [ Информация контрагента ] ([ Account Info ]) имеет вид, который представлен на рисунке ниже.

**Account Info**

Name	Logo
Vertigo Systems	

Если в реестре раздела [ Контрагенты ] ([ Accounts ]) не выбрана запись, то отчет [ Информация контрагента ] ([ Account Info ]) не активен в выпадающем меню кнопки [ Печать ] ([ Print ]) панели инструментов.

Accounts

NEW ACCOUNT ACTIONS ▾

Filters/folders Tag

**Vertigo Systems**

Primary contact Peter Moore

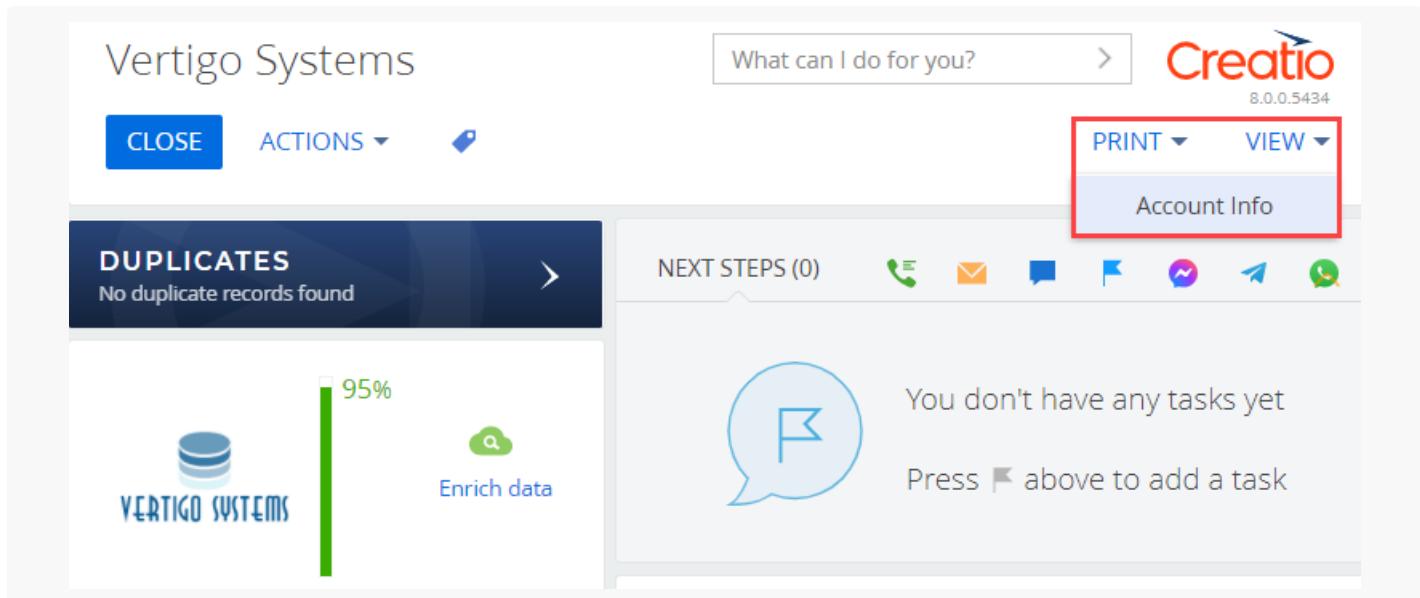
Web www.vertigosys.com Primary phone +44 (20) 3427 1374 Type Customer

Address 83 Ashton Street City London Country United Kingdom

PRINT ▾ VIEW ▾

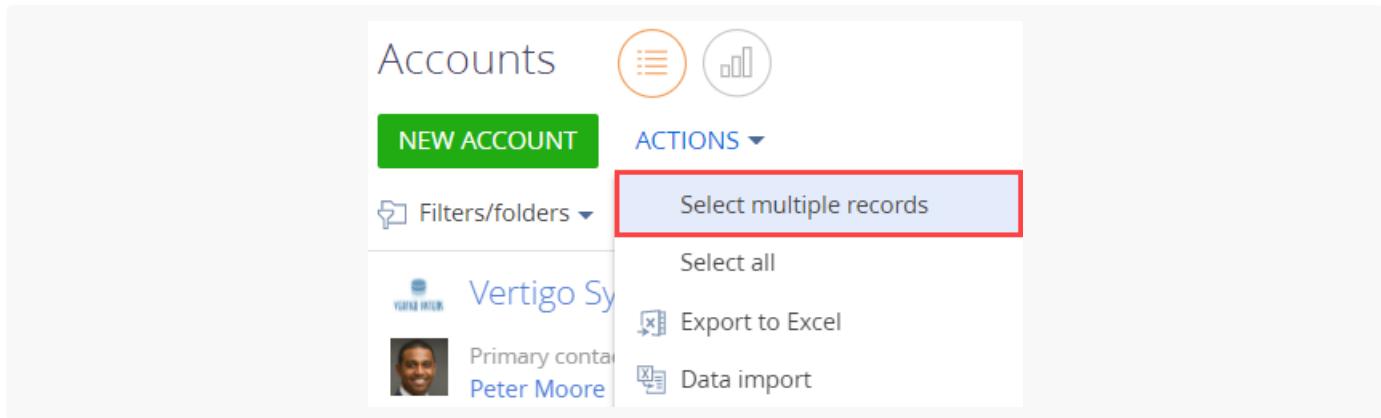
Account Info

В результате выполнения примера на страницу контрагента также добавлен отчет [ Информация контрагента ] ([ Account Info ]). Отчет доступен в выпадающем меню кнопки [ Печать ] ([ Print ]) панели инструментов страницы контрагента.



Чтобы **посмотреть результат выполнения примера по нескольким контрагентам**:

1. Обновите страницу раздела [ Контрагенты ] ([ Accounts ]).
2. Выберите несколько записей раздела. Для этого на панели инструментов раздела выполните действие [ Действия ] —> [ Выбрать несколько записей ] ([ Actions ] —> [ Select multiple records ]).



3. В реестре раздела выберите несколько контрагентов.
4. Выберите отчет [ Информация контрагента ] ([ Account Info ]) в выпадающем меню кнопки [ Печать ] ([ Print ]) панели инструментов раздела.

Отчет [ Информация контрагента ] ([ Account Info ]) по нескольким записям имеет вид, который представлен на рисунке ниже.

<u>Account Info</u>	
Name	Logo
Vertigo Systems	
Sunrise Investments	
RealWay	

## Синхронизация почты с MS Exchange



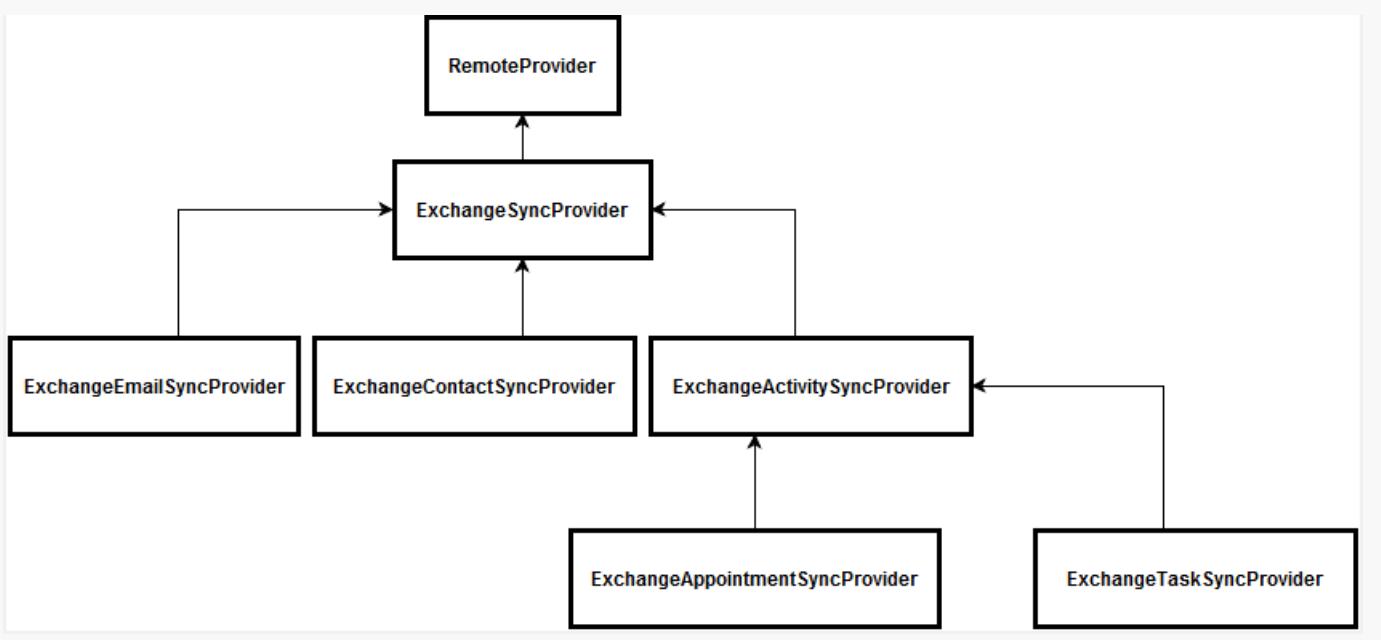
Сложный

В отличие от общего механизма синхронизации, почта синхронизируется только из MS Exchange в Creatio. В силу того что письма после отправки уже не могут быть изменены, в синхронизации участвуют только ранее не синхронизированные письма. Основным отличием механизма синхронизации почты от других интеграций является механизм поиска письма в Creatio. Так как одно и то же письмо может быть синхронизировано от имени любого из получателей, и даже по протоколу IMAP, то для поиска ранее синхронизированных писем нельзя использовать метаданные синхронизации. Для поиска письма используется тема, дата отправки и тело письма. Из тела письма удаляется разметка и пробельные символы. Для ускорения поиска используется md5 хэш, который хранится в колонке `MailHash` объекта `Activity`.

Вторым отличием данной синхронизации является то, что деталь вложения синхронизируется отдельным процессом, после того как обработаны все письма. Это сделано для того, чтобы время на скачивание вложения не влияло на время сохранения писем.

## Реализация интеграции

Как описано в [статье](#), для того чтобы реализовать интеграцию с использованием данного механизма, необходим класс, реализующий логику работы с внешним хранилищем (наследник `RemoteProvider`) и класс, реализующий интерфейс `IRemoteItem`, который представляет один экземпляр элемента синхронизации (в нашем случае — `email MS Exchange`).

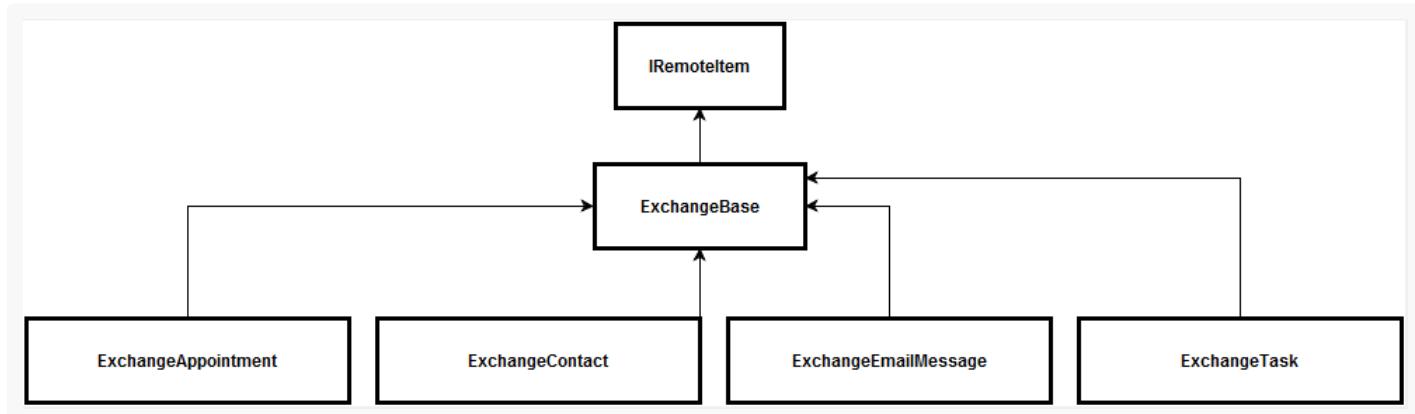


Класс `ExchangeEmailSyncProvider` является провайдером для работы с внешним хранилищем exchange, в нем реализована логика по выбору данных из MS Exchange.

Класс `ExchangeEmailMessage` реализует интерфейс `IRemoteItem` и в нем реализована логика заполнения данных в объектах Creatio.

Класс `ExchangeUtility` содержит утилитные методы для использования библиотеки EWS API и методы, используемые для загрузки вложений писем.

Класс `ExchangeEmailMessageUtility` содержит утилитные методы для преобразования справочных значений из полей email.



## Синхронизируемые данные

Соответствие объектов Creatio и полей класса `EmailMessage` MS Exchange отображено в таблице.

Соответствие объектов Creatio и полей класса `EmailMessage` MS Exchange

<b>Объект Creatio</b>	<b>Поле объекта</b>	<b>Соответствующее поле EmailMessage</b>
Activity	Title	Subject
	Body	Body.Text
	Sender	Sender
	Recipient	ToRecipients
	CopyRecipient	CcRecipients
	BlindCopyRecipient	BccRecipients
	SendDate	DateTimeSpent
	Priority	Importance
	DueDate, StartDate	DateTimeReceived
ActivityFile	Name	Name
	Data	Content
	Size	Content.Length

## Логика заполнения участников письма

Для того чтобы письмо правильно отображалось только у пользователей, которые его синхронизировали, реализован следующий механизм заполнения детали [ Участники активности ]. Условно эту логику можно разделить на две части:

1. Создание участников для нового письма.
2. Актуализация списка участников при его изменении (в том числе, повторной синхронизации).

## Создание участников для нового письма

Основным значением, которое влияет на то, кто попадет в участники письма, является деталь [ Средства связи ] контакта. Если у контакта на детали [ Средства связи ] есть email, который указан в одном из адресных полей письма ([ от ], [ кому ], [ копия ], [ скрытая копия ]), то контакт может быть добавлен в участники. Дополнительно выполняется проверка, существует ли для этого контакта пользователь системы, не являющийся пользователем портала. Пользователь добавляется в участники только тогда, когда он синхронизирует это письмо. Это позволяет добавить в участники переписки всех внешних контактов, и только тех пользователей, которые синхронизировали это письмо.

## Актуализация списка участников письма

Для того чтобы после синхронизации уже существующего письма пользователь попал в участники письма, при изменении письма происходит актуализация участников — все участники, которые не являются пользователями Creatio, удаляются с детали, и затем выполняется алгоритм заполнения детали для нового письма. Таким образом, пользователи, которые ранее синхронизировали письмо, остаются в участниках, новый пользователь добавляется, а список контактов актуализируется.

## Логика выбора данных для синхронизации

При выборе писем для синхронизации из папок MS Exchange используется следующий набор фильтров: выбрать письма, которые изменены после даты последней синхронизации почты, и не являются черновиками. Для папок синхронизации существует ограничение: папка " удаленные " и папка " Конфликтующие элементы " не участвуют в синхронизации. При выборе писем не учитывается наличие метаданных синхронизации. Направление изменений всегда " сохранить изменения в Creatio ". При обработке каждого письма вначале проверяется наличие письма в Creatio. Если письмо уже существует в Creatio, выполняется обновление участников, если нет, то создается новое письмо. В конце сессии синхронизации в планировщик добавляется задание на синхронизацию вложений.

## Интеграция с Oktell

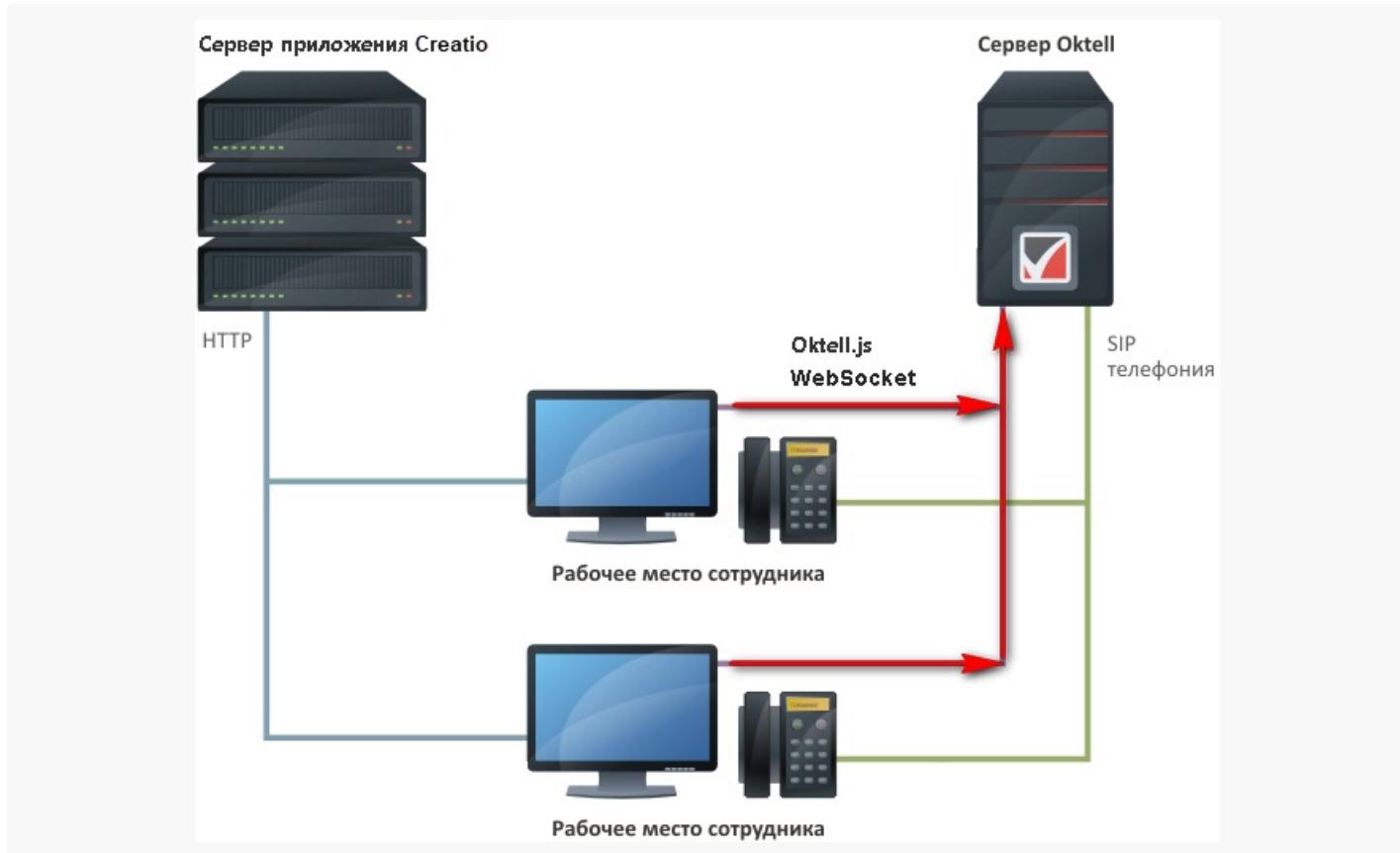


Сложный

Интеграция Oktell с Creatio реализована на клиентском уровне с помощью библиотеки `oktell.js`.

Исходный код библиотеки `oktell.js` находится в конфигурационной схеме `OktellModule` пакета `ctvBase`.

Сервер Oktell взаимодействует с телефонами и с конечными клиентами (браузерами). При таком способе интеграции в Creatio не требуется наличие собственного WebSocket-сервера. Каждый клиент подключается по WebSocket-протоколу непосредственно к серверу Oktell. Сервер приложения Creatio занимается формированием страниц и предоставлением данных из базы данных приложения. Непосредственная взаимосвязь между серверами Creatio и Oktell отсутствует, доступ не требуется, клиенты самостоятельно обрабатывают и объединяют данные двух систем. По такому принципу реализованы Web-клиент Oktell и плагин `oktell.js`, доступный для встраивания в другие проекты.



## Oktell.js

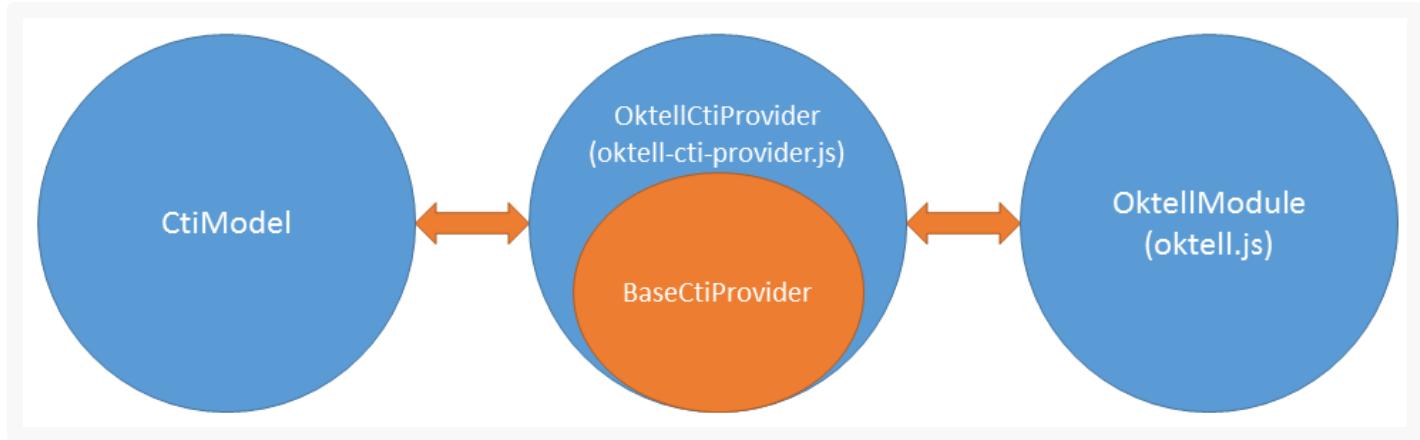
`Oktell.js` — javascript-библиотека для встраивания функциональности управления звонками в CRM-систему. `Oktell.js` использует протокол Oktell WebSocket для соединения с сервером Oktell. Преимущество этого протокола заключается в создании постоянного асинхронного соединения с сервером, которое позволяет без задержек получать события с сервера Oktell и выполнять определенные команды. Поскольку протокол Oktell WebSocket достаточно сложен для реализации, библиотека `Oktell.js` обворачивает внутри себя методы WebSocket-протокола, предоставляя простую функциональность для управления.

## Передача голоса между абонентами

Голос при разговоре между операторами `oktell Creatio` передается по протоколу [Session Initiation Protocol \(SIP\)](#). Для этого требуется использование либо [IP-телефона](#), либо установленного на компьютер оператора [софтфона](#).

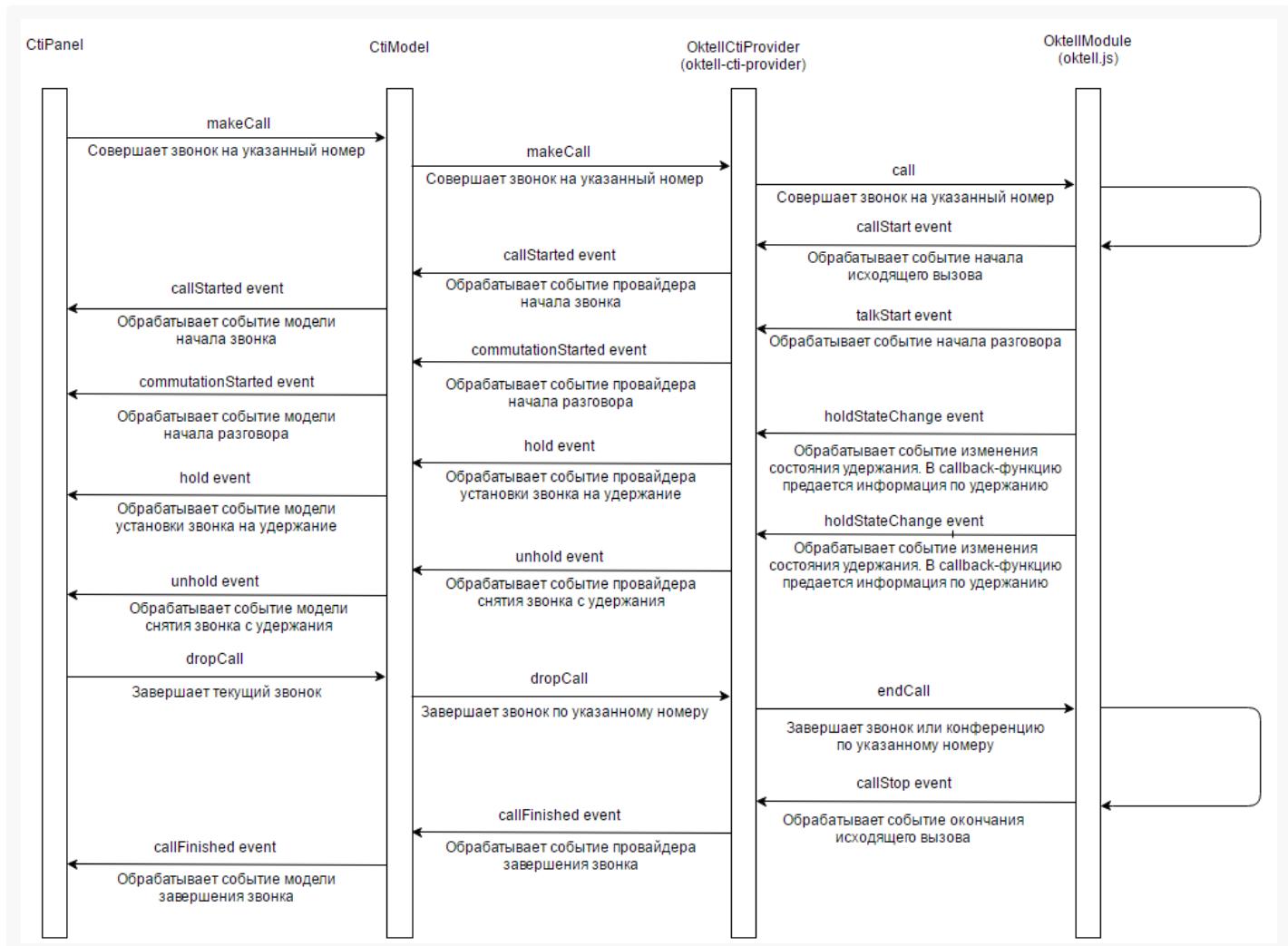
## Взаимодействие компонентов

Взаимодействие с библиотекой `oktell.js` осуществляется с помощью класса `OktellCtiProvider`, который является связующим звеном между `CtiModel` и `OktellModule`, в котором находится код `oktell.js`. Класс `OktellCtiProvider` реализует интерфейс класса [`BaseCtiProvider`](#).

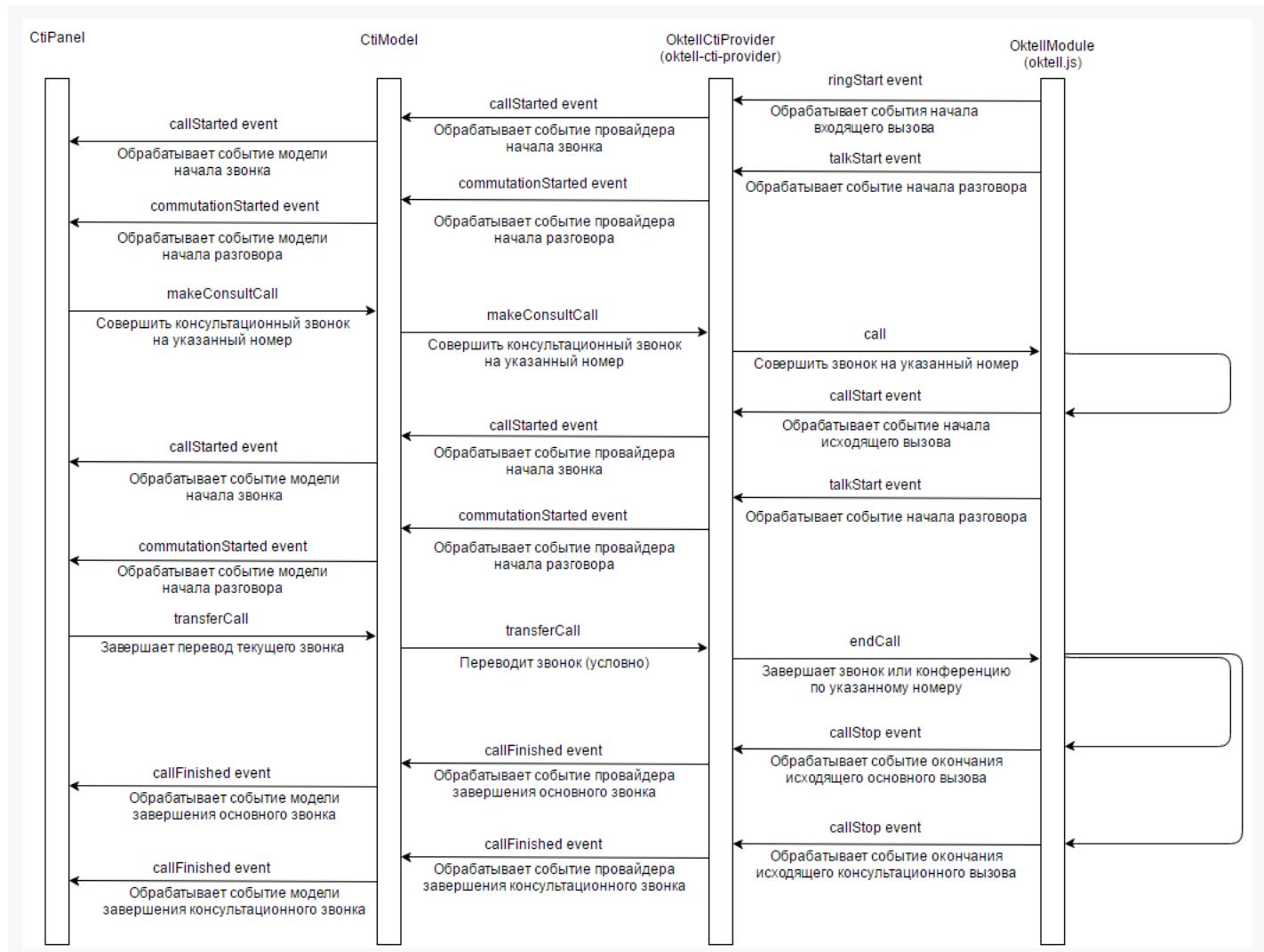


Примеры взаимодействия CtModel, OktellCtiProvider и OktellModule:

Исходящий звонок оператора абоненту с установкой звонка на удержание абонентом, снятием с удержания абонентом и завершением звонка оператором



Входящий звонок абонента 1 оператору с консультационным звонком абоненту 2, с последующим соединением оператором абонента 1 и абонента 2



Список поддерживаемых событий класса библиотеки `oktell.js` перечислен в таблице.

Список поддерживаемых событий класса библиотеки `oktell.js`

Событие	Описание
connect	Событие успешного соединения с сервером
connectError	Событие ошибки соединения с сервером в методе connect. Коды ошибок такие же, как для callback-функции метода connect
disconnect	Событие закрытия соединения с сервером. В callback-функцию передается объект с описанием причины разрыва соединения
statusChange	Событие изменения состояния агента. В callback-функцию передается два строковых параметра — новое и прошлое состояние
ringStart	Событие начала входящего вызова
ringStop	Событие завершения входящего вызова
backRingStart	Событие начала обратного вызова
backRingStop	Событие завершения обратного вызова
callStart	Событие начала исходящего вызова
callStop	Событие смены UUID звонка
talkStart	Событие начала разговора
talkStop	Событие окончания разговора
holdAbonentLeave	Событие выхода абонента из удержания. В callback-функцию передается объект <code>abonent</code> с информацией по абоненту
holdAbonentEnter	Событие входа абонента в удержание. В callback-функцию передается объект <code>abonent</code> с информацией по абоненту
holdStateChange	Событие изменения состояния удержания. В callback-функцию передается информация по удержанию
stateChange	Событие изменения состояния линии
abonentsChange	Событие изменения списка текущих абонентов
flashstatechanged	Низкоуровневое событие изменения состояния удержания
userstatechanged	Низкоуровневое событие изменения состояния пользователя

# Портал самообслуживания



Портал самообслуживания (Self Service Portal, SSP) является неотъемлемой функциональной частью продуктов [Service Creatio, enterprise edition](#), [Service Creatio, customer center edition](#), а также всех производных пакетов (bundle) продуктов, в которые они входят.

Портал представляет собой отдельное рабочее место (Workplace), в которое автоматически попадает пользователь портала при входе через стандартное окно авторизации приложения.

По умолчанию в поставляемом продукте портальному пользователю доступно два раздела — [*Обращения портала*] и [*База знаний*], а также [*Главная страница портала*], которая содержит общую сводную информацию и выполняет роль единого рабочего места пользователя портала.

Раздел [*Обращения портала*] служит для самостоятельной регистрации обращений пользователем, просмотра статуса своих обращений, внесения дополнительной информации по обращению, а также для получения информации по решению своего обращения. По умолчанию портальный пользователь имеет доступ только к тем обращениям, в которых он указан в качестве контакта. На странице обращения пользователь может внести дополнительную информацию, публиковать сообщения, вести переписку по своему обращению с сотрудниками службы поддержки. История переписки по обращению отображается на странице обращения на вкладке [*Обработка*].

Раздел [*База знаний*] служит для самостоятельного поиска пользователем справочной информации или решения по своему вопросу. Данный раздел по умолчанию может наполняться только сотрудниками поддержки из основного интерфейса системы.

## Устройство портала

С точки зрения разработки, портал является преднастроенным отдельным рабочим местом. Это рабочее место по умолчанию недоступно обычным пользователям приложения. Пользователь системы с типом "Пользователь портала" при авторизации автоматически попадает в данное рабочее место, а именно на Главную страницу портала.

Разделы и страницы портала — это обычные [разделы](#) и [страницы записи](#), и работают с теми же сущностями ([Entity](#)), что и страницы, доступные в основном интерфейсе системы. В частности, страница обращения на портале имеет более простой вид, и не содержит большинства полей. В конфигурации это две разные страницы записи — [CasePage](#) и [PortalCasePage](#).

Система прав доступа на портале имеет только одно небольшое отличие. Чтобы дать портальным пользователям доступ к определенным сущностям (EntitySchema), необходимо дополнительно указать эти сущности в справочнике [*Список объектов доступных портальному пользователю*]. Лицензии портала самообслуживания ограничивают количество записей, которое может быть добавлено в данный справочник. По умолчанию это 70 записей.

## Работа с мастером страниц на портале

Портальному пользователю недоступна функциональность мастеров страниц, деталей и разделов. Эта функциональность может быть вызвана пользователем с правами администратора из основного интерфейса системы следующим образом:

1. Перейти в раздел [ Настройка рабочих мест ] дизайнера системы.
2. Выбрать рабочее место [ Портал ] и нажать [ Открыть ].
3. Выбрать требуемый раздел и нажать кнопку [ Настройка раздела ].

После этого откроется стандартный мастер разделов.

## Настройка портала и портальных пользователей

Для того чтобы начать пользоваться порталом, необходимо выполнить следующие действия:

1. Убедиться, что в файле `web.config` загрузчика приложения (т. н. "внешний" `web.config`) опция `/configuration/terrasoft/auth` имеет следующий вид.

### Опция `/configuration/terrasoft/auth` файла `web.config`

```
<terrasoft>
  <auth providerNames="InternalUserPassword,SSPUserPassword" ...>
  ...
</terrasoft>
```

Данная настройка отвечает за возможность авторизации в системе портальных пользователей.

2. Создать контакт для пользователя.
3. Создать пользователя с типом "Пользователь портала". Заполнить необходимые поля.
4. Раздать необходимые лицензии пользователю.

У портального пользователя в профиле должен быть указан корректный часовой пояс. По умолчанию у нового пользователя он не указан. Настройка выполняется самим пользователем портала в его профиле. Отталкиваясь от этого значения, в системе будут отображаться все даты и времена.

## Ограничение доступа к веб-сервисам для пользователей портала

Основной сценарий использования портала — предоставить доступ большому количеству внешних по отношению к организации пользователей к ограниченному количеству данных из системы. Для решения таких задач возникает потребность управлять тем, какие пользователи (портальные или пользователи компании) имеют доступ к [веб-сервисам приложения](#).

**Важно.** Данная статья актуальна для версии системы 7.13.2 и новее.

## Префиксы маршрутов конфигурационных веб-сервисов

Для управления доступом к веб-сервисам приложения в Creatio реализован механизм префиксов маршрута. Используя определенные `ServiceRoute` атрибуты класса сервиса, можно задать необходимый префикс маршрута.

## Префиксы маршрутов конфигурационных веб-сервисов

Доступ	Атрибут	Префикс маршрута	Пример кода
Только для пользователей Портала самообслуживания	SspServiceRoute	/ssp	<pre>[ServiceContract] [SspServiceRoute] public class SspOnlyService {}</pre>
			<p><b>Пример вызова</b></p> <pre>~/ssp/rest/SspOnlyService ~/ssp/soap/SspOnlyService</pre>
Только для внутренних пользователей	DefaultServiceRoute	отсутствует	<pre>[ServiceContract] public class InternalService {}</pre>
	или		<p><b>ИЛИ</b></p> <pre>[ServiceContract] [DefaultServiceRoute] public class InternalService {}</pre>
	не указывать ни один ServiceRoute атрибут		<p><b>Пример вызова</b></p> <pre>~/rest/InternalService ~/soap/InternalService</pre>
И для внутренних пользователей, и для Портала	одновременно атрибуты DefaultServiceRoute и	/ssp или отсутствует	<pre>[ServiceContract] [DefaultServiceRoute] [SspServiceRoute]</pre>

`SspServiceRoute`

```
public class CommonService
{
}
```

### Пример вызова

`~/rest/CommonService`  
`~/soap/CommonService`  
`~/ssp/rest/CommonService`  
`~/ssp/soap/CommonService`

атрибут `ServiceRoute`  
 с указанием  
 префикса (например,  
`"custom"`)

`ServiceRoute("custom")`

произвольный  
 префикс  
 маршрута  
 сервиса

```
[ServiceContract]
[ServiceRoute("custom")]
public class CustomPrefixS
{
}
```

### Пример вызова

`~/custom/rest/CustomPrefix`  
`~/custom/soap/CustomPrefix`

**На заметку.** Допустимо использовать одновременно несколько атрибутов `ServiceRoute`, `SspServiceRoute` и `DefaultServiceRoute`. В результате будут созданы маршруты для сервиса со всеми вариантами префиксов.

## Ограничение доступа портальных пользователей к внутреннему API

Если пользователь портала (`SspUserConnection`) обратится к сервису по маршруту без префикса `"/ssp"`, сервис вернет в ответ страницу с кодом 403 (`Forbidden`).

## Ограничение доступа внутренних пользователей к портальному API

Если внутренний пользователь приложения (`UserConnection`) обратится к сервису по маршруту с префиксом `"/ssp"`, сервис вернет в ответ страницу с кодом 403 `Forbidden`.

## ServiceStack сервисы ядра

Методы сервисов `ServiceStack` ядра (`DataService`, `ManagerService` и другие) по умолчанию доступны только для внутренних пользователей.

Для того, чтобы какой-то из методов был доступен также и на Портале, необходимо пометить такой метод атрибутом `SspServiceAccess` - в этом случае метод будет иметь дополнительный маршрут с префиксом вида `~/DataService/ssp/...`

Если нужно, чтобы логика метода отличалась для Портала, необходимо создать новый сервис, указав для него атрибут `SspServiceAccess`, в конструктор которого передать имя оригинального метода.

Например:

```
[SspServiceAccess(nameof(SelectQuery))]
public class SspSelectQuery : SelectQuery
{
}
```

создает контракт, метод которого будет зарегистрирован по пути `~/DataService/ssp/SelectQuery`.

Доступ к методам `ServiceStack` с префиксом “`ssp`” запрещен внутренним пользователям, а к методам `ServiceStack` без префикса “`ssp`” - пользователям портала.

## Пример использования миксина PortalMessagePublisherExtensions

 Сложный

**Пример.** Использовать миксин `PortalMessagePublisherExtensions`.

## Реализация примера

Реализовать пример использования миксина `PortalMessagePublisherExtensions`.

`PortalMessagePublisherExtensions`

```
define("CaseSectionActionsDashboard", ["PortalMessagePublisherExtensions"], function() {
    return {
        mixins: {
            /**
             * @class PortalMessagePublisherExtensions extends tabs and publishers configs.
             */
            PortalMessagePublisherExtensions: "Terrasoft.PortalMessagePublisherExtensions"
        },
        methods: {
```

```

    /**
     * @inheritDoc Terrasoft.SectionActionsDashboard#getExtendedConfig
     * @override
     */
    getExtendedConfig: function() {
        // Получение объекта конфигурации вкладок из родительского метода.
        var config = this.callParent(arguments);
        // Вызов метода миксина, добавление конфигурации портальной вкладки.
        this.mixins.PortalMessagePublisherExtensions.extendTabsConfig.call(this, config)
        // Возвращение расширенного объекта конфигурации.
        return config;
    },

    /**
     * @inheritDoc Terrasoft.SectionActionsDashboard#getSectionPublishers
     * @override
     */
    getSectionPublishers: function() {
        // Получение коллекции издателей сообщений из родительского метода.
        var publishers = this.callParent(arguments);
        // Вызов метода миксина, добавление портального канала.
        this.mixins.PortalMessagePublisherExtensions.extendSectionPublishers.call(this,
        // Возвращение расширенной коллекции издателей сообщений.
        return publishers;
    }
},
diff: /**SCHEMA_DIFF*/[
{
    "operation": "insert",
    "name": "PortalMessageTab",
    "parentName": "Tabs",
    "propertyName": "tabs",
    "values": {
        "items": []
    }
},
{
    "operation": "insert",
    "name": "PortalMessageTabContainer",
    "parentName": "PortalMessageTab",
    "propertyName": "items",
    "values": {
        "itemType": this.Terrasoft.ViewItemType.CONTAINER,
        "classes": {
            "wrapClassName": ["portal-message-content"]
        },
        "items": []
    }
}
]

```

```

    },
    {
        "operation": "insert",
        "name": "PortalMessageModule",
        "parentName": "PortalMessageTab",
        "propertyName": "items",
        "values": {
            "classes": {
                "wrapClassName": [ "portal-message-module", "message-module" ]
            },
            "itemType": this.Terrasoft.ViewItemType.MODULE,
            "moduleName": "PortalMessagePublisherModule",
            "afterrender": {
                "bindTo": "onMessageModuleRendered"
            },
            "afterrerender": {
                "bindTo": "onMessageModuleRendered"
            }
        }
    }
}
]/**SCHEMA_DIFF*/
);
});

```

# Изменить доступ к сервису для пользователей портала



Сложный

В приложении существует набор базовых сервисов, доступ к которым есть только у внутренних пользователей.

Для изменения доступа к базовому сервису необходимо:

1. В пользовательском пакете создать сервис с настройкой доступа для портальных пользователей.
2. Добавить для него те методы базового сервиса, которые должны быть доступны для портальных пользователей.
3. Изменить пользовательские или расширить базовые клиентские схемы, изменив вызов базового сервиса на вызов созданного сервиса (см. шаг 1).
4. Скомпилировать конфигурацию.

**Пример.** Изменить доступ к сервису для пользователей портала.

## Реализация примера

Пример исходного кода сервиса, который расширяет доступ к базовому сервису `ActivityUtilService` представлен ниже.

```
PartnerActivityUtilService

namespace Terrasoft.Configuration
{
    using System;
    using System.IO;
    using System.Runtime.Serialization;
    using System.ServiceModel;
    using System.ServiceModel.Activation;
    using System.ServiceModel.Web;
    using Terrasoft.Configuration.FileUpload;
    using Terrasoft.Core.Factories;
    using Terrasoft.Web.Common;
    using Terrasoft.Web.Common.ServiceRouting;

    [ServiceContract]
    // Сервис доступен и для внутренних пользователей и для Портала.
    [DefaultServiceRoute]
    [SspServiceRoute]
    [AspNetCompatibilityRequirements(RequirementsMode = AspNetCompatibilityRequirementsMode.Req
public class PartnerActivityUtilService: BaseService {
    // Базовый сервис, доступ к которому необходимо расширить.
    private static readonly ActivityUtilService _baseService = new ActivityUtilService();

    [OperationContract]
    [WebInvoke(Method = "POST", BodyStyle = WebMessageBodyStyle.Wrapped, RequestFormat = Web
    public Guid CreateActivityFileEntity(JsonActivityFile jsonActivityFile) {
        return _baseService.CreateActivityFileEntity(jsonActivityFile);
    }
    [OperationContract]
    [WebInvoke(Method = "POST", BodyStyle = WebMessageBodyStyle.Wrapped, RequestFormat = Web
    public Guid CreateFileEntity(JsonEntityFile jsonEntityFile) {
        return _baseService.CreateFileEntity(jsonEntityFile);
    }
}
}
```

## Миксин PortalMessagePublisherExtensions

JS



**Миксин** — это класс-примесь, предназначенный для расширения функциональности других классов. Это отдельно созданные классы с дополнительной функциональностью.

Миксин `PortalMessagePublisherExtensions` предназначен для расширения схемы `SectionActionsDashboard` и ее наследников. Он позволяет расширить конфигурацию вкладок `SectionActionsDashboard` вкладкой порталных сообщений `PortalMessageTab` и добавить соответствующий канал сообщений `Portal`. Миксин реализован в пакете `PortalMessagePublisher` и доступен в продукте `ServiceEnterprise`, а также в бандлах, в которые входит этот продукт.

## Методы

`extendTabsConfig(config) : Object`

Расширяет конфигурацию вкладок `SectionActionsDashboard` вкладкой порталных сообщений `PortalMessageTab`.

Возвращает дополненный объект (`Object`) конфигурации вкладок `SectionActionsDashboard`.

### Параметры

<code>config (Object)</code>	Объект конфигурации вкладок <code>SectionActionsDashboard</code> .
------------------------------	--

`extendSectionPublishers(publishers) : Array`

Добавляет порталный канал (`Portal`) в коллекцию издателей сообщений.

Возвращает дополненную коллекцию издателей сообщений (`Array`).

### Параметры

<code>publishers (Array)</code>	Коллекция издателей сообщений.
---------------------------------	--------------------------------

# Отправка email-сообщений



**Важно.** Возможность отправки email-сообщений с явным указанием учетных данных доступна в Creatio версии 7.16 и выше.

В Creatio вы можете отправлять email-сообщения не только пользовательскими средствами, но и средствами разработки. Это реализуется следующими способами:

- С существующей учетной записи.

- С явным указанием учетных данных.

Отправка email-сообщения реализована с помощью бизнес-процесса. Для настройки бизнес-процесса используются элементы [\[Автогенерируемая страница\]](#) ([\[Auto-generated page\]](#)) и [\[Задание-сценарий\]](#) ([\[Script task\]](#)).

## Алгоритм отправки email-сообщения с существующей учетной записи

1. Создать бизнес-процесс, который должен включать обязательные элементы [ Автогенерируемая страница ] и [ Задание-сценарий ].
2. Создать конфиг отправляемого email-сообщения.
3. Добавить вложение (не обязательно).
4. Запустить бизнес-процесс для выполнения отправки.

### Создание конфига отправляемого email-сообщения

Для создания конфига отправляемого email-сообщения необходимо использовать класс `Terrasoft.Mail.Sender.EmailMessage`. Для формирования валидного email-сообщения необходимо заполнить параметры.

#### Формирование валидного email-сообщения

```
var message = new Terrasoft.Mail.Sender.EmailMessage {
    // Email-адрес отправителя.
    From = "Sender@email.com",
    // Email-адреса получателей.
    To = List<string>{ "first@recipient.co", "second@recipient.co" },
    // Копия (не обязательно).
    Cc = List<string>{ "first@recipient.co", "second@recipient.co" },
    // Скрытая копия (не обязательно).
    Bcc = List<string>{ "first@recipient.co", "second@recipient.co" },
    // Тема письма.
    Subject = "Message subject",
    // Тело письма.
    Body = "Body",
    // Приоритет, значения из перечисления Terrasoft.Mail.Sender.EmailPriority.
    Priority = Terrasoft.Mail.Sender.EmailPriority.Normal
};
```

### Добавление вложения (не обязательно)

В email-сообщение можно добавить вложение. Для этого необходимо заполнить поле [ *Attachments* ]. Вложения представляют собой список экземпляров `Terrasoft.Mail.Sender.EmailAttachments`.

## Добавление вложения

```
// Создание вложения.
var attachment = new Terrasoft.Mail.Sender.EmailAttachment {
    // Идентификатор вложения.
    Id = new Guid("844F0837-EAA0-4F40-B965-71F5DB9EAE6E"),
    // Имя вложения.
    Name = "attachName.txt",
    // Данные.
    Data = byteData
};
// Добавление вложения в письмо.
message.Attachments.Add(attachment);
```

## Отправка email-сообщения

Для отправки письма необходимо использовать метод `Send` класса `EmailSender` с переданными параметрами email-сообщения и конфига подключения.

