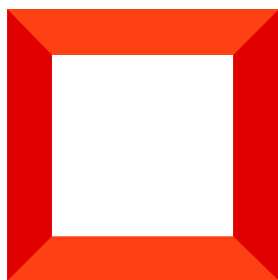
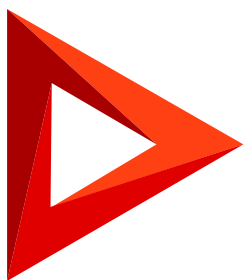


Основные элементы интерфейса

Версия 7.18



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

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

Содержание

Общие принципы работы	11
Главное меню	12
Раздел	14
Интерфейс элемента "Раздел"	14
Реестр раздела	15
Аналитика раздела	18
Действия раздела	20
Фильтры	23
Теги	26
Создать новый раздел	26
Удалить раздел	28
Создать новый раздел	29
1. С помощью мастера разделов создать раздел и добавить его в нужное рабочее место	29
2. Добавить необходимые колонки в схему