### Отправка email-сообщения

```
// Отправка сформированного email-сообщения. Для игнорирования прав доступа при отправке
// требуется присвоить параметру ignoreRights значение true.
emailSender.Send(message, ignoreRights);
```

## Алгоритм отправки email-сообщения с явным указанием учетных данных

1. Создать бизнес-процесс, который должен включать обязательные элементы [ Автогенерируемая страница ] и [ Задание-сценарий ].
2. Создать экземпляр класса `EmailClientFactory`.
3. Создать экземпляр класса `EmailSender`.
4. Создать конфиг подключения к почтовому ящику.
5. Создать конфиг отправляемого email-сообщения.
6. Добавить вложение (не обязательно).
7. Запустить бизнес-процесс для выполнения отправки.

## Создание экземпляра класса EmailClientFactory

Для создания экземпляра класса `EmailClientFactory` необходимо наличие пользовательского подключения `UserConnection`.

**Создание экземпляра класса EmailClientFactory**

```
var emailClientFactory = ClassFactory.Get<IEmailClientFactory>(
    new ConstructorArgument("userConnection", UserConnection));
```

**Создание экземпляра класса EmailSender**

Для создания экземпляра класса `EmailSender` необходимо передать в конструктор созданный экземпляр `EmailClientFactory` и пользовательское подключение `UserConnection`.

**Создание экземпляра класса EmailClientFactory**

```
var emailSender = ClassFactory.Get<IEmailSender>(
    new ConstructorArgument("emailClientFactory", emailClientFactory),
    new ConstructorArgument("userConnection", UserConnection));
```

**Создание конфига подключения к почтовому ящику**

Для создания конфига подключения к почтовому ящику используется класс `EmailContract.DTO.Credentials`. Чтобы создать конфиг, заполните параметры.

**Создание конфига подключения к почтовому ящику**

```
var credentialConfig = new EmailContract.DTO.Credentials {
    // Имя или IP-адрес сервера исходящей почты.
    ServiceUrl = "mail service host",
    // Может быть пустым для некоторых протоколов.
    Port = "Port",
    // Использовать протокол SSL для шифрования подключения.
    UseSsl = false,
    // Имя пользователя почтового ящика.
    UserName = "EmailUserName",
    // Пароль пользователя почтового ящика.
    Password = "UserPassword",
    // Тип почтового сервера (Exchange либо Imap/Ssmtp).
    ServerTypeId = EmailDomain.IntegrationConsts.ExchangeMailServerTypeId ||
        EmailDomain.IntegrationConsts.ImapMailServerTypeId,
    // Почтовый ящик отправителя.
    SenderEmailAddress = "sender@test.com"
};
```

**Важно.** Параметры `ServiceUrl`, `UserName`, `Password`, `ServerTypeId`, `SenderMailbox` являются

обязательными для заполнения.

## Создание конфига отправляемого email-сообщения

Для создания конфига отправляемого email-сообщения необходимо использовать класс `EmailContract.DTO.Email`. Для формирования валидного email-сообщения заполните параметры.

### Создание конфига отправляемого email-сообщения

```
var message = new EmailContract.DTO.Email {
    // Email-адрес отправителя.
    Sender = "Sender@email.com",
    // Email-адреса получателей.
    Recipients = List<string>{ "first@recipient.co", "second@recipient.co" },
    // Тема письма.
    Subject = "Message subject",
    // Тело письма.
    Body = "Body",
    // Приоритет, значения из перечисления EmailContract.EmailImportance.
    Importance = EmailContract.EmailImportance.High
};
```

## Добавление вложения (не обязательно)

В email-сообщение можно добавить вложение. Для этого необходимо заполнить поле [ *Attachments* ]. Вложения представляют собой список экземпляров `EmailContract.DTO.Attachment`.

### Добавление вложения

```
// Создание вложения.
var attachment = new EmailContract.DTO.Attachment {
    Name = "FileName",
    Id = "844F0837-EAA0-4F40-B965-71F5DB9EAE6E"
};
// Установка данных для вложения.
attachment.SetData(byteData);
// Добавление вложения в письмо.
message.Attachments.Add(attachment);
```

## Отправка email-сообщения

Для отправки письма необходимо использовать метод `Send` класса `EmailSender` с переданными аргументами email-сообщения и конфига подключения.

## Отправка email-сообщения

```
// Отправка сформированного email-сообщения с параметрами подключения к почтовому
// ящику отправителя. Для игнорирования прав доступа при отправке
// требуется присвоить параметру ignoreRights значение true.
emailSender.Send(message, credentialConfig, ignoreRights);
```

# Отправить email-сообщение с существующей учетной записи



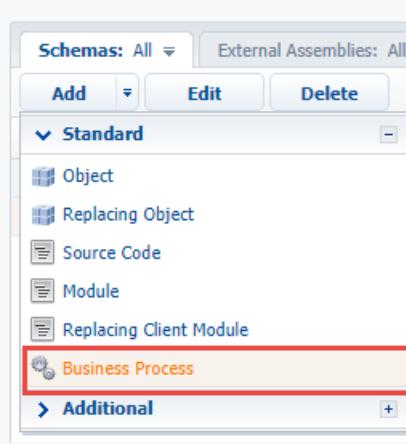
Сложный

**Пример.** Создайте бизнес-процесс в ходе которого будет открываться страница, содержащая параметры email-сообщения для заполнения и дальнейшей отправки email-сообщения с существующей учетной записи.

## Алгоритм реализации примера

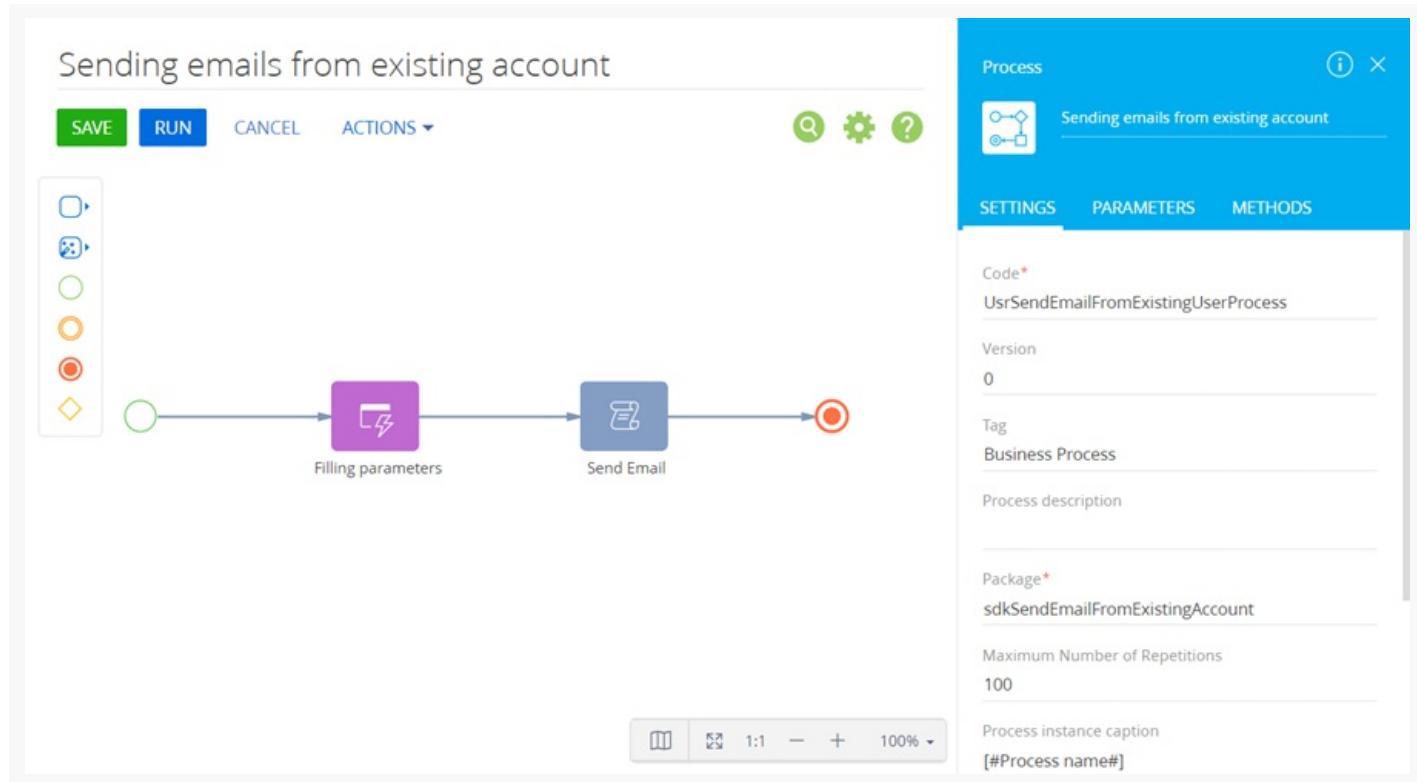
### 1. Создайте бизнес-процесс

В разделе [ Конфигурация ] ([ Configuration ]) выполните действие [ Добавить ] —> [ Бизнес-процесс ] ([ Add ] —> [ Business process ]).



В открывшемся дизайнере процессов установите следующие значения на панели настройки свойств процесса:

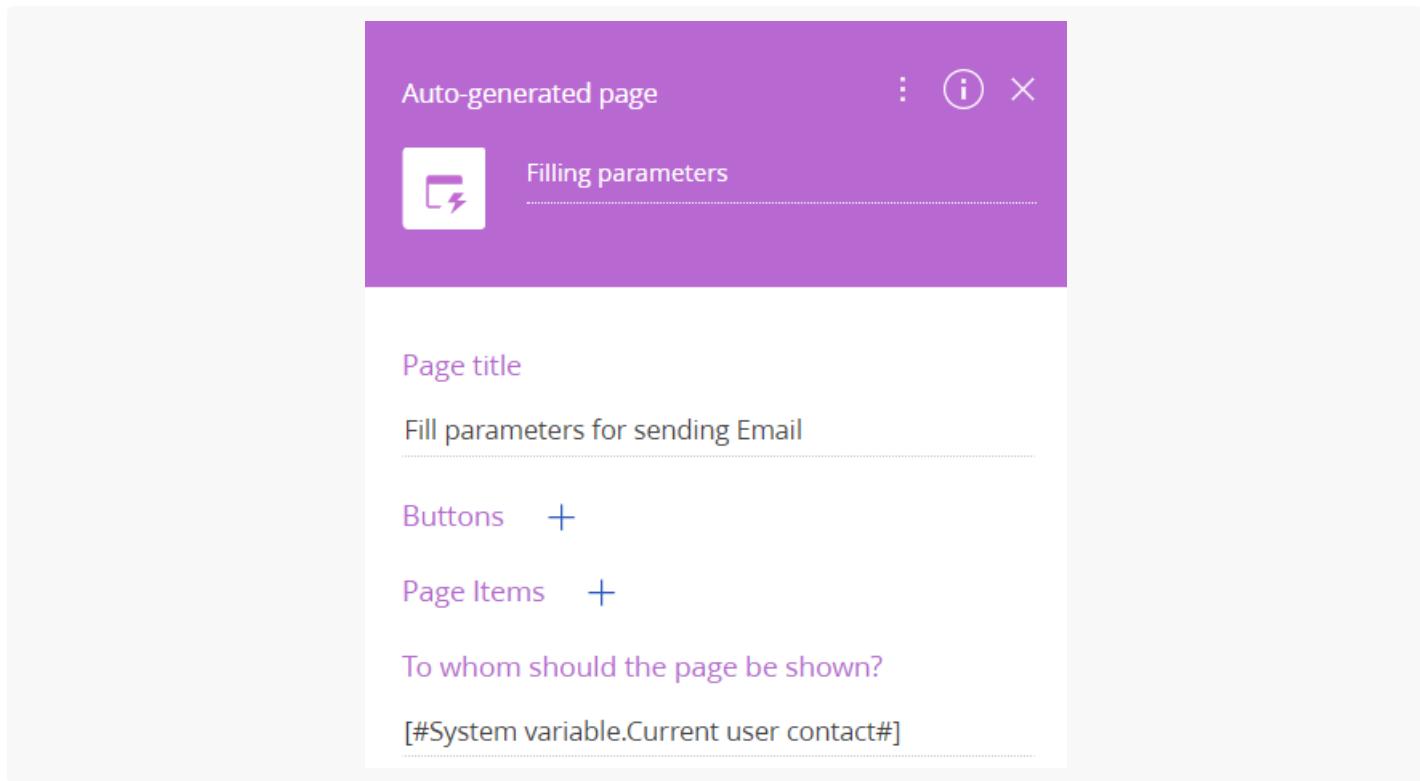
- [ Заголовок ] ([ Title ]) — "Sending emails from existing account".
- [ Код ] ([ Code ]) — "UsrSendEmailFromExistingUserProcess".



## 2. Добавьте элемент [ Автогенерируемая страница ]

С помощью элемента [ Автогенерируемая страница ] ([ *Auto-generated page* ]) в ходе выполнения процесса можно открыть произвольную страницу, которая создана пользователем системы. Для этого элемента добавьте подпись [ *Filling parameters* ] и установите следующие свойства:

- [ *Название страницы* ] ([ *Page title* ]) — "Fill parameters for sending Email".
- [ *Кому отобразить страницу?* ] ([ *To whom should the page be shown?* ]) — выберите "Formula" и установите "[#System variable.Current user contact#]".



### 3. Добавьте кнопку на страницу

Для добавления кнопки [ *Continue* ] на страницу в блоке [ *Кнопки* ] ([ *Buttons* ]) нажмите **+** и введите следующие параметры:

- [ *Название* ] ([ *Caption* ]) — "Continue".
- [ *Код* ] ([ *Code* ]) — "ContinueButton".
- [ *Стиль* ] ([ *Style* ]) — выберите "Green".
- Установите признак [ *Активная* ] ([ *Active* ]).
- Установите признак [ *Выполняет проверку значений* ] ([ *Performs value validation* ]).

**Buttons** +

**Caption\***  
Continue

**Code\***  
ContinueButton

**Style\***  
Green

Generate signal

Active

Performs value validation

**SAVE** **CANCEL**

Нажмите [ Сохранить ] ([ Save ]).

#### 4. Добавьте элементы на страницу

Для добавления на страницу элемента, который будет содержать почтовый ящик отправителя email-сообщения, в блоке [ Параметры страницы ] ([ Page Items ]) нажмите +, выберите тип [ Справочник ] ([ Selection field ]) и введите следующие параметры:

- [ Заголовок ] ([ Title ]) — "Sender Mailbox".
- [ Код ] ([ Code ]) — "SenderId".
- [ Источник данных ] ([ Data source ]) — выберите "Mailbox synchronization settings".
- [ Представление ] ([ View ]) — выберите "Drop down list".

The screenshot shows a configuration dialog for a 'Page Item'. The title bar says 'Page Items' with a plus sign icon. The main area contains the following fields:

- Title\***: Sender Mailbox
- Code\***: SenderMailbox
- Required**: An unchecked checkbox.
- Data source\***: Mailbox synchronization settings
- View\***: Drop down list
- Value**: A large empty text area.

At the bottom right are two buttons: a blue 'SAVE' button and a light blue 'CANCEL' button.

Нажмите [ Сохранить ] ([ Save ]).

Для добавления на страницу элемента, который будет содержать почтовый ящик получателя email-сообщения, в блоке [ Параметры страницы ] ([ Page Items ]) нажмите **+**, выберите тип [ Строковое поле ] ([ Text field ]) и введите следующие параметры:

- [ Заголовок ] ([ Title ]) — "Recipient (many recipients separated by semicolon ";")".
- [ Код ] ([ Code ]) — "Recipient".
- Установите признак [ Обязательное ] ([ Required ]).

The screenshot shows a configuration dialog for an email message element. The fields are as follows:

- Title\***: Recipient (many recipients separated by se...)
- Code\***: Recipient
- Is multiline
- Required
- Value**: (empty text area)

At the bottom right are two buttons: **SAVE** (blue) and **CANCEL**.

Нажмите [ Сохранить ] ([ Save ]).

Для добавления на страницу элемента, который будет содержать тему email-сообщения, в блоке [ Параметры страницы ] ([ Page Items ]) нажмите +, выберите тип [ Строковое поле ] ([ Text field ]) и введите следующие параметры:

- [ Заголовок ] ([ Title ]) — "Subject".
- [ Код ] ([ Code ]) — "Subject".
- Установите признак [ Обязательное ] ([ Required ]).

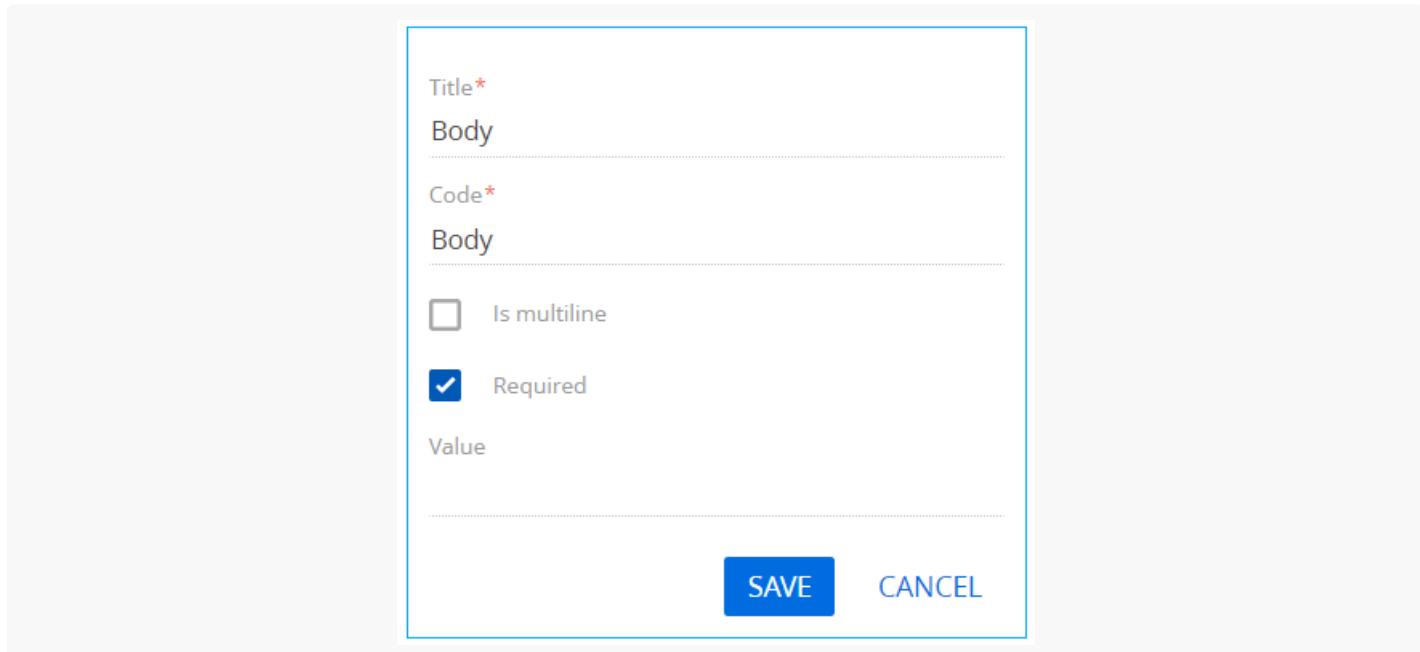
The screenshot shows the same configuration dialog as above, but with the **Title\*** field set to "Subject". The other fields remain the same.

Нажмите [ Сохранить ] ([ Save ]).

Для добавления на страницу элемента, который будет содержать тело email-сообщения, в блоке

[ Параметры страницы ] ([ Page Items ]) нажмите **+**, выберите тип [ Строковое поле ] ([ Text field ]) и введите следующие параметры:

- [ Заголовок ] ([ Title ]) — "Body".
- [ Код ] ([ Code ]) — "Body".
- Установите признак [ Обязательное ] ([ Required ]).



Нажмите [ Сохранить ] ([ Save ]).

## 5. Добавьте элемент [ Задание-сценарий ]

Свойству [ Заголовок ] ([ Title ]) элемента [ Задание-сценарий ] ([ Script task ]) присвойте значение "Send Email". Элемент должен выполнять программный код.

```
Send Email

// Id выбранного почтового ящика.
var mailBoxSettingId = Get<Guid>("SenderId");
// Создание экземпляра EmailClientFactory.
var emailClientFactory = ClassFactory.Get<IEmailClientFactory>(
    new ConstructorArgument("userConnection", UserConnection));
// Создание экземпляра IEmailSender.
var emailSender = ClassFactory.Get<IEmailSender>(
    new ConstructorArgument("emailClientFactory", emailClientFactory),
    new ConstructorArgument("userConnection", UserConnection));

var entity = UserConnection.EntitySchemaManager.GetInstanceByName("MailboxSyncSettings").CreateEntity();
if (entity.FetchFromDB("Id", mailBoxSettingId, new List<string> { "SenderId" })) {
    // Получение почтового адреса отправителя из выбранного почтового ящика.
```

```

var senderEmailAddress = entity.GetTypedColumnValue<string>("SenderEmailAddress");
// Заполнение параметров отправляемого сообщения.
var message = new Terrasoft.Mail.Sender.EmailMessage {
    // Email-адрес отправителя.
    From = senderEmailAddress,
    // Email-адреса получателей.
    To = Get<string>("Recipient").Split(';').ToList<string>(),
    // Копия (не обязательно).
    // Скрытая копия (не обязательно).
    // Bcc = List<string>{ "first@recipient.co", "second@recipient.co" },
    // Тема письма.
    Subject = Get<string>("Subject"),
    // Тело письма.
    Body = Get<string>("Body"),
    // Приоритет, значения из перечисления Terrasoft.Mail.Sender.EmailPriority.
    Priority = Terrasoft.Mail.Sender.EmailPriority.Normal
};
// Дополнительно можно прикреплять вложения (в примере используются тестовые значения).
// Создание вложения.
var attachment = new Terrasoft.Mail.Sender.EmailAttachment {
    // Идентификатор вложения.
    Id = Guid.NewGuid(),
    // Название файла.
    Name = "test.txt",
    // Данные.
    Data = Encoding.ASCII.GetBytes("some test text")
};
// Добавление вложения в письмо.
message.Attachments.Add(attachment);
// Отправка письма.
emailSender.Send(message);
}

return true;

```

## 6. Добавьте параметры

Для добавления параметра бизнес-процесса, который будет содержать почтовый ящик получателя email-сообщения, на вкладке [ Параметры ] ([ Parameters ]) панели настройки свойств процесса выполните действие [ Добавить параметр ] —> [ Текст ] ([ Add parameters ] —> [ Text ]) и установите следующие свойства параметра:

- [ Заголовок ] ([ Title ]) — "Recipient".
- [ Код ] ([ Code ]) — "Recipient".
- [ Значение ] ([ Value ]) — нажмите ⚡ —> [ Параметр процесса ] ([ Process parameter ]) и выберите элемент процесса "Recipient (many recipients separated by semicolon ";)".

Title\*

Recipient

Code\*

Recipient

Data type\*

Text (500 characters)

Value

[#Filling parameters.Recipient (many recipi...]

**SAVE**   **CANCEL**

Нажмите [ Сохранить ] ([ Save ]).

Для добавления параметра бизнес-процесса, который будет содержать тему email-сообщения, на вкладке [ Параметры ] ([ Parameters ]) панели настройки свойств процесса выполните действие [ Добавить параметр ] —> [ Текст ] ([ Add parameters ] —> [ Text ]) и установите следующие свойства параметра:

- [ Заголовок ] ([ Title ]) — "Subject".
- [ Код ] ([ Code ]) — "Subject".
- [ Значение ] ([ Value ]) — нажмите ⚡ —> [ Параметр процесса ] ([ Process parameter ]) и выберите элемент процесса "Subject".

Title\*

Subject

Code\*

Subject

Data type\*

Text (500 characters)

Value

[#Filling parameters.Subject#]

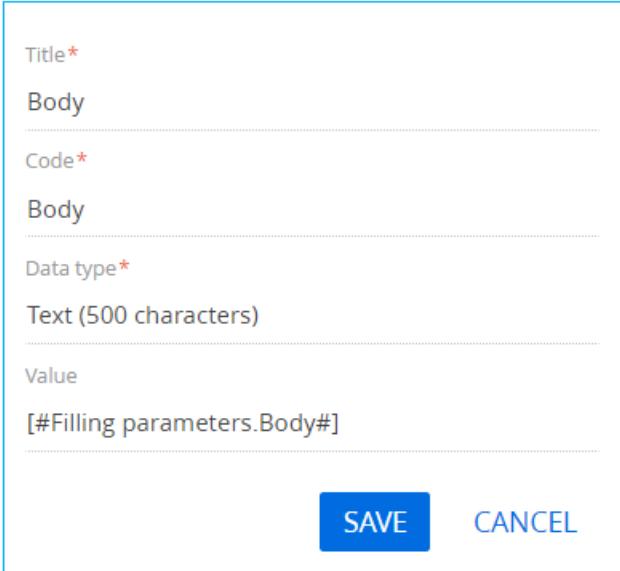
**SAVE**   **CANCEL**

Нажмите [ Сохранить ] ([ Save ]).

Для добавления параметра бизнес-процесса, который будет содержать тело email-сообщения, на

вкладке [ Параметры ] ([ Parameters ]) панели настройки свойств процесса выполните действие [ Добавить параметр ] —> [ Текст ] ([ Add parameters ] —> [ Text ]) и установите следующие свойства параметра:

- [ Заголовок ] ([ Title ]) — "Body".
- [ Код ] ([ Code ]) — "Body".
- [ Значение ] ([ Value ]) — нажмите  —> [ Параметр процесса ] ([ Process parameter ]) и выберите элемент процесса "Body".



The screenshot shows the 'Text' configuration dialog. It contains the following fields:

- Title\***: Body
- Code\***: Body
- Data type\***: Text (500 characters)
- Value**: [#Filling parameters.Body#]

At the bottom right are two buttons: **SAVE** (highlighted in blue) and **CANCEL**.

Нажмите [ Сохранить ] ([ Save ]).

Для добавления параметра бизнес-процесса, который будет содержать почтовый ящик отправителя email-сообщения, на вкладке [ Параметры ] ([ Parameters ]) панели настройки свойств процесса выполните действие [ Добавить параметр ] —> [ Другое ] —> [ Уникальный идентификатор ] ([ Add parameters ] —> [ Other ] —> [ Unique identifier ]) и установите следующие свойства параметра:

- [ Заголовок ] ([ Title ]) — "Sender Mailbox".
- [ Код ] ([ Code ]) — "SenderMailbox".
- [ Значение ] ([ Value ]) — нажмите  —> [ Параметр процесса ] ([ Process parameter ]) и выберите элемент процесса "Sender Mailbox".

The screenshot shows a configuration dialog box with the following fields:

- Title\***: Sender Mailbox
- Code\***: SenderMailbox
- Data type\***: Unique identifier
- Value**: [#Filling parameters.Sender Mailbox#]

At the bottom right are two buttons: **SAVE** (in blue) and **CANCEL**.

Нажмите [ Сохранить ] ([ Save ]).

## 7. Добавьте методы

Для добавления методов бизнес-процесса на вкладке [ Методы ] ([ Methods ]) панели настройки свойств процесса в блоке [ Usings ] нажмите **+** и в поле [ Пространство имён ] ([ Name Space ]) добавьте значение `Terrasoft.Configuration`. Нажмите [ Сохранить ] ([ Save ]).

Таким же способом добавьте следующие пространства имен:

- `Terrasoft.Mail.Sender`
- `Terrasoft.Core.Factories`
- `Terrasoft.Core`
- `Terrasoft.Mail`
- `IntegrationApi`
- `System.Linq`

Сохраните все изменения в дизайнере процессов.

## 8. Запустите бизнес-процесс

**Важно.** Для успешного запуска бизнес-процесса необходимо предварительно [добавить учетную запись отправителя в приложение Creatio](#).

После запуска бизнес-процесса по кнопке [ Запустить ] ([ Run ]) будет открыта страница для заполнения параметров email-сообщения.

Fill parameters for sending Email

[CONTINUE](#) [CLOSE](#)

---

Sender Mailbox

Recipient (many recipients separated by semicolon ";")\*

Subject\*

Body\*

Для отправки email-сообщения с существующей учетной записи нажмите [ *Continue* ].

## Отправить email-сообщение с явным указанием учетных данных



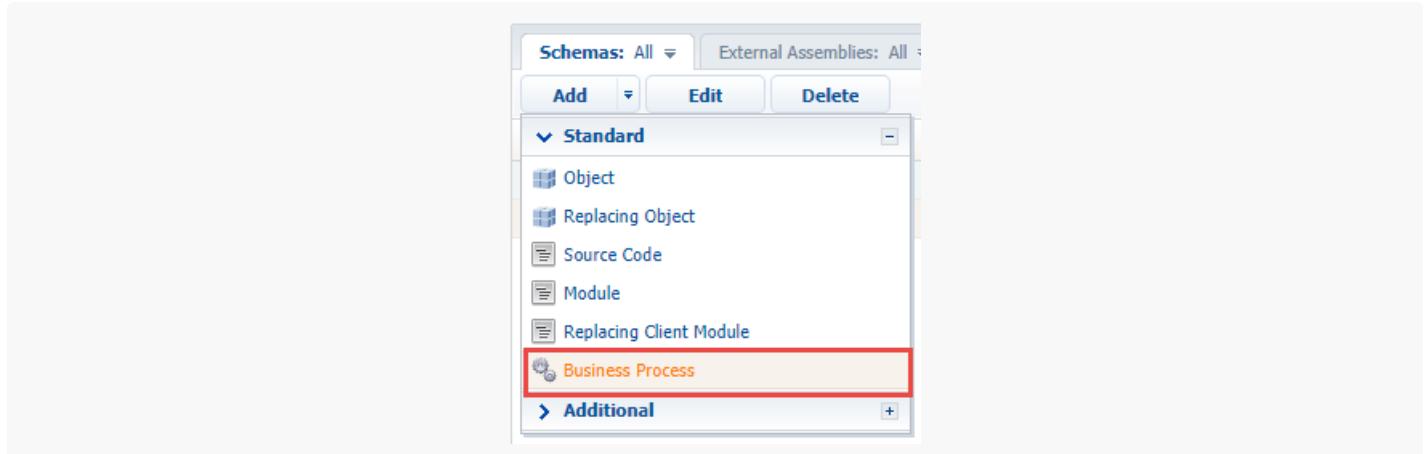
Сложный

**Пример.** Создайте бизнес-процесс в ходе которого будет открываться страница, содержащая параметры email-сообщения для заполнения и дальнейшей отправки email-сообщения с явным указанием учетных данных.

### Алгоритм реализации примера

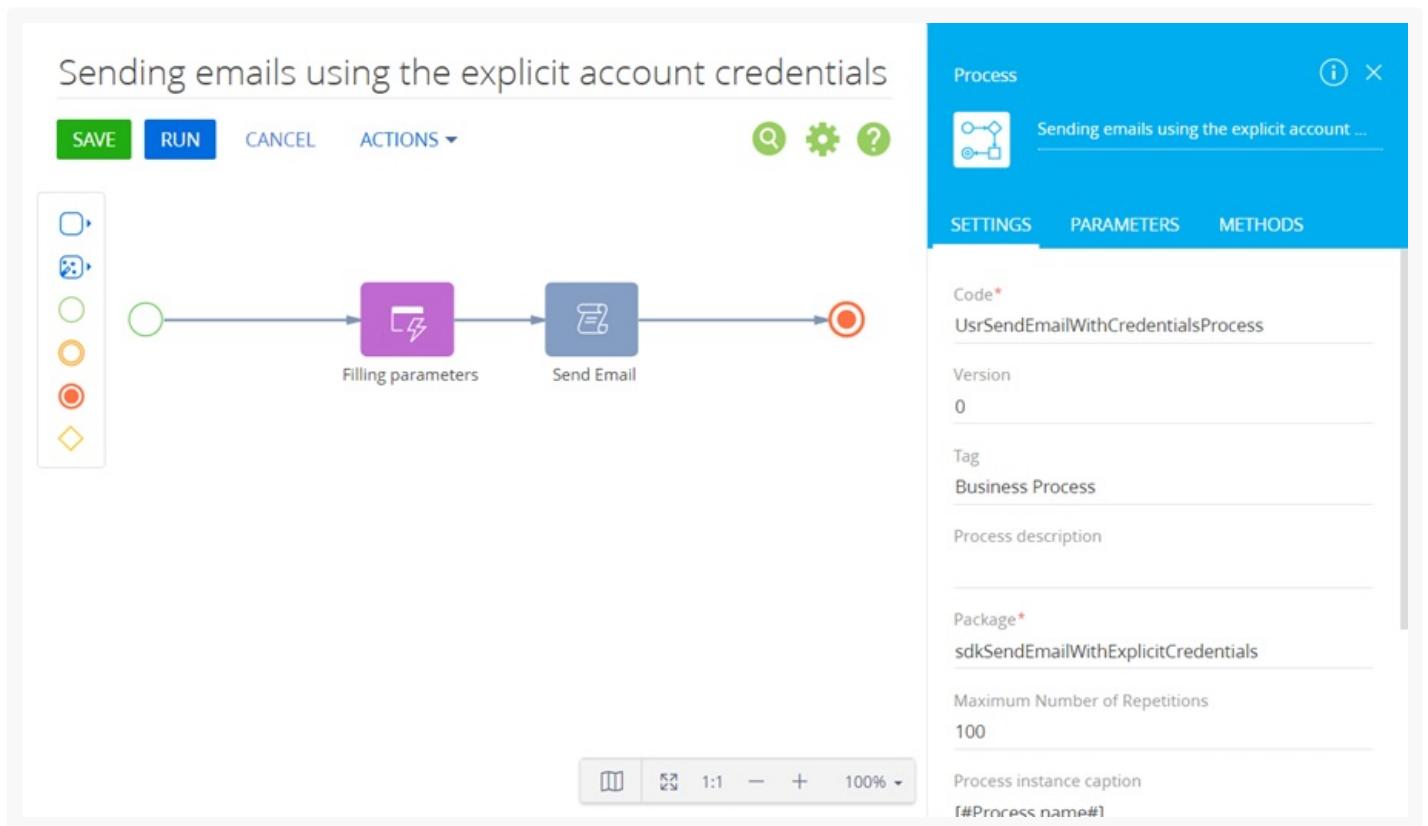
#### 1. Создайте бизнес-процесс

В разделе [ *Конфигурация* ] ([ *Configuration* ]) выполните действие [ *Добавить* ] —> [ *Бизнес-процесс* ] ([ *Add* ] —> [ *Business process* ]).



В открывшемся дизайнере процессов установите следующие значения на панели настройки свойств процесса:

- [ Заголовок ] ([ Title ]) — "Sending emails using the explicit account credentials".
- [ Код ] ([ Code ]) — "UsrSendEmailWithCredentialsProcess".

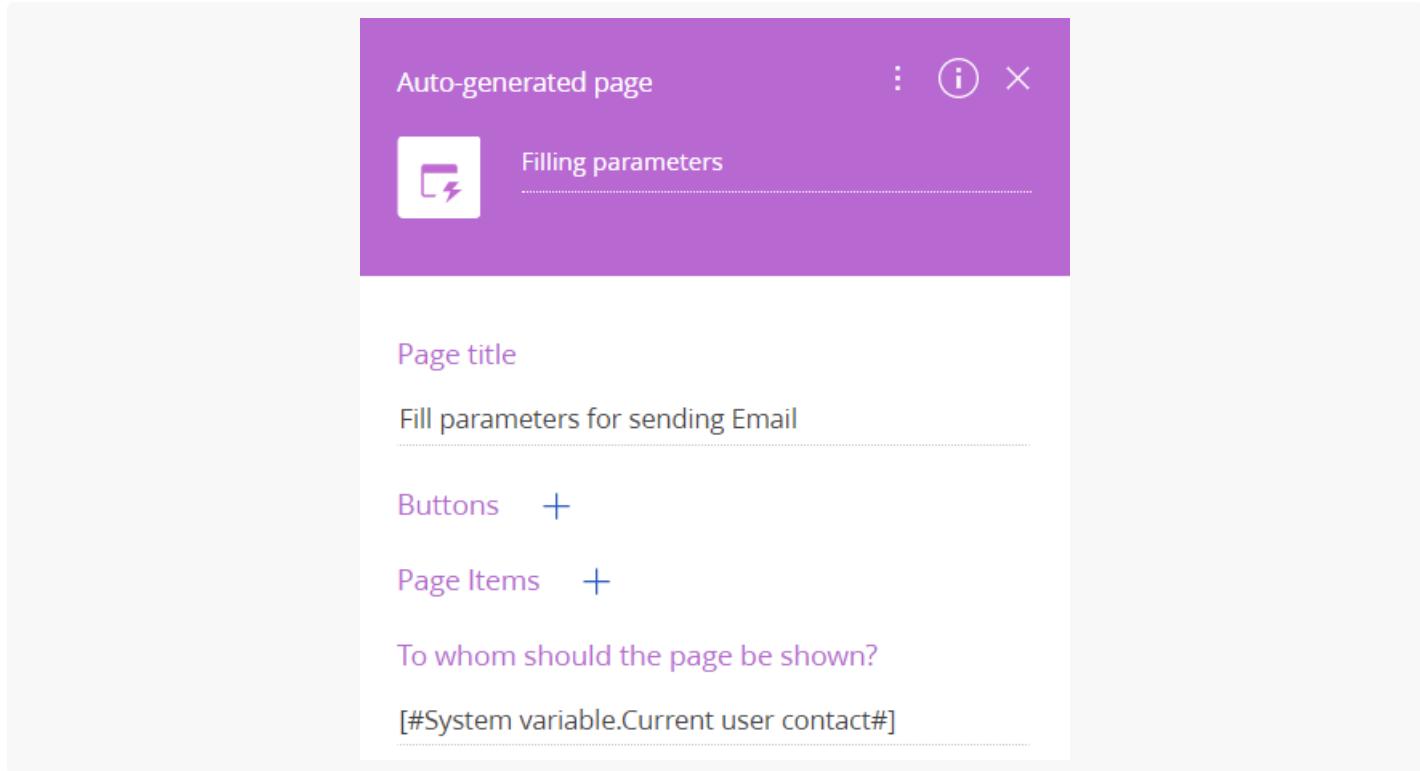


## 2. Добавьте элемент [ Автогенерируемая страница ]

С помощью элемента [ Автогенерируемая страница ] ([ Auto-generated page ]) в ходе выполнения процесса можно открыть произвольную страницу, которая создана пользователем системы. Для этого элемента добавьте подпись [ Filling parameters ] и установите следующие свойства:

- [ Название страницы ] ([ Page title ]) — "Fill parameters for sending Email".

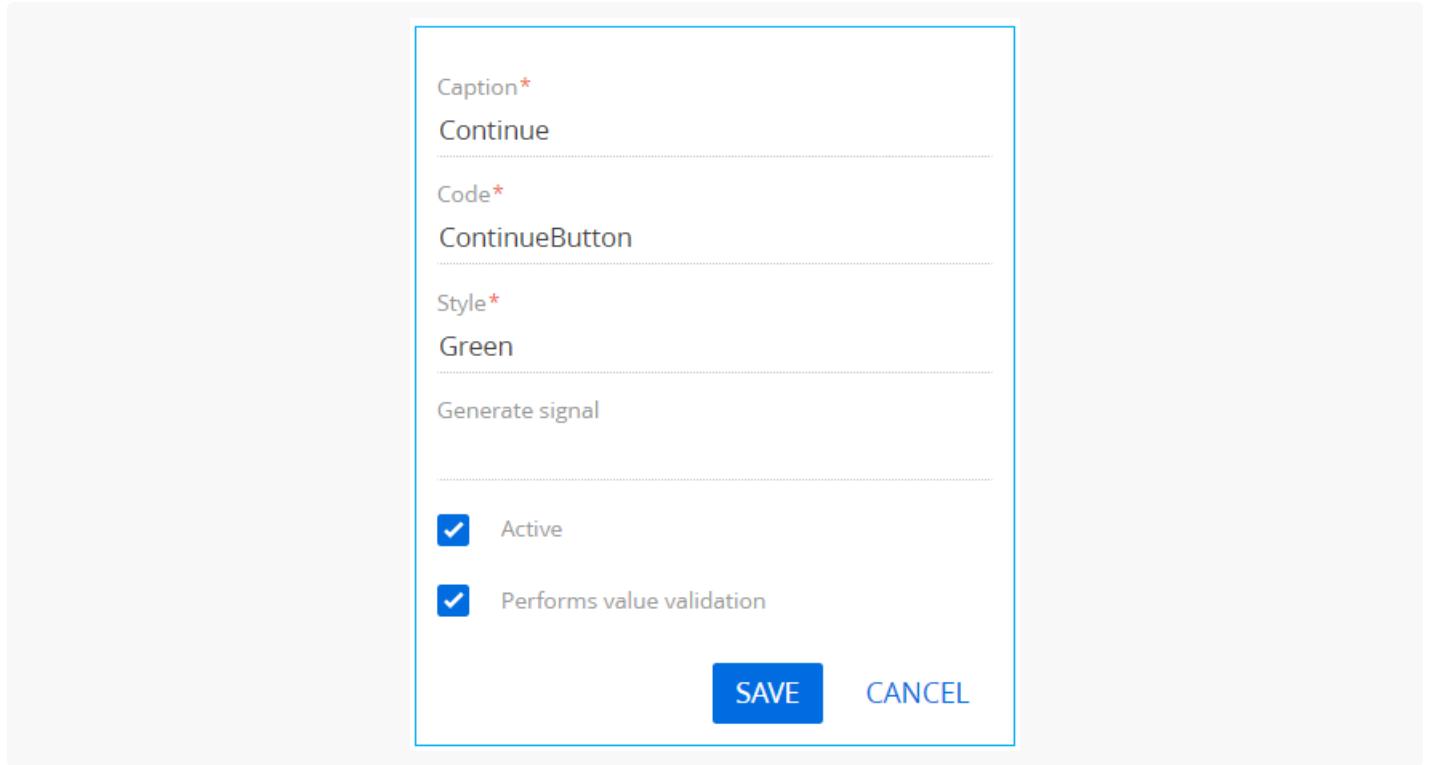
- [ Кому отобразить страницу? ] ([ To whom should the page be shown? ]) — выберите "Formula" и установите [#System variable.Current user contact#].



### 3. Добавьте кнопку на страницу

Для добавления кнопки [ Continue ] на страницу, в блоке [ Кнопки ] ([ Buttons ]) нажмите + и введите следующие параметры:

- [ Название ] ([ Caption ]) — "Continue".
- [ Код ] ([ Code ]) — "ContinueButton".
- [ Стиль ] ([ Style ]) — выберите "Green".
- Установите признак [ Активная ] ([ Active ]).
- Установите признак [ Выполняет проверку значений ] ([ Performs value validation ]).



Нажмите [ Сохранить ] ([ Save ]).

#### 4. Добавьте элементы на страницу

Для добавления на страницу элемента, который будет содержать почтовый ящик отправителя email-сообщения, в блоке [ Параметры страницы ] ([ Page Items ]) нажмите **+**, выберите тип [ Строковое поле ] ([ Text field ]) и введите следующие параметры:

- [ Заголовок ] ([ Title ]) — "Sender Mailbox".
- [ Код ] ([ Code ]) — "SenderMailbox".
- Установите признак [ Обязательное ] ([ Required ]).

Page Items +

Title\*  
Sender Mailbox

Code\*  
SenderMailbox

Is multiline

Required

Value

SAVE CANCEL

Нажмите [ Сохранить ] ([ Save ]).

Для добавления на страницу элемента, который будет содержать имя отправителя email-сообщения, в блоке [ Параметры страницы ] ([ Page Items ]) нажмите +, выберите тип [ Строковое поле ] ([ Text field ]) и введите следующие параметры:

- [ Заголовок ] ([ Title ]) — "User Name".
- [ Код ] ([ Code ]) — "UserName".
- Установите признак [ Обязательное ] ([ Required ]).

Title\*  
User Name

Code\*  
UserName

Is multiline

Required

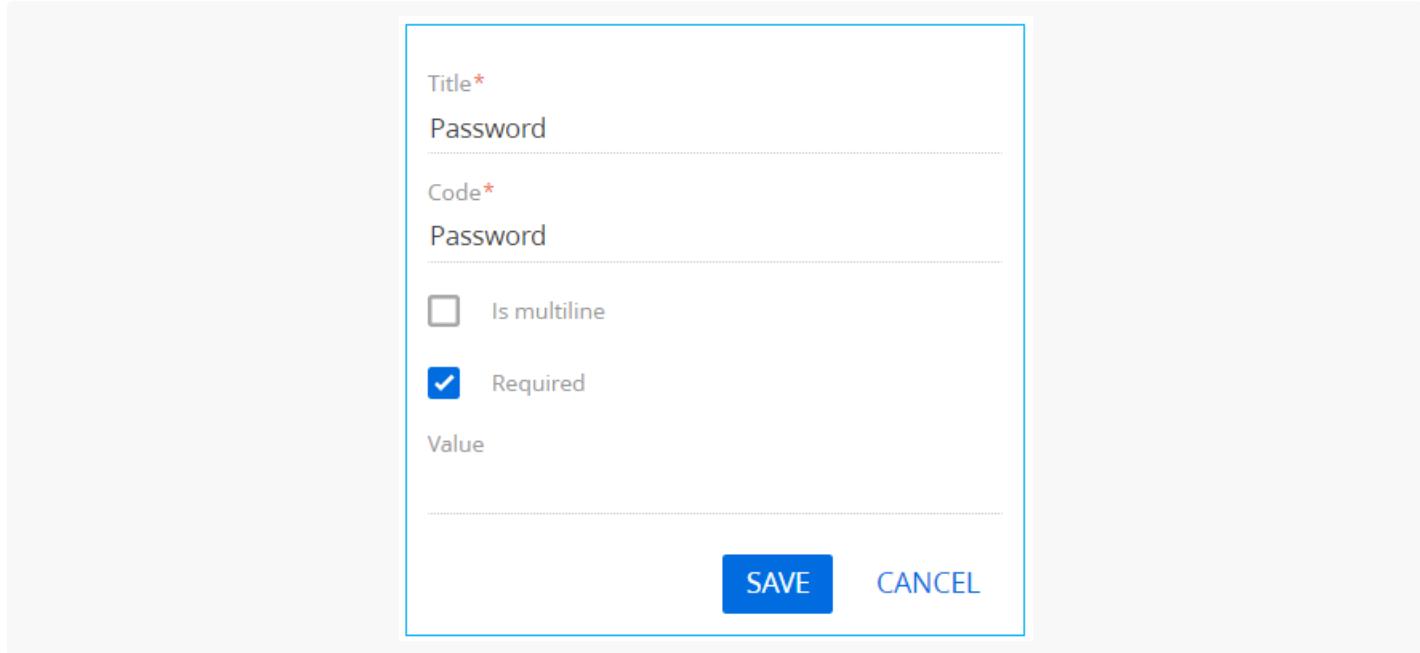
Value

SAVE CANCEL

Нажмите [ Сохранить ] ([ Save ]).

Для добавления на страницу элемента, который будет содержать пароль к почтовому ящику отправителя email-сообщения, в блоке [ Параметры страницы ] ([ Page Items ]) нажмите **+**, выберите тип [ Строковое поле ] ([ Text field ]) и введите следующие параметры:

- [ Заголовок ] ([ Title ]) — "Password".
- [ Код ] ([ Code ]) — "Password".
- Установите признак [ Обязательное ] ([ Required ]).



Нажмите [ Сохранить ] ([ Save ]).

Для добавления на страницу элемента, который будет содержать адрес почтового сервера отправителя email-сообщения, в блоке [ Параметры страницы ] ([ Page Items ]) нажмите **+**, выберите тип [ Строковое поле ] ([ Text field ]) и введите следующие параметры:

- [ Заголовок ] ([ Title ]) — "Service Url".
- [ Код ] ([ Code ]) — "ServiceUrl".
- Установите признак [ Обязательное ] ([ Required ]).

Title\*  
Service Url

Code\*  
ServiceUrl

Is multiline

Required

Value

SAVE CANCEL

Нажмите [ Сохранить ] ([ Save ]).

Для добавления на страницу элемента, который будет содержать номер порта почтового провайдера отправителя email-сообщения, в блоке [ Параметры страницы ] ([ Page Items ]) нажмите +, выберите тип [ Целое число ] ([ Integer ]) и введите следующие параметры:

- [ Заголовок ] ([ Title ]) — "Port".
- [ Код ] ([ Code ]) — "Port".

Title\*  
Port

Code\*  
Port

Required

Value

SAVE CANCEL

Нажмите [ Сохранить ] ([ Save ]).

Для добавления на страницу элемента, который будет содержать криптографический протокол для обеспечения безопасной связи, в блоке [ Параметры страницы ] ([ Page Items ]) нажмите +, выберите тип [ Логическое ] ([ Boolean ]) и введите следующие параметры:

- [ Заголовок ] ([ Title ]) — "Use Ssl".
- [ Код ] ([ Code ]) — "UseSsl".

The screenshot shows a configuration dialog for a page item. It contains three input fields:

- Title\***: Value is "Use Ssl".
- Code\***: Value is "UseSsl".
- Value**: This field is empty.

At the bottom of the dialog are two buttons: "SAVE" (in blue) and "CANCEL".

Нажмите [ Сохранить ] ([ Save ]).

Для добавления на страницу элемента, который будет содержать почтовый ящик получателя email-сообщения, в блоке [ Параметры страницы ] ([ Page Items ]) нажмите **+**, выберите тип [ Строковое поле ] ([ Text field ]) и введите следующие параметры:

- [ Заголовок ] ([ Title ]) — "Recipient (many recipients separated by semicolon ';')".
- [ Код ] ([ Code ]) — "Recipient".
- Установите признак [ Обязательное ] ([ Required ]).

The screenshot shows a configuration dialog for a page item. It contains four input fields:

- Title\***: Value is "Recipient (many recipients separated by se...)".
- Code\***: Value is "Recipient".
- Required**: This checkbox is checked.
- Value**: This field is empty.

At the bottom of the dialog are two buttons: "SAVE" (in blue) and "CANCEL".

Нажмите [ Сохранить ] ([ Save ]).

Для добавления на страницу элемента, который будет содержать тему email-сообщения, в блоке

[ Параметры страницы ] ([ Page Items ]) нажмите **+**, выберите тип [ Строковое поле ] ([ Text field ]) и введите следующие параметры:

- [ Заголовок ] ([ Title ]) — "Subject".
- [ Код ] ([ Code ]) — "Subject".
- Установите признак [ Обязательное ] ([ Required ]).

The screenshot shows a configuration dialog for a page item. The item is titled 'Subject' and has the code 'Subject'. It is set to be required (indicated by a checked checkbox) and is not multiline (indicated by an unchecked checkbox). The 'Value' field is empty. At the bottom are 'SAVE' and 'CANCEL' buttons.

Нажмите [ Сохранить ] ([ Save ]).

Для добавления на страницу элемента, который будет содержать тело email-сообщения, в блоке [ Параметры страницы ] ([ Page Items ]) нажмите **+**, выберите тип [ Строковое поле ] ([ Text field ]) и введите следующие параметры:

- [ Заголовок ] ([ Title ]) — "Body".
- [ Код ] ([ Code ]) — "Body".
- Установите признак [ Обязательное ] ([ Required ]).

The screenshot shows a configuration dialog box with a light gray background and a blue border. Inside, there are two sections for 'Title' and 'Code', each with a 'Body' input field. Below these is a checkbox labeled 'Is multiline' which is unchecked. Underneath is another checkbox labeled 'Required' which is checked. At the bottom right are two buttons: a blue 'SAVE' button and a light blue 'CANCEL' button.

Нажмите [ Сохранить ] ([ Save ]).

Для добавления на страницу элемента, который будет содержать тип почтового провайдера отправителя email-сообщения, в блоке [ Параметры страницы ] ([ Page Items ]) нажмите **+**, выберите тип [ Справочник ] ([ Selection field ]) и введите следующие параметры:

- [ Заголовок ] ([ Title ]) — "Type of mail server".
- [ Код ] ([ Code ]) — "ServerTypeId".
- Установите признак [ Обязательное ] ([ Required ]).
- [ Источник данных ] ([ Data source ]) — выберите "Mail service provider type".
- [ Представление ] ([ View ]) — выберите "Drop down list".

The screenshot shows a configuration dialog box with the following fields:

- Title\***: Type of mail server
- Code\***: ServerTypeId
- Required**: Checked (indicated by a blue checkmark)
- Data source\***: Mail service provider type
- View\***: Drop down list
- Value**: (empty field)

At the bottom right are two buttons: **SAVE** (blue) and **CANCEL**.

Нажмите [ Сохранить ] ([ Save ]).

## 5. Добавьте элемент [ Задание-сценарий ]

Свойству [ Заголовок ] ([ Title ]) элемента [ Задание-сценарий ] ([ Script task ]) присвойте значение "Send Email". Элемент должен выполнять программный код.

```
Send Email

// Создание экземпляра EmailClientFactory.
var emailClientFactory = ClassFactory.Get<EmailClientFactory>(new ConstructorArgument("userConnection"));
// Установка параметров подключения к почтовому сервису.
var credentialConfig = new EmailContract.DTO.Credentials {
    ServiceUrl = Get<string>("ServiceUrl"),
    Port = Get<int>("Port"),
    UseSsl = Get<bool>("UseSsl"),
    UserName = Get<string>("UserName"),
    Password = Get<string>("Password"),
    ServerTypeId = Get<Guid>("ServerTypeId"),
    SenderEmailAddress = Get<string>("SenderMailbox")
};
// Создание экземпляра IEmailSender.
var emailSender = ClassFactory.Get<IEmailSender>(new ConstructorArgument("emailClientFactory", emailClientFactory),
    new ConstructorArgument("userConnection", UserConnection));
// Установка параметров отправляемого сообщения.
var message = new EmailContract.DTO.Email {
```

```

    Sender = credentialConfig.SenderEmailAddress,
    Recipients = Get<string>("Recipient").Split(';').ToList<string>(),
    Subject = Get<string>("Subject"),
    Body = Get<string>("Body"),
    Importance = EmailContract.EmailImportance.Normal,
    // Если тело письма было сформировано в формате HTML.
    IsHtmlBody = true,
    // Дополнительно можно указать список получателей в копии, в т.ч. скрытых
    // CopyRecipients = new List<string> { "user@mail.service" },
    // BlindCopyRecipients = new List<string> { "user@mail.service" }
};

// Создание вложения.
var attachment = new EmailContract.DTO.Attachment {
    // Идентификатор вложения.
    Id = Guid.NewGuid().ToString(),
    // Название файла.
    Name = "test.txt",
};

// Данные (используется тестовое значение данных в виде текста).
byte[] data = Encoding.ASCII.GetBytes("some test text");
// Добавление данных во вложение.
attachment.SetData(data);
// Добавление вложения в сообщение.
message.Attachments.Add(attachment);
// Отправка email-сообщения.
emailSender.Send(message, credentialConfig);
return true;

```

## 6. Добавьте параметры

Для добавления параметра бизнес-процесса, который будет содержать адрес почтового сервера отправителя email-сообщения, на вкладке [ Параметры ] ([ Parameters ]) панели настройки свойств процесса выполните действие [ Добавить параметр ] —> [ Текст ] ([ Add parameters ] —> [ Text ]) и установите следующие свойства параметра:

- [ Заголовок ] ([ Title ]) — "Service Url".
- [ Код ] ([ Code ]) — "ServiceUrl".
- [ Значение ] ([ Value ]) — нажмите  —> [ Параметр процесса ] ([ Process parameter ]) и выберите элемент процесса "Service Url".

Title\*

Service Url

Code\*

ServiceUrl

Data type\*

Text (500 characters)

Value

[#Filling parameters.Service Url#]

**SAVE**   **CANCEL**

Нажмите [ Сохранить ] ([ Save ]).

Для добавления параметра бизнес-процесса, который будет содержать номер порта почтового провайдера отправителя email-сообщения, на вкладке [ Параметры ] ([ Parameters ]) панели настройки свойств процесса выполните действие [ Добавить параметр ] —> [ Целое число ] ([ Add parameters ] —> [ Integer ]) и установите следующие свойства параметра:

- [ Заголовок ] ([ Title ]) — "Port".
- [ Код ] ([ Code ]) — "Port".
- [ Значение ] ([ Value ]) — нажмите —> [ Параметр процесса ] ([ Process parameter ]) и выберите элемент процесса "Port".

Title\*

Port

Code\*

Port

Data type\*

Integer

Value

[#Filling parameters.Port#]

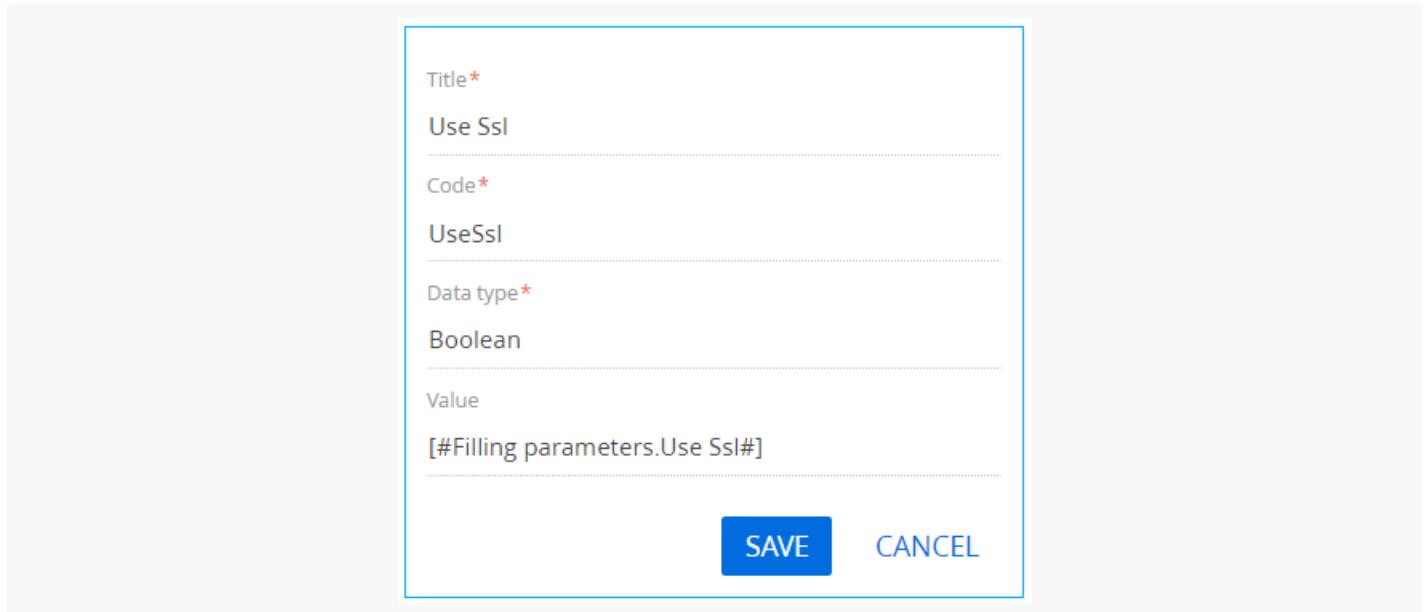
**SAVE**   **CANCEL**

Нажмите [ Сохранить ] ([ Save ]).

Для добавления параметра бизнес-процесса, который будет содержать криптографический протокол

для обеспечения безопасной связи, на вкладке [ Параметры ] ([ Parameters ]) панели настройки свойств процесса выполните действие [ Добавить параметр ] —> [ Логическое ] ([ Add parameters ] —> [ Boolean ]) и установите следующие свойства параметра:

- [ Заголовок ] ([ Title ]) — "Use Ssl".
- [ Код ] ([ Code ]) — "Use Ssl".
- [ Значение ] ([ Value ]) — нажмите  —> [ Параметр процесса ] ([ Process parameter ]) и выберите элемент процесса "Use Ssl".



Нажмите [ Сохранить ] ([ Save ]).

Для добавления параметра бизнес-процесса, который будет содержать имя отправителя email-сообщения, на вкладке [ Параметры ] ([ Parameters ]) панели настройки свойств процесса выполните действие [ Добавить параметр ] —> [ Текст ] ([ Add parameters ] —> [ Text ]) и установите следующие свойства параметра:

- [ Заголовок ] ([ Title ]) — "User Name".
- [ Код ] ([ Code ]) — "UserName".
- [ Тип данных ] ([ Data type ]) — выберите "Text (250 characters)".
- [ Значение ] ([ Value ]) — нажмите  —> [ Параметр процесса ] ([ Process parameter ]) и выберите элемент процесса "User Name".

The dialog box contains the following fields:

- Title\***: User Name
- Code\***: UserName
- Data type\***: Text (250 characters)
- Value**: [#Filling parameters.User Name#]

At the bottom are two buttons: **SAVE** (highlighted in blue) and **CANCEL**.

Нажмите [ Сохранить ] ([ Save ]).

Для добавления параметра бизнес-процесса, который будет содержать пароль к почтовому ящику отправителя email-сообщения, на вкладке [ Параметры ] ([ Parameters ]) панели настройки свойств процесса выполните действие [ Добавить параметр ] —> [ Текст ] ([ Add parameters ] —> [ Text ]) и установите следующие свойства параметра:

- [ Заголовок ] ([ Title ]) — "Password".
- [ Код ] ([ Code ]) — "Password".
- [ Тип данных ] ([ Data type ]) — выберите "Text (250 characters)".
- [ Значение ] ([ Value ]) — нажмите —> [ Параметр процесса ] ([ Process parameter ]) и выберите элемент процесса "Password".

The dialog box contains the following fields:

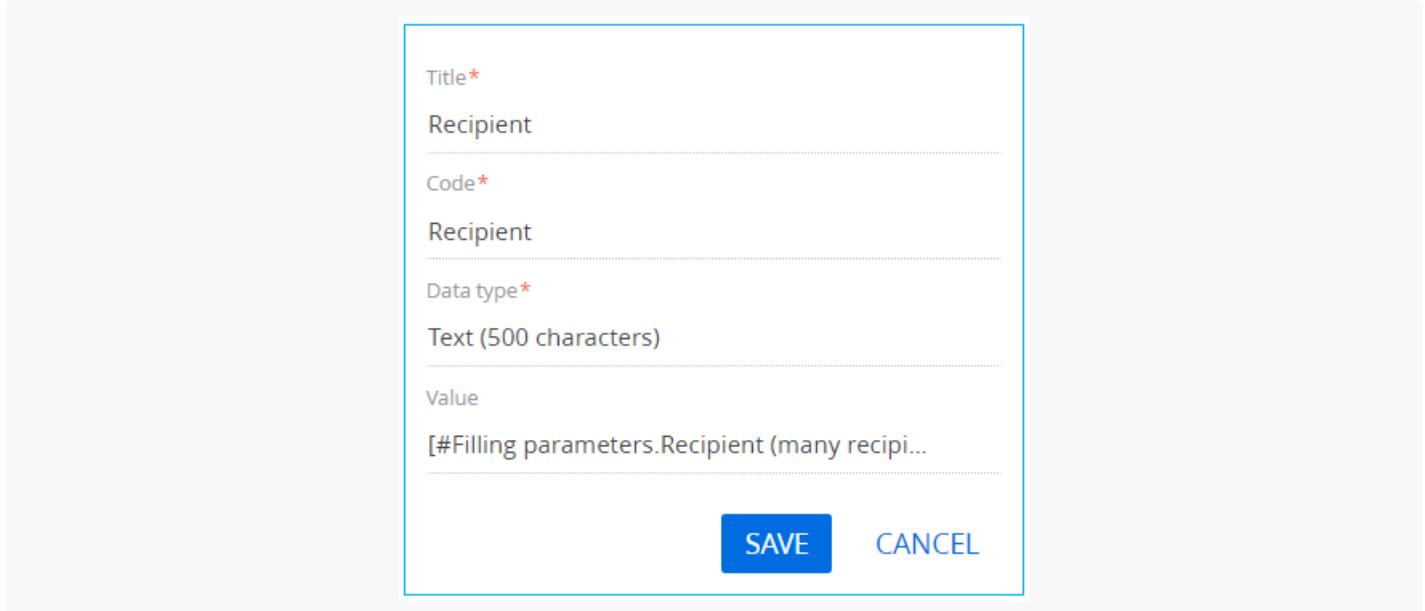
- Title\***: Password
- Code\***: Password
- Data type\***: Text (250 characters)
- Value**: [#Filling parameters.Password#]

At the bottom are two buttons: **SAVE** (highlighted in blue) and **CANCEL**.

Нажмите [ Сохранить ] ([ Save ]).

Для добавления параметра бизнес-процесса, который будет содержать почтовый ящик получателя email-сообщения, на вкладке [ Параметры ] ([ Parameters ]) панели настройки свойств процесса выполните действие [ Добавить параметр ] —> [ Текст ] ([ Add parameters ] —> [ Text ]) и установите следующие свойства параметра:

- [ Заголовок ] ([ Title ]) — "Recipient".
- [ Код ] ([ Code ]) — "Recipient".
- [ Значение ] ([ Value ]) — нажмите  —> [ Параметр процесса ] ([ Process parameter ]) и выберите элемент процесса "Recipient (many recipients separated by semicolon ";" )".



Нажмите [ Сохранить ] ([ Save ]).

Для добавления параметра бизнес-процесса, который будет содержать тип почтового провайдера отправителя email-сообщения, на вкладке [ Параметры ] ([ Parameters ]) панели настройки свойств процесса выполните действие [ Добавить параметр ] —> [ Другое ] —> [ Уникальный идентификатор ] ([ Add parameters ] —> [ Other ] —> [ Unique identifier ]) и установите следующие свойства параметра:

- [ Заголовок ] ([ Title ]) — "Type of mail server".
- [ Код ] ([ Code ]) — "ServerTypeId".
- [ Значение ] ([ Value ]) — нажмите  —> [ Параметр процесса ] ([ Process parameter ]) и выберите элемент процесса "Type of mail server".

Title\*  
Type of mail server

Code\*  
ServerTypeid

Data type\*  
Unique identifier

Value  
[#Filling parameters.Type of mail server#]

**SAVE**   **CANCEL**

Нажмите [ Сохранить ] ([ Save ]).

Для добавления параметра бизнес-процесса, который будет содержать почтовый ящик отправителя email-сообщения, на вкладке [ Параметры ] ([ Parameters ]) панели настройки свойств процесса выполните действие [ Добавить параметр ] —> [ Текст ] ([ Add parameters ] —> [ Text ]) и установите следующие свойства параметра:

- [ Заголовок ] ([ Title ]) — "Sender Mailbox".
- [ Код ] ([ Code ]) — "SenderMailbox".
- [ Тип данных ] ([ Data type ]) — выберите "Text (250 characters)".
- [ Значение ] ([ Value ]) — нажмите —> [ Параметр процесса ] ([ Process parameter ]) и выберите элемент процесса "Sender Mailbox".

Title\*  
Sender Mailbox

Code\*  
SendMailbox

Data type\*  
Text (250 characters)

Value  
[#Filling parameters.Sender Mailbox#]

**SAVE**   **CANCEL**

Нажмите [ Сохранить ] ([ Save ]).

## 7. Добавьте методы

Для добавления методов бизнес-процесса на вкладке [ *Методы* ] ([ *Methods* ]) панели настройки свойств процесса в блоке [ *Usings* ] нажмите и в поле [ *Пространство имён* ] ([ *Name Space* ]) добавьте значение `Terrasoft.Mail.Sender`. Нажмите [ *Сохранить* ] ([ *Save* ]).

Таким же способом добавьте следующие пространства имен:

- `Terrasoft.Core.Factories`
- `System.Linq`

Сохраните все изменения в дизайнере процессов.

## 8. Запуск бизнес-процесса

После запуска бизнес-процесса по кнопке [ *Запустить* ] ([ *Run* ]) будет открыта страница для заполнения параметров email-сообщения.

Fill parameters for sending Email

**CONTINUE**    **CLOSE**

Sender Mailbox\*

User Name\*

Password\*

Service Url\*

Port 0

Use Ssl

Recipient (many recipients separated by semicolon ";")\*

Subject\*

Body\*

Type of mail server\*

Для отправки email-сообщения с явным указанием учетных данных нажмите [ *Continue* ].

# Web-To-Object



**Web-to-Object** — это механизм реализации простых односторонних интеграций с Creatio. С его помощью можно создавать записи в разделах Creatio (лиды, обращения, заказы и т. д.), просто отправив необходимые данные сервису Web-to-Object.

Наиболее распространенные **случаи использования** сервиса Web-to-Object:

- Интеграция Creatio с пользовательскими лендингами и веб-формами. Вызов сервиса выполняется из лендинга (особым образом настроенная пользовательская страница со специальной web-формой), после отправки заполненной web-формы посетителем.
- Интеграция с внешними системами, в результате работы которых должны создаваться объекты Creatio.

С помощью функциональности Web-to-Object можно настроить регистрацию в Creatio практически любых объектов. Например, в Creatio может быть зарегистрирован лид, контакт, заказ, обращение.

Для работы с лендингами в Creatio предусмотрен раздел [ *Лендинги и web-формы* ]. Этот раздел входит во все продукты Creatio, однако он может быть не включен по умолчанию в рабочие места некоторых продуктов (например, данный раздел не включен в рабочие места линейки продуктов Sales Creatio).

Каждая запись раздела [ *Лендинги и web-формы* ] соответствует определенному лендингу. На странице записи предусмотрена вкладка [ *Настройка лендинга* ].

## Реализация сервиса Web-to-Object

Основная функциональность механизма Web-To-Object содержится в пакете `WebForm` и является общей для всех продуктов. В зависимости от продукта Creatio эта функциональность расширена механизмами `Web-to-Lead` (пакет `WebLeadForm`), `Web-to-Order` (пакет `WebOrderForm`) и [`Web-to-Case`](#) (пакет `WebCaseForm`).

Для обработки данных, отправленных web-формой лендинга, в пакете `WebForm` реализован конфигурационный сервис `GeneratedObjectWebFormService` (класс

`Terrasoft.Configuration.GeneratedWebService`). Данные веб-формы лендинга принимаются в качестве аргумента метода `public string SaveWebFormObjectData(FormData formData)`. Затем они передаются в метод `public void HandleForm(FormData formData)` Класса `Terrasoft.Configuration.WebFormHandler`, в котором и выполняется создание соответствующего объекта системы.

## Внешний API сервиса Web-to-Object

**Для использования сервиса необходимо отправить POST-запрос по адресу**

[Путь к приложению *Creatio*]/0/ServiceModel/GeneratedObjectWebFormService.svc/SaveWebFormObjectDa

**Например**

`http://mycreatio.com/0/ServiceModel/GeneratedObjectWebFormService.svc/SaveWebFormObjectData`

Тип содержимого запроса — application/json. Кроме необходимых cookies, в содержимое запроса нужно добавить JSON-объект, содержащий данные веб-формы. Пример JSON-объекта:

```
{
    "formData": {
        "formId": "d66ebbf6-f588-4130-9f0b-0ba3414dafb8",
        "formFieldsData": [
            {"name": "Name", "value": "John Smith"},
            {"name": "Email", "value": "j.smith@creatio.com"},
            {"name": "Zip", "value": "00000"},
            {"name": "MobilePhone", "value": "0123456789"},
            {"name": "Company", "value": "Creatio"},
            {"name": "Industry", "value": ""},
            {"name": "FullJobTitle", "value": "Sales Manager"},
            {"name": "UseEmail", "value": ""},
            {"name": "City", "value": "Boston"},
            {"name": "Country", "value": "USA"},
            {"name": "Commentary", "value": ""},
            {"name": "BpmHref", "value": "http://localhost/Landing/"},
            {"name": "BpmSessionId", "value": "0ca32d6d-5d60-9444-ec34-5591514b27a3"}
        ]
    }
}
```

## Интеграция с внешними системами

Для интеграции с внешними системами необходимо:

- Добавить запись в разделе [Лендинги и web-формы].
- Получить из конфигурационного объекта созданной записи адрес к сервису (свойство `serviceUrl`) и идентификатор (свойство `landingId`).
- Реализовать во внешней системе отправку POST-запроса к сервису Web-to-Object по полученному адресу. В запрос добавить необходимые данные в виде JSON-объекта. Свойству `formId` отправляемого JSON-объекта установить значение полученного идентификатора.

# Настроить web-форму для создания пользовательского объекта



Используя web-форму посадочной страницы (лендинга) стороннего сайта можно создать пользовательский объект в приложении Creatio. Подробнее о лендингах можно узнать из блока статей ["Раздел \[Лендинги и web-формы\]"](#).

Общий порядок действий при создании пользовательского объекта через web-форму:

1. Зарегистрировать новый тип лендинга.
2. Добавить страницу записи web-формы.
3. Связать новый тип лендинга с созданной страницей записи.
4. Актуализировать наполнение скриптами для страницы web-формы.
5. Создать и настроить лендинг раздела **[Лендинги и web-формы]** (**[Landing pages and web forms]**).
6. Развернуть и настроить посадочную страницу, содержащую web-форму.
7. Зарегистрировать пользовательский объект в справочнике **[Настройки объекта для элемента кампании "Лендинг"]** (**[Entity settings for campaign landing element]**).

**Важно.** Трекинг событий сайта работает только для лидов. Для пользовательских объектов трекинг событий сайта не работает.

**Пример.** Через web-форму посадочной страницы создайте пользовательский объект [ Контакт ] ([ Contact ]).

## 1. Зарегистрируйте новый тип лендинга

Для регистрации нового типа лендинга выполните следующие действия:

1. Перейдите в дизайнер системы по кнопке . В блоке **[Настройка системы]** (**[System setup]**) перейдите по ссылке **[Справочники]** (**[Lookups]**).
2. Выберите справочник **[Типы лендингов]** (**[Landing types]**).
3. Создайте новую запись.

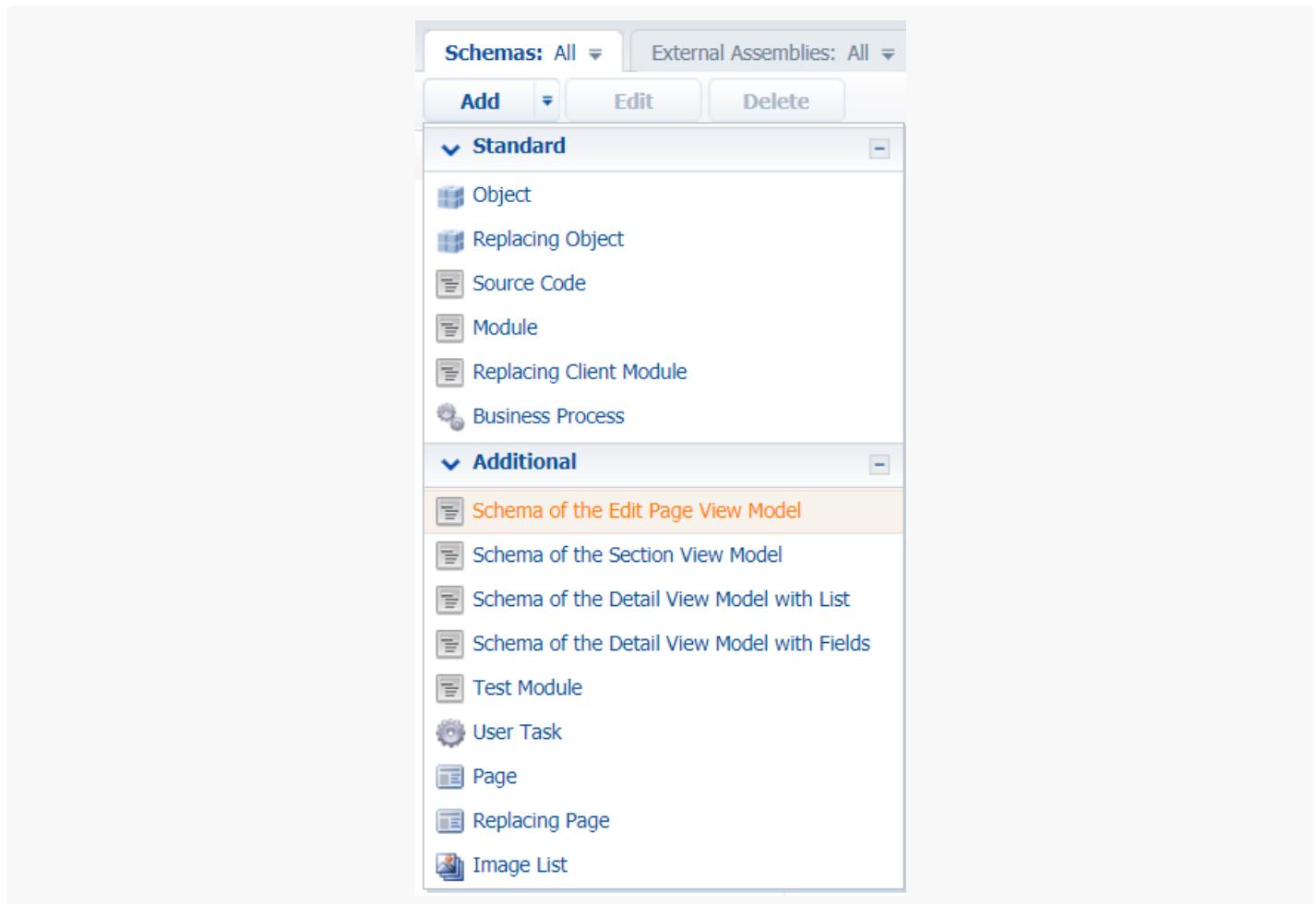
Для создаваемой записи установите:

- **[Название]** (**[Name]**) — "Contact";
- **[Объект]** (**[Object]**) — "Contact".

Name	Description	Object	Order
Contact		Contact	
Case		Case	
Event participant		Event participant	
Lead		Lead	

## 2. Добавьте страницу web-формы

В разделе [ Конфигурация ] ([ Configuration ]) пользовательского пакета на вкладке [ Схемы ] ([ Schemas ]) выполните действие [ Добавить ] —> [ Схема модели представления карточки ] ([ Add ] —> [ Schema of the Edit Page View Module ]). Процесс создания схемы модели представления карточки описан в статье "[Создать клиентскую схему](#)".



Для создаваемой схемы модели представления карточки установите:

- **[Заголовок] ([Title])** — "ContactGeneratedWebFormPage";
- **[Название] ([Name])** — "UsrContactGeneratedWebFormPage";
- **[Родительский объект] ([Parent object])** — "Edit page, landing".

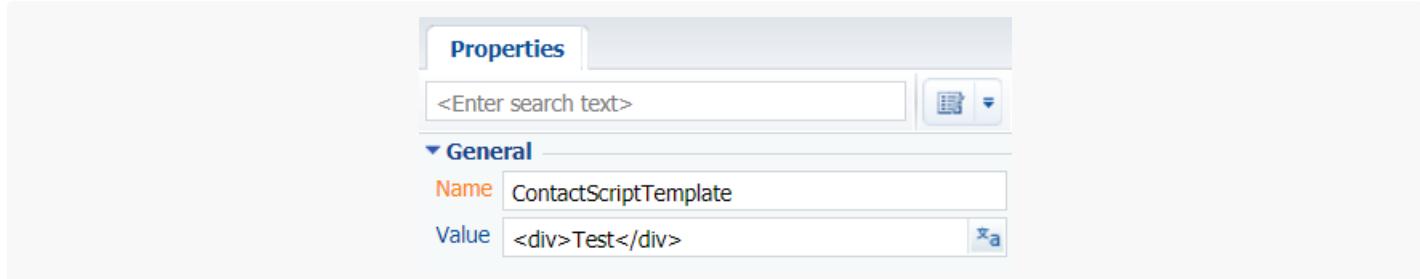
Properties	
<Enter search text>	
<b>General</b> Title: ContactGeneratedWebFormPage Name: UsrContactGeneratedWebFormPage Package: sdkWebFormPackage	
<b>Inheritance</b> Parent object: Edit page, landing ( WebForm ) <input type="checkbox"/> Replace parent	

### UsrContactGeneratedWebFormPage.js

```
// UsrContactGeneratedWebFormPage – уникальное название схемы.
define("UsrContactGeneratedWebFormPage", ["UsrContactGeneratedWebFormPageResources"],
function() {
    return {
        details: /**SCHEMA_DETAILS*/{}/**SCHEMA_DETAILS*/,
        methods: {
            /**
             * @inheritDoc BaseGeneratedWebFormPageV2#getScriptTemplateFromResources
             * @overridden
             */
            getScriptTemplateFromResources: function() {
                var scriptTemplate;
                if (this.getIsFeatureEnabled("OutboundCampaign")) {
                    // ContactScriptTemplate – имя локализуемой строки.
                    scriptTemplate = this.get("Resources.Strings.ContactScriptTemplate");
                } else {
                    scriptTemplate = this.get("Resources.Strings.ScriptTemplate");
                }
                return scriptTemplate;
            }
        },
        diff: /**SCHEMA_DIFF*/[]/**SCHEMA_DIFF*/
    };
});
```

После внесения изменений сохраните схему.

Добавьте локализируемую строку `ContactScriptTemplate`. В значение строки добавьте `<div>Test</div>`. Больше информации о работе с локализуемыми строками содержится в статье "[Работа с локализуемыми ресурсами](#)".



После внесения изменений сохраните схему.

### 3. Свяжите новый тип лендинга с созданной страницей записи

Чтобы связать новый тип лендинга с созданной страницей записи, добавьте запись в таблицу **[dbo.SysModuleEdit]** базы данных. Для этого выполните следующий SQL-запрос:

## Запрос для добавления записи в таблицу dbo.SysModuleEdit

```
-- Parameters of new landing page
DECLARE @editPageName nvarchar(250) = N'UsrContactGeneratedWebFormPage'; -- название созданной с
DECLARE @landingTypeName NVARCHAR(250) = N'Contact'; -- название типа лендинга
DECLARE @actionCaption NVARCHAR(250) = N'Contact form'; -- название типа лендинга в разделе при

-- Set system parameters based on new landing page
DECLARE @generatedWebFormEntityUID uniqueidentifier = '41AE7D8D-BEC3-41DF-A6F0-2AB0D08B3967';
DECLARE @cardSchemaUID uniqueidentifier = (select top 1 UID from SysSchema where Name = @editPageName);
DECLARE @pageCaption nvarchar(250) = (select top 1 Caption from SysSchema where Name = @editPageName);
DECLARE @sysModuleEntityId uniqueidentifier = (select top 1 Id from SysModuleEntity where SysEntityType = 'Page');
DECLARE @landingTypeId uniqueidentifier = (SELECT TOP 1 Id FROM LandingType WHERE Name = @landingTypeName);

-- Adding new Landing page variant to application interface
INSERT INTO SysModuleEdit
(Id, SysModuleEntityId, TypeColumnName, UseModuleDetails, CardSchemaUID, ActionKindCaption, ActionCaption)
VALUES
(NEWID(), @sysModuleEntityId, @landingTypeId, 1, @cardSchemaUID, @actionCaption, @editPageName,
```

**Важно.** 41AE7D8D-BEC3-41DF-A6F0-2AB0D08B3967 — неизменный идентификатор сущности GeneratedWebForm в таблице [ **dbo.SysSchema** ] базы данных для любого кейса добавления посадочной страницы для пользовательской сущности.

После выполнения скрипта очистите кэш браузера. В результате в разделе [ **Лендинги и web-формы** ] ([ **Landing pages and web forms** ]) появится возможность добавить новый тип лендинга [ **Contact form** ]. Но при открытии страницы лендинга будет отсутствовать скрипт, который необходимо добавить в код посадочной страницы.

Реестр раздела [ **Лендинги и web-формы** ] ([ *Landing pages and web forms* ])

Landing pages and web forms

**ACTIONS**

Version	Type	Created on	Leads
ing-	Lead	6/3/2015 12:30 PM	0
	Description	The page gives access to the demo version of bpm'online marketing	Status Active
Contact form	Type Lead	Created on 12/2/2015 8:28 AM	Leads 0
Bpm'online marketing – free trial	Description	Form under construction	Status Inactive
Website URL	http://www.terrasoft.ru/demo/ru-marketing-trial/redirect		

Страница лендинга

New record

**ACTIONS**

Name*	Website domains*	ATTACHMENTS AND NOTES	DEFAULT VALUES	HISTORY	FEED

**LANDING SETUP**

**STEP 1. Set up a redirection URL (optional)**

Redirection URL

**STEP 2. Copy the code and configure and map the fields**

**STEP 3. Insert the customized code into the landing page source code. Set up a function to create the object on form submit**

Place the edited code on your landing page. Insert the following code into the <form> tag of your form.

```
onSubmit="createObject(); return false"
```

## 4. Актуализируйте наполнение скриптами для страницы web-формы

Значение переменной содержит экранированный html с тегами `<script>` и другую информацию по настройке связи полей web-формы — колонок создаваемой сущности. Это значение должно быть локализуемым. Для этого выполните следующий SQL-запрос:

```
-- Landing edit page schema name
DECLARE @editPageName nvarchar(250) = N'UsrContactGeneratedWebFormPage'; -- название созданной с

-- region Scripts' structure
DECLARE @sqriptPrefix nvarchar(max) = N'<div style="font-family: "Courier New"; monosp'
DECLARE @sqriptDelimiter nvarchar(max) = N'<br>&ampnbsp&ampnbsp&ampnbsp&ampnbsp&ampnbsp&ampnbsp&ampnbsp''
DECLARE @sqriptSuffix nvarchar(max) = N'<br>&ampnbsp&ampnbsp&ampnbsp&ampnbsp&ampnbsp},<br>&ampnbsp&ampnbsp&ampnbsp&ampnbsp&nbs
-- endregion

-- region Scripts' variables
DECLARE @sqriptNameColumn nvarchar(max); -- объявляем переменные для колонок, которые будем мапи
DECLARE @sqriptEmailColumn nvarchar(max);
DECLARE @scriptResult nvarchar(max);
-- endregion

-- Adding entity columns.
SET @sqriptNameColumn = N'"Name":&ampnbsp"<span style="color: #0c0cec;">css-selectc
SET @sqriptEmailColumn = N'"Email":&ampnbsp"<span style="color: #0c0cec;">css-selec

-- Concat result scripts.
SET @scriptResult = @sqriptPrefix + @sqriptNameColumn + @sqriptDelimiter + @sqriptEmailColumn + 

-- Set new localizable scripts value for resource with name like '%ScriptTemplate'
UPDATE SysLocalizableValue
SET [Value] = @scriptResult
WHERE SysSchemaId = (SELECT TOP 1 Id FROM SysSchema WHERE [Name] = @editPageName)
and [Key] like '%ScriptTemplate.Value'
```

В блоке `Adding entity columns` добавьте названия колонок сущности, которые будут заполняться значениями из web-формы.

**Важно.** Символы двойных кавычек ("") и пробела ( ) необходимо заменить экранируемыми "" и .

**Важно.** Для добавления поля необходимо добавить переменную `(@sqriptИмяПеременнойColumn)` и [конкатенировать](#) ее в `scriptResult`.

Если после выполнения всех настроек появилась необходимость получать значения других полей (кроме `Name` и `Email`), то необходимо повторно выполнить скрипт из данного пункта, но в блоке `Adding entity columns` прописать все необходимые колонки, включая существующие. При повторном выполнении скрипта произойдет обновление ранее созданных настроек.

После выполнения скрипта необходимо открыть созданную в конфигурации схему и пересохранить ее, чтобы пересохнулились ресурсы. В результате в разделе **[ Лендинги и web-формы ] ([ Landing pages and web forms ])** при выборе **[ Contact form ]** будет отображена страница лендинга со скриптом,

который необходимо разместить в коде посадочной страницы.

The screenshot shows the Creatio application's interface for setting up a landing page. On the left, there is a sidebar with navigation links for Marketing, Contacts, Campaigns, Email, Landing pages and web forms (which is selected), Events, Leads, Accounts, Dashboards, and Marketing plans. The main area is titled "New record" and contains fields for Name\*, Website domains\*, Description, and Status\* (set to Active). To the right, a tab bar includes LANDING SETUP (selected), ATTACHMENTS AND NOTES, DEFAULT VALUES, HISTORY, and FEED. Below the tabs, three steps are outlined: STEP 1. Set up a redirection URL (optional) with a note about replacing CSS-selectors; STEP 2. Copy the code and configure and map the fields, showing a snippet of JavaScript code for mapping fields; and STEP 3. Insert the customized code into the landing page source code. A sidebar on the right contains icons for user profile, settings, and help.

Скрипт содержит конфигурационный объект `config`, в котором определены следующие свойства:

- `fields` — содержит объект со свойствами `Name` и `Email`, значения которых должны совпадать с селекторами атрибутов `id` соответствующих полей web-формы.
- `landingId` — содержит идентификатор лендинга в базе данных.
- `serviceUrl` — содержит URL службы, по которому будут отправляться данные web-формы.
- `onSuccess` — содержит функцию-обработчик успешного создания контакта. Необязательное свойство.
- `onError` — содержит функцию-обработчик ошибки создания контакта. Необязательное свойство.

Конфиг, который будет формироваться, представлен ниже.

```
var config = {
    fields: {
        "Name": "css-selector", // Имя контакта
        "Email": "css-selector", // Email контакта
    },
    landingId: "b73790ab-acb1-4806-baea-4342a1f3b2a8",
    serviceUrl: "http://localhost:85/0/ServiceModel/GeneratedObjectWebFormService.svc/SaveWebFor
    redirectUrl: ""}),
};
```

## 5. Создайте и настройте лендинг раздела [Лендинги и

## web-формы] ([Landing pages and web forms])

Чтобы создать новую запись лендинга необходимо в разделе [ **Лендинги и web-формы** ] ([ **Landing pages and web forms** ]) выбрать [ **Contact form** ]. Добавление записи в раздел [ **Лендинги и web-формы** ] ([ **Landing pages and web forms** ]) описано в статье "[Как добавить новую запись в разделе \[Лендинги и web-формы\]](#)".

Для создаваемой записи установите:

- [Название] ([Name]) — "Contact";
- [Домены сайта] ([Website domains]) — "http://localhost:85/Landing/LandingPage.aspx";
- [Состояние] ([Status]) — "Active".

Для применения изменений сохраните страницу.

## 6. Разверните и настройте посадочную страницу, содержащую web-форму

Чтобы создать посадочную страницу для лендинга необходимо в любом текстовом редакторе при помощи html-разметки создать обычную посадочную страницу, содержащую web-форму. Создание посадочной страницы и добавление скрипта созданного лендинга в код посадочной страницы описаны в статье "[Как связать лендинг на сайте с Creatio](#)".

Для регистрации в Creatio данных контакта, отправляемых через web-форму, в код посадочной страницы необходимо добавить следующие поля (html-элемент `<input>`):

- Имя контакта.
- Email контакта.

Для каждого поля необходимо указать атрибуты `name` и `id`.

Чтобы при отправке данных web-формы в Creatio создавался новый объект [**Контакт**] ([**Contact**]), в посадочную страницу добавьте скрипт на языке JavaScript. Исходный код скрипта скопируйте из поля [**ШАГ 2. Скопируйте код и настройте в нем соответствие полей**] ([**STEP 2. Copy the code and configure and map the fields**]) страницы лендинга (рис. 8).

Конфигурационный объект `config` передается в качестве аргумента функции `createObject()`, которая должна выполняться при отправке web-формы.

Чтобы функция `createObject()` была вызвана при отправке web-формы, в тег `<form>` web-формы посадочной страницы добавьте атрибут `onSubmit="createObject(); return false"` из поля [**ШАГ 3.**

**Вставьте настроенный код на страницу лендинга. Настройте запуск функции создания объекта по submit формы**] ([**STEP 3. Insert the customized code into the landing page source code. Set up a function to create the object on form submit**]) страницы лендинга.

### Исходный код посадочной страницы

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <!--ШАГ 2-->
    <!--Эту часть необходимо скопировать из поля ШАГ 2 страницы редактирования лендинга-->
    <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.min.js"></script>
    <script src="https://webtracking-v01.bpmonline.com/JS/track-cookies.js"></script>
    <script src="https://webtracking-v01.bpmonline.com/JS/create-object.js"></script>
    <script>

    /**
     * Замените выражение в кавычках "css-selector" в коде ниже значением селектора элемента на Вашей странице.
     * Вы можете использовать #id или любой другой CSS селектор, который будет точно определять нужный элемент.
     * Пример: "Email": "#MyEmailField".
     * Если Ваша лендинговая страница не содержит одного или нескольких полей из приведенных ниже, уберите их.
     */
    var config = {
        fields: {
            "Name": "#name-field", // Имя контакта
            "Email": "#email-field", // Email контакта
        },
        landingId: "b73790ab-acb1-4806-baea-4342a1f3b2a8",
        serviceUrl: "http://localhost:85/0/ServiceModel/GeneratedObjectWebFormService.svc/SaveWebFormObject",
        redirectUrl: "",
        onSuccess: function(response) {
            window.alert(response.resultMessage);
        },
        onError: function(response) {
            window.alert(response.resultMessage);
        }
    };
    /**

```

```

* Функция ниже создает объект из введенных данных.
* Привяжите вызов этой функции к событию "onSubmit" формы или любому другому элементу события
* Пример: <form class="mainForm" name="landingForm" onSubmit="createObject(); return false">
/*
function createObject() {
    landing.createObjectFromLanding(config)
}
/**/
* Функция ниже инициализирует лендинг из параметров URL.
*/
function initLanding() {
    landing.initLanding(config)
}
jQuery(document).ready(initLanding)
</script>
<!--ШАГ 2-->

</head>
<body>
<h1>Landing web-page</h1>
<div>
    <h2>Contact form</h2>
    <form method="POST" class="mainForm" name="landingForm" onSubmit="createObject(); return fal
        Name:<br>
        <input type="text" name="Name" id="name-field"><br>
        Email:<br>
        <input type="text" name="Email" id="email-field"><br><br><br>
        <input type="submit" value="Submit">
    </font>
</form>
</div>
</body>
</html>

```

Откройте посадочную страницу. Для создаваемого контакта установите:

- **[Name]** — "New User";
- **[Email]** — "new\_user@creatio.com".

# Landing web-page

## Contact form

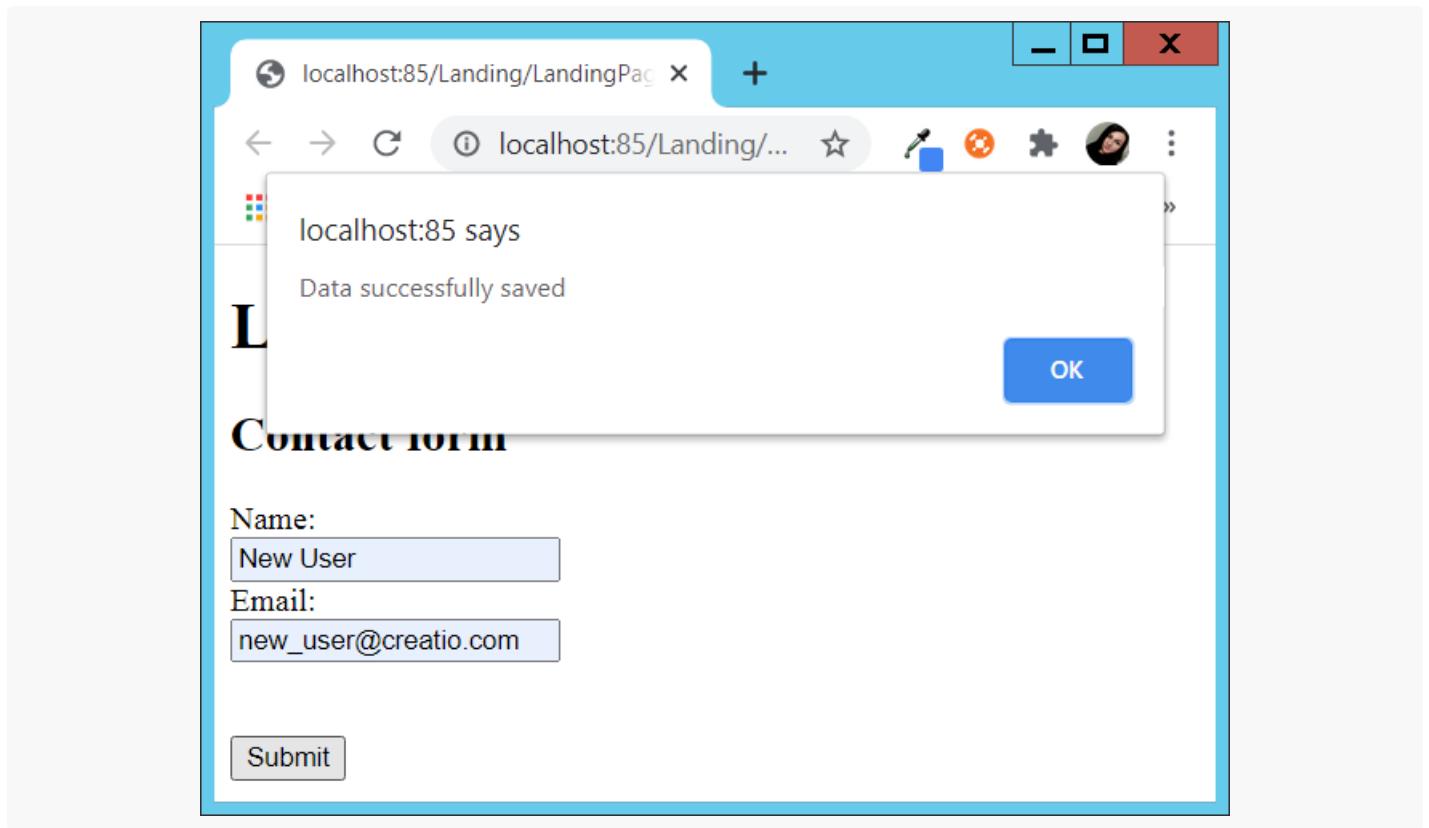
Name:

Email:

Для создания контакта нажмите [ **Submit** ].

**Важно.** Контакт с посадочной страницы будет добавлен в Creatio только в том случае, если страница размещена на сайте, имя которого указано в поле [**Домены сайта**] ([**Website domains**]) лендинга.

Если разместить страницу на локальном сервере компьютера, обслуживающего [зарезервированное доменное имя localhost](#) (как указано в настройке лендинга), то скрипт для создания контакта с посадочной страницы лендинга отработает корректно.



В результате в системе будет автоматически создан контакт с указанными параметрами.

Name	Job title	Business phone
Andrew Baker (sample)	Specialist	+1 617 440 2031
New User	Email	+1 617 221 5187
Supervisor	a.baker@ac.com	
Supervisor		
SysPortalConnection		

## 7. Зарегистрируйте пользовательский объект в справочнике

Для использования пользовательского лендинга в маркетинговых кампаниях (элемент **[Добавить из лендинга] ([Landing page])**), нужно зарегистрировать его в справочнике.

Регистрация объекта в справочнике **[Настройки объекта для элемента кампании "Лендинг"]**

### ([Entity settings for campaign landing element]):

- Перейдите в дизайнер системы по кнопке .
- В блоке **[Настройка системы]** перейдите по ссылке **[Справочники] ([Lookups])**.
- Откройте справочник **[Настройки объекта для элемента кампании "Лендинг"] ([Entity settings for campaign landing element])**.
- Нажмите **[Добавить] ([Add])**. Для создаваемого объекта заполните поля:
  - [Название] ([Caption])** — пользовательское название объекта.
  - [Объект] ([Entity object])** — создаваемый посадочной страницей объект системы.
  - [Путь к Контакту] ([Path to Contact])** — путь к колонке, которая связывает запись с контактом.
  - [Путь к идентификатору веб-формы] ([Path to WebForm])** — путь к колонке, которая связывает запись с посадочной страницей.
- Сохраните объект.

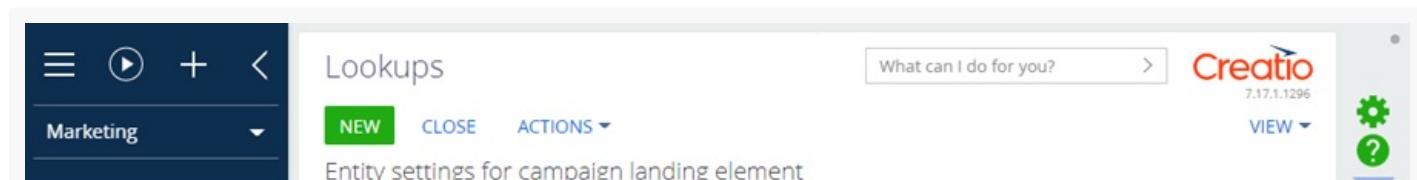
Пример заполнения полей для объектов **[Лид] ([Lead])** и **[Участник мероприятия] ([Event participant])** представлен в таблице.

Полей объектов **[Лид] ([Lead])** и **[Участник мероприятия] ([Event participant])**

Имя поля	Значение поля объекта	
	<b>[Лид] ([Lead])</b>	<b>[Участник мероприятия] ([Event participant])</b>
<b>[Название] ([Caption])</b>	Lead	Event participant
<b>[Объект] ([Entity object])</b>	Lead	Event participant
<b>[Путь к Контакту] ([Path to Contact])</b>	QualifiedContact	Contact
<b>[Путь к идентификатору веб-формы] ([Path to WebForm])</b>	WebForm	GeneratedWebForm

Для создаваемой записи установите:

- [Название] (Caption)** — "Contact";
- [Объект] ([Entity object])** — "Lead";
- [Путь к Контакту] ([Path to Contact])** — "QualifiedContact";
- [Путь к идентификатору веб-формы] ([Path to WebForm])** — "WebForm".



Caption	Entity object	Path to Contact	Path to WebForm
Contact	Lead	QualifiedContact	WebForm
Event participant	Event participant	Contact	GeneratedWebForm
Lead	Lead	QualifiedContact	WebForm

Пользовательский лендинг "Contact" доступен к выбору в элементе **[Добавить из лендинга] ([Landing page])** маркетинговых кампаний.

## Реализовать обработчик для создания сущности с помощью web-формы

Сложный

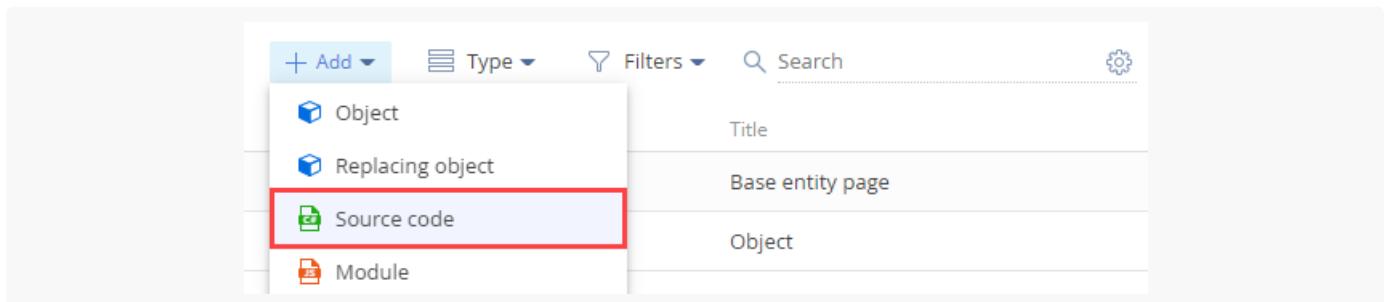
В сущности `Contact` присутствует обязательная текстовая колонка `CustomRequiredTextColumn`. При создании участника мероприятия (`EventTarget`) при помощи web-формы в приложении выполняется поиск соответствующего контакта. Если контакт не найден, то по умолчанию создается новый контакт. При сохранении контакта возникает ошибка, поскольку не заполнено обязательное поле `CustomRequiredTextColumn`. Чтобы успешно **сохранить контакт**, необходимо реализовать пользовательский обработчик перед созданием участника мероприятия.

**Пример.** Реализовать пользовательский обработчик перед созданием участника мероприятия.

Перед выполнением примера настройте web-форму для создания пользовательского объекта. В web-форму добавьте обязательное пользовательское поле `CustomRequiredTextColumn`. Для этого воспользуйтесь инструкцией, которая приведена в статье [Настроить web-форму для создания пользовательского объекта](#).

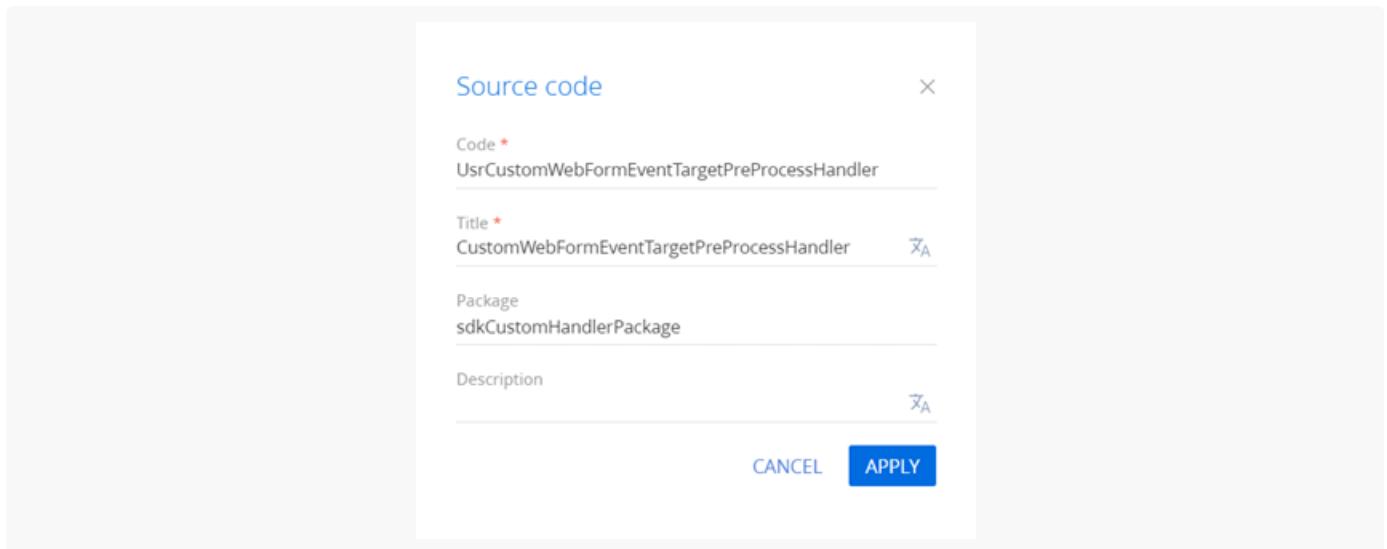
## 1. Реализовать пользовательский обработчик

1. [Перейдите в раздел \[ Конфигурация \]](#) ([ Configuration ]) и выберите пользовательский [пакет](#), в который будет добавлена схема.
2. На панели инструментов реестра раздела нажмите [ Добавить ] —> [ Исходный код ] ([ Add ] —> [ Source code ]).



3. В дизайнере схем заполните свойства схемы:

- [ Код ] ([ Code ]) — "UsrCustomWebFormEventTargetPreProcessHandler".
- [ Заголовок ] ([ Title ]) — "CustomWebFormEventTargetPreProcessHandler".



Для применения заданных свойств нажмите [ Применить ] ([ Apply ]).

4. Реализуйте **пользовательский обработчик перед созданием участника мероприятия**.
  - а. В дизайнере схем добавьте пространство имён `Terrasoft.Configuration`.

- b. С помощью директивы `using` добавьте пространства имен, типы данных которых будут задействованы в классе.
- c. Добавьте название класса, которое соответствует названию схемы (свойство [ Код ] ([ Code ])).
- d. В качестве родительского класса укажите класс `WebFormEventTargetPreProcessHandler`.

Исходный код схемы `UsrCustomWebFormEventTargetPreProcessHandler` типа [ Исходный код ] ([ Source code ]) представлен ниже.

#### UsrCustomWebFormEventTargetPreProcessHandler

```
namespace Terrasoft.Configuration
{
    using System;
    using System.Linq;
    using Core.Entities;
    using Core;
    using GeneratedWebService;

    #region Class: UsrCustomWebFormEventTargetPreProcessHandler
    /// <summary>
    /// Executes custom pre event target saving processing.
    /// </summary>
    /// <seealso cref="Terrasoft.Configuration.IGeneratedWebFormPreProcessHandler" />
    public class CustomWebFormEventTargetPreProcessHandler: WebFormEventTargetPreProcessHandler
    {

        #region Properties: Private
        private UserConnection _userConnection { get; set; }
        private FormData _formData { get; set; }
        #endregion

        #region Methods: Private
        private string GetCustomRequiredColumnValue(string customColumnName) {
            var customFormField = this._formData.formFieldsData
                .FirstOrDefault(x => x.name == customColumnName);
            if (customFormField == null) {
                throw new Exception($"There is no required form field {customColumnName}");
            }
            if (string.IsNullOrEmpty(customFormField?.value)) {
                throw new Exception($"Required value is empty for field {customColumnName}");
            }
            return customFormField.value;
        }
        #endregion

        #region Methods: Protected
        /// <summary>
```

```

/// Creates contact entity with custom required text column filled with form value.
/// </summary>
/// <param name="contactId">Unique identifier of contact.</param>
/// <param name="contactNameField">Required contact name form field.</param>
protected override void CreateContactEntity(Guid contactId, FormFieldsData contactName)
{
    EntitySchema contactSchema = _userConnection.EntitySchemaManager.GetInstanceByName("Contact");
    Entity contact = contactSchema.CreateEntity(_userConnection);
    contact.SetDefColumnValues();
    contact.SetColumnValue("Id", contactId);
    contact.SetColumnValue("Name", contactNameField.value);

    // set value for custom required column.
    var customRequiredColumnName = nameof(Contact.CustomRequiredTextColumn);
    var customRequiredColumnValue = GetCustomRequiredColumnValue(customRequiredColumnName);
    contact.SetColumnValue(customRequiredColumnName, customRequiredColumnValue);

    contact.Save(false);
}

#endregion

#region Methods: Public
/// <inheritdoc/>
/// Override for base method implementation to init <see cref="UserConnection"/> instance
public new FormData Execute(UserConnection userConnection, FormData formData,
    IWebFormImportParamsGenerator paramsGenerator) {
    _userConnection = userConnection;
    _formData = formData;
    return base.Execute(userConnection, formData, paramsGenerator);
}
#endregion
}
#endregion
}

```

- На панели инструментов дизайнера нажмите [ Сохранить ] ([ Save ]), а затем [ Опубликовать ] ([ Publish ]).

## 2. Зарегистрировать пользовательский обработчик в базе данных

Реализация пользовательского обработчика предполагает его обязательную регистрацию в таблице `[WebFormProcessHandlers]` базы данных.

**Способы** регистрации пользовательского обработчика в базе данных:

- С помощью справочника.

Чтобы зарегистрировать пользовательский обработчик в базе данных **с помощью справочника**:

- Перейдите в дизайнер системы по кнопке .
- В блоке [ Настойка системы ] ([ System setup ]) перейдите по ссылке [ Справочники ] ([ Lookups ]).
- Добавьте новую запись для обработчика в справочник для сущности [ Обработчики веб-форм ] ([ Web form process handlers ]). По умолчанию этот справочник не добавлен в перечень справочников приложения. Чтобы добавить справочник [ Обработчики веб-форм ] ([ Web form process handlers ]) в приложение, создайте справочник и в качестве объекта выберите объект [ Обработчики веб-форм ] ([ Web form process handlers ]).
- Заполните поля справочника:**
  - [ Имя объекта ] — "EventTarget".
  - [ FullClassName ] — "Terrasoft.Configuration.CustomWebFormEventTargetPreProcessHandler, Terrasoft.Configuration".
  - Установите признак [ Активный ] ([ isActive ]).
- С помощью SQL-запроса.

Чтобы зарегистрировать пользовательский обработчик в базе данных **с помощью SQL-запроса**, выполните следующий SQL-запрос.

### SQL-запрос

```
INSERT INTO WebFormProcessHandlers (Id, EntityName, FullClassName, IsActive)
VALUES (NEWID(), N'EventTarget', 'Terrasoft.Configuration.CustomWebFormEventTargetPreProc
```

Поскольку схема наследуется от коробочного обработчика и базовая логика вызывается в пользовательской схеме, то необходимо отключить коробочный обработчик.

### Способы отключения коробочного обработчика:

- С помощью справочника.

Чтобы отключить коробочный обработчик **с помощью справочника**:

- Перейдите в дизайнер системы по кнопке .
- В блоке [ Настойка системы ] ([ System setup ]) перейдите по ссылке [ Справочники ] ([ Lookups ]).
- Откройте справочник и для сущности [ Участник мероприятия ] ([ EventTarget ]), у которого в поле [ FullClassName ] установлено значение "Terrasoft.Configuration.CustomWebFormEventTargetPreProcessHandler, Terrasoft.Configuration", отключите признак [ Активный ] ([ isActive ]).

- С помощью SQL-запроса.

Чтобы отключить коробочный обработчик **с помощью SQL-запроса**, выполните следующий SQL-запрос.

### SQL-запрос

```
UPDATE WebFormProcessHandlers
SET IsActive = 0
```

```
WHERE FullClassName = 'Terrasoft.Configuration.WebFormEventTargetPreProcessHandler', Terrasoft
```

## Результат выполнения примера

Чтобы **посмотреть результат выполнения примера**, перезапустите пул приложения.

В результате новый контакт создается при отправке формы с заполненными обязательными полями, среди которых присутствует поле `CustomRequiredTextColumn`.

## Лид



Сложный

## Создать пользовательские напоминания и уведомления

В версии 7.12.0 переработан механизм отправки уведомлений и напоминаний.

Ранее для отправки пользовательского уведомления требовалось:

- Создать класс, реализующий интерфейс `INotificationProvider` или наследующий абстрактный класс `BaseNotificationProvider`.
- В созданный класс добавить логику для выбора пользовательских напоминаний приложением.
- Зарегистрировать класс в таблице `NotificationProvider`.

При этом уведомления отправлялись раз в минуту, вызывая все классы из таблицы `NotificationProvider`.

Начиная с версии 7.12.0 достаточно создать уведомление или напоминание с нужными параметрами.

После этого приложение либо отправит уведомление мгновенно, либо отобразит напоминание в указанное время.

Чтобы отправлять пользовательские напоминания:

1. В пользовательском пакете [создайте схему \[ Исходный код \]](#), в которой определите класс для формирования текста напоминания и всплывающего окна. Класс должен реализовывать интерфейс `IRemindingTextFormatter` (объявлен в схеме `IRemindingTextFormatter` пакета `Base`).
2. Заместите нужный объект (например, объект [ Лид ]) и задайте в нем логику отправки напоминания.
3. Заместите схему вкладки напоминаний `ReminderNotificationsSchema` для отображения напоминаний по требуемому объекту.

## Создать пользовательское напоминание о дате актуализации продажи в лиде



Средний

### Описание примера

Создать пользовательское напоминание о дате актуализации продажи в лиде. Дата должна быть

указана в поле [ *Дата следующей актуализации* ] на вкладке [ *Информация о сделке* ].

## Исходный код

Пакет с реализацией примера для Sales Creatio, enterprise edition можно скачать по [ссылке](#).

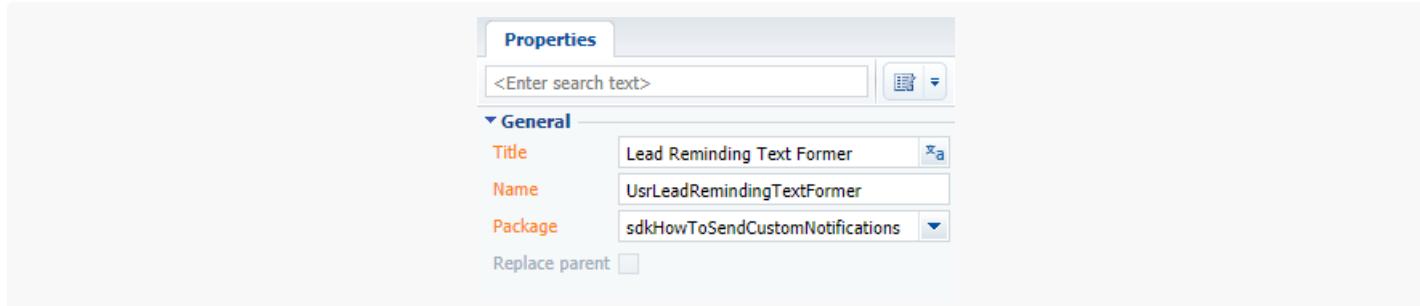
## Алгоритм реализации примера

### 1. Создать класс для формирования текста напоминания и всплывающего окна

1. Создайте в пользовательском пакете схему [ *Исходный код* ] (см. "[Создать схему \[ Исходный код \]](#)"). Установите для нее следующие значения свойств (рис. 1):

- [Заголовок] ([Title]) — "Конструктор текста напоминаний лида" ("Lead Reminding Text Former").
- [Название] ([Name]) — "UsrLeadRemindingTextFormer".

Рис. 1. — Свойства схемы [ *Исходный код* ]



2. Используя контекстное меню вкладки [ *Структура* ] ([ *Structure* ]), добавьте в схему две локализуемые строки (рис. 2). Свойства локализуемых строк приведены в таблице 1.

Рис. 2. — Добавление локализуемых строк в схему

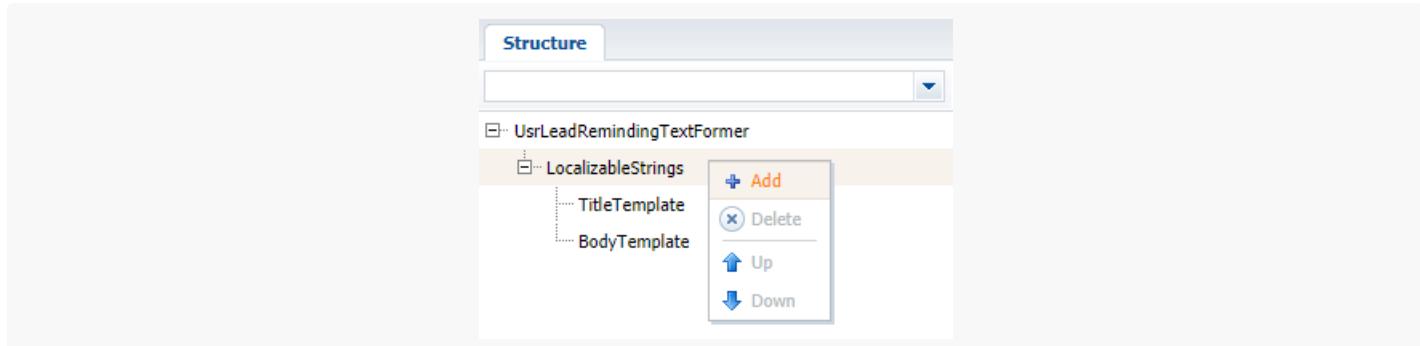


Табл. 1. — Свойства локализуемых строк

[Название] ([Name])	[Значение] ([Value])
TitleTemplate	Необходимо актуализировать продажу (You need to update the sale)
BodyTemplate	Лид {0} требует актуализации информации по продаже (Lead {0} requires update of sales information)

3. На вкладку [ Исходный код ] ([ *Source code* ]) добавьте реализацию класса для формирования текста напоминания и всплывающего окна:

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

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

        public UsrLeadRemindingTextFormer(UserConnection userConnection)
        {
            UserConnection = userConnection;
        }
        // Формирует текст напоминания из коллекции входящих параметров и локализуемой строки Вс
        public string GetBody(IDictionary<string, object> formParameters)
        {
            formParameters.CheckArgumentNull("formParameters");
            var bodyTemplate = UserConnection.GetLocalizableString(ClassName, "BodyTemplate");
            var leadName = (string)formParameters["LeadName"];
            var body = string.Format(bodyTemplate, leadName);
            return body;
        }
        // Формирует заголовок напоминания из имени класса и локализуемой строки TitleTemplate.
        public string GetTitle(IDictionary<string, object> formParameters)
        {
            return UserConnection.GetLocalizableString(ClassName, "TitleTemplate");
        }
    }
}
```

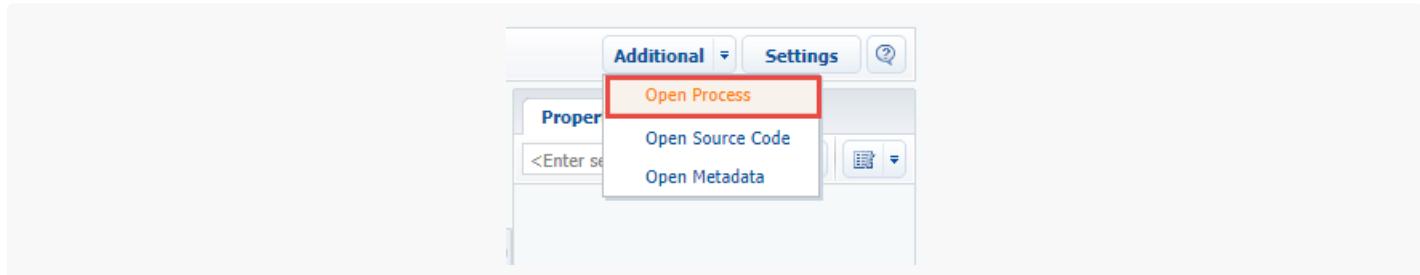
4. Сохраните и опубликуйте схему.

## 2. Заместить объект [Лид] и задать в нем логику отправки

## НАПОМИНАНИЯ

- Создайте замещающую схему объекта [ Лид ] (см. раздел "Создание замещающей схемы объекта" статьи "[Создать схему объекта](#)").
- Используя команду [ Дополнительно ] ([ Additional ]) — [ Открыть процесс ] ([ Open process ]), перейдите в дизайнер встроенного процесса замещающего объекта [ Лид ] (рис. 3).

Рис. 3. — Команда перехода в дизайнер встроенного процесса



- Используя контекстное меню вкладки [ Структура ] ([ Structure ]), добавьте параметр процесса `GenerateReminding` (рис. 4) со следующими свойствами (рис. 5):

- [Заголовок] ([Title]) — "Сгенерировать напоминание" ("Generate Reminding").
- [Название] ([Name]) — "GenerateReminding".
- [Тип данных] ([Data type]) — "Boolean".

Рис. 4. — Добавление параметра процесса

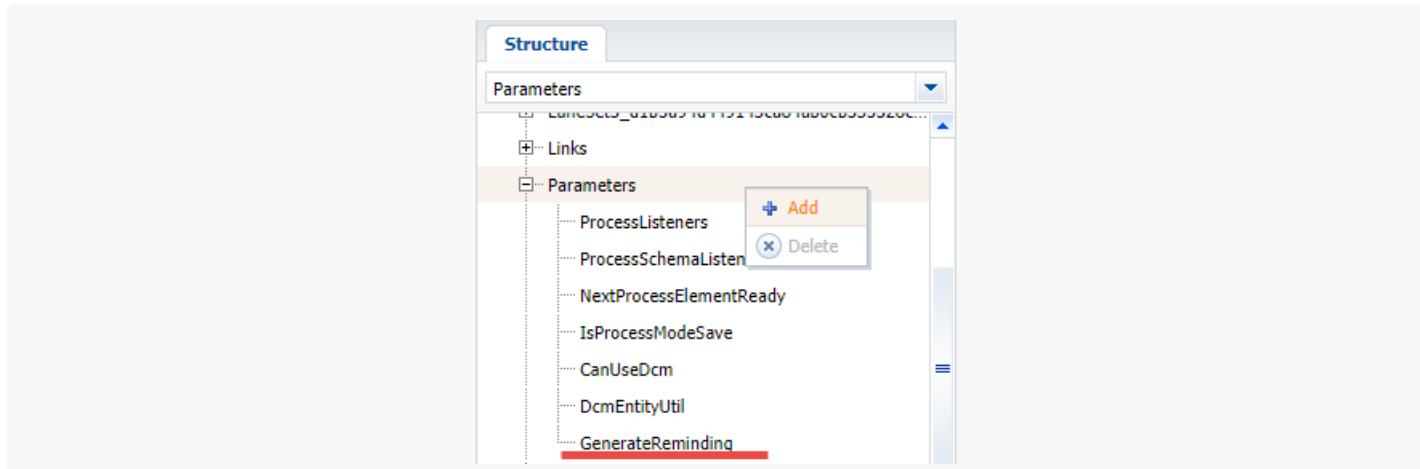
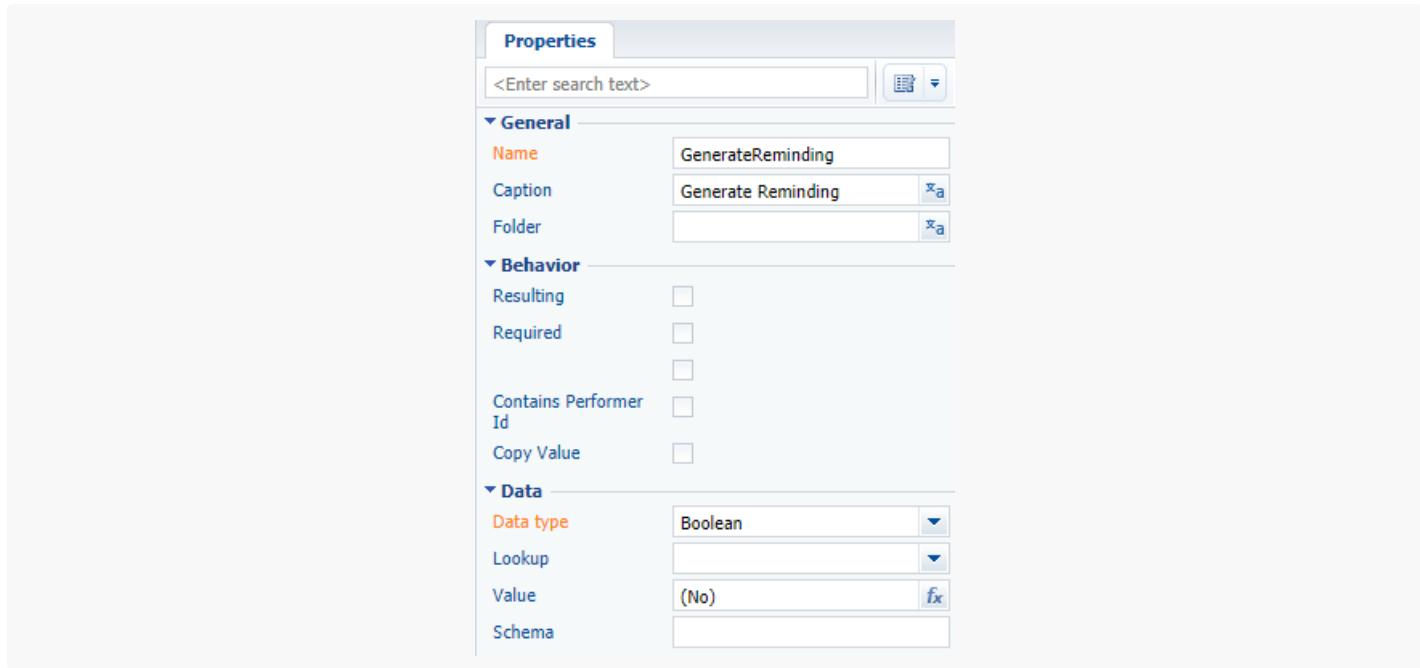


Рис. 5. — Свойства параметра процесса



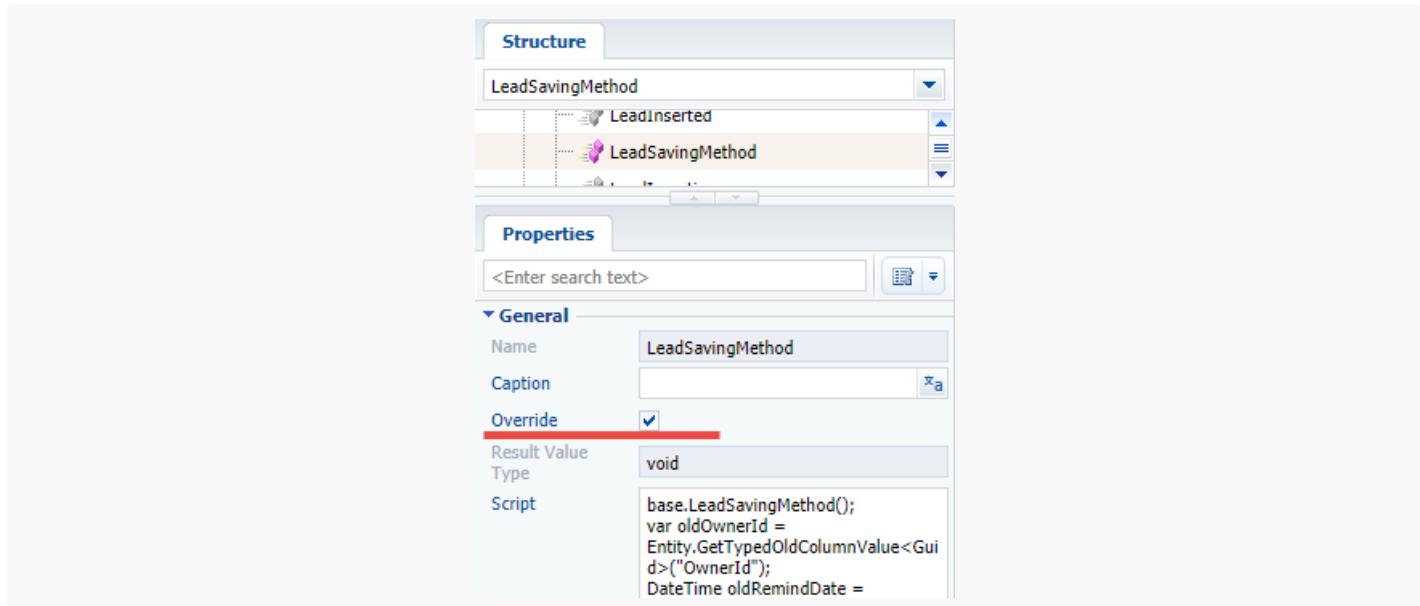
4. На вкладке [ Структура ] ([ *Structure* ]), в группе [ *Methods* ] выберите метод `LeadSavingMethod()`, который вызывается перед сохранением объекта (рис. 6). Укажите признак [ *Переопределен* ] ([ *Override* ]) и добавьте следующий исходный код в тело метода:

```
// Вызов базовой реализации метода.
base.LeadSavingMethod();
// Получение идентификатора ответственного.
var oldOwnerId = Entity.GetTypedOldColumnValue<Guid>("OwnerId");
// Получение даты следующей актуализации.
DateTime oldRemindDate = Entity.GetTypedOldColumnValue<DateTime>("NextActualizationDate");
// Сравнение старого значения идентификатора ответственного и нового.
bool ownerChanged = !Entity.OwnerId.Equals(oldOwnerId);
// Сравнение старого значения даты актуализации и нового.
bool remindDateChanged = !Entity.NextActualizationDate.Equals(oldRemindDate);
// Параметру процесса GenerateReminding присвоить true, если были изменены или ответственный,
// или дата следующей актуализации.
GenerateReminding = ownerChanged || remindDateChanged;
```

### **На заметку.**

Для отображения редактора исходного кода тела метода дважды кликните по названию метода на вкладке [ *Структура* ].

Рис. 6. — Свойства метода `LeadSavingMethod`



5. На вкладке [ Структура ] ([ *Structure* ]), в группе [ *Methods* ] выберите метод `LeadSavedMethod()`, который вызывается после сохранения объекта (рис. 7). Укажите признак [ Переопределен ] ([ *Override* ]) и добавьте следующий исходный код в тело метода:

```

base.LeadSaved();
// Проверки необходимости генерации напоминания.
if (!GenerateReminding) {
    return;
}
DateTime remindTime = Entity.NextActualizationDate;
if (Entity.OwnerId.Equals(Guid.Empty) || remindTime.Equals(default(DateTime))) {
    return;
}
// Создание экземпляра класса UsrLeadRemindingTextFormer.
IRemindingTextFormer textFormer =
    ClassFactory.Get<UsrLeadRemindingTextFormer>(new ConstructorArgument("userConnection", UserC
// Получение названия льда.
string leadName = Entity.LeadName;
// Получение текста напоминания.
string subjectCaption = textFormer.getBody(new Dictionary<string, object> {
    {"LeadName", leadName}
});
// Получение заголовка напоминания.
string popupTitle = textFormer.getTitle(null);
// Конфигурирование напоминания.
var remindingConfig = new RemindingConfig(Entity);
// Автор сообщения – текущий контакт.
remindingConfig.AuthorId = UserConnection.CurrentUser.ContactId;
// Целевой получатель – ответственный листа.
remindingConfig.ContactId = Entity.OwnerId;
// Тип – напоминание.

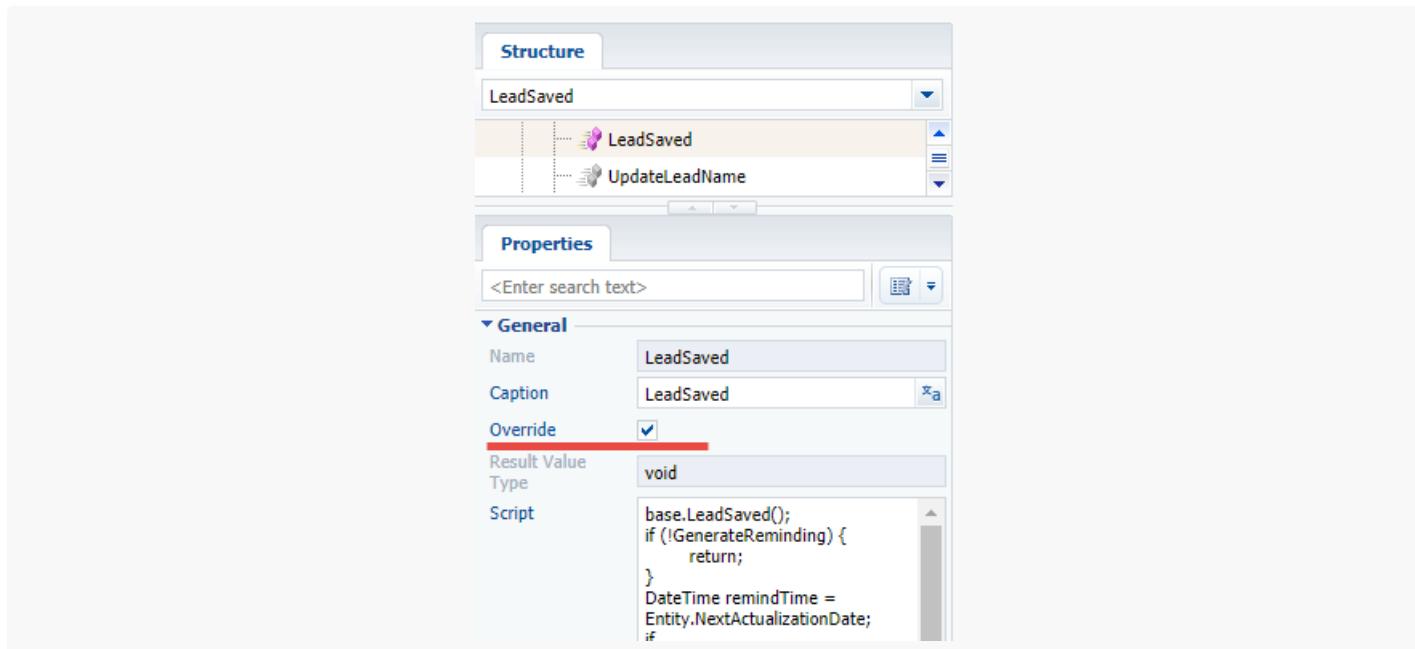
```

```

remindingConfig.NotificationTypeId = RemindingConsts.NotificationTypeRemindingId;
// Дата отправки напоминания – дата актуализации продажи в лиде.
remindingConfig.RemindTime = remindTime;
// Текст напоминания.
remindingConfig.Description = subjectCaption;
// Заголовок напоминания.
remindingConfig.PopupTitle = popupTitle;
// Создание утилитного класса напоминания.
var remindingUtilities = ClassFactory.Get<RemindingUtilities>();
// Создание напоминания.
remindingUtilities.CreateReminding(UserConnection, remindingConfig);

```

Рис. 7. — Свойства метода LeadSavedMethod



### На заметку.

Чтобы напоминания отображались на вкладке системных уведомлений , необходимо в коде метода LeadSavedMethod вместо

```

remindingConfig.NotificationTypeId = RemindingConsts.NotificationTypeRemindingId; указать
remindingConfig.NotificationTypeId = RemindingConsts.NotificationTypeNotificationId;

```

6. Сохраните и опубликуйте схему встроенного процесса объекта [ Лид ]. Затем опубликуйте схему объекта [ Лид ].

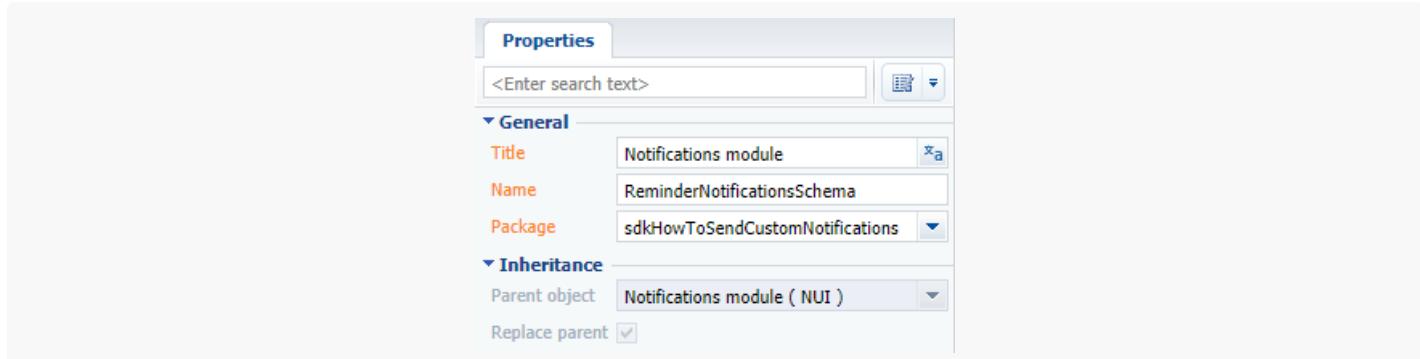
## 3. Заместить схему вкладки напоминаний ReminderNotificationsSchema

1. Чтобы отображались напоминания по требуемому объекту, создайте в [пользовательском пакете](#) замещающую схему `ReminderNotificationsSchema` и добавьте в нее необходимую программную логику. Как

создать замещающую клиентскую схему описано в статье "[Создать клиентскую схему](#)". Свойства замещающей схемы (рис. 8):

- [Заголовок] ([Title]) — "Модуль уведомлений" ("Notifications module").
- [Название] ([Name]) — "ReminderNotificationsSchema".

Рис. 8. — Свойства схемы ReminderNotificationsSchema



2. На вкладку [ Исходный код ] ([ *Source code* ]) схемы добавьте следующий исходный код:

```
define("ReminderNotificationsSchema", ["ReminderNotificationsSchemaResources"],
function() {
    return {
        entitySchemaName: "Reminding",
        methods: {
            // Определяет, относится ли напоминание к лицу.
            getIsLeadNotification: function() {
                return this.get("SchemaName") === "Lead";
            },
            // Возвращает заголовок напоминания.
            getNotificationSubjectCaption: function() {
                var caption = this.get("Description");
                return caption;
            }
        },
        // Массив модификаций модели представления.
        diff: [
            // Основной контейнер напоминания.
            {
                "operation": "insert",
                "name": "NotificationleadItemContainer",
                "parentName": "Notification",
                "propertyName": "items",
                "values": {
                    "itemType": Terrasoft.ViewItemType.CONTAINER,
                    "wrapClass": [
                        "reminder-notification-item-container"
                    ],

```

```
// Отображается только для лидов.
"visible": {"bindTo": "getIsLeadNotification"},  
"items": []  
}  
},  
// Контейнер для заголовка.  
{  
    "operation": "insert",  
    "name": "NotificationItemleadTopContainer",  
    "parentName": "NotificationleadItemContainer",  
    "propertyName": "items",  
    "values": {  
        "itemType": Terrasoft.ViewItemType.CONTAINER,  
        "wrapClass": ["reminder-notification-item-top-container"],  
        "items": []  
    }  
},  
// Изображение.  
{  
    "operation": "insert",  
    "name": "NotificationleadImage",  
    "parentName": "NotificationItemleadTopContainer",  
    "propertyName": "items",  
    "values": {  
        "itemType": Terrasoft.ViewItemType.BUTTON,  
        "className": "Terrasoft.ImageView",  
        "imageSrc": {"bindTo": "getNotificationImage"},  
        "classes": {"wrapClass": ["reminder-notification-icon-class"]}  
    }  
},  
// Отображение даты.  
{  
    "operation": "insert",  
    "name": "NotificationDate",  
    "parentName": "NotificationItemleadTopContainer",  
    "propertyName": "items",  
    "values": {  
        "itemType": Terrasoft.ViewItemType.LABEL,  
        "caption": {"bindTo": "getNotificationDate"},  
        "classes": {"labelClass": ["subject-text-labelClass"]}  
    }  
},  
// Отображение текста напоминания.  
{  
    "operation": "insert",  
    "name": "NotificationleadSubject",  
    "parentName": "NotificationItemleadTopContainer",  
    "propertyName": "items",  
    "values": {
```

```

        "itemType": Terrasoft.ViewItemType.LABEL,
        "caption": {"bindTo": "getNotificationSubjectCaption"},
        "click": {"bindTo": "onNotificationSubjectClick"},
        "classes": {"labelClass": ["subject-text-labelClass", "label-link", "lat
    }
}
];
};

});

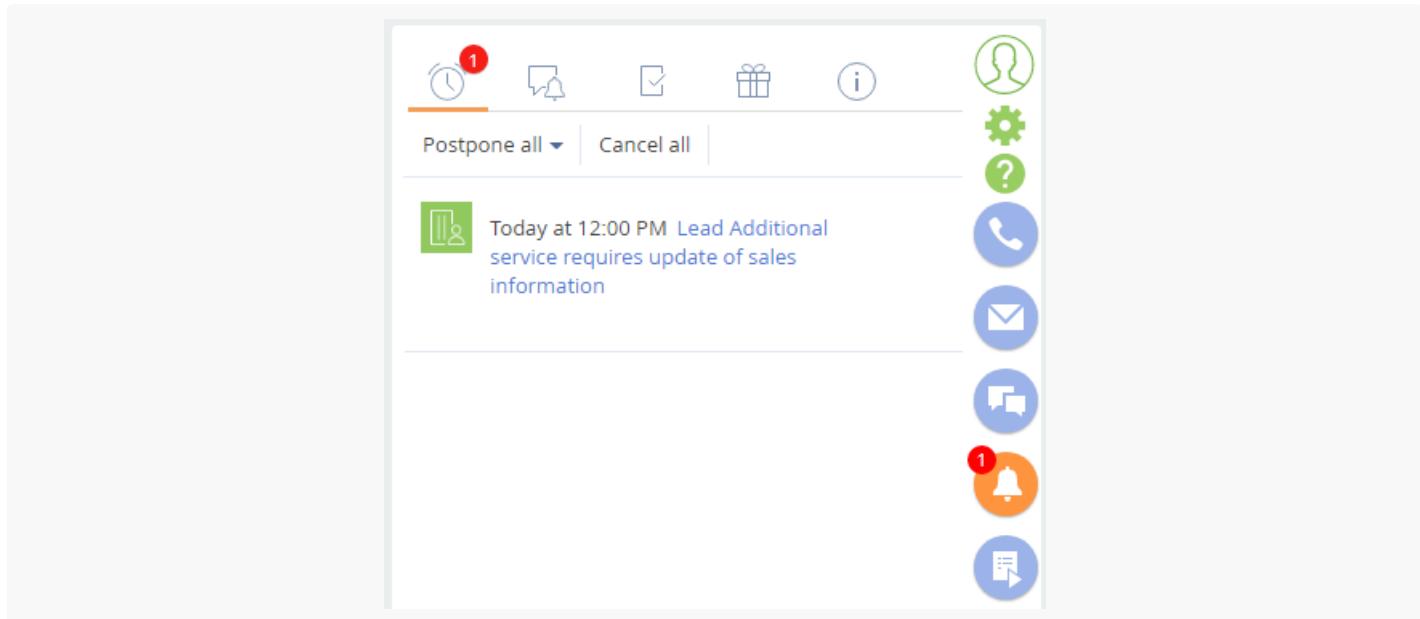
```

### 3. Сохраните схему.

В результате выполнения примера для лидов, у которых указаны ответственный и дата следующей актуализации (рис. 9), будут приходить напоминания о необходимости актуализации (рис. 10).

Рис. 9. — Лид с указанными ответственным и датой следующей актуализации

Рис. 10. — Напоминание об актуализации лода



## Планирование

 Сложный

Функциональность Sales Creatio позволяет планировать объемы продаж компании и анализировать фактическое достижение целей. Используя раздел [ Планирование ], можно формировать различные планы по ключевым срезам, данные о которых внесены в систему, и рассчитывать фактически полученные значения. Это позволяет анализировать выполнение продаж по выбранному периоду и оценивать эффективность отдела продаж в целом на основе сводных таблиц раздела [ Планирование ].

Подробнее возможности раздела описаны в статье "[Планирование](#)" документации пользователя.

## Изменить логику расчета в разделе [Планирование]

 Средний

**Пример.** В разделе Планирование изменить логику расчета факта: производить расчет на основании счетов, а не продаж.

### 1. Скопировать исходный код модуля построения плана

Для этого необходимо в разделе [ Конфигурация ] ([ Configuration ]) ввести название схемы `ForecastBuilder` в строку поиска (1). После двойного клика по строке результата поиска (2) в дизайнере модуля откроется схема модуля.

ForecastBuilder			
Schemas: All	External Assemblies: All	SQL Scripts: All	Data: All
Add	Edit	Delete	Package Dependencies
<a href="#">Name <sup>1</sup></a>		<a href="#">Package</a>	
ForecastBuilder	CoreForecast	<a href="#">Title <sup>2</sup></a>	

Затем нужно скопировать весь исходный код модуля из вкладки [ Исходный код ] ([ Source Code ]) и сохранить, например, в текстовом файле.

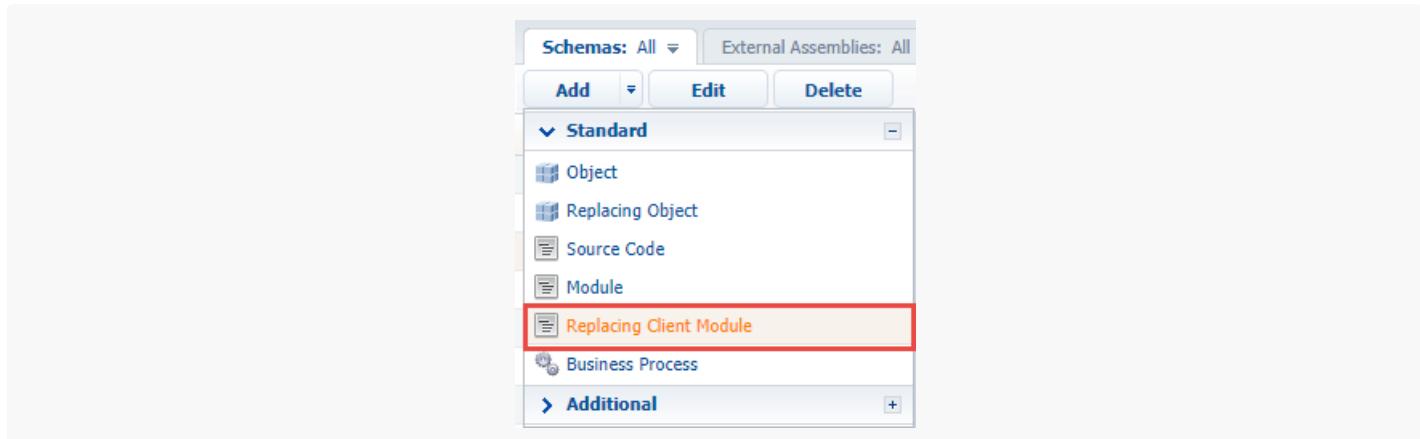
```

1 define("ForecastBuilder", ["ext-base", "ForecastBuilderResources", "RightUtilities", "Mask
2     "ConfigurationEnums", "ServiceHelper", "ImageCustomGeneratorV2", "AcademyUtili
3     ],);
4     function(Ext, resources, RightUtilities, MaskHelper, ConfigurationEnums, ServiceHe
5         AcademyUtilities) {
6
7             /**
8             * @class Terraso
9             * #####
10            */
11            Ext.define("Terraso", {
12                extend: "Terraso",
13                alternateClass: "Terraso"
14
15                /**
16                * #####
17                * @return {C
18                * generate: fun
19                *     return []
20                {
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37

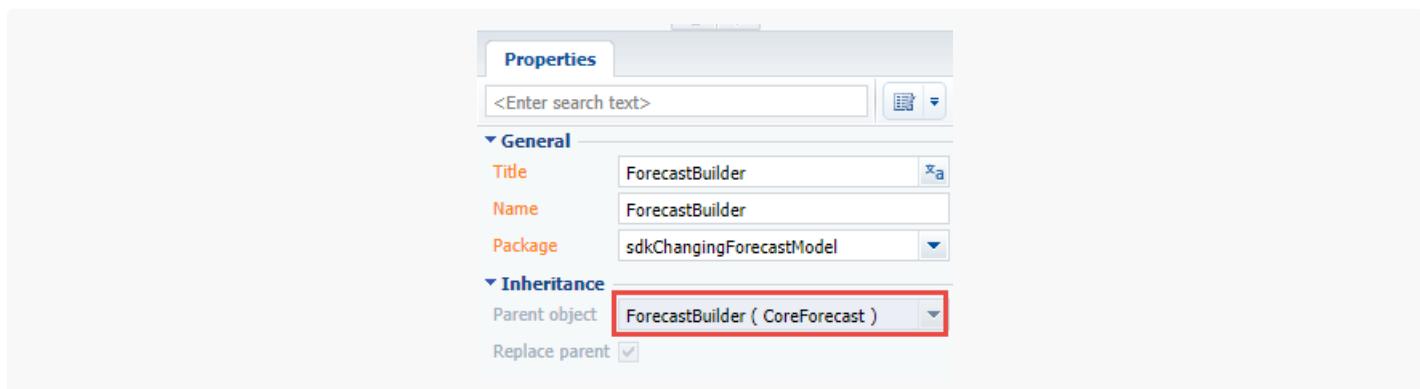
```

## 2. Создать замещающий модуль построения плана

Для этого необходимо в разделе [ Конфигурация ] ([ Configuration ]) выбрать пользовательский пакет и на вкладке [ Схемы ] ([ Schemas ]) выполнить команду [ Добавить ] ([ Add ]) — [ Замещающий клиентский модуль ] ([ Replacing Client Module ]). Создание замещающего клиентского модуля подробно описано в статье "Создать клиентскую схему".



Для созданной схемы модуля в свойстве [ Родительский объект ] ([ Parent object ]) нужно установить значение "ForecastBuilder". После этого автоматически будут применены значения всех свойств схемы, взятые из родительского модуля.



Во вкладке [ Исходный код ] ([ Source Code ]) схемы нужно добавить исходный код родительской схемы, скопированный на предыдущем шаге, и сохранить схему.

### 3. Изменить метод открытия страницы плана

В исходном коде созданной схемы модуля ForecastBuilder необходимо заменить значения массива `valuePairs` в методе `openForecastPage()` класса `Terrasoft.configuration.BaseForecastsViewModel` на значения, соответствующие схеме объекта [ Счет ] ([ Invoice ]). Исходный код изменений приведен ниже (предыдущие значения закомментированы).

```
...
openForecastPage: function(moduleId, operation) {
    ...
    var valuePairs = [
        {
            name: "EntitySchemaUID",
            value: "bfb313dd-bb55-4e1b-8e42-3d346e0da7c5" //value: "ae46fb87-c02c-4ae8-ad31-a923
        },
        {
    
```

```

        name: "EntitySchemaName",
        value: "Invoice" //value: "Opportunity"
    }
];
...
},
...

```

Значение идентификатора `EntitySchemaUID` можно получить, выполнив следующий SQL-запрос к базе данных Creatio :

```

select lower(UId), Name from SysSchema
Where name = 'Invoice'
and ExtendParent = 0

```

После внесения изменений, схему необходимо сохранить.

## 4. Внести изменения в хранимую процедуру `tsp_RecalculateForecastFact`

Хранимая процедура `tsp_RecalculateForecastFact` выполняет пересчет значений фактов за выбранный временной период.

**Важно.** При составлении и выполнении SQL-запроса к базе данных следует быть предельно внимательным. Выполнение неправильно составленного SQL-запроса может привести к необратимому повреждению существующих данных и нарушению работы системы.

Поскольку для подсчета значений нужно использовать счета, а не продажи, то в процедуре необходимо внести следующие изменения (предыдущие значения закомментированы).

1. Изменить значение, хранящееся в переменной `@CompletedId`.

```

--SET @CompletedId = '{60D5310C-5BE6-DF11-971B-001D60E938C6}'
SET @CompletedId = '{698D39FD-52E6-DF11-971B-001D60E938C6}'

```

В этой переменной хранится идентификатор статуса полностью оплаченного счета. Значение переменной можно получить, выполнив следующий SQL-запрос к базе данных Creatio:

```

select Id, Name from InvoicePaymentStatus
where Name = 'Paid'

```

2. Изменить запрос, результат которого сохраняется в переменной `@MaxDueDate`.

```
--SET @MaxDueDate = (SELECT Convert(Date, MAX(DueDate), 104) FROM Opportunity o WHERE o.StageId
SET @MaxDueDate = (SELECT Convert(Date, MAX(StartDate), 104) FROM Invoice o WHERE o.PaymentStatu
```

В запросе выполняется поиск самого нового счета среди всех оплаченных счетов.

3. Изменить выражение подзапроса, хранящееся в переменной `@SQLText`. Именно в этом подзапросе реализована логика расчета факта и потенциала.

```
--Исходное значение
/*SET @SQLText = N'
SELECT
    (SELECT SUM(ISNULL(fiv.[Value], 0))
     FROM [ForecastItemValue] fiv
     WHERE fiv.[ForecastItemId] = @P5
       AND fiv.[PeriodId] = @P6
       AND fiv.[ForecastIndicatorId] = @P7
    ) PlanAmount,
    (SELECT SUM(ISNULL(o.[Amount], 0))
     FROM [Opportunity] o
     WHERE o.[StageId] = @P1
       AND o.[DueDate] >= @P2
       AND o.[DueDate] < @P3
       AND o.' + @ColumnName + N' = @P4
    ) FactAmount,
    (SELECT SUM(ISNULL(o.[Amount], 0) * ISNULL(o.[Probability], 0) / 100)
     FROM [Opportunity] o
     INNER JOIN [OpportunityInStage] ois ON ois.[OpportunityId] = o.[Id]
     INNER JOIN [OpportunityStage] os ON os.[Id] = ois.[StageId]
     WHERE os.[End] = 1
       AND ois.[DueDate] >= @P2
       AND ois.[DueDate] < @P3
       AND ois.[Historical] = 0
       AND o.' + @ColumnName + N' = @P4
    ) PotentialAmount*/
--Новое значение
SET @SQLText = N'
SELECT
    (SELECT SUM(ISNULL(fiv.[Value], 0))
     FROM [ForecastItemValue] fiv
     WHERE fiv.[ForecastItemId] = @P5
       AND fiv.[PeriodId] = @P6
       AND fiv.[ForecastIndicatorId] = @P7
    ) PlanAmount,
    (SELECT SUM(ISNULL(o.[Amount], 0))
     FROM [Invoice] o
```

```

WHERE o.[PaymentStatusId] = @P1
AND o.[StartDate] >= @P2
AND o.[StartDate] < @P3
AND o.' + @ColumnName + N' = @P4
) FactAmount,
(SELECT SUM(ISNULL(o.[Amount], 0))
FROM [Invoice] o
INNER JOIN [InvoicePaymentStatus] os ON os.[Id] = o.[PaymentStatusId]
WHERE os.[FinalStatus] = 0
AND o.[StartDate] >= @P2
AND o.[StartDate] < @P3
AND o.' + @ColumnName + N' = @P4
) PotentialAmount'

```

Для применения изменений нужно запустить выполнить SQL-сценарий.

## Результат выполнения примера

В результате расчет фактов и потенциалов будет выполняться не по продажам, а по счетам.

Результат расчета фактов по продажам

	Expected	Actual	Closed, %	Pipeline
Alpha Business	10,000	12,520	125	0
Axiom	5,000	6,200	124	0
Fast Works	0	0	0	0

Результат расчета фактов по счетам

The screenshot shows a sales forecast interface with four tabs at the top: CUSTOM FORECAST (selected), FORECAST BY CUSTOMER, FORECAST BY SALES DIVISION, and FORECAST BY SALES MANAGER. Below the tabs is a green 'ADD' button and a 'ACTIONS ▾' dropdown. The main area displays a grid of data for three companies: Alpha Business, Axiom, and Fast Works. The grid has four columns: Expected, Actual, Closed, %, and Pipeline. Red boxes highlight the 'Expected' column for Alpha Business (10,000) and the entire row for Axiom.

	1 Expected	April 2017 Actual	2 Closed, %	Pipeline
Alpha Business	10,000	11,300	113	19,502
Axiom	5,000	0	0	15,350
Fast Works	0	0	0	0

## Синхронизация с MS Exchange



Сложный

На базе механизма синхронизации Sync Engine в конфигурации реализована интеграция с различными сущностями MS Exchange по протоколу EWS (Exchange Web Services).

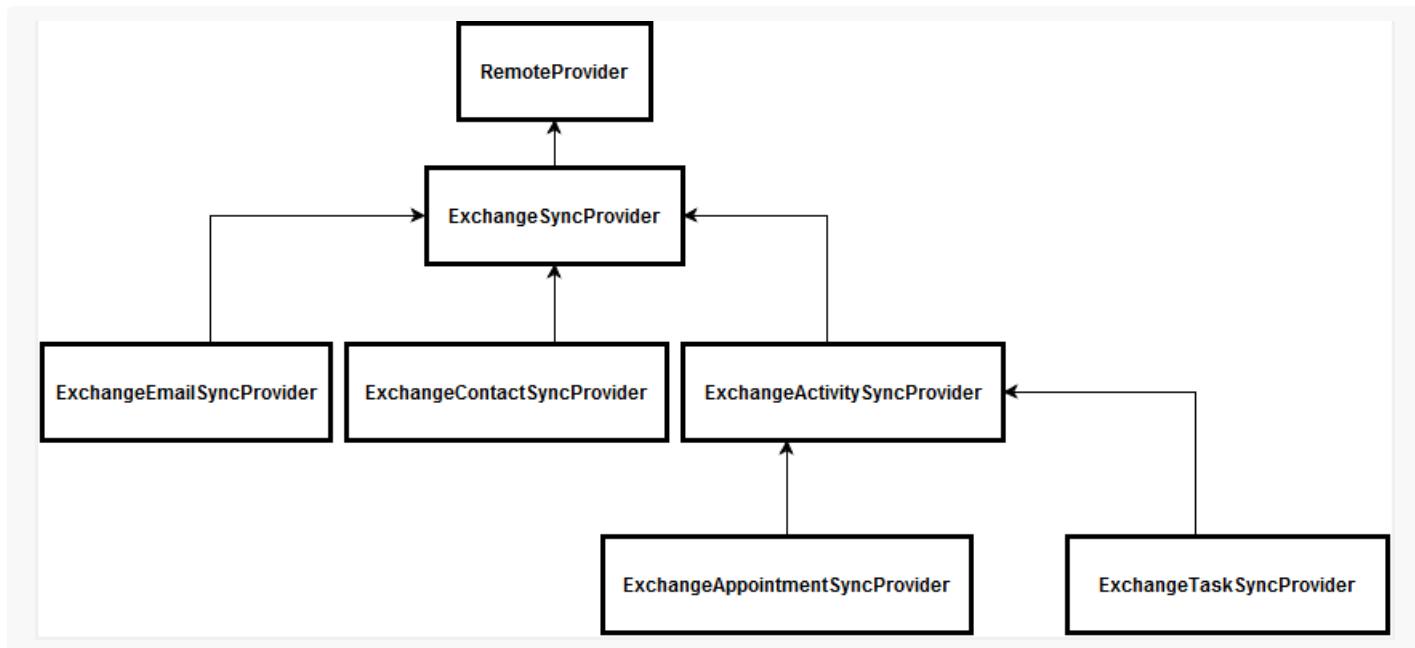
### Синхронизация задач с MS Exchange

Алгоритм синхронизации задач происходит в три этапа:

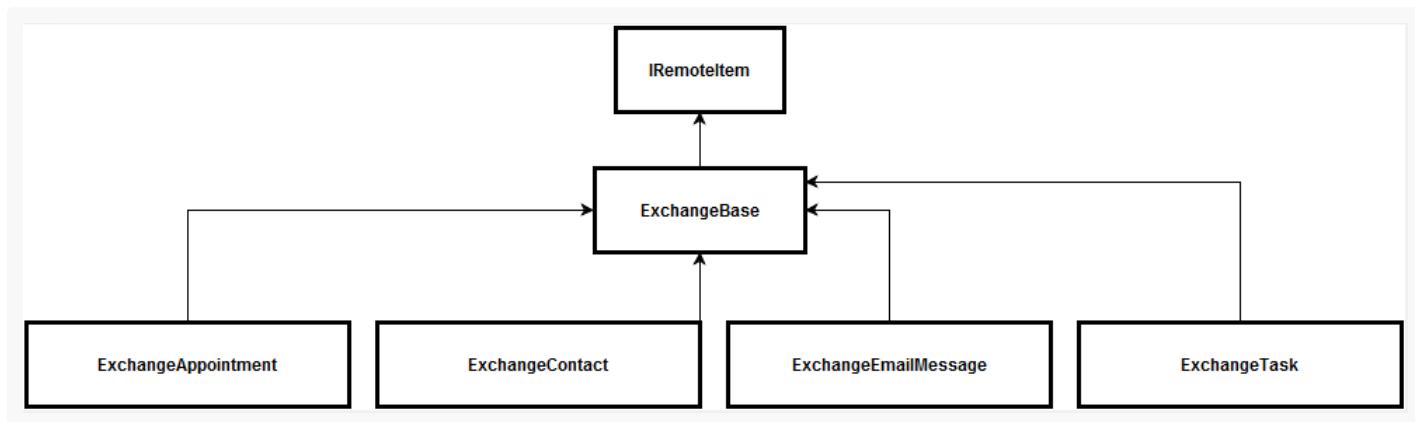
- Получение изменений из MS Exchange, их применение;
- Получение изменений из Creatio, их применение;
- Создание новых задач из Creatio в MS Exchange.

### Реализация интеграции

Как описано в [статье](#), для того, чтобы реализовать интеграцию с использованием данного механизма, необходим класс, реализующий логику работы с внешним хранилищем (наследник `RemoteProvider`). Вся иерархия классов провайдера изображена на рисунке. Также необходим класс, реализующий интерфейс `IRemoteItem`, который представляет один экземпляр элемента синхронизации (в данном случае — задачу MS Exchange). Вся иерархия `RemoteItem` изображена на рисунке.



Класс `ExchangeTaskSyncProvider` является провайдером для работы с внешним хранилищем MS Exchange, в нем реализована логика по выбору данных и сохранению изменений в Creatio и MS Exchange. Класс `ExchangeTask` реализует интерфейс `IRemoteItem`. В нем реализована логика заполнения данных в соответствующих системах.



## Синхронизируемые данные

Соответствие объектов Creatio и полей класса `ExchangeTask` отображено в таблице.

Соответствие объектов Creatio и полей класса `ExchangeTask`

<b>Объект Creatio</b>	<b>Поле объекта</b>	<b>Поле <code>ExchangeTask</code></b>
Activity	Title	Subject
	StartDate	StartDate
	DueDate	CompleteDate или DueDate в зависимости от того, завершена задача или нет.
	Priority	Importance
	Status	Status
	RemindToOwner	IsRemindSet
	RemindToOwnerDate	ReminderDueBy
	Notes	Body.Text

## Логика выбора данных для синхронизации

Для выбора изменений списка задач из отобранных для синхронизации папок MS Exchange используются следующие условия: выбрать задачи MS Exchange, которые были изменены после даты последней синхронизации задач, или задачу MS Exchange, которая не была отмечена, как ранее синхронизированная. Для задач MS Exchange, которые были изменены, находятся соответствующие активности в Creatio. Актуальные изменения применяются в соответствующей системе.

При выборе измененных активностей Creatio выбираются активности:

- имеющие запись в метаданных синхронизации как задача MS Exchange;
- автором которых является текущий пользователь;
- дата последнего изменения которых больше чем дата последней синхронизации.

При выборе новых активностей Creatio, устанавливается набор общих фильтров и фильтры, настраиваемые пользователем. Основными фильтрами являются:

1. Тип активности не равен "email".
2. У активности не установлен признак [ Отображать в расписании ].

Пользователь может указать группы активностей, которые будут экспортirоваться из Creatio.

## Заполнение полей [ Дата начала ], [ Дата завершения ] (опционально)

В объекте `ExchangeTask` есть несколько особенностей при работе с датами начала и завершения:

- В этих полях хранятся значения без времени. При изменении задачи в MS Exchange после синхронизации в Creatio применится дата из задачи MS Exchange, а время — из активности Creatio.
- Для даты завершения в `ExchangeTask` есть два поля — дата завершения (`due date`) и дата выполнения (`complete date`). В зависимости от статуса задачи, актуальной является одна из них — для не завершенных используется дата завершения, для завершенных — дата выполнения. Дата выполнения устанавливается текущей датой, пока задача не будет завершена.
- Дата начала и дата завершения — поля, необязательные для заполнения в MS Exchange. Если какое-то из них не установлено, используется текущая дата. Из-за этого могут возникать конфликты, так как в Creatio и дата начала, и дата завершения обязательны для заполнения, и дата начала должна быть меньше даты завершения.

## Синхронизация контактов с MS Exchange

Алгоритм синхронизации контактов происходит в три этапа:

1. Получение изменений из MS Exchange, их применение;
2. Получение изменений из Creatio, их применение;
3. Создание новых контактов из Creatio в MS Exchange.

## Реализация интеграции

Как описано в [статье](#), для того, чтобы реализовать интеграцию с использованием данного механизма, необходим класс, реализующий логику работы с внешним хранилищем, — наследник `RemoteProvider` и класс, реализующий интерфейс `IRemoteItem`, который представляет один экземпляр элемента синхронизации (в данном случае — контакт MS Exchange).

Схема иерархии `RemoteProvider`

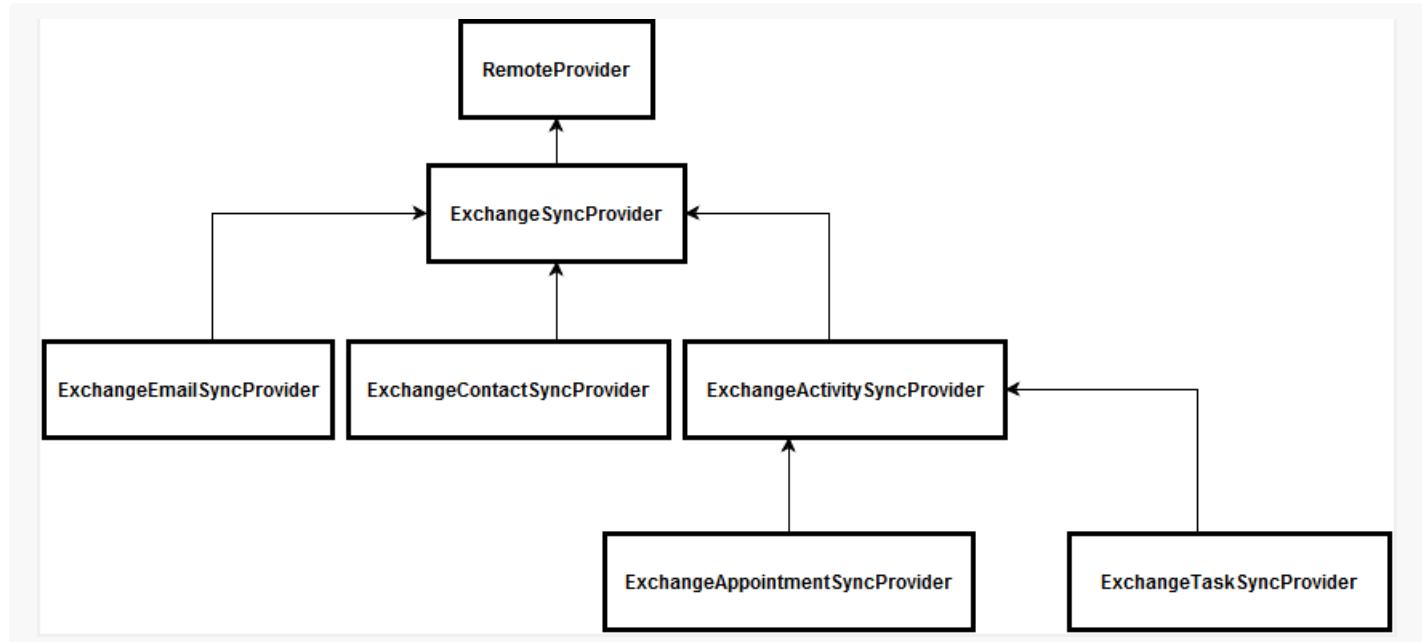
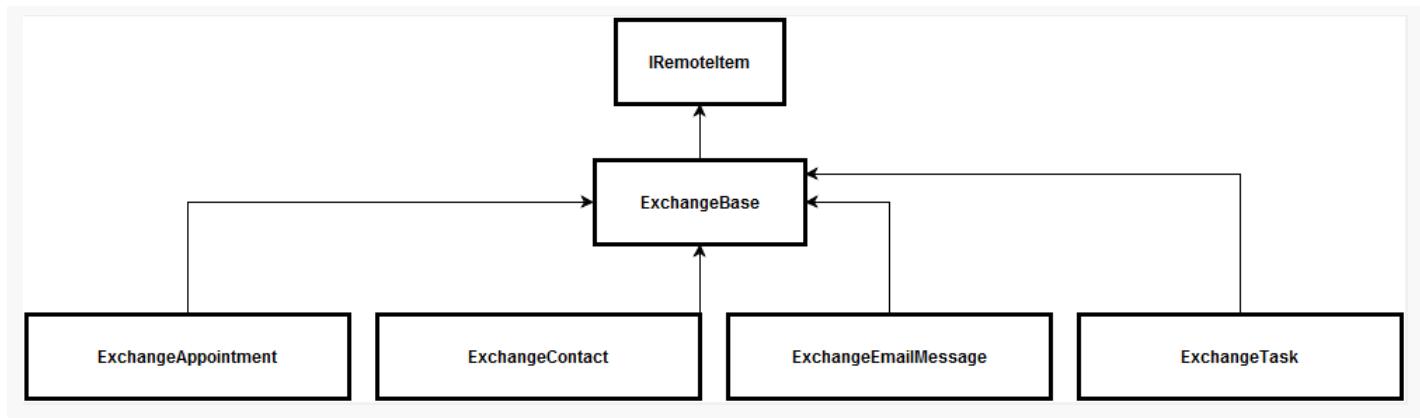


Схема иерархии `Remoteltem`



Для синхронизации контактов также используются следующие классы:

- Класс `ExchangeContactSyncProvider` является провайдером для работы с внешним хранилищем MS Exchange, в нем реализована логика по выбору данных и сохранению изменений в Creatio и MS Exchange.
- Класс `ExchangeContact` реализует интерфейс `IRemoteItem`. В нем реализована логика заполнения данных в соответствующих системах.
- Класс `ExchangeAddressDetailsSynchronizer` содержит утилитные методы для преобразования данных об адресах контактов.
- Класс `ExchangeEmailAddressDetailsSynchronizer` содержит утилитные методы для преобразования данных об email-адресах контактов.
- Класс `ExchangePhoneNumbersDetailsSynchronizer` содержит утилитные методы для преобразования данных о телефонах контактов.

Логика заполнения деталей вынесена в отдельные классы, так как существуют значительные отличия в форматах хранения данных в Creatio и MS Exchange, и требуется дополнительное преобразование.

## Синхронизируемые данные

Соответствие объектов Creatio и полей класса Contact MS Exchange отображено в таблице.

Соответствие объектов Creatio и полей класса Contact MS Exchange

<b>Объект Creatio</b>	<b>Поле объекта</b>	<b>Поле класса Contact MS Exchange</b>
Contact	Name	DisplayName
	Surname	Surname
	GivenName	GivenName
	MiddleName	MiddleName
	Account	CompanyName
	JobTitle	JobTitle
	Department	Department
	BirthDate	Birthday
	SalutationType	TitleTag
	Gender	GenderTag
ContactCommunication	Number	Значение в коллекции PhoneNumbers
	CommunicationType	Ключ для значения в коллекции PhoneNumbers
ContactAddresses	City	Поле City элемента коллекции PhysicalAddresses
	Country	Поле CountryOrRegion элемента коллекции PhysicalAddresses
	Region	Поле State элемента коллекции PhysicalAddresses
	Address	Поле Street элемента коллекции PhysicalAddresses
	Zip	Поле PostalCode элемента коллекции PhysicalAddresses
	AddressType	Ключ для значения в коллекции PhysicalAddresses

Соответствие типов средств связи представлено в таблице.

Соответствие типов средств связи Creatio и MS Exchange

Тип средства связи Creatio	Тип средства связи MS Exchange
Email	EmailAddress1, EmailAddress2, EmailAddress3
WorkPhone	BusinessPhone, BusinessPhone2
HomePhone	HomePhone
MobilePhone	MobilePhone

Соответствие типов адресов представлено в таблице.

Соответствие типов адресов Creatio и MS Exchange

Тип адреса Creatio	Тип адреса MS Exchange
HomeAddress	Home
BusinessAddress	Business

## Логика выбора данных для синхронизации

Для выбора изменений в списке контактов из отобранных для синхронизации папок MS Exchange используется следующий набор фильтров: выбирается контакт MS Exchange, который был изменен после даты последней синхронизации контактов, или контакт MS Exchange, который не был отмечен, как ранее синхронизированный. Для контактов, которые были изменены, находится соответствующий контакт в Creatio. Самые актуальные изменения применяются в соответствующей системе.

При выборе измененных контактов Creatio выбираются контакты:

- автором которых является текущий пользователь;
- дата последнего изменения которых больше чем дата последней синхронизации;
- контакт не использовался в первом этапе синхронизации.

На правила выбора новых контактов Creatio, помимо стандартного фильтра по автору и фильтра отсутствия метаданных синхронизации, влияют пользовательские настройки. Доступно три варианта настройки:

- Синхронизировать контакты сотрудников. При выборе этой настройки к запросу будет добавлен фильтр по колонке [тип контакта], и синхронизированы будут те контакты, у которых указан тип "Сотрудник".
- Синхронизировать контакты клиентов. При выборе этой настройки к запросу будет добавлен фильтр по колонке [тип контакта], и синхронизированы будут те контакты, у которых указан тип "Клиент".
- Синхронизировать контакты из определенных групп. При выборе этой настройки к запросу будут добавлены фильтры выбранных групп контактов.

## Дополнительные возможности

Возможность использовать расширенные ключи для контактов во внешнем хранилище

Ввиду особенностей работы MS Exchange возможна ситуация, когда при большом количестве контактов MS Exchange часть из них получит одинаковые идентификаторы, и, как следствие, синхронизация может некорректно определить соответствующий контакт в Creatio. Для обхода подобной ситуации реализована функциональность расширенных внешних ключей, которую можно включить при помощи настройки [ Использовать комбинированные идентификаторы для синхронизируемых контактов MS Exchange ]. Код настройки — `UseComplexExchangeContactId`. После ее включения может понадобиться повторная синхронизация.

## Синхронизация встреч с MS Exchange

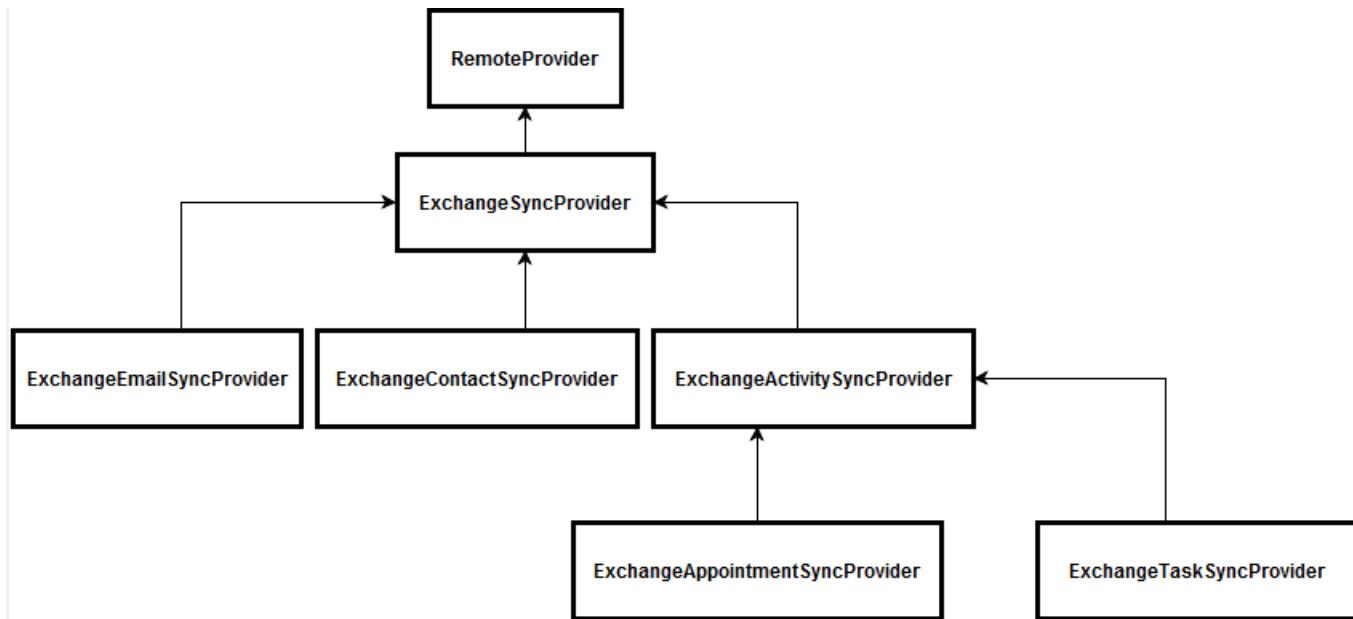
Синхронизация встреч Creatio происходит только для новых активностей или при изменении полей `Title`, `Location`, `StartDate`, `DueDate`, `Priority`, `Notes`. По этим полям формируется хэш, который хранится в дополнительных параметрах метаданных (в поле `ExtraParameters`). Если встреча была изменена в Creatio, и хэш из `ExtraParameters` не совпадает с новым хэшем, то эта встреча должна быть синхронизирована.

Алгоритм синхронизации встреч происходит в три этапа:

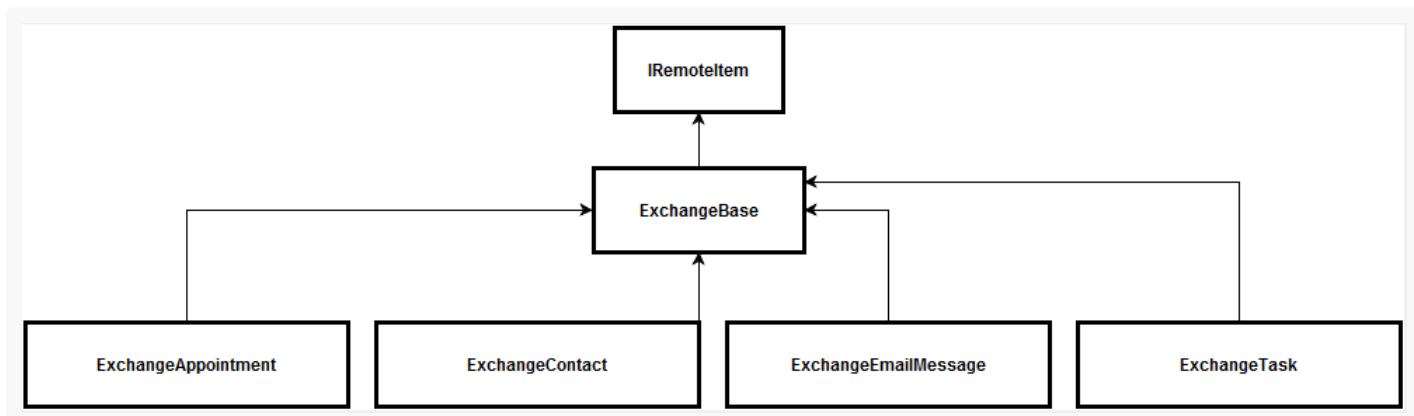
1. Получение изменений из MS Exchange, их применение;
2. Получение изменений из Creatio, их применение;
3. Создание новых встреч из Creatio в MS Exchange.

## Реализация интеграции

Как описано в [статье](#), для того чтобы реализовать интеграцию с использованием данного механизма, необходим класс, реализующий логику работы с внешним хранилищем, — наследник `RemoteProvider`, и класс, реализующий интерфейс `IRemoteItem`, который представляет один экземпляр элемента синхронизации (в данном случае — встречу MS Exchange).



Класс `ExchangeAppointmentSyncProvider` является провайдером для работы с внешним хранилищем Exchange. В нем реализована логика по выбору данных и сохранению изменений в Creatio и Exchange. Класс `ExchangeAppointment` реализует интерфейс `IRemoteItem`, и в нем реализована логика заполнения данных в соответствующих системах.



## Синхронизируемые данные

Соответствие объектов Creatio и полей класса `ExchangeAppointment` отображено в таблице.

Соответствие объектов Creatio и полей класса **ExchangeAppointment**

<b>Объект Creatio</b>	<b>Поле объекта</b>	<b>Соответствующее поле MS Exchange Appointment</b>
Activity	Title	Subject
	Location	Location
	StartDate	StartDate
	DueDate	CompleteDate или DueDate в зависимости от того, завершена встреча или нет.
	Priority	Importance
	Status	Заполняется по алгоритму:  Если статус не указан и дата окончания больше, чем текущая, — статус в Creatio устанавливается как "Новая встреча".  Если дата окончания меньше, чем текущая дата, — статус устанавливается как закрытая встреча со статусом "Информация получена".
	RemindToOwner	IsReminderSet
	RemindToOwnerDate	ReminderDueBy
	Notes	Body.Text
ActivityParticipant	InviteResponse	Если в MS Exchange установлен признак, что встреча принята или пользователь является организатором встречи, то в Creatio устанавливается признак "Встреча подтверждена". В противном случае устанавливается признак того, что встреча отменена.

## Логика выбора данных для синхронизации

Для выбора изменений в списке встреч из отобранных для синхронизации папок MS Exchange используется следующий набор фильтров: выбираются встречи MS Exchange, которые были изменены после даты последней синхронизации встреч, или встречи MS Exchange, которые не были отмечены, как ранее синхронизированные. Для встреч, которые были изменены, находятся соответствующие активности в Creatio. Самые актуальные изменения применяются в соответствующей системе.

При выборе измененных активностей Creatio выбираются активности:

- которые имеют запись в метаданных синхронизации как встречи MS Exchange по `RemoteId` (определяется уникальным идентификатором встречи в календаре `iCalId`);
- дата последнего изменения которых больше, чем дата последней синхронизации;
- одна встреча в Creatio соответствует нескольким встречам в MS Exchange для каждого участника.

При выборе новых активностей Creatio устанавливается набор общих фильтров и фильтры, настраиваемые пользователем. Основными фильтрами являются:

1. Тип активности не равен "Email".
2. У активности установлен признак [ *Отображать в расписании* ].

Пользователь может указать группы активностей, которые будут экспортirоваться из Creatio.

## Логика выбора участников встречи

При синхронизации активности из MS Exchange в Creatio на деталь [ Участники ] добавляются те контакты, у которых на детали [ Средства связи ] есть email из списка участников встречи в MS Exchange.

При синхронизации активностей из Creatio в MS Exchange участники встречи для MS Exchange заполняются основными email-адресами контактов.

# Интеграция с Webitel



Сложный

Интеграция [Webitel](#) реализована на клиенте в виде отдельных модулей Creatio. Состав модулей, входящих в интеграцию:

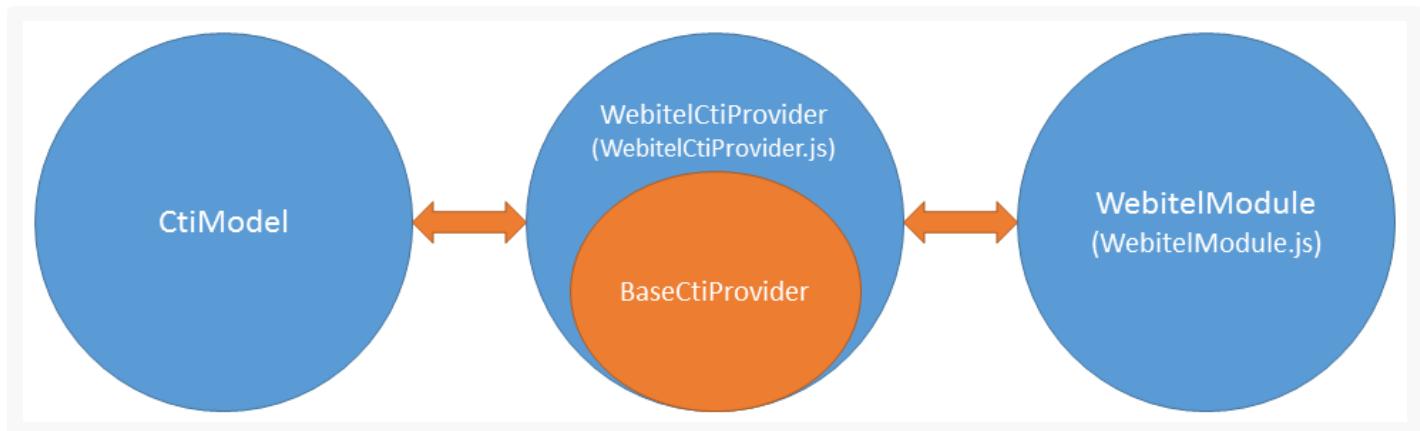
Пакет `WebitelCore` — модули, содержащие низкоуровневые взаимодействия с АТС Webitel с использованием модуля `Verto`, а также СТИ-панели на странице приложения Creatio.

Пакет `WebitelCollaborations` — реализация базовых интерфейсов для работы с Webitel в Creatio. Содержит модуль `WebitelCtiProvider`, реализующий класс `WebitelCtiProvider`, коннектор к Webitel, страницу настройки параметров подключения, справочник для редактирования пользователей Webitel непосредственно в Creatio, необходимые привязки данных.

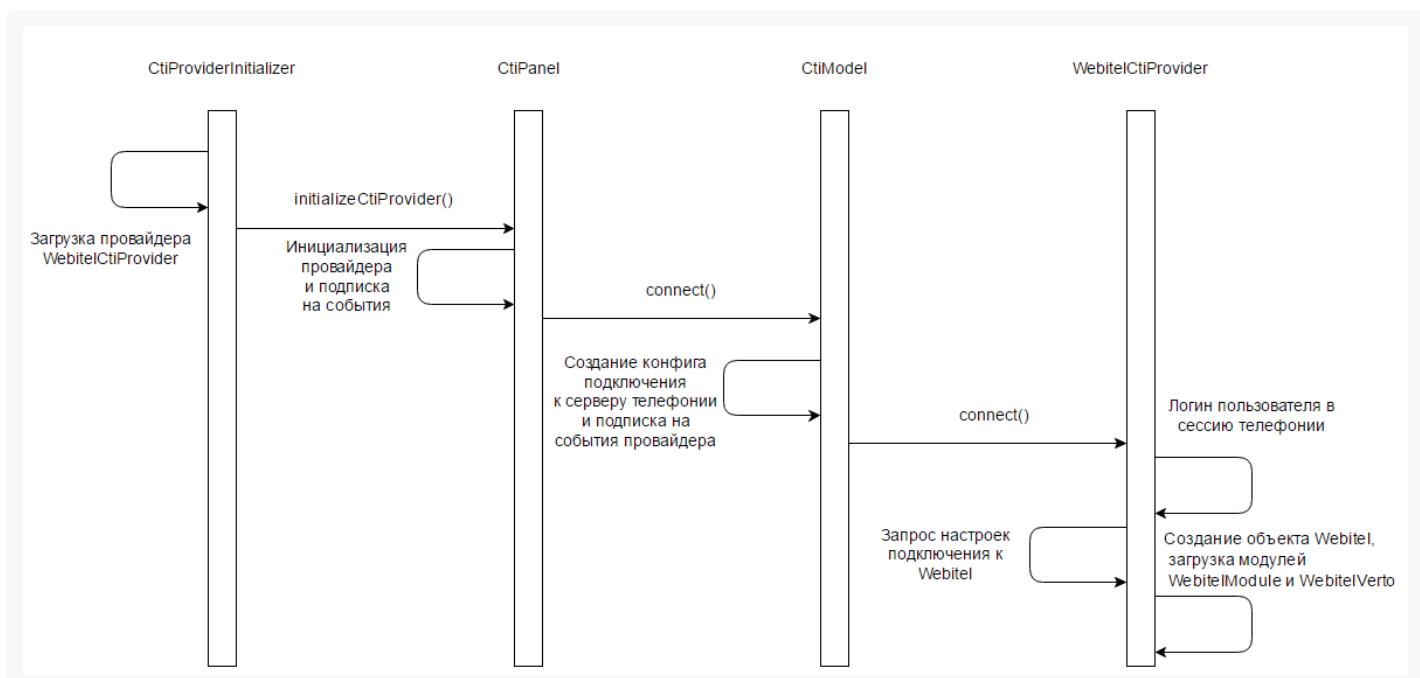
Подробно с архитектурой платформы Webitel можно ознакомиться в [документации](#).

## Взаимодействие компонентов

Класс `WebitelCtiProvider` (наследник класса `Terrasoft.BaseCtiProvider`) реализует необходимое взаимодействие между `ctiModel` и низкоуровневым глобальным объектом `Webitel` (модуль `WebitelCore.WebitelModule.js`).



Интеграция осуществляется следующим образом. Если пользователь установил системную настройку библиотеки интеграции Webitel, `ctiProviderInitializer` загружает модуль `WebitelCtiProvider`. Далее вызывается метод `init` в `WebitelCtiProvider`, который осуществляет логин пользователя в сессию телефонии (метод `LogInMsgServer` сервиса `MsgUtilService.svc`). Если логин был успешным, вызывается метод `connect`, который проверяет, что еще нет существующего подключения (свойство `this.isConnected` имеет значение `false` и `this.webitel` — пустое). После этого `connect` запрашивает настройки подключения к Webitel, которые хранятся в системных настройках `webitelConnectionString` и `webitelWebrtcConnectionString`.

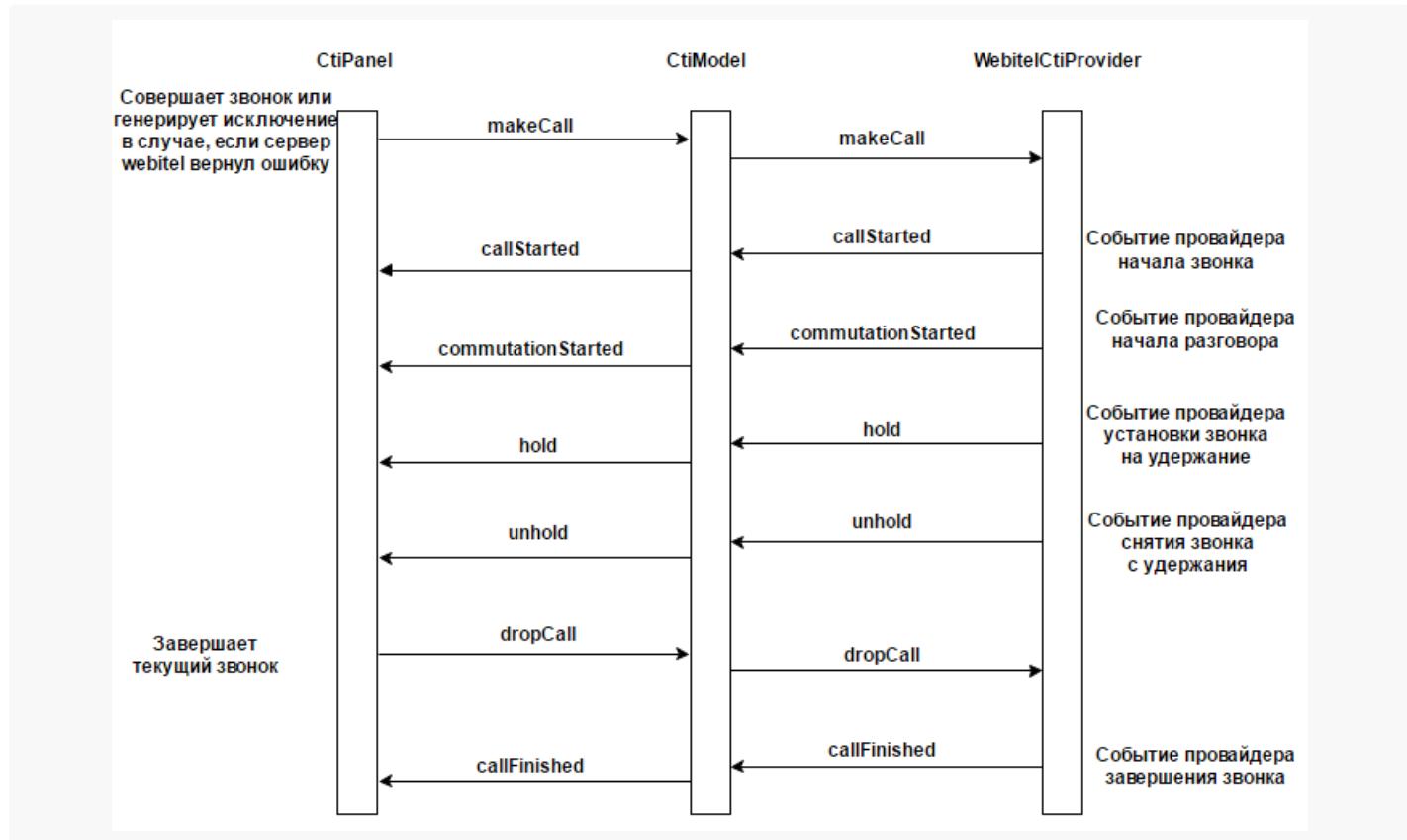


После получения системных настроек происходит получение настроек пользователя из справочника [ Пользователи Webitel ], используя метод `GetUserConnection` клиентского сервиса `WUserService`. После получения пользовательских настроек загружается модуль `WebitelModule` и `WebitelVerto`, если в настройках пользователя установлен признак [ *Use web phone* ]. Затем вызывается метод `onConnected`, в котором создается глобальный объект `Webitel`, где свойства заполняются настройками подключения. Происходит подписка на события объекта `Webitel` и вызывается метод `connect`, который осуществляет подключение по WebSocket, авторизацию в ATC Webitel и остальные низкоуровневые операции по

подключению. При возникновении события `onConnect` подключение считается успешным, и у пользователя появляется возможность работать со звонками. Во время работы коннектора `WebitelCtiProvider` реагирует на события объекта `Webitel`, обрабатывает их и, при необходимости, генерирует события коннектора, описанные в классе `Terrasoft.BaseCtiProvider`. Для управления звонками `WebitelCtiProvider` реализует абстрактные методы класса `Terrasoft.BaseCtiProvider`, используя методы объекта `Webitel`.

## Примеры взаимодействия CtiPanel, CtiModel и WebitelCtiProvider

Исходящий звонок оператора абоненту с установкой звонка на удержание абонентом, снятия с удержания абонентом и завершения звонка оператором.



## Список портов Webitel

- 871 — порт WebSocket для подключения к серверу Webitel и получения событий.
- 5060 и 5080 — сигнальные порты для подключенииз SIP-телефонов и провайдеров телефонии.
- 5066 — порт для подключения Web-телефона, сигнальный порт WebRTC.
- 4004 — порт для получения записей разговоров.

## События Webitel

События WebitelCtiProvider

Событие	Описание
onNewCall	Событие начала нового звонка.
onAcceptCall	Событие поднятия трубки.
onHoldCall	Событие удержания звонка.
onUnholdCall	Событие снятия звонка с удержания.
onDtmfCall	Событие тонового набора.
onBridgeCall	Событие соединения с каналом.
onUuidCall	Событие смены UUID звонка.
onHangupCall	Событие завершения звонка.
onNewWebRTCCall	Событие новой WebRTC-сессии.

## Обращения



Легкий

Creatio предоставляет возможность реализовать собственную логику получения временных параметров для расчета сроков в обращении. При расчете или перерасчете сроков в обращении вместо одной из базовых стратегий расчета будет использоваться стратегия, реализованная разработчиком.

Выбор конкретного правила для расчета происходит с использованием справочника [ Правила расчета сроков по обращениям ] ([ Case deadline calculation schemas ]).

Чтобы **добавить новое правило расчета**:

1. Создайте схему объекта, в которую добавьте колонки, необходимые для хранения расчетного времени реакции и разрешения, а также ссылки на календарь, сервисный договор и сервис.
2. Создайте справочник на основе созданной схемы объекта и заполните его значениями, которые необходимы для расчета временных параметров.
3. Добавьте схему исходного кода, в которой объявили класс-наследник абстрактного класса `BaseTermStrategy`. В классе реализуйте пользовательский механизм получения временных параметров реакции и разрешения обращения.
4. Подключите новое правило.

## Добавить новое правило расчета сроков в обращении

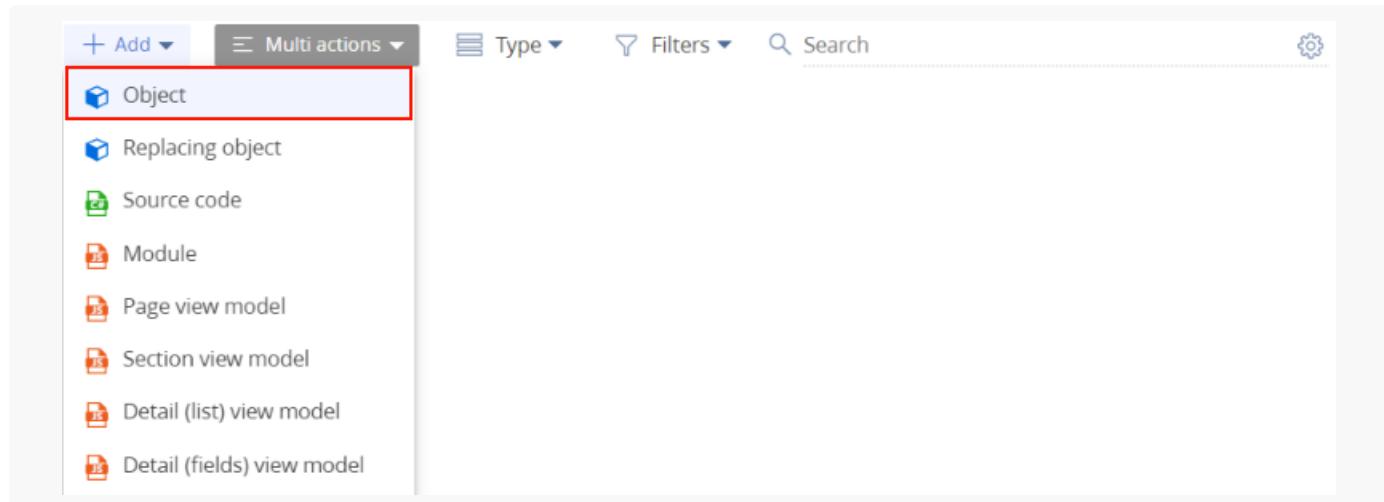


**Пример.** Добавить пользовательское правило для расчета временных параметров обращений по сервису Восстановление утерянных данных согласно договора [ 78 — *Elite Systems* ]. Для нового правила установить следующие значения:

- время реакции — 2 календарных часа;
- время разрешения — 1 рабочий день;
- используемый календарь — [ *Типовой календарь* ].

## 1. Создать схему объекта

1. [Перейдите в раздел \[ Конфигурация \]](#) ([ Configuration ]) и выберите пользовательский [пакет](#), в который будет добавлена схема.
2. На панели инструментов реестра раздела нажмите [ Добавить ] —> [ Объект ] ([ Add ] —> [ Object ]).



3. В дизайнере схем заполните **свойства схемы**:

- [ Код ] ([ Code ]) — "UsrServiceTestTerms".
- [ Заголовок ] ([ Title ]) — "ServiceTestTerms".
- [ Родительский объект ] ([ Parent object ]) — выберите [ Базовый объект ] ([ BaseEntity ]).

**General**

Code *	Title *
UsrServiceTestTerms	ServiceTestTerms
Package	Description
Custom	<input type="button" value="X"/>

**Inheritance**

Parent object	<input type="checkbox"/> Replace parent
BaseEntity	

Для применения заданных свойств нажмите [ Применить ] ([ *Apply* ]).

## 2. Добавить в объект колонки

В схеме объекта [ *ServiceTestTerms* ] создайте набор колонок со следующими свойствами:

Свойства добавляемых колонок

Название	Заголовок	Тип	Описание
UsrReactionTimeUnit	Единица измерения времени реакции (Response time unit)	Справочник [Единица времени] ([Time unit])	Указывает единицу измерения времени (календарные дни, часы и т.д.), по которой будет рассчитан параметр [Время реакции].
UsrReactionTimeValue	Значение времени реакции (Response time)	Целое (Integer)	Колонка для хранения значения срока реакции.
UsrSolutionTimeUnit	Единица измерения времени разрешения (Resolve time unit)	Справочник [Единица времени] ([Time unit])	Указывает единицу измерения времени (календарные дни, часы и т.д.), по которой будет рассчитан параметр [Время разрешения].
UsrSolutionTimeValue	Время разрешения (Resolution time)	Целое (Integer)	Колонка для хранения значения срока разрешения.
UsrCalendarId	Используемый календарь (Calendar that is used)	Справочник [Календарь] ([Calendar])	Календарь, по которому будут считаться сроки в обращении.
UsrServicePactId	Сервисный договор (Service agreement)	Справочник [Сервисный договор] ([Service agreement])	Ссылка на объект [Сервисный договор]. Добавлена для возможности осуществлять фильтрацию.
UsrServiceItemId	Сервис (Service)	Справочник [Сервис] ([Service])	Ссылка на объект [Сервис]. Добавлена для возможности осуществлять фильтрацию.

Опубликуйте схему объекта нажав кнопку [ Опубликовать ] ([ Publish ]).

## 2. Создать справочник и заполнить его необходимыми

## значениями

Для расчета сроков реакции и разрешения обращения необходимо предоставить их конкретные значения. Для этого на основе добавленной схемы [создайте справочник](#) со следующими свойствами:

- [ Название ] ([ Name ]) — "Пользовательские сроки реакции и разрешения" ("Custom response and resolution deadlines").
- [ Объект ] ([ Object ]) — "ServiceTestTerms".

The screenshot shows a form with the following fields:

- Name\*: Custom response and resolution deadlines
- Object\*: ServiceTestTerms
- List page
- Description

В наполнение созданного справочника добавьте запись с данными по условиям примера:

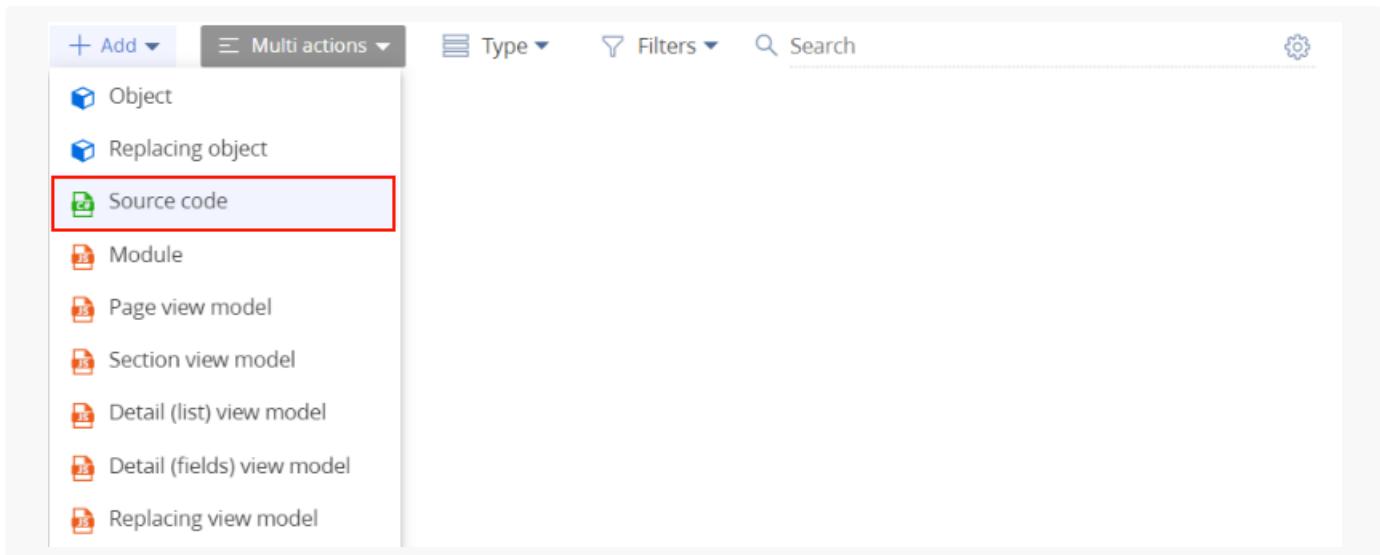
The screenshot shows the 'Lookups' screen with the following details:

- Header: Lookups, NEW, CLOSE, ACTIONS ▾, What can I do for you? ▾, Creatio, VIEW ▾
- Title: Custom response and resolution deadlines
- Filter: Filters/folders ▾
- Table headers: Response time unit, Response ..., Resolve time unit, Resolutio..., Service, Calendar that is used, Service agreement
- Table data:
 

Calendar hours	2	Calendar days	1	Lost data recovery	Default calendar	78 — Elite Systems
----------------	---	---------------	---	--------------------	------------------	--------------------

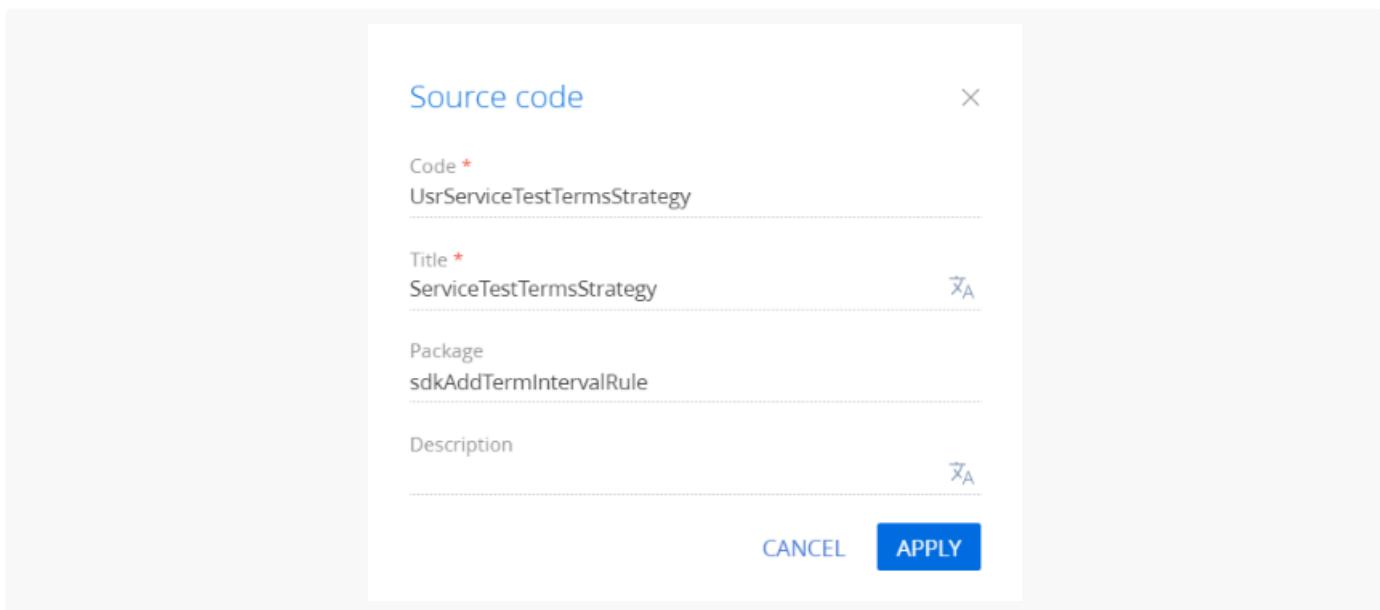
## 3. Реализовать класс для получения временных параметров

1. [Перейдите в раздел \[ Конфигурация \]](#) ([ Configuration ]) и выберите пользовательский [пакет](#), в который будет добавлена схема.
2. На панели инструментов реестра раздела нажмите [ Добавить ] —> [ Исходный код ] ([ Add ] —> [ Source code ]).



3. В дизайнере схем заполните **свойства схемы**:

- [Код] ([Code]) — "UsrServiceTestTermsStrategy".
- [Заголовок] ([Title]) — "ServiceTestTermsStrategy".



Для применения заданных свойств нажмите [Применить] ([Apply]).

4. Реализуйте логику получения временных параметров.

- В исходный код схемы добавьте класс-наследник абстрактного класса `BaseTermStrategy` пакета `Calendar`. В классе реализуйте параметризованный конструктор со следующими параметрами:
  - `UserConnection userConnection` — текущее подключение пользователя;
  - `Dictionary args,>` — аргументы, на основании которых будет выполняться расчет.
- Реализуйте объявленный в базовом классе абстрактный метод `GetTermInterval()`. Этот метод в качестве входного параметра принимает маску уже заполненных значений, на основании которой

будет принято решение о заполнении конкретных временных параметров возвращаемого класса `TermInterval`, реализующего интерфейс `ITermInterval`.

### UsrServiceTestTermsStrategy.cs

```
namespace Terrasoft.Configuration
{
    using System;
    using System.Collections.Generic;
    using Terrasoft.Common;
    using Terrasoft.Configuration.Calendars;
    using Terrasoft.Core;
    using Terrasoft.Core.Entities;
    using CalendarsTimeUnit = Calendars.TimeUnit;
    using SystemSettings = Terrasoft.Core.Configuration.SysSettings;
    public class ServiceTestTermsStrategy: BaseTermStrategy<CaseTermInterval, CaseTermStates>
    {
        // Класс-контейнер для хранения данных, полученных из точки входа.
        protected class StrategyData
        {
            public Guid ServiceItemId {
                get;
                set;
            }
            public Guid ServicePactId {
                get;
                set;
            }
        }
        // Поле, хранящее данные, полученные из точки входа.
        protected StrategyData _strategyData;
        // Параметризованный конструктор, необходимый для корректной
        // инициализации классом-селектором.
        public ServiceTestTermsStrategy(UserConnection userConnection, Dictionary<string, object> args)
            : base(userConnection) {
            _strategyData = args.ToObject<StrategyData>();
        }
        // Метод, который получает данные и возвращает их в экземпляре класса CaseTermInterval.
        public override CaseTermInterval GetTermInterval(CaseTermStates mask) {
            var result = new CaseTermInterval();
            // Создание EntitySchemaQuery запроса.
            var esq = new EntitySchemaQuery(UserConnection.EntitySchemaManager, "UsrServiceTestTerms");
            // Добавление колонок в запрос.
            string reactionTimeUnitColumnName = esq.AddColumn("UsrReactionTimeUnit.Code").Name;
            string reactionTimeValueColumnName = esq.AddColumn("UsrReactionTimeValue").Name;
            string solutionTimeUnitColumnName = esq.AddColumn("UsrSolutionTimeUnit.Code").Name;
            string solutionTimeValueColumnName = esq.AddColumn("UsrSolutionTimeValue").Name;
            string calendarColumnName = esq.AddColumn("UsrCalendarId.Id").Name;
        }
    }
}
```

```
// Добавление фильтров в запрос.
var esqFirstFilter = esq.CreateFilterWithParameters(FilterComparisonType.Equal, "CaseTermStates", "ContainsResponse", "1");
var esqSecondFilter = esq.CreateFilterWithParameters(FilterComparisonType.Equal, "CaseTermStates", "ContainsResolve", "1");
esq.Filters.Add(esqFirstFilter);
esq.Filters.Add(esqSecondFilter);
// Выполнение и обработка результатов запроса.
EntityCollection entityCollection = esq.GetEntityCollection(UserConnection);
if (entityCollection.IsEmpty()) {
    // Добавление к возвращаемому значению времени реакции.
    if (!mask.HasFlag(CaseTermStates.ContainsResponse)) {
        result.ResponseTerm = new TimeTerm {
            Type = entityCollection[0].GetTypedColumnValue<CalendarsTimeUnit>(rea
                Value = entityCollection[0].GetTypedColumnValue<int>(reactionTimeValu
                CalendarId = entityCollection[0].GetTypedColumnValue<Guid>(calendarCo
            );
    }
    // Добавление к возвращаемому значению времени разрешения.
    if (!mask.HasFlag(CaseTermStates.ContainsResolve)) {
        result.ResolveTerm = new TimeTerm {
            Type = entityCollection[0].GetTypedColumnValue<CalendarsTimeUnit>(sol
                Value = entityCollection[0].GetTypedColumnValue<int>(solutionTimeValu
                CalendarId = entityCollection[0].GetTypedColumnValue<Guid>(calendarCo
            );
    }
}
return result;
}
```

Опубликуйте схему, нажав кнопку [ Опубликовать ] ([ Publish ]).

#### 4. Подключить новое правило

[Добавьте значение в справочник](#) [ Правила расчета сроков по обращениям ] ([ Case deadline calculation schemas ]). В колонке [ Обработчик ] ([ Handler ]) укажите полное квалифицированное имя созданного класса (с указанием пространства имен).

При необходимости в колонке [ Альтернативное правило ] ([ *Alternative schema* ]) можно указать правило, по которому будут рассчитаны сроки в том случае, если расчет сроков с учетом текущего правила не представляется возможным. При этом следует учитывать, что если любой из временных параметров не будет рассчитан классом стратегии, то будет создан экземпляр класса альтернативной стратегии. Если же и альтернативная стратегия не сможет рассчитать сроки, то будет создана ее альтернативная стратегия, таким образом формируется очередь правил.

Для добавленной записи необходимо установить признак [ По умолчанию ] ( [ Default ] ).

Пример добавленной в справочник [ Правила расчета сроков по обращениям ] ([ Case deadline calculation schemas ]) записи показан на рисунке.

Lookups

Case deadline calculation schemas

Filters/folders ▾

Name	Description	Handler	Default	Alternative schema
Strategy for 78 - Elite systems		Terrasoft.Configuration.ServiceTestTermsStrategy	<input checked="" type="checkbox"/>	<input type="button" value="Search"/>
By priority in SLA level		Terrasoft.Configuration.CaseTermStrategyByPriority...	No	By priority <input type="button" value="Edit"/> <input type="button" value="Delete"/>
By service		Terrasoft.Configuration.CaseTermStrategyByService	No	

## Результат выполнения примера

В результате для обращений по сервисному договору [ 78 — Elite Systems ] для сервиса [ Восстановление утерянных данных ] ([ Lost data recovery ]) будут применены новые правила расчета сроков реакции и разрешения.

Case #SR00000016

What can I do for you?

Resolution time  
6/11/2018 11:44 AM 1d 00:00

Priority  
Medium

Contact

Account  
Elite Systems

SLA  
78 — Elite Systems

Category  
Incident

Service  
Lost data recovery

Configuration item

Assignees group

New  In progress  Waiting for response  Resolved  Closed

NEXT STEPS (0)

< PROCESSING CLOSURE AND FEEDBACK CASE INFORMATION TIMELINE ATTACHMENTS FEED >

Subject\* \_\_\_\_\_

Description \_\_\_\_\_

Source Call Support line 1st-line support

**Terms**

Registration date* 6/10/2018 <input type="button" value="Lock"/> 11:44 AM	Resolution time 6/11/2018 11:44 AM
Response time 6/10/2018 1:44 PM	First resolution time _____
Actual response time _____	Actual resolution time _____
Remaining: 02:00 <input type="button" value="Clock"/>	Remaining: 1d 00:00 <input type="button" value="Clock"/>
Closed on _____	

# Интеграция с Asterisk



Для взаимодействия с сервером [Asterisk](#) используется AMI ([Asterisk Manager API](#)). Manager API позволяет клиентским программам соединяться с сервером Asterisk, используя TCP/IP протокол. Данный API ([Application Programming Interface](#)) позволяет считывать события, происходящие в автоматической телефонной станции (АТС), и отправлять команды управления звонком.

**На заметку.** Приложение поддерживает интеграцию с Asterisk версии 13.

Для коммуникации между АТС Asterisk и подсоединенным Manager API клиентом используется простой текстовый построчный протокол вида: "параметр: значение". Окончание строки определяется последовательностью перевода строки и возврата каретки ([CRLF](#)).

**На заметку.** В дальнейшем для набора строк вида "параметр: значение", после которых идет пустая строка, содержащая только CRLF, для упрощения будет использоваться термин "пакет".

## Настройка конфигурационного файла сервиса Messaging Service для интеграции Creatio с Asterisk

Для работы интеграции с Creatio необходимо установить Terrasoft Messaging Service (TMS) на выделенном компьютере, который будет использоваться в качестве сервера данной интеграции. В конфигурационном файле `Terrasoft.Messaging.Service.exe.config` необходимо установить следующие параметры для коннектора Asterisk.

### Установка параметров для коннектора Asterisk

```
<asterisk filePath="" url="Имя_или_адрес_сервера_Asterisk" port="Порт_сервера_Asterisk" userName=
```

## Порты для интеграции Creatio с Asterisk

- TMS принимает от браузера WebSocket подключение на порт 2013 по протоколу TCP.
- TMS подключается к серверу Asterisk по умолчанию на порт 5038.

## Сервис Terrasoft Messaging Service для интеграции Creatio с Asterisk

Интеграционная часть Messaging Service реализована в ядре основного решения Creatio в библиотеке `Terrasoft.Messaging.Asterisk`.

Основные классы библиотеки:

- `AsteriskAdapter` — класс, преобразующий события Asterisk в высокоуровневые события модели звонка, используемой в интеграции Creatio.
- `AsteriskManager` — класс, использующийся для создания и удаления пользовательского соединения с

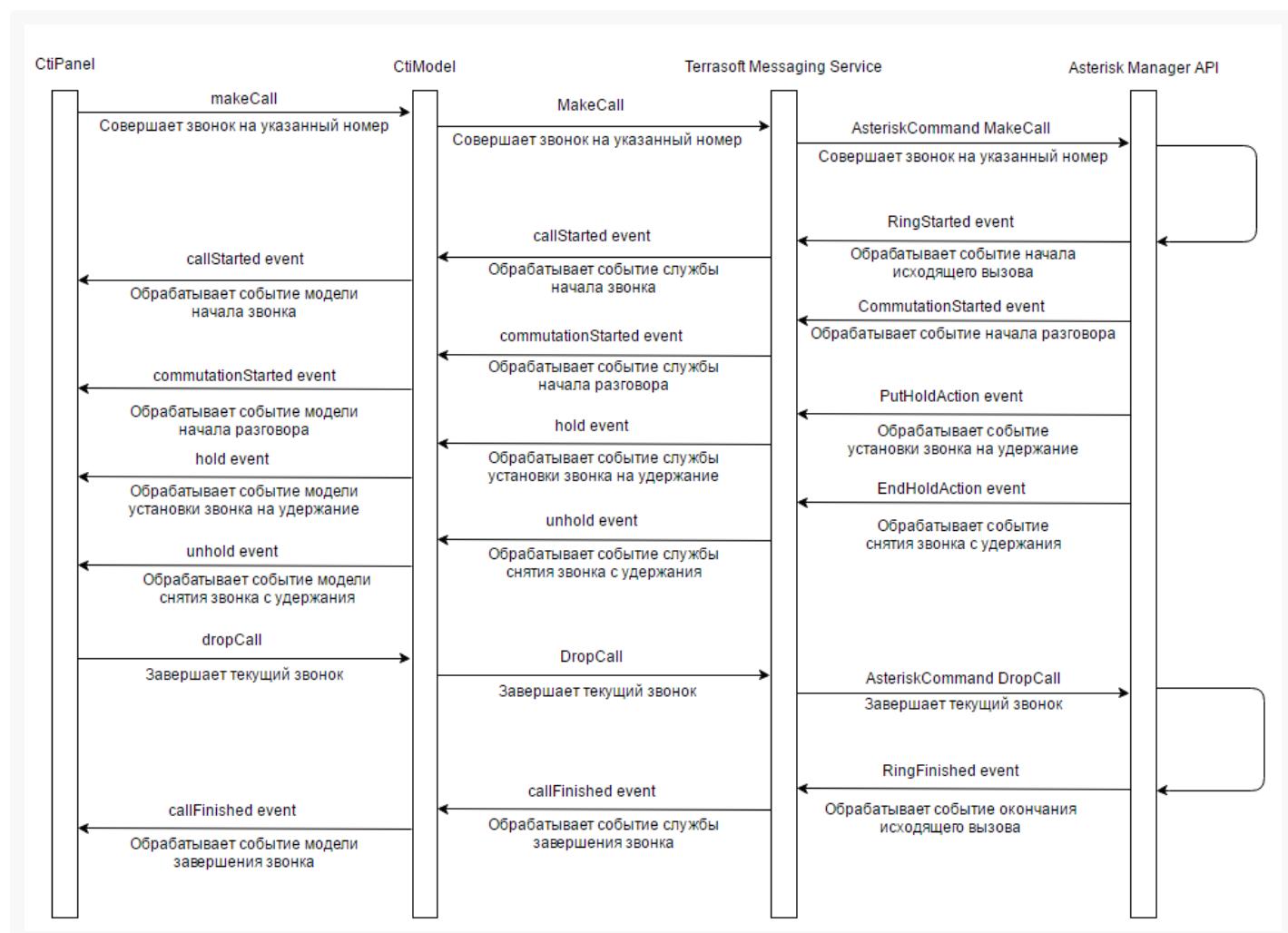
сервером Asterisk.

- `AsteriskConnection` — класс, представляющий собой пользовательское соединение при интеграции с Asterisk.
- `AsteriskClient` — класс, использующийся для отправки команд на сервер Asterisk.

## Пример взаимодействия CtModel, Terrasoft Messaging Service и Asterisk Manager API

Исходящий звонок оператора абоненту с установкой звонка на удержание абонентом, снятия с удержания абонентом и завершения звонка оператором.

Последовательность возникновения событий при звонке для данного примера:



В таблице приведен пример обработки событий — как данные события интерпретируются TMS, какие значения из приведенных событий используются при обработке входящего звонка.

События Asterisk

Asterisk log	TMS	Action
Сообщество начало и поборолось		

```
{
  Event: newchannel
  Channel: <channel_name>
  UniqueID: <unique_id>
}
```

Создается канал и добавляется в коллекцию

```
new AsteriskChannel({
  Name: <channel_name>
  UniqueID: <unique_id>
});
```

```
{
  Event: Hold
  UniqueID: <unique_id>
  Status: "On"
}
```

В коллекции каналов ищется канал по `<unique_id>` и с помощью метода `fireEvent` генерируется событие

`PutHoldAction`

```
{
  Event: Hold
  UniqueID: <unique_id>
  Status: "Off"
}
```

В коллекции каналов ищется канал по `<unique_id>` и с помощью метода `fireEvent` генерируется событие

`EndHoldAction`

```
{
  Event: Hangup
  UniqueID: <unique_id>
}
```

В коллекции каналов ищется канал по `<unique_id>` и с помощью метода `fireEvent` генерируется событие

`RingFinished`

```
{
  Event: Dial
  SubEvent: Begin
  UniqueID: <unique_id>
}
```

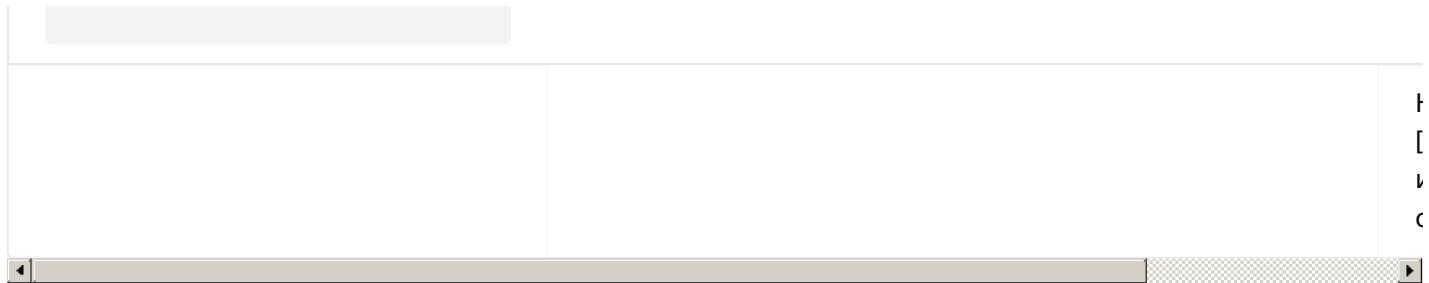
В коллекции каналов ищется канал по `<unique_id>`, заполняются данные и с помощью метода `fireEvent` генерируется событие

`RingStarted`

```
{
  Event: Bridge
  UniqueID: <unique_id>
}
```

В коллекции каналов ищется канал по `<unique_id>` и с помощью метода `fireEvent` генерируется событие

`CommutationStarted`



## События Asterisk

Подробный список событий и информация об их параметрах приведены в [документации Asterisk](#).

# Сервис машинного обучения



Сложный

## Общие принципы работы сервиса машинного обучения

Сервис машинного обучения (сервис прогнозирования значений справочного поля) использует методы статистического анализа для обучения на основании набора исторических данных. Историческими данными могут быть, например, обращения в службу поддержки за год. При этом используется текст обращения, а также дата и категория контрагента. Результирующим является поле [ Группа ответственных ].

## Схема взаимодействия Creatio с сервисом прогнозирования

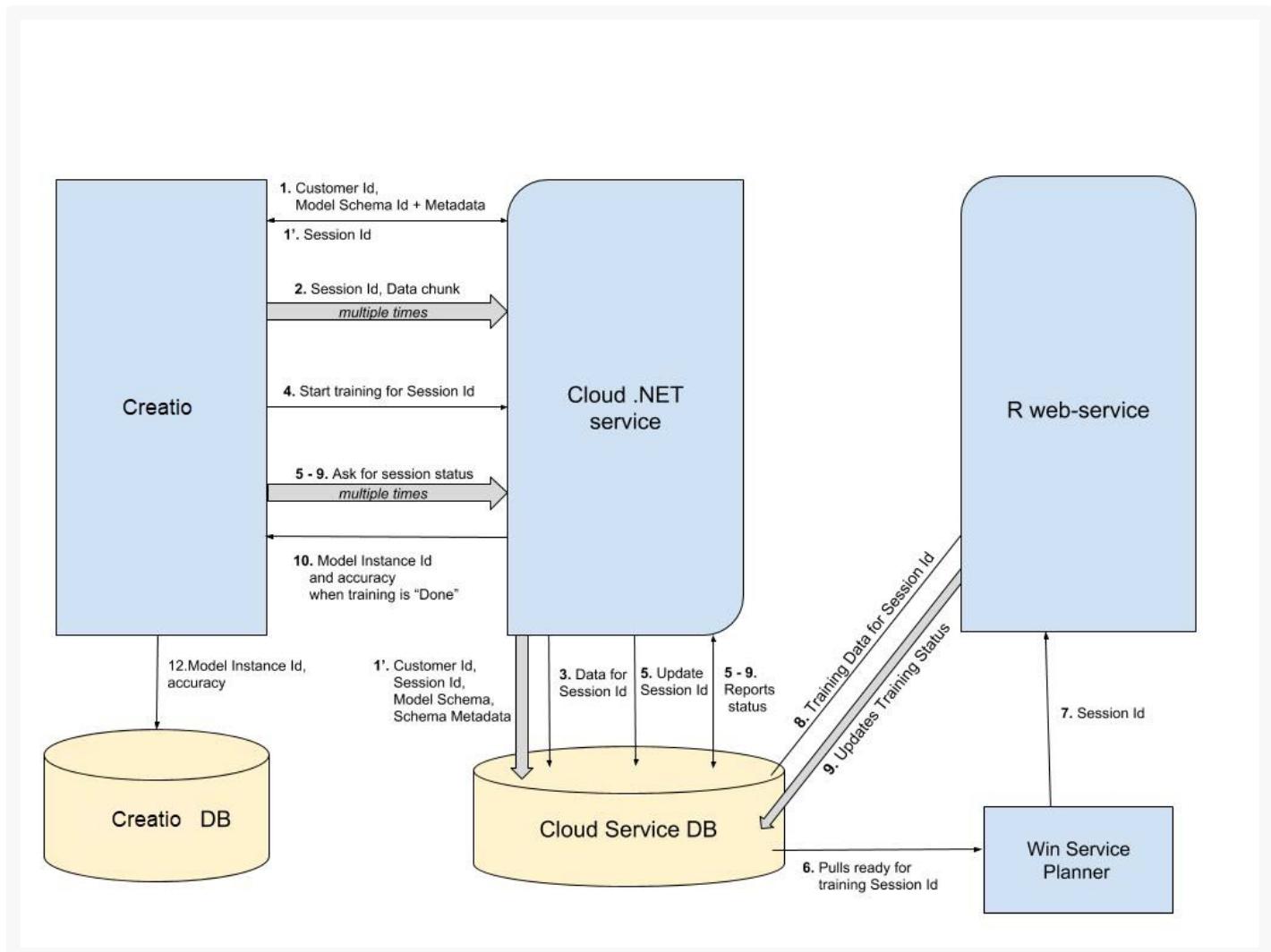
Существует два основных этапа работы Creatio для каждой модели: обучение и прогнозирование.

**Модель прогнозирования** — это алгоритм, который строит прогнозы и позволяет автоматически принимать полезное решение на основе исторических данных.

### Обучение

На этапе обучения выполняется "тренировка" сервиса. Основные шаги обучения:

- Установление сессии передачи данных и обучения.
- Последовательная выборка порции данных для модели и их загрузка в сервис.
- Запрос на постановку в очередь для обучения.
- Training engine для обучения модели обрабатывает очередь, обучает модель и сохраняет ее параметры во внутреннее хранилище.
- Creatio периодически опрашивает сервис для получения статуса модели.
- Как только для статуса модели установлено значение Done — модель готова для прогнозирования.



## Прогнозирование

Задача прогнозирования выполняется через вызов облачного сервиса с указанием Id экземпляра модели и данных для прогноза. Результат работы сервиса — набор значений с вероятностями, который сохраняется в Creatio в таблице `MLPrediction`.

Если существует прогноз в таблице `MLPrediction` по определенной записи сущности, то на странице записи автоматически отображаются спрогнозированные значения для поля.



## Настройки и типы данных Creatio для работы с сервисом прогнозирования

## Настройки Creatio

Настройки Creatio, которые необходимо выполнить для работы с сервисом прогнозирования, описаны в статье [Заполнить настройки Creatio](#).

### Расширение логики обучения модели

Описанная выше цепочка классов вызывает и создает экземпляры друг друга посредством IOC контейнера `Terrasoft.Core.Factories.ClassFactory`.

Если необходимо заменить логику какого-либо компонента, то нужно реализовать соответствующий интерфейс. При запуске приложения следует выполнить привязку интерфейса в собственной реализации.

Интерфейсы для расширения логики:

- `IMLModelTrainerJob` — реализация этого интерфейса позволит изменить набор моделей для обучения.
- `IMLModelTrainer` — отвечает за логику загрузки данных для обучения и обновления статуса моделей.
- `IMLServiceProxy` — реализация этого интерфейса позволит выполнять запросы к произвольному предиктивному сервису.

### Вспомогательные классы для прогнозирования

Вспомогательные (утилитные) классы для прогнозирования позволяют реализовать два базовых кейса:

1. Прогнозирование в момент создания или обновления записи какой-либо сущности на сервере.
2. Прогнозирование при изменении сущности на странице записи.

В случае прогнозирования на стороне сервера Creatio — создается бизнес-процесс, который реагирует на сигнал создания/изменения сущности, вычитывает набор полей и вызывает сервис прогнозирования. При получении корректного результата он сохраняет набор значений поля с вероятностями в таблицу `MLClassificationResult`. При необходимости бизнес-процесс записывает отдельное значение (например, с наибольшей вероятностью) в соответствующее поле сущности.

## Составление запросов на выборку данных для модели машинного обучения

Для выборки тренировочных данных или данных для прогнозирования сервиса машинного обучения используется экземпляр класса `Terrasoft.Core.DB.Select`. Он динамически интерпретируется с помощью `Terrasoft.Configuration.ML.QueryInterpreter`.

**Важно.** Интерпретатор `QueryInterpreter` не допускает использования лямбда-выражений.

При составлении выражения запроса в качестве аргумента типа `Terrasoft.Core.UserConnection` в конструкторе `Select` следует использовать предоставляемую переменную `userConnection`. Также обязательным в выражении запроса является наличие колонки с псевдонимом “`Id`” — уникальным идентификатором экземпляра целевого объекта.

Так как выражение `Select` может быть довольно сложным, то для упрощения его читаемости были добавлены следующие возможности:

- Динамическое добавление типов для интерпретатора.
- Использование локальных переменных.
- Использование утилитного класса `Terrasoft.Configuration.QueryExtensions`.

## Динамическое добавление типов для интерпретатора

Существует возможность динамически добавлять типы для интерпретатора. Для этого класс `QueryInterpreter` предоставляет методы `RegisterConfigurationType` и `.RegisterType`. Их можно напрямую использовать в выражении. Например, вместо непосредственного использования идентификатора типа:

```
new Select(userConnection)
    .Column("Id")
    .Column("Body")
    .From("Activity")
    .Where("TypeId").IsEqual(Column.Parameter("E2831DEC-CFC0-DF11-B00F-001D60E938C6"));
```

можно использовать название константы из динамически зарегистрированного перечисления:

```
RegisterConfigurationType("ActivityConsts");
new Select(userConnection)
    .Column("Id")
    .Column("Body")
    .From("Activity")
    .Where("TypeId").IsEqual(Column.Parameter(ActivityConsts.EmailTypeID));
```

## Использование локальных переменных

Существует возможность использовать локальные переменные для предотвращения дублирования кода и более удобного структурирования. Ограничение: тип переменной должен быть статически вычисляем и определяется ключевым словом `var`.

Например, запрос с повторяющимся использованием делегатов:

```
new Select(userConnection)
    .Column("Id")
    .Column("Body")
    .From("Activity")
    .Where("CreatedOn").IsGreater(Func.DateAddMonth(-1, Func.CurrentDateTime()))
    .And("StartDate").IsGreater(Func.DateAddMonth(-1, Func.CurrentDateTime()));
```

можно переписать следующим образом:

```
var monthAgo = Func.DateAddMonth(-1, Func.CurrentDateTime());

new Select(userConnection)
    .Column("Id")
    .Column("Body")
    .From("Activity")
    .Where("StartDate").IsGreater(monthAgo)
    .And("ModifiedOn").IsGreater(monthAgo);
```

## Подключение веб-сервиса к функциональности машинного обучения

**Важно.** Возможность подключения пользовательского веб-сервиса к функциональности машинного обучения доступна в Creatio версии 7.16.2 и выше.

Задачи машинного обучения (классификация, скоринг, числовая регрессия) или другие подобные задачи (например, прогнозирование оттока клиентов) можно реализовать с помощью веб-сервиса. Эта статья описывает процесс подключения к Creatio пользовательского веб-сервиса, реализующего задачу машинного обучения.

Общий порядок действий при подключении пользовательского веб-сервиса к сервису машинного обучения:

1. Создайте веб-сервис — движок машинного обучения.
2. Расширьте список задач сервиса машинного обучения.
3. Реализуйте модель машинного обучения.

### Создание веб-сервиса — движка машинного обучения

Пользовательский веб-сервис должен реализовать контракт для обучения и выполнения прогнозов по существующей модели. Пример контракта можно посмотреть через Swagger сервиса машинного обучения Creatio.

#### Адрес контракта сервиса машинного обучения Creatio

<https://demo-ml.bpmonline.com/swagger/index.html#/MLService>

Обязательные методы:

- `/session/start` — начало сессии обучения модели.

- `/data/upload` — передача данных в рамках открытой сессии обучения.
- `/session/info/get` — получение информации о состоянии сессии.
- `<пользовательский метод начала обучения>` — метод, который будет вызван Creatio по завершении процесса передачи данных. Процесс обучения модели не должен быть завершен до завершения выполнения этого метода. Процесс обучения может длиться произвольное количество времени (десятки минут и даже часы). Когда обучение завершено, метод `/session/info/get` должен будет вернуть состояние сессии обучения `Done` или `Error` в зависимости от результата обучения. Кроме этого, если модель успешно обучена, то необходимо вернуть информацию об экземпляре модели — `ModelSummary`: тип метрики, значение метрики, идентификатор экземпляра и другие.
- `<пользовательский метод прогнозирования>` — метод произвольной сигнатуры, который будет выполнять прогнозирование данных на основании идентификатора экземпляра обученной модели.

Процесс разработки веб-сервиса с использованием IDE Microsoft Visual Studio описан в [статье](#).

## Расширение списка задач сервиса машинного обучения

Для расширения списка задач машинного обучения в Creatio необходимо дополнить справочник `[MLProblemType]` новой записью. Необходимо указать следующие параметры:

- `Service endpoint Url` — адрес работающего сервиса машинного обучения.
- `Training endpoint` — путь метода начала обучения.
- `Prediction endpoint` — путь метода прогнозирования.

**Важно.** Для дальнейшей работы необходимо знать идентификатор созданной записи. Посмотреть `Id` можно в таблице `[dbo.MLProblemType]` базы данных.

## Реализация модели машинного обучения

Для настройки и отображения модели машинного обучения, возможно, потребуется расширить схему карточки `MLModelPage`.

### Реализация IMLPredictor

Необходимо реализовать метод `Predict`, который на вход получит данные, выгруженные из системы по объекту (в виде `Dictionary<key, value>`, где `key` — название поля, а `value` — его значение), а на выход вернет результат прогноза. Метод может использовать прокси-класс, реализующий интерфейс `IMLServiceProxy`, который упростит вызов веб-сервиса.

### Реализация IMLEntityPredictor

Необходимо инициализировать свойство `ProblemTypeId` идентификатором созданной записи в справочнике `MLProblemType`. Также необходимо реализовать следующие методы:

- `SaveEntityPredictedValues` — метод получает результат прогнозирования и должен сохранить его для

объекта (`entity`) системы, для которого выполнялось прогнозирование. Если возвращаемое значение типа `double` или подобно результату классификации, можно воспользоваться методами вспомогательного класса `PredictionSaver`.

- `SavePrediction` (опционально) — метод сохраняет результат прогнозирования в привязке к экземпляру обученной модели и идентификатору объекта (`entityId`). Для базовых задач в системе существуют объекты `MLPrediction` и `MLClassificationResult`.

## Расширение IMLServiceProxy и MLServiceProxy (опционально)

Можно расширить существующий интерфейс `IMLServiceProxy` и его реализацию методом прогнозирования для текущей задачи. В частности, класс `MLServiceProxy` содержит обобщенный метод `Predict`, который принимает контракты для входящих данных и результата прогнозирования.

## Реализация IMLBatchPredictor

В случае, если сервис вызывается с набором данных (500 шт.), следует реализовать интерфейс `IMLBatchPredictor`. Необходимо реализовать следующие методы:

- `FormatValueForSaving` — метод возвращает значение, преобразованное для сохранения результата прогноза в базе данных. В случае пакетного прогнозирования для ускорения операции сохранения запись обновляется в базе данных методом `Update`, а не через экземпляры `Entity`.
- `SavePredictionResult` — определяет, как будет сохраняться результат прогнозирования в системе для каждой записи. Для базовых задач в системе существуют объекты `MLPrediction` и `MLClassificationResult`.

# Примеры запросов на выборку данных для модели машинного обучения



Сложный

## Использование утилитного класса `Terrasoft.Configuration.QueryExtensions`

Утилитный класс `Terrasoft.Configuration.QueryExtensions` предоставляет несколько расширяющих методов для `Terrasoft.Core.DB.Select`. Это позволяет составлять более компактные запросы.

Для всех расширяющих методов в качестве аргумента `object sourceColumn` могут быть использованы следующие типы (они будут трансформированы в `Terrasoft.Core.DB.QueryColumnExpression`):

- `System.String` — название колонки в формате "`TableAlias.ColumnAlias as ColumnAlias`" (где `TableAlias` и `ColumnAlias` опциональны) или "\*" — все колонки.
- `Terrasoft.Core.DB.QueryColumnExpression` — будет добавлен без изменений.
- `Terrasoft.Core.DB.IQueryColumnExpressionConvertible` — будет сконвертировано.
- `Terrasoft.Core.DB.Select` — будет рассмотрен как подзапрос.

**Важно.** Если тип не поддерживается, то будет сгенерировано исключение.

`public static Select Cols(this Select select, params object[] sourceColumns)`

Добавляет в запрос указанные колонки или подвыражения.

Используя расширяющий метод `Cols()`, вместо громоздкого выражения:

```
new Select(userConnection)
    .Column("L", "Id")
    .Column("L", "QualifyStatusId")
    .Column("L", "LeadTypeId")
    .Column("L", "LeadSourceId")
    .Column("L", "LeadMediumId").As("LeadChannel")
    .Column("L", "BusinesPhone").As("KnownBusinessPhone")
    .From("Lead").As("L");
```

можно записать:

```
new Select(userConnection).Cols(
    "L.Id",
    "L.QualifyStatusId",
    "L.LeadTypeId",
    "L.LeadSourceId",
    "L.LeadMediumId AS LeadChannel",
    "L.BusinesPhone AS KnownBusinessPhone")
    .From("Lead").As("L");
```

`public static Select Count(this Select select, object sourceColumn)`

Добавляет в запрос агрегирующую колонку для вычисления количества непустых значений.

Например, вместо:

```
var activitiesCount = new Select(userConnection)
    .Column(Func.Count(Column.Asterisk()))
    .From("Activity")
```

можно записать:

```
var activitiesCount = new Select(userConnection)
```

```
.Count("*") // Здесь можно указать и название колонки.  
.From("Activity")
```

**public static Select Coalesce(this Select select, params object[] sourceColumns)**

Добавляет в запрос колонку с функцией определения первого значения, не равного `NULL`.

Например, вместо:

```
new Select(userConnection)  
.Cols("L.Id")  
.Column(Func.Coalesce(  
    Column.SourceColumn("L", "CountryId"),  
    Column.SourceColumn("L", "CountryId"),  
    Column.SourceColumn("L", "CountryId")))  
.As("CountryId")  
.From("Lead").As("L")  
.LeftOuterJoin("Contact").As("C").On("L", "QualifiedContactId").IsEqual("C", "Id")  
.LeftOuterJoin("Account").As("A").On("L", "QualifiedAccountId").IsEqual("A", "Id");
```

можно записать:

```
new Select(userConnection)  
.Cols("L.Id")  
.Coalesce("L.CountryId", "C.CountryId", "A.CountryId").As("CountryId")  
.From("Lead").As("L")  
.LeftOuterJoin("Contact").As("C").On("L", "QualifiedContactId").IsEqual("C", "Id")  
.LeftOuterJoin("Account").As("A").On("L", "QualifiedAccountId").IsEqual("A", "Id");
```

**public static Select DateDiff(this Select select, DateDiffQueryFunctionInterval interval, object startDateExpression, object endDateExpression)**

Добавляет в запрос колонку с определением разницы дат.

Например, вместо:

```
new Select(_userConnection)  
.Cols("Id")  
.Column(Func.DateDiff(DateDiffQueryFunctionInterval.Day,  
    Column.SourceColumn("L", "CreatedOn"), Func.CurrentDateTime()).As("LeadAge")  
.From("Lead").As("L");
```

можно записать:

```
var day = DateDiffQueryFunctionInterval.Day;
new Select(userConnection)
    .Cols("L.Id")
    .DateDiff(day, "L.CreatedOn", Func.CurrentDateTime()).As("LeadAge")
    .From("Lead").As("L");
```

**public static Select IsNull(this Select select, object checkExpression, object replacementValue)**

Добавляет в запрос колонку с функцией замены значения `NULL` замещающим выражением.

Например, вместо:

```
new Select(userConnection).Cols("Id")
    .Column(Func.IsNull(
        Column.SourceColumn("L", "CreatedOn"),
        Column.SourceColumn("L", "ModifiedOn")))
    .From("Lead").As("L");
```

можно записать:

```
new Select(userConnection).Cols("L.Id")
    .IsNull("L.CreatedOn", "L.ModifiedOn")
    .From("Lead").As("L");
```

## Подключить веб-сервис к функциональности машинного обучения

 Средний

**Пример.** Подключить к Creatio пользовательский аналог предиктивного скоринга.

### Алгоритм реализации примера

#### 1. Создайте веб-сервис — движок машинного обучения

Пример реализации веб-сервиса машинного обучения для ASP.Net Core 3.1 можно скачать по [ссылке](#).

Реализуйте веб-сервис машинного обучения `MLService`.

Задайте обязательные методы:

- `/session/start`
- `/data/upload`
- `/session/info/get`
- `/fakeScorer/beginTraining`
- `/fakeScorer/predict`

Полностью исходный код представлен ниже.

```
MLService

namespace FakeScoring.Controllers
{
    using System;
    using System.Collections.Generic;
    using System.Linq;
    using System.Net;
    using Microsoft.AspNetCore.Mvc;
    using Terrasoft.ML.Interfaces;
    using Terrasoft.ML.Interfaces.Requests;
    using Terrasoft.ML.Interfaces.Responses;

    [ApiController]
    [Route("")]
    public class MLService : Controller
    {
        public const string FakeModelInstanceId = "{BFC0BD71-19B1-47B1-8BC4-D761D9172667}";

        private List<ScoringOutput.FeatureContribution> GenerateFakeContributions(DatasetValue r
            var random = new Random(42);
            return record.Select(columnValue => new ScoringOutput.FeatureContribution {
                Name = columnValue.Key,
                Contribution = random.NextDouble(),
                Value = columnValue.Value.ToString()
            }).ToList();
        }

        [HttpGet("ping")]
        public JsonResult Ping() {
            return new JsonResult("Ok");
        }

        /// <summary>
        /// Handshake request to service with the purpose to start a model training session.
    }
}
```

```

/// </summary>
/// <param name="request">Instance of <see cref="StartSessionRequest"/>.</param>
/// <returns>Instance of <see cref="StartSessionResponse"/>.</returns>
[HttpPost("session/start")]
public StartSessionResponse StartSession(StartSessionRequest request) {
    return new StartSessionResponse {
        SessionId = Guid.NewGuid()
    };
}

/// <summary>
/// Uploads training data.
/// </summary>
/// <param name="request">The upload data request.</param>
/// <returns>Instance of <see cref="JsonResult"/>.</returns>
[HttpPost("data/upload")]
public JsonResult UploadData(UploadDataRequest request) {
    return new JsonResult(string.Empty) {
        StatusCode = (int) HttpStatusCode.OK
    };
}

/// <summary>
/// Begins fake scorer training on uploaded data.
/// </summary>
/// <param name="request">The scorer training request.</param>
/// <returns>Simple <see cref="JsonResult"/>.</returns>
[HttpPost("fakeScorer/beginTraining")]
public JsonResult BeginScorerTraining(BeginScorerTrainingRequest request) {
    // Start training work here. It doesn't have to be done by the end of this request.
    return new JsonResult(string.Empty) {
        StatusCode = (int) HttpStatusCode.OK
    };
}

/// <summary>
/// Returns current session state and model statistics, if training is complete.
/// </summary>
/// <param name="request">Instance of <see cref="GetSessionInfoRequest"/>.</param>
/// <returns>Instance of <see cref="GetSessionInfoResponse"/> with detailed state info.</returns>
[HttpPost("session/info/get")]
public GetSessionInfoResponse GetSessionInfo(GetSessionInfoRequest request) {
    var response = new GetSessionInfoResponse {
        SessionState = TrainSessionState.Done,
        ModelSummary = new ModelSummary {
            DataSetSize = 100500,
            Metric = 0.79,
            MetricType = "Accuracy",
            TrainingTimeMinutes = 5,
        }
    };
}

```

```
        ModelInstanceId = new Guid(FakeModelInstanceId)
    }
};

return response;
}

/// <summary>
/// Performs fake scoring prediction.
/// </summary>
/// <param name="request">Request object.</param>
/// <returns>Scoring rates.</returns>
[HttpPost("fakeScorer/predict")]
public ScoringResponse Predict(ExplainedScoringRequest request) {
    List<ScoringOutput> outputs = new List<ScoringOutput>();
    var random = new Random(42);
    foreach (var record in request.PredictionParams.Data) {
        var output = new ScoringOutput {
            Score = random.NextDouble()
        };
        if (request.PredictionParams.PredictContributions) {
            output.Bias = 0.03;
            output.Contributions = GenerateFakeContributions(record);
        }
        outputs.Add(output);
    }
    return new ScoringResponse { Outputs = outputs };
}
}
```

## 2. Расширьте список задач машинного обучения

Для расширения списка задач машинного обучения выполните следующие действия:

1. Перейдите в дизайнер системы по кнопке . В блоке [ Настстройка системы ] ([ *System setup* ]) перейдите по ссылке [ Справочники ] ([ *Lookups* ]).  
2. Выберите справочник [ Задачи машинного обучения ] ([ *ML problem types* ]).  
3. Создайте новую запись.

Для записи установите:

- [ Название ] ([ *Name* ]) — "Fake scoring";
  - [ Url сервиса ] ([ *Service endpoint Url* ]) — "http://localhost:5000/";
  - [ Метод сервиса для обучения модели ] ([ *Training endpoint* ]) — "/fakeScorer/beginTraining".

The screenshot shows the 'Lookups' screen in the Creatio application. The left sidebar is titled 'Marketing' and lists various entities: Contacts, Campaigns, Email, Landing pages and web forms, Events, Leads, Accounts, Dashboards, and Marketing plans. The main area is titled 'ML problem types' and contains a table with the following data:

Name	Description	Service endpoint Url	Training endpoint
Numeric prediction			/regressor/beginTraining
Predictive scoring	Predictive scoring		/scorer/beginTraining
Fake scoring		http://localhost:5000/	/fakeScorer/beginTraining
Recommendation	Recommendation based on Coll...		/cf/beginTraining
Lookup prediction	Entity lookup field prediction		/classifier/beginTraining

Идентификатор созданной записи — "319c39fd-17a6-453a-bceb-57a398d52636".

### 3. Реализуйте модель машинного обучения

В разделе [ Конфигурация ] ([ Configuration ]) пользовательского пакета на вкладке [ Схемы ] ([ Schemas ]) выполните действие [ Добавить ] —> [ Схема модели представления карточки ] ([ Add ] —> [ Schema of Edit Page View Model ]). Созданный модуль должен наследовать функциональность базовой страницы модели машинного обучения `MLModelPage`, которая определена в пакете `ML`. Для этого укажите эту схему в качестве родительской для создаваемой схемы.

Для создаваемой [схемы объекта](#) установите:

- [ Заголовок ] ([ Title ]) — "FakeScoringMLModelPage";
- [ Название ] ([ Name ]) — "UsrMLModelPage".
- [ Родительский объект ] ([ Parent object ]) — выберите `MLModelPage`.

The screenshot shows the 'Properties' dialog box for the 'FakeScoringMLModelPage' schema. The 'General' section contains the following fields:

- Title: FakeScoringMLModelPage
- Name: UsrMLModelPage
- Package: sdkMLExtensionPackage

The 'Inheritance' section contains the following field:

- Parent object: MLModelPage

Переопределите базовый метод `getIsScoring`, чтобы вид создаваемой карточки соответствовал карточке базового предиктивного скоринга. Полностью исходный код представлен ниже.

**MLModelPage**

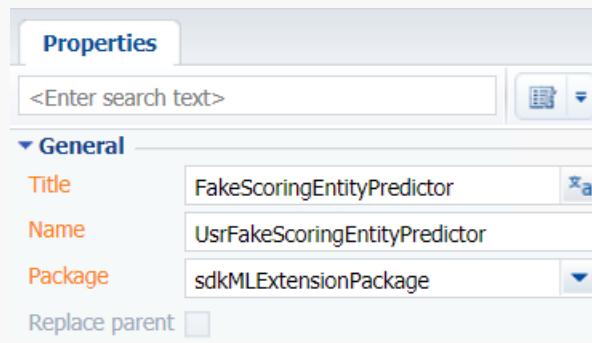
```
define("MLModelPage", [],
  function() {
    return {
      entitySchemaName: "MLModel",
      properties: {
        FakeScoringProblemTypeId: "319c39fd-17a6-453a-bceb-57a398d52636"
      },
      methods: {
        getIsScoring: function() {
          const parentResult = this.callParent(arguments);
          return parentResult || this.getLookupValue("MLProblemType") === this.FakeScc
        }
      }
    });
});
```

После внесения изменений сохраните и опубликуйте схему.

В разделе [ Конфигурация ] ([ Configuration ]) пользовательского пакета на вкладке [ Схемы ] ([ Schemas ]) выполните действие [ Добавить ] —> [ Исходный код ] ([ Add ] —> [ Source Code ]).

Для создаваемой [схемы объекта](#) установите:

- [ Заголовок ] ([ Title ]) — "FakeScoringEntityPredictor";
- [ Название ] ([ Name ]) — "UsrFakeScoringEntityPredictor".



Реализуйте метод `Predict`, который получает данные, выгруженные из системы по объекту, и возвращает результат прогноза. Метод использует прокси-класс, реализующий интерфейс `IMLServiceProxy`, который упрощает вызов веб-сервиса.

Инициализируйте свойство `ProblemTypeId` идентификатором созданной записи в справочнике `MLProblemType` и реализуйте методы `SaveEntityPredictedValues` и `SavePrediction`. Полностью исходный код представлен ниже.

**FakeScoringEntityPredictor**

```

namespace Terrasoft.Configuration.ML
{
    using System;
    using System.Collections.Generic;
    using Core;
    using Core.Factories;
    using Terrasoft.ML.Interfaces;

    [DefaultBinding(typeof(IMLEntityPredictor), Name = "319C39FD-17A6-453A-BCEB-57A398D52636")]
    [DefaultBinding(typeof(IMLPredictor<ScoringOutput>), Name = "319C39FD-17A6-453A-BCEB-57A398D52636")]
    public class FakeScoringEntityPredictor: MLBaseEntityPredictor<ScoringOutput>, IMLEntityPredictor<ScoringOutput>
    {

        public FakeScoringEntityPredictor(UserConnection userConnection) : base(userConnection)
        {}

        protected override Guid ProblemTypeId => new Guid("319C39FD-17A6-453A-BCEB-57A398D52636")

        protected override ScoringOutput Predict(IMLServiceProxy proxy, MLModelConfig model,
            Dictionary<string, object> data) {
            return proxy.FakeScore(model, data, true);
        }

        protected override List<ScoringOutput> Predict(MLModelConfig model,
            IList<Dictionary<string, object>> dataList, IMLServiceProxy proxy) {
            return proxy.FakeScore(model, dataList, true);
        }

        protected override void SaveEntityPredictedValues(MLModelConfig model, Guid entityId,
            ScoringOutput predictedResult) {
            var predictedValues = new Dictionary<MLModelConfig, double> {
                { model, predictedResult.Score }
            };
            PredictionSaver.SaveEntityScoredValues(model.EntitySchemaId, entityId, predictedValues);
        }

        protected override void SavePrediction(MLModelConfig model, Guid entityId, ScoringOutput predictedResult) {
            PredictionSaver.SavePrediction(model.Id, model.ModelInstanceId, entityId, predictedResult);
        }
    }
}

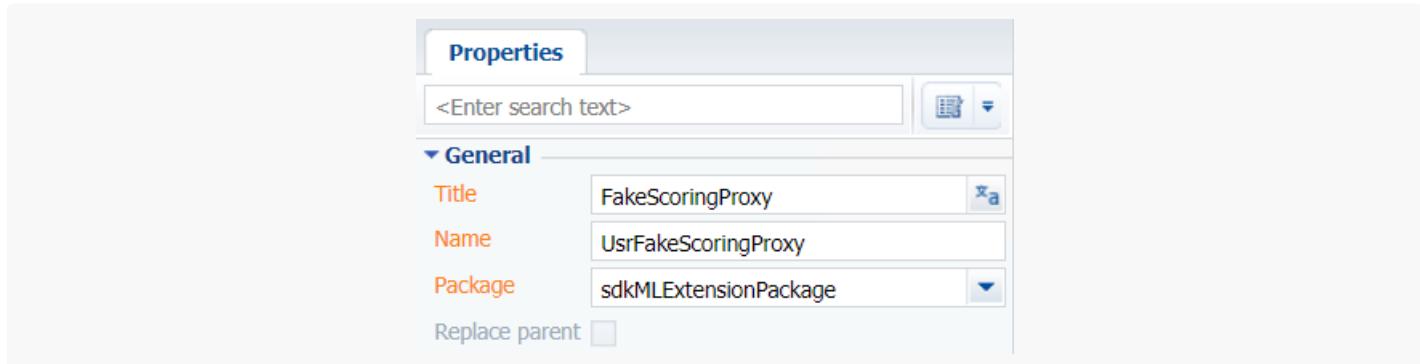
```

После внесения изменений сохраните и опубликуйте схему.

В разделе [ Конфигурация ] ([ Configuration ]) пользовательского пакета на вкладке [ Схемы ] ([ Schemas ]) выполните действие [ Добавить ] —> [ Исходный код ] ([ Add ] —> [ Source Code ]).

Для создаваемой [схемы объекта](#) установите:

- [ Заголовок ] ([ Title ]) — "FakeScoringProxy";
- [ Название ] ([ Name ]) — "UsrFakeScoringProxy".



Расширьте существующий интерфейс `IMLServiceProxy` и его реализацию методом прогнозирования для текущей задачи. В частности класс `MLServiceProxy` содержит обобщенный метод `Predict`, который принимает контракты для входящих данных и результата прогнозирования.

Полностью исходный код представлен ниже.

```
IMLServiceProxy

namespace Terrasoft.Configuration.ML
{
    using System;
    using System.Collections.Generic;
    using System.Linq;
    using Terrasoft.ML.Interfaces;
    using Terrasoft.ML.Interfaces.Requests;
    using Terrasoft.ML.Interfaces.Responses;

    public partial interface IMLServiceProxy
    {
        ScoringOutput FakeScore(MLModelConfig model, Dictionary<string, object> data, bool predictContributions);
    }

    public partial class MLServiceProxy : IMLServiceProxy
    {
        public ScoringOutput FakeScore(MLModelConfig model, Dictionary<string, object> data, bool predictContributions)
        {
            ScoringResponse response = Predict<ExplainedScoringRequest, ScoringResponse>(model.ModelId, new List<Dictionary<string, object>> { data }, model.PredictionEndpoint,
                new List<Dictionary<string, object>> { data }, model.PredictionEndpoint,
                predictContributions);
            return response.ScoringOutput;
        }
    }
}
```

```
        100, predictContributions);
    return response.Outputs.FirstOrDefault();
}

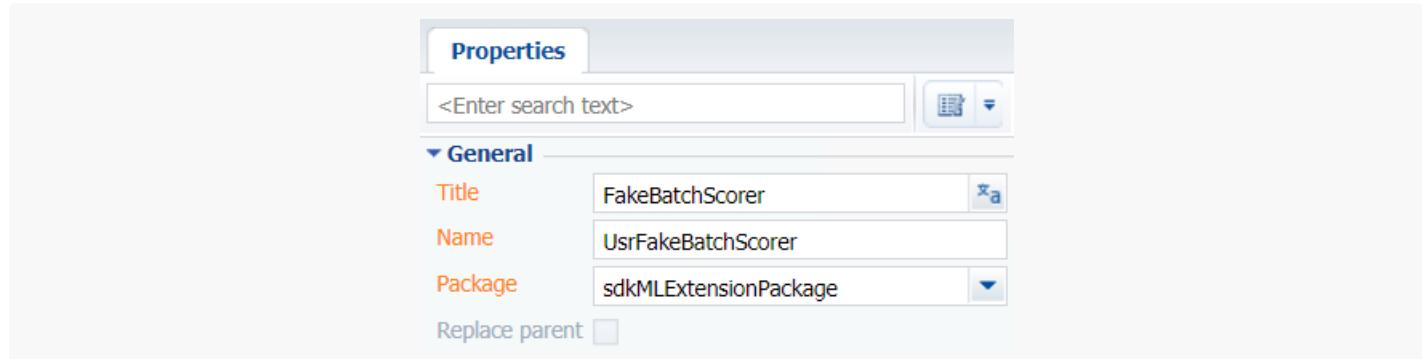
public List<ScoringOutput> FakeScore(MLModelConfig model, IList<Dictionary<string, object>> dataList, bool predictContributions) {
    int count = Math.Min(1, dataList.Count);
    int timeout = Math.Max(ScoreTimeoutSec * count, BatchScoreTimeoutSec);
    ScoringResponse response = Predict<ScoringRequest, ScoringResponse>(model.ModelInstance,
        dataList, model.PredictionEndpoint, timeout, predictContributions);
    return response.Outputs;
}
}
```

После внесения изменений сохраните и опубликуйте схему.

В разделе [ Конфигурация ] ([ Configuration ]) пользовательского пакета на вкладке [ Схемы ] ([ Schemas ]) выполните действие [ Добавить ] → [ Исходный код ] ([ Add ] → [ Source Code ]).

Для создаваемой [схемы объекта](#) установите:

- [ Заголовок ] ([ *Title* ]) — "FakeBatchScorer";
  - [ Название ] ([ *Name* ]) — "UsrFakeBatchScorer".



Для использования функциональности пакетного прогнозирования реализуйте интерфейс `IMLBatchPredictor`, методы `FormatValueForSaving` и `SavePredictionResult`.

Полностью исходный код представлен ниже.

```
namespace Terrasoft.Configuration.ML  
{  
    using System;  
    using Core;  
    using Terrasoft.Core.Factories;  
    using Terrasoft.ML.Interfaces;
```

```
[DefaultBinding(typeof(MLBatchPredictor)), Name = "319C39FD-17A6-453A-BCEB-57A398D52636")]
public class FakeBatchScorer : MLBatchPredictor<ScoringOutput>
{
    public FakeBatchScorer(UserConnection userConnection) : base(userConnection) {}

    protected override object FormatValueForSaving(ScoringOutput scoringOutput) {
        return Convert.ToInt32(scoringOutput.Score * 100);
    }

    protected override void SavePredictionResult(Guid modelId, Guid modelInstanceId, Guid entityId, ScoringOutput value) {
        PredictionSaver.SavePrediction(modelId, modelInstanceId, entityId, value);
    }
}
```

После внесения изменений сохраните и опубликуйте схему.

# Реализовать пользовательскую предиктивную модель



Сложный

## Описание кейса

Реализовать автоматическое прогнозирование колонки [ Категория ] ([ AccountCategory ]) по значениям полей [ Страна ] ([ Country ]), [ Количество сотрудников ] ([ EmployeesNumber ]), [ Отрасль ] ([ Industry ]) во время сохранения записи контрагента. При этом должны выполняться следующие условия:

- Обучение модели необходимо формировать на основании записей о контрагентах за последние 90 дней.
- Переобучение производить каждые 30 дней.
- Допустимое значение точности прогнозирования для модели в целом — 0.6.

### Важно.

Для выполнения этого кейса необходимо удостовериться, что установлено корректное значение для системной настройки [ API ключ облачных сервисов Creatio ] ([ Creatio cloud services API key ], код `CloudServicesAPIKey`) и установлен адрес предиктивного сервиса в поле [ Url сервиса ] ([ Service endpoint URL ]) записи справочника [ Задачи машинного обучения ] ([ ML problem types ]).

**На заметку.** Описанный кейс можно выполнить, используя UI-инструменты карточки (поля и фильтры), а также элемент бизнес-процесса «Прогнозирование данных». Примеры реализации прогнозирования с помощью встроенных возможностей системы описаны в разделе [«Предиктивный анализ данных»](#).

## Алгоритм выполнения кейса

### 1. Обучение модели

Для обучения модели необходимо:

- Добавить запись в справочник [ *Модели машинного обучения* ] ( `MLModel` ). Значения полей записи приведены в таблице 1.

Табл. 1. — Значения полей записи справочника `MLModel`

Поле	Значение
Название	Прогнозирование категории контрагента
Задача машинного обучения	Прогнозирование справочного поля
Идентификатор корневого объекта	Контрагент
Нижний порог допустимого качества	0.6
Частота переобучения	30
Метаданные выборки для обучения	<pre>{     "inputs": [         {             "name": "CountryId",             "type": "Lookup",             "isRequired": true         },         {             "name": "EmployeesNumberId",             "type": "Lookup",             "isRequired": true         }     ] }</pre>

```

        },
        {
            "name": "IndustryId",
            "type": "Lookup",
            "isRequired": true
        }
    ],
    "output": {
        "name": "AccountCategoryId",
        "type": "Lookup",
        "displayName": "AccountCategory"
    }
}

```

Выражение для  
выборки данных  
обучения

```

new Select(userConnection)
    .Column("a", "Id").As("Id")
    .Column("a", "CountryId")
    .Column("a", "EmployeesNumberId")
    .Column("a", "IndustryId")
    .Column("a", "AccountCategoryId")
    .Column("c", "Name").As("AccountCategory")
    .From("Account").As("a")
    .InnerJoin("AccountCategory").As("c").On("c", "Id").IsEqual("a", "Accoun
    .Where("a", "CreatedOn").IsGreater(Column.Parameter(DateTime.Now.AddDays

```

Примеры составления запросов можно посмотреть в статье [Составление запросов  
данных для модели машинного обучения](#).

Флаг,  
включающий  
прогнозирование  
по модели

Отметить

2. Выполнить действие [ Запланировать задание обучения моделей ] на странице справочника [ Модели машинного обучения ] ( `MLModel` ).

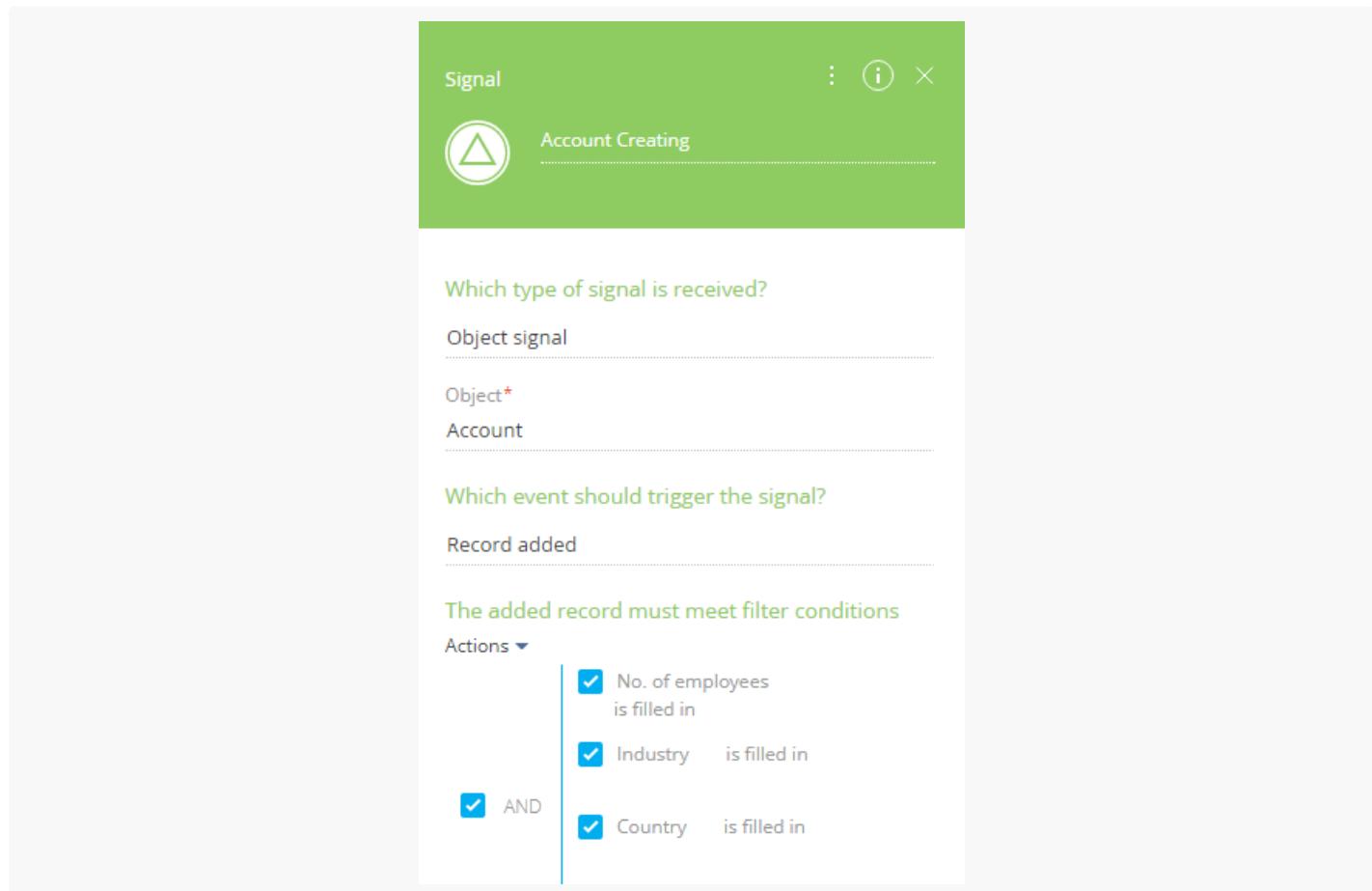
Далее необходимо подождать пока поле [ Статус обработки моделей ] пройдет цепочку значений `DataTransfer`, `QueuedToTrain`, `Training`, `Done`. Процесс ожидания может занять несколько часов (зависит от объема передаваемых данных и общей нагрузки на сервис предиктивных моделей).

## 2. Выполнение прогнозов

Для выполнения прогнозов необходимо:

1. В пользовательском пакете создать бизнес-процесс, для которого стартовым сигналом будет событие сохранения объекта [ Контакт ]. При этом нужно проверить заполненность значениями необходимых полей (рис. 1).

Рис. 1. — Свойства стартового сигнала



2. В бизнес-процесс добавить параметр-справочник `MLModelId`, ссылающийся на сущность [ Модель машинного обучения ]. В качестве значения нужно выбрать запись с созданной моделью [ Прогнозирование категории Контрагента ].

3. В бизнес-процесс добавить параметр-справочник `RecordId`, ссылающийся на сущность [ Контрагент ]. В качестве значения нужно выбрать ссылку на параметр `RecordId` элемента [ Сигнал ].

4. На диаграмму бизнес-процесса добавить элемент [ Задание-сценарий ] и добавить в него следующий исходный код:

```
var userConnection = Get<UserConnection>("UserConnection");
// Получение Id записи контрагента
Guid entityId = Get<Guid>("RecordId");
// Получение id модели.
var modelId = Get<Guid>("MLModelId");
var connectionArg = new ConstructorArgument("userConnection", userConnection);
// Объект для вызова прогнозирования
var predictor = ClassFactory.Get<MLEntityPredictor>(connectionArg);
```

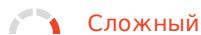
```
// Загрузка модели.
// Вызов сервиса прогнозирования. Данные сохраняются в MLPrediction и в случае высокой вероятности
predictor.PredictEntityValueAndSaveResult(modelId, entityId);
return true;
```

После сохранения и компиляции процесса для новых контрагентов будет выполняться прогнозирование. Полученный прогноз будет отображаться на странице контрагента.

### Важно.

При такой реализации замедляется сохранение отдельной записи контрагента, поскольку вызов сервиса прогнозирования происходит около 2 с. Это может уменьшить производительность массовых операций с сохранением, например при импорте значений из Excel.

## Класс IMLModelTrainerJob c#



Сложный

Интерфейсы `IJobExecutor`, `IMLModelTrainerJob` используются для задания синхронизации моделей.

Оркестрирует обработку моделей на стороне Creatio, запуская сессии передачи данных, стартуя обучения, а также проверяя статус моделей, обрабатываемых сервисом. Экземпляры по умолчанию запускаются планировщиком заданий через стандартный метод `Execute` интерфейса `IJobExecutor`.

### Методы

#### `IMLModelTrainerJob.RunTrainer()`

Виртуальный метод, инкапсулирующий логику синхронизации. Базовая реализация этого метода выполняет следующую последовательность действий:

- Выборка моделей для обучения — отбираются записи из `MLModel` по следующему фильтру:

- Поля `MetaData` и `TrainingsetQuery` заполнены.
- Поле `Status` находится в состоянии `NotStarted`, `Done`, `Error` или не заполнено.
- `TrainFrequency` больше 0.
- От даты последней попытки обучения (`TriedToTrainon`) прошло `TrainFrequency` дней.  
Для каждой записи этой выборки производится передача данных в сервис с помощью тренера предиктивной модели (см. ниже).

- Выборка ранее отправленных на обучение моделей и, при необходимости, обновление их статуса.

По каждой подходящей модели начинается сессия передачи данных по выборке. В процессе сессии пакетами по 1000 записей отправляются данные. Для каждой модели объем выборки ограничивается до 75 000 записей.

# Класс IMLModelTrainer



Сложный

Интерфейс `IMLModelTrainer` является тренером предиктивной модели.

Отвечает за общую обработку отдельно взятой модели на этапе обучения. Коммуникация с сервисом обеспечивается с помощью прокси к предиктивному сервису (см. ниже).

## Методы

---

`IMLModelTrainer.StartTrainSession()`

Устанавливает сессию обучения для модели.

---

`IMLModelTrainer.UploadData()`

Передает данные в соответствии с выборкой модели пакетами по 1000 записей. Выборка ограничивается объемом в 75 000 записей.

---

`IMLModelTrainer.BeginTraining()`

Обозначает завершение передачи данных и сообщает сервису о необходимости постановки модели в очередь обучения.

---

`IMLModelTrainer.UpdateModelState`

Запрашивает у сервиса текущее состояние модели и при необходимости обновляет `Status`.

Если обучение было успешным (`Status` содержит значение `Done`), то сервис возвращает метаданные по обученному экземпляру, в частности — точность получившегося экземпляра. Если точность выше или равна нижнему порогу (`MetricThreshold`) — идентификатор нового экземпляра записывается в поле `ModelInstanceUID`.

# Класс IMLServiceProxy



Сложный

Интерфейс `IMLServiceProxy` является прокси к предиктивному сервису. Класс-обертка для http-запросов к предиктивному сервису.

## Методы

---

`IMLServiceProxy.UploadData()`

Передает пакет данных для сессии обучения.

---

`IMLServiceProxy.BeginTraining()`

Вызывает сервис постановки обучения в очередь.

---

`IMLServiceProxy.GetTrainingSessionInfo()`

Запрашивает у сервиса текущее состояние для сессии обучения.

---

`IMLServiceProxy.Classify(Guid modelInstanceId, Dictionary<string, object> data)`

Вызывает для ранее обученного экземпляра модели сервис прогнозирования значения поля для отдельно взятого набора значений полей. В параметре `Dictionary data` в качестве ключа передается имя поля, которое обязательно должно совпадать с именем, указанным в поле `MetaData` справочника моделей. При успешном результате метод возвращает список значений типа `ClassificationResult`.

## Свойства типа ClassificationResult

---

### Value

Значение поля.

---

### Probability

Вероятность данного значения в диапазоне [0:1]. Сумма вероятностей для одного списка результатов близка к 1 (значения около 0 могут быть опущены).

---

### Significance

Уровень важности данного прогноза.

#### Возможные значения ( Significance )

High	Данное значение поля имеет явное преимущество по сравнению с другими значениями из списка. Такой уровень может быть только у одного элемента в списке предсказанных.
Medium	Значение поля близко к нескольким другим высоким значениям в списке. Например, два значения в списке имеют вероятность 0,41 и 0,39, все остальные значительно меньше.
None	Нерелевантные значения с низкими вероятностями.

# Класс IMLEntityPredictor [C#](#)



Сложный

Утилитный класс, помогающий для отдельной сущности произвести прогнозирование значения поля по указанной модели (одной или нескольким).

## Методы

`PredictEntityValueAndSaveResult(Guid modelId, Guid entityId)`

По `Id` модели и `Id` сущности выполняет прогнозирование и записывает результат в результирующее поле объекта. Работает с любой задачей машинного обучения: классификация, скоринг, прогнозирование числового поля.

`ClassifyEntityValues(List<Guid> modelIds, Guid entityId)`

По `Id` модели (или списку нескольких моделей, созданных для одного и того же объекта) и `Id` сущности выполняет классификацию и возвращает словарь, ключ которого — объект модели, значения — список спрогнозированных результатов.

# Класс IMLPredictionSaver [C#](#)



Сложный

Утилитный класс, помогающий сохранить результаты прогнозирования в объект системы.

## Методы

`SaveEntityPredictedValues(Guid schemaUid, Guid entityId, Dictionary<MLModelConfig, List<ClassificationResult> onSetEntityValue)`

Сохраняет результаты классификации (`MLEntityPredictor.ClassifyEntityValues`) в объект системы. По умолчанию сохраняет только результат, у которого `Significance` равен `"High"`. Но есть возможность переопределить это поведение с помощью переданного делегата `onSetEntityValue`. Если делегат вернет `false`, то значение не будет записано в объект системы.

# Справочник MLModel [C#](#)



Сложный

Содержит информацию о выбранных данных для модели, периоде обучения, текущем статусе обучения и т.д.

## Поля справочника MLModel

Назначение основных полей справочника `MLModel` приведено в таблице.

Основные поля справочника `MLModel`

Поле	Тип данных	Назначение
<code>Name</code>	Строка	Название модели.
<code>ModelInstanceId</code>	Уникальный идентификатор	Идентификатор текущего экземпляра модели.
<code>TrainedOn</code>	Дата/время	Время, когда экземпляр был обучен.
<code>TriedToTrainOn</code>	Дата/время	Время последней попытки обучения.
<code>TrainFrequency</code>	Целое	Частота переобучения модели (дни).
<code>MetaData</code>	Строка	Метаданные с типами колонок выборки. Записываются JSON-формате следующей структуры:

**Структура метаданных**

```
{
  inputs: [
    {
      name: "Имя поля 1 в выборке данных",
      type: "Text",
     isRequired: true
    },
    {
      name: "Имя поля 2 в выборке данных",
      type: "Lookup"
    },
    //...
  ],
  output: {
    name: "Результирующее поле",
    type: "Lookup",
    displayName: "Имя колонки для отображения"
  }
}
```

Здесь:

- `inputs` — набор входящих колонок для модели.
- `output` — колонка, значение которой модели получит

- `outptr` — колонка, значение которой модель должна предсказывать.

В описании колонок поддерживаются следующие атрибуты:

- `name` — имя поля из выражения `TrainingsetQuery`.
- `type` — тип данных для движка обучения.

#### Поддерживаемые значения

<code>Text</code>	Текстовая колонка.
<code>Lookup</code>	Колонка со справочным типом.
<code>Boolean</code>	Логический тип.
<code>Numeric</code>	Числовой тип.
<code>DateTime</code>	Дата и время.

- `isRequired` — обязательность значения поля (`true` / `false`). По умолчанию — `false`.

`TrainingsetQuery`

Строка

C#-выражение выборки тренировочных данных. Данное выражение должно возвращать экземпляр класса `Terrasoft.Core.DB.Select`.

#### Пример

```
(Select)new Select(userConnection)
    .Column("Id")
    .Column("Symptoms")
    .Column("CreatedOn")
    .From("Case", "c")
    .OrderByDesc("c", "CreatedOn")
```

**Важно.** В выражении для выборки обязательно должна выбираться колонка типа "Уникальный идентификатор". Эта колонка должна называться или иметь псевдоним `Id`.

**Важно.** Если выражение для выборки содержит колонку для сортировки, то эта колонка

		обязательно должна присутствовать в результирующей выборке.
RootSchemaUUID	Уникальный идентификатор	Ссылка на схему объекта, для которого будет выполняться прогноз.
Status	Строка	Статус обработки модели (передача данных, обучение, готова для прогноза).
InstanceMetric	Число	Метрика качества для текущего экземпляра модели.
MetricThreshold	Число	Нижний порог допустимого качества модели.
PredictionEnabled	Логическое	Флаг, включающий прогнозирование по данной модели
TrainSessionId	Уникальный идентификатор	Текущая сессия обучения.
MLProblemType	Уникальный идентификатор	Задача машинного обучения (определяет алгоритм и service url, с помощью которого будет обучаться модель)

## Сервис обогащения контактов из email

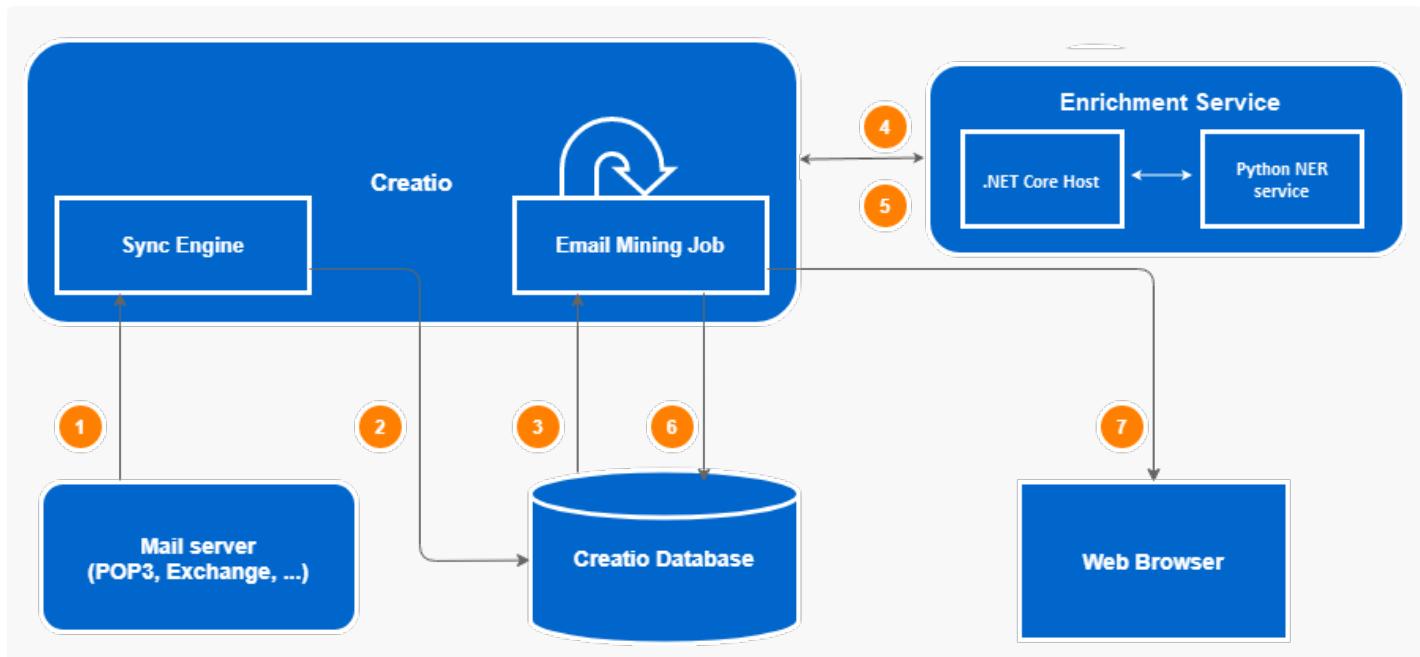


Сложный

В версии 7.10.0 появилась функциональность обогащения контактных данных информацией из email. Основная задача функциональности — обнаружение в письмах информации, которой можно обогатить данные контактов/контрагентов.

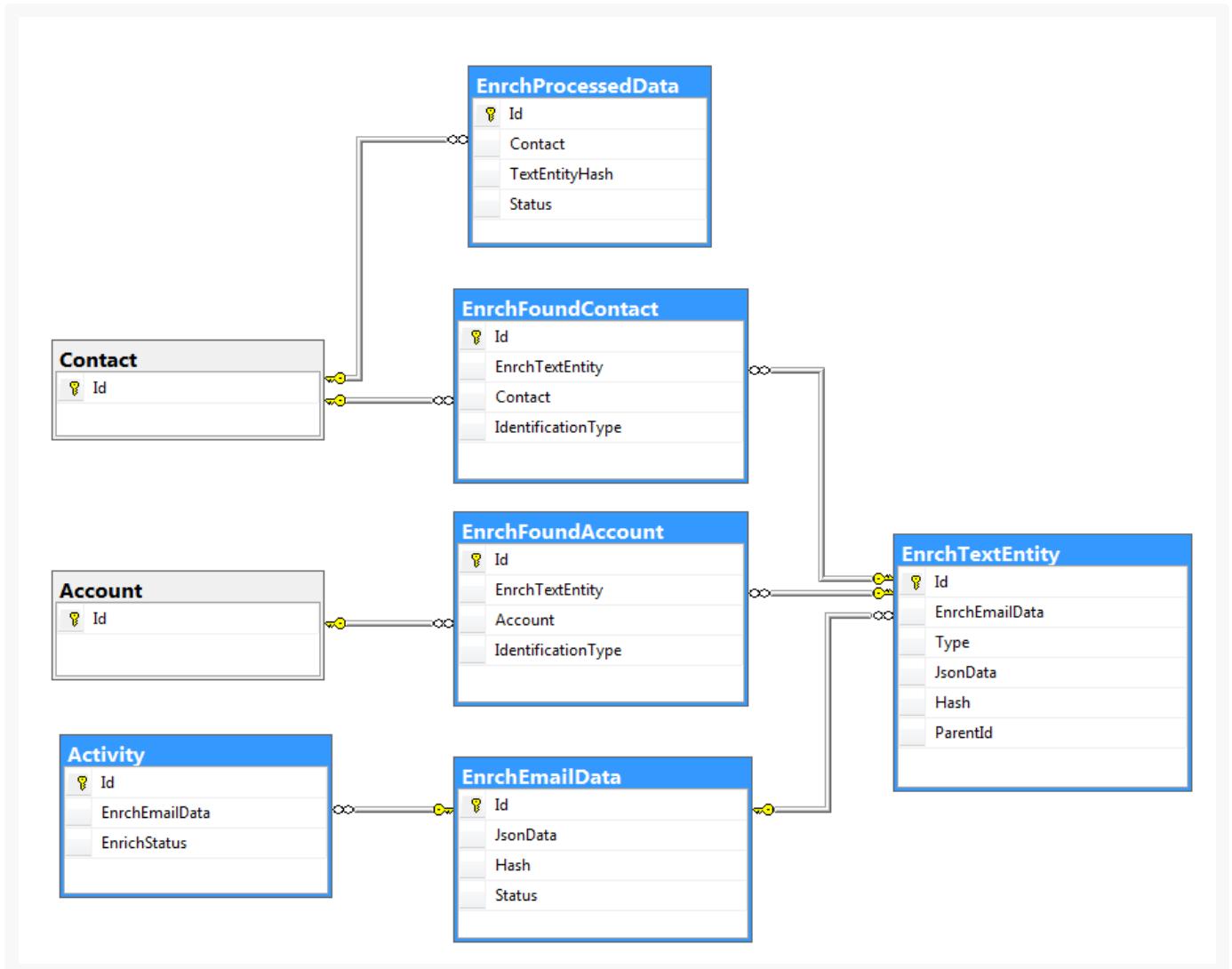
### Процесс обогащения контакта/контрагента

Процесс обогащения контакта/контрагента:



Основные этапы процесса обогащения контакта информацией из email:

- Существующий [механизм синхронизации Sync Engine](#) производит синхронизацию с почтовым сервером. Почтовый сервер передает Sync Engine новые письма.
- Sync Engine сохраняет полученные письма в базе данных в виде активностей с типом `Email`.
- Планировщик Creatio периодически выполняет задание, которое запускает процесс `Email Mining Job`. Этот процесс выбирает из базы данных порцию последних (по дате создания) активностей с типом `Email`, которые ранее не были им обработаны. Из каждой записи активности выбирается тело письма и его формат (plain-текст или html).
- Процесс `Email Mining Job` по каждому выбранному письму отправляет http-запрос в облачный сервис обогащения данных `Enrichment Service`.
- `Enrichment Service` выполняет следующие операции:
  - выделяет из письма цепочку отдельных сообщений (ответов);
  - для каждого сообщения выделяет подпись (signature);
  - из подписи выделяет сущности (entity extraction) — контакт (ФИО, должность), телефоны, email- и web-адреса, социальные сети, другие средства связи, адреса, название организации. Эти данные `Enrichment Service` возвращает в http-ответе в виде определенной структуры в формате JSON.
- Процесс `Email Mining Job` разбирает полученную от сервиса структуру и сохраняет ее в сыром виде в таблицах Creatio.

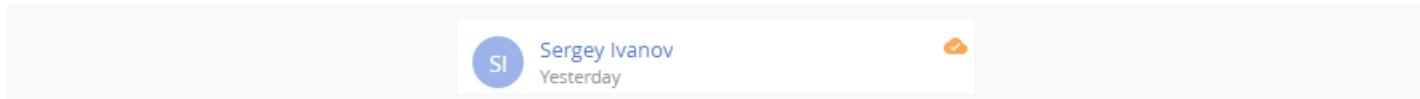


Основное назначение таблиц:

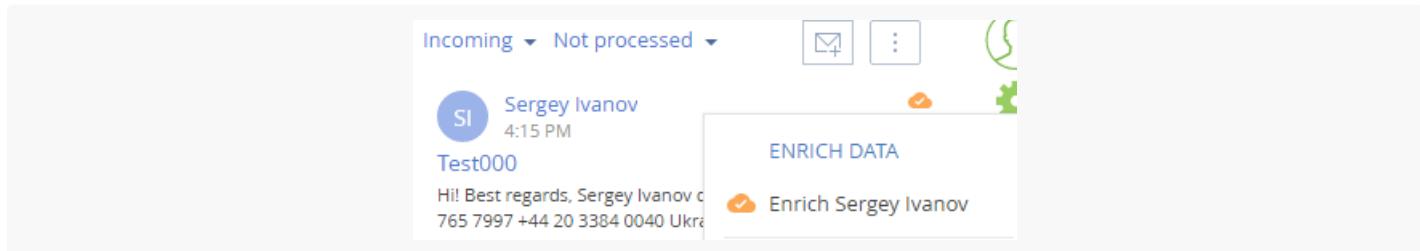
- [EnrichTextEntity] — хранит информацию об одной сущности, выделенной из письма. Поле `Type` определяет тип этой сущности (контакт, коммуникация, адрес, организация и т.п.). Сами данные хранятся в формате JSON в поле `JsonData`.
- [EnrichEmailData] — определяет набор информации для обогащения, выделенный из одного письма.
- [EnrichFoundContact] — контакт в Creatio, идентифицированный по выделенным из email данным. Хранит ссылку на контакт Creatio и [EnrichTextEntity] типа Контакт.
- [EnrichFoundAccount] — аналогично таблице [EnrichFoundContact] хранит информацию об идентифицированном контрагенте Creatio.
- [Activity] — в существующую таблицу активностей добавлены поля, которые отображают связь между активностями типа `Email` и объектами [EnrichEmailData] с текущим статусом процесса выделения информации.
- [EnrichProcessedData] — содержит информацию об уже обработанных данных, принятых либо отвергнутых пользователем в процессе обогащения.

7. Процесс `Email Mining Job` уведомляет о завершении извлечения информации из письма. Сообщения

отправляются по каналам websocket пользователям, которые в коммуникационной панели видят обрабатываемые письма. Если в письме есть информация, которой можно обогатить связанный контакт, либо на основании которой можно создать новый контакт, то в интерфейсе приложения в правом верхнем углу письма отображается соответствующий значок.



Для такого письма доступно действие, которое позволит обогатить или создать новый контакт системы.



## Системные настройки

Системные настройки процесса обогащения:

- `TextParsingService` — адрес cloud-сервиса обогащения данных `Enrichment Service`. Для клиентов on-demand заполняется автоматически. Обязательна для заполнения.
- `CloudServicesAPIKey` — ключ для доступа к API cloud-сервиса. Для клиентов on-demand заполняется автоматически. Обязательна для заполнения.
- `EmailMiningPackageSize` — количество обрабатываемых за один раз писем. Процесс `Email Mining Job` при каждом запуске будет обрабатывать столько писем, сколько указано в системной настройке. Значение по умолчанию — 10.
- `EmailMiningPeriodMin` — периодичность (в минутах) запуска задания `Email Mining Job`.

**Важно.** Если значение `EmailMiningPeriodMin` меньше или равно нулю, то задание не будет запланировано и функциональность будет отключена. Для повторного включения необходимо установить значение настройки  $\geq 1$ , перезапустить application pool приложения Creatio, после чего перейти на страницу логина и войти в приложение.

- `EmailMiningIdentificationActualPeriod` — период актуальности (в днях) идентификации контактов/контрагентов. Если по истечении указанного срока по ранее идентифицированному контакту будет обработано новое письмо, то идентификация будет произведена повторно.

## Последовательность идентификации

### Идентификация контактов

1. Поиск по ФИО.
2. Поиск по фамилии и имени.
3. Поиск по email-адресам. В расчет принимаются только те email-адреса, которые не принадлежат бесплатным или временным почтовым сервисам.
4. Поиск по телефонам. Поиск происходит только по последним цифрам телефонных номеров контакта.

Если на каком-то из этапов идентификации обнаружено совпадение данных, то процесс идентификации будет остановлен.

## Идентификация контрагентов

1. Поиск по колонке [ Название ] или [ Альтернативное название ] без учета регистра символов.
2. Поиск по web-адресу.
3. Поиск по домену из email-адресов. Учитываются только email-адреса, которые не принадлежат бесплатным или временным почтовым сервисам. Из email-адреса выделяется домен и выполняется поиск средств связи контрагента по фильтру “начинается с” одного из представлений домена:  
<http://<домен>>, <https://<домен>>, <http://www.<домен>>, <https://www.<домен>>, <www.<домен>>, <<домен>>.

Если на каком-то из этапов идентификации обнаружено совпадение данных, то процесс идентификации будет остановлен.

## Хэширование информации

Извлеченная из письма информация [хэшируется](#). В результате в таблицах [EnrchTextEntity] и [EnrchEmailData] в поле [Hash] записывается строковое значение хэша, которое однозначно идентифицирует данную единицу или набор извлеченных данных в системе. Это позволяет реализовать два важных улучшения — экономию ресурсов при повторной идентификации контактов/контрагентов из набора извлеченной информации и группировку полученной информации для контакта.

## Повторная идентификация контактов/контрагентов

Например, в систему поступило письмо, в подписи которого указан контакт с ФИО “Иванов Иван Иванович”, телефоном “123-45-67” и адресом “ул. Пушкина, 47, оф. 3”. Для текущего набора данных на основании его содержимого система рассчитала хэш “Hash1” и записала его в поле [Hash] таблицы [EnrchEmailData]. Идентификация контакта выявила в системе контакт “Иванов Иван” и записала полученный результат в таблицу [EnrchFoundContact].

Через некоторое время в систему поступило еще одно письмо с подписью, в которой упоминается контакт “Иванов Иван Иванович” с теми же телефоном и адресом. Система рассчитала для текущего набора данных такой же хэш, как и в прошлый раз — “Hash1”, т.к. входящие данные хэширования не изменились. Вместо того, чтобы создавать новые записи в таблицах [EnrchEmailData], [EnrchTextEntity] и повторно идентифицировать этот контакт, система нашла по полю [Hash] ранее созданную запись в таблице [EnrchEmailData] по полю [Hash] и записала ссылку на эту запись в таблице [Activity].

Таким образом экономится объем хранящихся данных и, что более важно, не производятся ресурсоемкие запросы идентификации контакта.

## Группировка выделенной информации для контакта

Поскольку каждая единица выделенной информации [EnrichTextEntity] имеет хэш-код, основанный на ее содержимом, при обогащении данных существующего контакта появляется возможность использовать информацию, найденную во всех письмах, в которых он участвовал. При выборке данных для обогащения они группируются по полю [Hash] и не дублируются.

# Планировщик заданий Quartz



Сложный

## Рекомендации по настройке планировщика заданий

Все политики Quartz по обработке не отработавших вовремя заданий можно разделить условно на три группы: `Ignore misfire policy` (Игнорировать), `Run immediately and continue` (Выполнить немедленно и продолжать) и `Discard and wait for next` (Отменить и ожидать следующего задания). Далее приведены рекомендации, в каких случаях использовать ту или иную политику.

### Ignore misfire policy

Эта политика представлена одноименной константой `MisfireInstruction.IgnoreMisfirePolicy = -1`. Ее рекомендуется применять в том случае, когда необходимо гарантировать, что все запуски триггера будут обязательно выполнены, даже если имеется несколько просроченных запусков триггера. Например, есть задание A с периодичностью в 2 минуты. Из-за нехватки рабочих потоков Quartz или выключения планировщика время следующего старта задания A (`NEXT_FIRE_TIME`) отстает на 10 минут от текущего времени. Если необходимо, чтобы все 5 просроченных запусков задания обязательно выполнились, то следует использовать политику `IgnoreMisfirePolicy`.

Эту политику рекомендуется применять для триггеров, которые при каждом запуске используют некоторые уникальные данные, и важно, чтобы все запуски триггера были рано или поздно выполнены. Например, есть задание B, которое выполняется с интервалом в 1 час и генерирует отчет за интервал времени от `PREV_FIRE_TIME` до `PREV_FIRE_TIME + 1` час. При этом планировщик был выключен на 8 часов. В таком случае необходимо, чтобы после включения планировщика все 8 просроченных запусков задания B были в конечном итоге запущены, и таким образом сгенерированы отчеты за каждый из часов простоя.

Следует также отметить, что применение этой политики к триггерам, которые не привязаны к каким-либо уникальным данным, может привести к ненужному загромождению очереди планировщика и ухудшению общей производительности приложения. Например, для каждого из пользователей Create настроена синхронизация почты Exchange с интервалом в 1 минуту. При этом 1.5 часа выполнялось обновление. После обновления Quartz, прежде чем приступить к заданиям, запланированным на текущее время, будет выполнять для каждого из пользователей синхронизацию почты 90 раз. Хотя достаточно выполнить просроченную синхронизацию почты однократно, а затем вернуться к выполнению задания согласно расписанию.

### Run immediately and continue

К этой группе политик относятся:

- `SimpleTrigger.FireNow`;
- `SimpleTrigger.RescheduleNowWithExistingRepeatCount`;
- `SimpleTrigger.RescheduleNowWithRemainingRepeatCount`;
- `CronTrigger.FireOnceNow`;
- `CalendarIntervalTrigger.FireOnceNow`.

Эти политики следует применять, если необходимо просроченное задание выполнить как можно скорее, но один раз, а далее выполнять задание согласно расписанию. В такую категорию входят, например, задания по синхронизации почты (такие, как `<user>@<server>_LoadExchangeEmailsProcess_<userId>`, `SyncImap_<user>@<server>_<userId>`), а также задачи `RemindingCountersJob`, `SyncWithLDAPProcess`.

Например, для каждого из пользователей настроена синхронизация почты с интервалом 5 минут, запускаемая триггерами `<user>@<server>_LoadExchangeEmailProcess_<userId>Trigger`. При этом сайт обновлялся с 1:30 до 2:43. Когда сайт был запущен (в 2:43), время следующего запуска для триггеров `<user>@<server>_LoadExchangeEmailsProcess_<userId>Trigger` будет обновлено на текущее (т. е., на 2:43). Соответственно, просроченные задания будут запущены один раз в 2:43 и далее будут выполняться согласно расписанию (т. е., в 2:48, 2:53, 2:58 и т. д.).

## Discard and wait for next

К этой группе политик относятся:

- `SimpleTrigger.RescheduleNextWithRemainingCount`;
- `SimpleTrigger.RescheduleNextWithExistingCount`;
- `CronTrigger.DoNothing`;
- `CalendarIntervalTrigger.DoNothing`.

Эти политики следует применять для заданий, которые необходимо выполнять строго в определенное время. Например, есть задание по сбору статистики, запускаемое каждый день в 3 часа ночи, когда нет активных пользователей на сайте (используется `CronTrigger`). Это задание достаточно ресурсоемкое и длительное, и в рабочее время его запускать нельзя, потому что это может замедлить работу пользователей. В таком случае нужно применить политику `CronTrigger.DoNothing`. В итоге, если по какой-то причине задание не было выполнено вовремя, следующий запуск будет запланирован на 3 часа ночи следующих суток.

## Конфигурирование Quartz

### thread count

Если планировщик не успевает своевременно обрабатывать задания или некоторые задания не выполнились ни разу, то, возможно, поможет увеличение числа потоков Quartz. Для этого необходимо установить необходимое количество потоков в файле `Web.config` загрузчика приложения.

**Установка необходимого количества потоков в файле `Web.config` загрузчика приложения**

```
<add key="quartz.threadPool.threadCount" value="5" />
```

**Важно.** Файл `Web.config` загрузчика приложения расположен в корневом каталоге установленного приложения Creatio.

## misfireThreshold

Если увеличение числа потоков Quartz нежелательно (например, из-за ограниченных ресурсов), то оптимизировать выполнение заданий может изменение настройки `misfireThreshold` в файле `Web.config` загрузчика приложения.

Например, есть приложение с числом заданий, намного большим, чем число потоков. При этом большинство заданий выполняются с достаточно малым интервалом (1 минута). Значение настройки `misfireThreshold` равно одной минуте, число потоков равно 3.

### Изменение настройки `misfireThreshold` в файле `Web.config` загрузчика приложения

```
<add key="quartz.jobStore.misfireThreshold" value="60000" />
<add key="quartz.threadPool.threadCount" value="3" />
```

Также для большинства заданий используются политики из группы `Run immediately and continue`. Это означает следующее. Quartz для большинства заданий, у которых `MISFIRE_INSTR` не равно `-1` и `NEXT_FIRE_TIME` меньше текущего времени на 1 минуту (60000 мс), будет регулярно устанавливать время следующего запуска на текущее время. Это означает, что будет теряться изначальный порядок запланированных заданий, потому что всем заданиям будет установлено время запуска на текущее время. Как следствие, увеличивается вероятность того, что Quartz чаще будет брать в работу одни и те же задания, при этом игнорируя другие.

На рисунке отображена очередь планировщика после 15 минут работы. Задания, у которых значение `PREV_FIRE_TIME` равно `NULL`, ни разу не выполнялись. Очевидно, что таких заданий достаточно много.

Затем следует увеличить значение `misfireThreshold` до 10 минут (очистив при этом `PREV_FIRE_TIME` в `QRTZ_TRIGGERS`).

TRIGGER_NAME	TRIGGER_GROUP	NEXT_FIRE_TIME	PREV_FIRE_TIME	Last repeat interval, min	TRIGGER_STATE	TRIGGER_TYPE	START_TIME	MISFIRE_INSTR	...
MinutelyJob_15Trigger	TestGroup	2017-11-21 19:17:20.860	NULL	NULL	WAITING	SIMPLE	2017-11-21 19:17:20.860	3	...
MinutelyJob_16Trigger	TestGroup	2017-11-21 19:17:20.863	NULL	NULL	WAITING	SIMPLE	2017-11-21 19:17:20.863	3	...
MinutelyJob_17Trigger	TestGroup	2017-11-21 19:17:20.867	NULL	NULL	WAITING	SIMPLE	2017-11-21 19:17:20.867	3	...
MinutelyJob_18Trigger	TestGroup	2017-11-21 19:17:20.867	NULL	NULL	WAITING	SIMPLE	2017-11-21 19:17:20.867	3	...
MinutelyJob_19Trigger	TestGroup	2017-11-21 19:17:20.870	NULL	NULL	WAITING	SIMPLE	2017-11-21 19:17:20.870	3	...
Syncimap_postulga.alexy@gmail.com_73b869f-34f3...	IMAP	2017-11-21 19:17:20.870	NULL	NULL	WAITING	SIMPLE	2017-11-21 19:17:20.870	3	...
MinutelyJob_5Trigger	TestGroup	2017-11-21 19:17:20.873	NULL	NULL	WAITING	SIMPLE	2017-11-21 19:17:20.873	3	...
Terrasoft.Configuration.Campaign.FalloverHandlerTrigger	Campaign.JobGroup	2017-11-21 19:17:20.873	NULL	NULL	WAITING	SIMPLE	2017-11-21 19:17:20.873	3	...
Terrasoft.Configuration.TriggerEmailFailoverHandlerTri...	Mailing	2017-11-21 19:17:20.877	NULL	NULL	WAITING	SIMPLE	2017-11-21 19:17:20.877	3	...
Terrasoft.Configuration.BulkEmailFailoverHandlerTrigger	Mailing	2017-11-21 19:17:20.880	NULL	NULL	WAITING	SIMPLE	2017-11-21 19:17:20.880	3	...
MinutelyJob_6Trigger	TestGroup	2017-11-21 19:17:20.880	NULL	NULL	WAITING	SIMPLE	2017-11-21 19:17:20.880	3	...
MinutelyJob_7Trigger	TestGroup	2017-11-21 19:17:20.883	NULL	NULL	WAITING	SIMPLE	2017-11-21 19:17:20.883	3	...
MinutelyJob_0Trigger	TestGroup	2017-11-21 19:17:20.883	2017-11-21 19:04:37.643	12.72065943000	WAITING	SIMPLE	2017-11-21 19:17:20.883	3	...
MinutelyJob_1Trigger	TestGroup	2017-11-21 19:17:20.887	2017-11-21 19:04:37.663	12.72035926333	WAITING	SIMPLE	2017-11-21 19:17:20.887	3	...
MinutelyJob_2Trigger	TestGroup	2017-11-21 19:17:20.887	2017-11-21 19:04:37.667	12.72034262833	WAITING	SIMPLE	2017-11-21 19:17:20.887	3	...
CESWebHooksSyncTrigger	Mailing	2017-11-21 19:17:20.890	2017-11-21 19:08:20.733	9.00257776333	WAITING	CAL_INT	2017-11-21 18:39:38.140	1	...
MandrillScheduledMailingTrigger	Mailing	2017-11-21 19:17:20.890	2017-11-21 19:08:20.740	9.00252771833	WAITING	CAL_INT	2017-11-21 18:39:38.197	1	...
MT_287467341633338083	DEFAULT	2017-11-21 19:18:20.857	NULL	NULL	WAITING	SIMPLE	2017-11-21 19:08:42.410	0	...
Terrasoft.Configuration.ML.MLModelTrainerJob_Tera...	DEFAULT	2017-11-21 19:18:20.860	NULL	NULL	WAITING	SIMPLE	2017-11-21 19:08:42.433	0	...
Terrasoft.Configuration.EmailMining.EmailMiningJob_T...	DEFAULT	2017-11-21 19:18:20.860	NULL	NULL	WAITING	SIMPLE	2017-11-21 19:08:42.450	0	...
Terrasoft.Configuration.NotificationsJobTrigger	NotificationsCountersGroup	2017-11-21 19:18:20.863	2017-11-21 19:08:20.727	10.00226696166	WAITING	SIMPLE	2017-11-21 19:18:20.863	3	...
A.Postulga@terrasoft.ru_LoadExchangeEmailsProces...	Exchange	2017-11-21 19:18:20.863	2017-11-21 19:08:20.743	10.00205025333	WAITING	SIMPLE	2017-11-21 19:18:20.863	3	...
MinutelyJob_8Trigger	TestGroup	2017-11-21 19:18:20.867	2017-11-21 19:08:20.743	10.00205029000	WAITING	SIMPLE	2017-11-21 19:18:20.867	3	...
MinutelyJob_9Trigger	TestGroup	2017-11-21 19:18:20.870	NULL	NULL	WAITING	SIMPLE	2017-11-21 19:18:20.870	3	...
MinutelyJob_10Trigger	TestGroup	2017-11-21 19:18:20.870	NULL	NULL	WAITING	SIMPLE	2017-11-21 19:18:20.870	3	...
MinutelyJob_11Trigger	TestGroup	2017-11-21 19:18:20.873	NULL	NULL	WAITING	SIMPLE	2017-11-21 19:18:20.873	3	...
MinutelyJob_12Trigger	TestGroup	2017-11-21 19:18:20.873	NULL	NULL	WAITING	SIMPLE	2017-11-21 19:18:20.873	3	...
c999bb9e-070e-411f-8d19-181e6393b6e8	DEFAULT	2017-11-21 19:18:25.517	NULL	NULL	WAITING	SIMPLE	2017-11-21 19:08:25.517	0	...
RemindingCountersJob_73b869f-34f3-4f20-ab4d-748...	RemindingCountersGroup	2017-11-21 19:18:48.613	2017-11-21 19:17:48.613	1.00000000000	BLOCKED	CAL_INT	2017-11-21 19:08:48.613	0	...
MinutelyJob_3Trigger	TestGroup	2017-11-21 19:19:20.723	2017-11-21 19:18:20.723	1.00000000000	BLOCKED	SIMPLE	2017-11-21 19:08:20.723	3	...
MinutelyJob_4Trigger	TestGroup	2017-11-21 19:19:20.723	2017-11-21 19:18:20.723	1.00000000000	BLOCKED	SIMPLE	2017-11-21 19:08:20.723	3	...
Terrasoft.Configuration.DelayedNotifyingTrigger	DelayedNotificationGroup	2017-11-21 19:23:20.697	2017-11-21 19:18:20.697	5.00000000000	WAITING	SIMPLE	2017-11-21 19:08:20.697	3	...
SyncWithLDAPProcessTrigger	LDAP	2017-11-21 19:43:35.327	NULL	NULL	WAITING	CAL_INT	2017-11-21 18:43:35.327	0	...
NotificationCleanerTrigger	NotificationCleanerGroup	2017-11-22 02:00:00.000	NULL	NULL	WAITING	CRON	2017-11-21 18:39:37.000	0	...
ActualizeActiveContactsTrigger	Mailing	2017-11-22 02:00:00.000	NULL	NULL	WAITING	CRON	2017-11-21 18:39:37.997	1	...
GenerateAnniversaryRemindingsTrigger	GenerateAnniversaryRem...	2017-11-22 03:00:00.000	NULL	NULL	WAITING	CRON	2017-11-21 19:08:25.000	0	...
Terrasoft.Configuration.DelayedNotificationCleaningTri...	DelayedNotificationGroup	2017-11-22 18:39:37.843	NULL	NULL	WAITING	SIMPLE	2017-11-21 18:39:37.843	3	...

## Увеличение значения misfireThreshold

```
<add key="quartz.jobStore.misfireThreshold" value="600000" />
```

После 15 минут работы планировщика очередь будет выглядеть следующим образом:

TRIGGER_NAME	TRIGGER_GROUP	NEXT_FIRE_TIME	PREV_FIRE_TIME	Last repeat interval, min	TRIGGER_STATE	TRIGGER_TYPE	START_TIME	MISFIRE_INSTR
MinutelyJob_13Trigger	TestGroup	2017-11-21 19:23:20.960	2017-11-21 19:22:20.960	1.00000000000	WAITING	SIMPLE	2017-11-21 19:20:20.960	3
Terasoft.Configuration.TriggerEmailFailoverHandlerTri...	Mailing	2017-11-21 19:25:20.980	2017-11-21 19:20:20.980	5.00000000000	WAITING	SIMPLE	2017-11-21 19:20:20.980	3
Terasoft.Configuration.BulkEmailFailoverHandlerTrigger	Mailing	2017-11-21 19:25:20.980	2017-11-21 19:20:20.980	5.00000000000	WAITING	SIMPLE	2017-11-21 19:20:20.980	3
RemindingCountersJob_73b869f-34f3-4f20-ab4d-748...	RemindingCountersGroup	2017-11-21 19:25:50.787	2017-11-21 19:24:50.787	1.00000000000	BLOCKED	CAL_INT	2017-11-21 19:22:50.787	0
MinutelyJob_3Trigger	TestGroup	2017-11-21 19:26:20.723	2017-11-21 19:25:20.723	1.00000000000	BLOCKED	SIMPLE	2017-11-21 19:08:20.723	3
MinutelyJob_4Trigger	TestGroup	2017-11-21 19:26:20.723	2017-11-21 19:25:20.723	1.00000000000	BLOCKED	SIMPLE	2017-11-21 19:08:20.723	3
Terasoft.Configuration.DelayedNotifyingTrigger	DelayedNotificationGroup	2017-11-21 19:28:20.697	2017-11-21 19:23:20.697	5.00000000000	WAITING	SIMPLE	2017-11-21 19:08:20.697	3
SyncImap_postulga.alexey@gmail.com_73b869f-34f3...	IMAP	2017-11-21 19:32:39.110	2017-11-21 19:20:20.973	12.30228216333	WAITING	SIMPLE	2017-11-21 19:32:39.110	3
MinutelyJob_5Trigger	TestGroup	2017-11-21 19:32:39.117	2017-11-21 19:20:20.977	12.30234882000	WAITING	SIMPLE	2017-11-21 19:32:39.117	3
MinutelyJob_6Trigger	TestGroup	2017-11-21 19:32:39.117	2017-11-21 19:20:20.983	12.30234896000	WAITING	SIMPLE	2017-11-21 19:32:39.117	3
MinutelyJob_7Trigger	TestGroup	2017-11-21 19:32:39.120	2017-11-21 19:20:20.983	12.30226555000	WAITING	SIMPLE	2017-11-21 19:32:39.120	3
MinutelyJob_0Trigger	TestGroup	2017-11-21 19:32:39.120	2017-11-21 19:20:20.987	12.30224884166	WAITING	SIMPLE	2017-11-21 19:32:39.120	3
MinutelyJob_1Trigger	TestGroup	2017-11-21 19:32:39.123	2017-11-21 19:20:20.987	12.30224826833	WAITING	SIMPLE	2017-11-21 19:32:39.123	3
MinutelyJob_2Trigger	TestGroup	2017-11-21 19:32:39.123	2017-11-21 19:20:20.994	12.30224890666	WAITING	SIMPLE	2017-11-21 19:32:39.123	3
Terasoft.Configuration.NotificationsJob Trigger	NotificationsCountersGroup	2017-11-21 19:32:39.127	2017-11-21 19:20:21.057	12.30116543500	WAITING	SIMPLE	2017-11-21 19:32:39.127	3
A.Postulga@terasoft.ru_LoadExchangeEmailsProces...	Exchange	2017-11-21 19:32:39.130	2017-11-21 19:20:21.060	12.30116547833	WAITING	SIMPLE	2017-11-21 19:32:39.130	3
MinutelyJob_8Trigger	TestGroup	2017-11-21 19:32:39.130	2017-11-21 19:20:21.060	12.30116555333	WAITING	SIMPLE	2017-11-21 19:32:39.130	3
MinutelyJob_9Trigger	TestGroup	2017-11-21 19:32:39.133	2017-11-21 19:20:21.063	12.30116544666	WAITING	SIMPLE	2017-11-21 19:32:39.133	3
MinutelyJob_10Trigger	TestGroup	2017-11-21 19:32:39.133	2017-11-21 19:20:21.067	12.30116546166	WAITING	SIMPLE	2017-11-21 19:32:39.133	3
MinutelyJob_11Trigger	TestGroup	2017-11-21 19:32:39.137	2017-11-21 19:20:21.067	12.30116550166	WAITING	SIMPLE	2017-11-21 19:32:39.137	3
MinutelyJob_12Trigger	TestGroup	2017-11-21 19:32:39.140	2017-11-21 19:20:21.070	12.30118216666	WAITING	SIMPLE	2017-11-21 19:32:39.140	3
CESWebHooksSyncTrigger	Mailing	2017-11-21 19:32:39.143	2017-11-21 19:20:38.140	12.01673093833	WAITING	CAL_INT	2017-11-21 18:39:38.140	1
MandrillScheduledMailingTrigger	Mailing	2017-11-21 19:32:39.143	2017-11-21 19:20:38.197	12.01579751166	WAITING	CAL_INT	2017-11-21 18:39:38.197	1
MinutelyJob_14Trigger	TestGroup	2017-11-21 19:32:39.147	2017-11-21 19:21:20.963	11.30308236666	WAITING	SIMPLE	2017-11-21 19:32:39.147	3
MinutelyJob_15Trigger	TestGroup	2017-11-21 19:32:39.150	2017-11-21 19:21:20.963	11.30308227500	WAITING	SIMPLE	2017-11-21 19:32:39.150	3
MinutelyJob_16Trigger	TestGroup	2017-11-21 19:32:39.150	2017-11-21 19:21:20.967	11.30308224833	WAITING	SIMPLE	2017-11-21 19:32:39.150	3
MinutelyJob_17Trigger	TestGroup	2017-11-21 19:32:39.153	2017-11-21 19:21:20.967	11.30308234666	WAITING	SIMPLE	2017-11-21 19:32:39.153	3
MinutelyJob_18Trigger	TestGroup	2017-11-21 19:32:39.210	2017-11-21 19:21:20.970	11.30399938833	WAITING	SIMPLE	2017-11-21 19:32:39.210	3
MinutelyJob_19Trigger	TestGroup	2017-11-21 19:32:39.213	2017-11-21 19:21:20.970	11.30401564500	WAITING	SIMPLE	2017-11-21 19:32:39.213	3
MT_520821574169140644	DEFAULT	2017-11-21 19:32:39.217	NULL	NULL	WAITING	SIMPLE	2017-11-21 19:22:30.010	0
Terasoft.Configuration.ML.MLModelTrainerJob_Tera...	DEFAULT	2017-11-21 19:32:39.227	NULL	NULL	WAITING	SIMPLE	2017-11-21 19:22:30.037	0
Terasoft.Configuration.EmailMining.EmailMiningJob_T...	DEFAULT	2017-11-21 19:32:39.227	NULL	NULL	WAITING	SIMPLE	2017-11-21 19:22:30.047	0
Terasoft.Configuration.CampaignFailoverHandlerTrigger	CampaignJobGroup	2017-11-21 19:35:20.977	2017-11-21 19:20:20.977	15.00000000000	WAITING	SIMPLE	2017-11-21 19:20:20.977	3
0778a81c-ef91-402a-826a-77ab0e3afb25	DEFAULT	2017-11-21 19:37:25.223	NULL	NULL	WAITING	SIMPLE	2017-11-21 19:22:25.223	0
SyncWithLDAPProcessTrigger	LDAP	2017-11-21 19:43:35.327	NULL	NULL	WAITING	CAL_INT	2017-11-21 18:43:35.327	0
NotificationCleanerTrigger	NotificationCleanerGroup	2017-11-22 02:00:00.000	NULL	NULL	WAITING	CRON	2017-11-21 18:39:37.000	0
ActualizeActiveContactsTrigger	Mailing	2017-11-22 02:00:00.000	NULL	NULL	WAITING	CRON	2017-11-21 18:39:37.997	1
GenerateAnniversaryRemindersTrigger	GenerateAnniversaryRem...	2017-11-22 03:00:00.000	NULL	NULL	WAITING	CRON	2017-11-21 19:22:25.000	0
Terasoft.Configuration.DelayedNotificationCleaningTri...	DelayedNotificationGroup	2017-11-22 18:39:37.843	NULL	NULL	WAITING	SIMPLE	2017-11-21 18:39:37.843	3

Очевидно, что ни разу не запущенных заданий стало намного меньше.

Увеличение `misfireThreshold` приводит к тому, что планировщик более равномерно выполняет задания.

Другими словами, выполняются практически все задания из очереди. Очевидно, что из-за нехватки потоков планировщик не успевает каждое из заданий выполнять через минуту. Это видно по колонке `[Last repeat interval]`, значение в которой равно `NEXT_FIRE_TIME - PREV_FIRE_TIME`, мин. Однако, при этом планировщик выполняет каждое из заданий.

## batchTriggerAcquisitionMaxCount

Увеличение `batchTriggerAcquisitionMaxCount` может оптимизировать работу планировщика, если не используется кластеризованная конфигурация Quartz (используется один узел планировщика).

## Политики Quartz для обработки не отработавших вовремя заданий

В Quartz существуют как политики, общие для всех типов триггеров, так и политики, специфичные для конкретного типа триггера. В таблице перечислены все политики, используемые для триггеров `SimpleTrigger`, `CronTrigger` и `CalendarIntervalTrigger`.

### Политики триггеров

Политика Quartz	Значение	Значение	Тип триггера
<code>MISFIRE_INSTR</code>	<code>Terrasoft.Core.Scheduler.AppSettingsMisfireInstruction</code>		

IgnoreMisfirePolicy

-1

IgnoreMisfirePolicy

для всех типов

**Описание поведения** IgnoreMisfirePolicy

Триггеры с `IgnoreMisfirePolicy` будут обязательно выполнены. Соответственно, для таких триггеров Quartz не будет обновлять время следующего запуска (`NEXT_FIRE_TIME`).

Все просроченные задания Quartz попытается выполнить как можно скорее, после чего вернется к изначальному расписанию триггера. Например, запланировано задание с триггером `SimpleTrigger` на 10 повторений. Исходные условия:

- `START_TIME = 9:00`;
- `REPEAT_COUNT = 9` (первое выполнение + 9 повторений);
- `REPEAT_INTERVAL = 0:15`.

Если планировщик был выключен с 8:50 до 9:20, то при включении Quartz попытается как можно скорее выполнить 2 просроченных задания (в 9:00 и 9:15). Далее он будет выполнять 8 оставшихся заданий по расписанию — в 9:30; 9:45 и т.д.

SmartPolicy

0

SmartPolicy

для всех типов

**Описание поведения** SmartPolicy

Используется Quartz по умолчанию для всех типов триггеров. В зависимости от типа и конфигурации триггера Quartz выберет соответствующую политику. В псевдокоде ниже приведен алгоритм выбора для версии Quartz 2.3.2.

```

if (TRIGGER_TYPE == 'SIMPLE') // Триггер Simple.
    if (REPEAT_COUNT == 0) // Без повторов.
        MISFIRE_INSTR = 1 // SimpleTrigger.FireNow
    else if (REPEAT_COUNT == -1) // Повторять бесконечно.
        MISFIRE_INSTR = 4 // SimpleTrigger.RescheduleNextWithRemainingCount
    else // Указано количество повторов.
        MISFIRE_INSTR = 2 // SimpleTrigger.RescheduleNowWithExistingRepeatCount
else if (TRIGGER_TYPE == 'CAL_INT') // Триггер CalendarInterval.
    MISFIRE_INSTR = 1 // CalendarIntervalTrigger.FireOnceNow
else if (TRIGGER_TYPE == 'CRON') // Триггер Cron.
    MISFIRE_INSTR = 1 // CronTrigger.FireOnceNow

```

SimpleTrigger.FireNow

1

FireNow

SimpleTrigger

**Описание поведения** SimpleTrigger.FireNow

Применяется для триггеров `SimpleTrigger` у которых значение `REPEAT_COUNT` равно 0 (триггеры, рассчитанные на 1 запуск). Если значение `REPEAT_COUNT` не равно 0, то будет применена политика

```
SimpleTrigger.RescheduleNowWithRemainingRepeatCount .
```

Например, запланировано задание с триггером `SimpleTrigger`. Исходные условия:

- `START_TIME = 9:00 ;`
- `REPEAT_COUNT = 0 .`

Если планировщик был выключен с 8:50 до 9:20, то при включении Quartz попытается как можно скорее выполнить задание. В результате, в 9:20 или чуть позже задание будет выполнено.

<code>SimpleTrigger.RescheduleNowWithExistingRepeatCount</code>	2	<code>RescheduleNowWithExistingRepeatCount</code>	<code>SimpleTrigger</code>
---	---	---	----------------------------

#### **Описание поведения `SimpleTrigger.RescheduleNowWithExistingRepeatCount`**

Планировщик пытается как можно скорее выполнить первое просроченное задание. Все остальные запуски триггера будут выполнены с интервалом `REPEAT_INTERVAL`.

Например, запланировано задание с триггером `SimpleTrigger` на 10 повторений. Исходные условия:

- `START_TIME = 9:00 ;`
- `REPEAT_COUNT = 9 ;`
- `REPEAT_INTERVAL = 0:15 .`

Если планировщик был выключен с 8:50 до 9:20, то при включении Quartz выполнит первое просроченное задание, назначенное на 9:00, (из заданий в 9:00 и 9:15) в 9:20. Оставшиеся 9 запусков выполнит в 9:35, 9:50 и т.д..

**На заметку.** Для методов `AppScheduler.ScheduleMinutelyJob` поведение `RescheduleNowWithExistingRepeatCount` идентично `RescheduleNowWithRemainingRepeatCount`, а поведение `RescheduleNextWithRemainingCount` идентично `RescheduleNextWithExistingCount`, поскольку используются триггеры с `REPEAT_COUNT = -1`.

<code>SimpleTrigger.RescheduleNowWithRemainingRepeatCount</code>	3	<code>RescheduleNowWithRemainingRepeatCount</code>	<code>SimpleTrigger</code>
--	---	--	----------------------------

#### **Описание поведения `SimpleTrigger.RescheduleNowWithRemainingRepeatCount`**

Планировщик пытается как можно скорее выполнить первое просроченное задание. Остальные просроченные задания игнорируются. Планировщик выполняет оставшиеся задания, которые не были просрочены, с интервалом `REPEAT_INTERVAL`.

Например, запланировано задание с триггером `SimpleTrigger` на 10 повторений. Исходные условия:

- `START_TIME = 9:00 ;`
- `REPEAT_COUNT = 9 ;`
- `REPEAT_INTERVAL = 0:15 .`

Если планировщик был выключен с 8:50 до 9:20, то при включении Quartz выполнит первое просроченное задание (из заданий в 9:00 и 9:15) в 9:20. Второе просроченное задание будет проигнорировано, и оставшиеся 8 запусков будут выполнены в 9:35, 9:50 и т.д.

SimpleTrigger.Reschedule NextWithRemainingCount	4	RescheduleNextWithRemaining Count	SimpleTrigger
--	---	--------------------------------------	---------------

#### Описание поведения SimpleTrigger.RescheduleNextWithRemainingCount

Планировщик игнорирует просроченные задания и ждет следующего планового запуска задания. При наступлении времени следующего запуска будут выполнены оставшиеся не просроченные задания с интервалом REPEAT\_INTERVAL .

Например, запланировано задание с триггером SimpleTrigger на 10 повторений. Исходные условия:

- START\_TIME = 9:00 ;
- REPEAT\_COUNT = 9 ;
- REPEAT\_INTERVAL = 0:15 .

Если планировщик был выключен с 8:50 до 9:20, то при включении Quartz выполнит оставшиеся 8 непросроченных заданий в 9:30; 9:45 и т.д.

SimpleTrigger.Reschedule NextWithExistingCount	5	RescheduleNextWithExistingCount	SimpleTrigger
---	---	---------------------------------	---------------

#### Описание поведения SimpleTrigger.RescheduleNextWithExistingCount

Планировщик будет ждать следующего времени запуска и выполнит все оставшиеся задания с интервалом REPEAT\_INTERVAL .

Например, запланировано задание с триггером SimpleTrigger на 10 повторений. Исходные условия:

- START\_TIME = 9:00 ;
- REPEAT\_COUNT = 9 ;
- REPEAT\_INTERVAL = 0:15 .

Если планировщик был выключен с 8:50 до 9:20, то при включении Quartz выполнит все 10 заданий в 9:30; 9:45 и т.д.

CronTrigger.FireOnceNow	1	-	CronTrigger
-------------------------	---	---	-------------

#### Описание поведения CronTrigger.FireOnceNow

Планировщик пытается как можно скорее выполнить первое просроченное задание. Остальные просроченные задания игнорируются. Оставшиеся непросроченные задания планировщик выполняет согласно с расписанием.

Например, запланировано задание с триггером CronTrigger : CRON\_EXPRESSION = '0 0 9-17 ? \* MON-FRI' (с понедельника по пятницу с 9:00 до 17:00). Если планировщик был выключен с 8:50 до 10:20, то при включении в 10:20 Quartz выполнит первое просроченное задание из двух (в 9:00 и 10:00).

Далее задания будут выполняться в 11:00, 12:00 и т.д.

CronTrigger.DoNothing	2	-	CronTrigger
-----------------------	---	---	-------------

#### **Описание поведения** CronTrigger.DoNothing

Планировщик игнорирует все просроченные задания. Оставшиеся непросроченные задания выполняются согласно с расписанием.

Например, запланировано задание с триггером CronTrigger : CRON\_EXPRESSION = '0 0 9-17 ? \* MON-FRI' (с понедельника по пятницу с 9:00 до 17:00). Если планировщик был выключен с 8:50 до 10:20, то при включении Quartz начнет выполнять задания с 11:00 (в 11:00, 12:00 и т.д.).

CalendarIntervalTrigger.	1	-	Calendar Interval Trigger
--------------------------	---	---	---------------------------------

#### **Описание поведения** CalendarIntervalTrigger.FireOnceNow

Поведение аналогично CronTrigger.FireOnceNow.

CalendarIntervalTrigger.	2	-	Calendar Interval Trigger
--------------------------	---	---	---------------------------------

#### **Описание поведения** CalendarIntervalTrigger.DoNothing

Поведение аналогично CronTrigger.DoNothing.

Особенности работы с планировщиком при использовании горизонтального масштабирования описаны в статье [Настроить горизонтальное масштабирование](#).

## Сервис бандлирования статического контента



Сложный

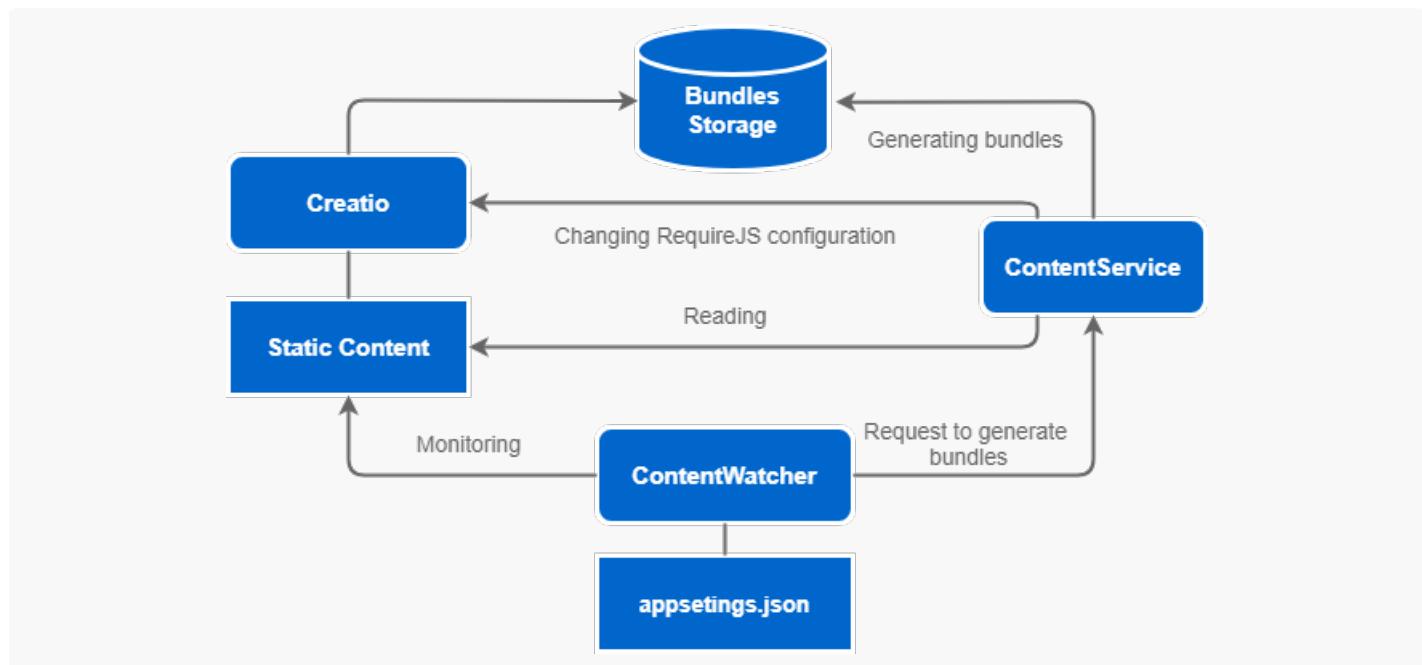
Для повышения производительности весь клиентский контент (исходный код клиентских схем, css-стили) генерируется в специальном каталоге приложения Creatio. Преимущества такого подхода приведены в [статье](#). Однако, из-за большого количества файлов браузеру необходимо выполнять большое количество запросов к приложению при его первоначальной загрузке. Чтобы избежать этого, обычно выполняется объединение всех однотипных файлов в один bundle-файл — бандлирование.

Для объединения однотипных файлов статического контента Creatio разработан сервис бандлирования статического контента.

## Структура и принцип работы

Приложение-наблюдатель (ContentWatcher) мониторит файлы в каталоге со статическим контентом и сообщает об изменениях в них веб-сервису. Веб-сервис (ContentService) выполняет перегенерацию bundle-файлов приложения при поступлении определенных запросов (например, от ContentWatcher или вручную). После бандлирования веб-сервис изменяет определенный конфигурационный файл Creatio таким образом, чтобы он содержал информацию о необходимости использовать bundle-файлы вместо оригинального статического контента.

Общая схема работы сервиса:



**На заметку.** Веб-сервис может быть установлен без приложения-наблюдателя (ContentWatcher). В таком случае запросы к ContentService на бандлирование или [минификацию](#) необходимо выполнять вручную.

**На заметку.** Компоненты сервиса могут быть установлены на том же компьютере, что и Creatio, или на отдельном компьютере. Если они установлены отдельно, то у них должен быть сетевой доступ к файлам статического контента Creatio.

## ContentService

Является .NET Core 2.1 веб-сервисом и имеет следующие операции (точки доступа):

- проверка работоспособности сервиса (метод `GET`).
- генерирует минифицированные bundle-файлы (метод `POST`).
- очищает bundle-файлы (метод `POST`).
- минифицирует контент (метод `POST`).

**Важно.** Каждая операция, (кроме `/`) не только выполняет действия над файлами, но и изменяет конфигурационный файл `_ContentBootstrap.js`.

Параметры операций:

- `ContentPath` — путь к статическому контенту приложения. Обязательный параметр.
- `OutputPath` — локальный или сетевой путь для `ContentService`, куда будут сохранятся бандлированные и/или минифицированные файлы. По умолчанию `ContentPath + "/bundles"`.
- `SchemasBundlesRequireUrl` — путь, который будет использоваться в конфигурационном файле `RequireJs` для бандлированных и/или минифицированных файлов. При изменении `OutputPath` со значения по умолчанию на любое другое этот параметр нужно тоже указывать. По умолчанию считается, что `OutputPath = ContentPath + "/bundles"`.
- `BundleMinLength` — пороговое значение длины одного бандла. Длина, при которой создается новый бандл. По умолчанию 204800Б (200КБ).
- `MinifyJs` — применять ли минификацию к JavaScript файлам (`true / false`). По умолчанию `true`.
- `MinifyCss` — применять ли минификацию к CSS файлам (`true / false`). По умолчанию `true`.
- `MinifyConfigs` — применять ли минификацию к конфигурационным файлам `RequireJs` (`true / false`). По умолчанию `true`.
- `ApplyBundlesForSchemas` — применять ли бандлирование к схемам (`true / false`). По умолчанию `true`.
- `ApplyBundlesForSchemasCss` — применять ли бандлирование к CSS схем (`true / false`). По умолчанию `false`.
- `ApplyBundlesForAloneCss` — применять ли бандлирование к отдельным CSS файлам, у которых нет связанного JavaScript-модуля (`true / false`). По умолчанию `false`.
- `ApplyBundlesForResources` — применять ли бандлирование к ресурсам схем (`true / false`). По умолчанию `true`.

## ContentWatcher

Является .NET Core 2.1 приложением. Запускается как служба (также может запускаться через .NET Core CLI Tools). Основная задача — наблюдать за изменением указанного в параметре `fileFilter` файла, который находится по пути, указанном в параметре `directory`. По умолчанию это файл `_MetaInfo.json`. Изменение файла сообщает об обновлении статического контента. При обнаружении изменений `ContentWatcher` оповещает `ContentService` о необходимости перегенерировать bundle-файлы.

Параметры запуска (указываются в `appsettings.json`):

- `ContentServiceUrl` — ссылка на приложение `ContentService`.
- `CloudServiceId` — идентификатор сервиса для считывания настроек из базы данных AzManager. Необязательный параметр. Необходим для считывания настроек облачной инфраструктуры Creatio.
- `DefaultFileFilter` — имя по умолчанию отслеживаемого файла.
- `AzManagerConnectionString` — строка подключения к базе данных AzManager. Необязательный параметр. Необходим для считывания настроек облачной инфраструктуры Creatio.

- `ConfigurationRefreshInterval` — период обновления конфигурации сервиса.
- `ContentWorkers` — массив конфигурационных объектов всех приложений, за которыми будет следить `ContentWatcher`. Свойства конфигурационных объектов:
  - `name` — имя для отслеживаемого сайта, которое будет отображено в логах работы службы.
  - `directory` — путь к отслеживаемому файлу для настраиваемого сайта.
  - `fileFilter` — имя отслеживаемого файла.
- `ContentWorkerArguments` — массив дополнительных параметров, которые будут передаваться в `ContentService`. Элементы массива — объекты ключ-значение. Свойства объектов:
  - `key` — Ключ. По умолчанию "contentPath".
  - `value` — путь к каталогу с файлами статического контента настраиваемого сайта.

### Пример конфигурационного файла `appsettings.json`

```
{
  "ContentServiceUrl": "http://localhost:9563/process-content",
  "ConfigurationRefreshInterval": "60000",
  "DefaultFileFilter": "_MetaInfo.json",
  "CloudServiceId": "151",
  "AzManagerConnectionString": "Data Source=azserver\\mssql2016;Initial Catalog=azmanager;Inte
  "ContentWorkers": [
    {
      "name": "My Creatio",
      "directory": "C:/Creatio/Terrasoft.WebApp/conf",
      "fileFilter": "_MetaInfo.json",
      "ContentWorkerArguments": [
        {
          "key": "contentPath",
          "value": "C:/Creatio/Terrasoft.WebApp/conf/content"
        }
      ]
    }
  ]
}
```

## Метрики InfluxDb

`ContentWatcher` и `ContentService` при необходимости могут записывать метрики успешных и неуспешных операций в `InfluxDb`. Для этого в соответствующем файле `appsettings.json` следует указать строку `InfluxDbConnectionString` с параметрами подключения:

- `url` — адрес сервера `InfluxDb`, обязательный параметр.
- `db` — имя базы данных, куда будут записываться метрики. Необязательный параметр, значение по

умолчанию: `content`.

- `user` — имя пользователя для соединения с сервером. Необязательный параметр, значение по умолчанию: `<пустая строка>`.
- `password` — пароль для соединения с сервером. Необязательный параметр, значение по умолчанию: `<пустая строка>`.
- `version` — версия сервера InfluxDb. Возможные значения: `Latest`, `v_1_3`, `v_1_0_0`, `v_0_9_6`, `v_0_9_5`, `v_0_9_2`, `v_0_8_x`. Необязательный параметр, значение по умолчанию: `Latest`.

### Пример

```
"InfluxDbConnectionString": "url=http://1.2.3.4:5678; db=content; user=User1; password=QwErTy; v
```

Метрики, записываемые `ContentWatcher`:

- `watcher_notifying_duration` — длительность оповещения `ContentService`.
- `watcher_error` — ошибки при загрузке настроек, мониторинге, оповещении `ContentService`.

Метрики, записываемые `ContentService`:

- `service_processing_duration` — длительность обработки контента.
- `service_error` — ошибки при генерации бандлов, удалении временных папок, очистке бандлов.

# Развернуть сервис в Docker



Сложный

## Системные требования

- сервер под управлением Linux OS (рекомендуются стабильные версии Ubuntu или Debian), с установленной и настроенной стабильной версией docker. С этого сервера должны быть разрешены запросы к хранилищу образов [Docker Hub](#).
- на сервере должны быть установлены Docker и Docker Compose (см. [документацию Docker](#)).

## Структура конфигурации сервисов

- `ets/content-watcher/appsettings.json` — конфигурационный файл ContentWatcher.
- `docker-compose.yml` — конфигурационный файл утилиты docker-compose.
- `.env` — файл с переменными окружения для запуска компонентов.

## Установка сервисов

1. В произвольный каталог (например, `/opt/services`) скачайте файлы конфигурации сервисов и

разархивируйте их.

2. В `./docker-compose/.env` укажите необходимые параметры:

- `ContentServicePort` — порт, на котором будет работать ContentService.
- `ContentPath` — каталог с сайтами/контентом сайтов, который будет смонтирован в контейнер.

3. В `./docker-compose/etc/content-watcher/appsettings.json` настройте список сайтов для отслеживание ContentWatcher (см. выше).

4. Из каталога, в котором находятся конфигурационные файлы (например, `/opt/services`) выполните команду:

#### **Команда для загрузки необходимых docker-образов сервисов**

```
sudo docker-compose pull
```

Результатом выполнения команды будет загрузка из hub.docker.com необходимых docker-образов сервисов.

**Важно.** Если на сервере запрещен доступ в интернет, скачайте на машине с открытым доступом все необходимые образы вручную (см.конфигурационный файл `docker-compose.yml`). Затем воспользуйтесь командами `sudo docker export` и `sudo docker import` для переноса образов в виде файлов на целевую машину.

5. Из каталога, в котором находятся конфигурационные файлы (например, `/opt/services`) выполните команду:

#### **Команда для запуска сервисов**

```
sudo docker-compose up -d
```

Результатом выполнения команды будет запуск сервисов.

6. Проверьте установку, отправив `GET`-запрос по адресу: `{IP-адрес сервера}: {ContentServicePort}`, где:

- `IP-адрес сервера` — IP-адрес сервера, на котором установлены сервисы.
- `ContentServicePort` — порт, на котором работает ContentService. Должен совпадать с портом, указанном в файле `./docker-compose/.env` (см. п. 2).

#### **Пример**

`10.17.17.7:9999`

Ожидаемый ответ

"Service is running"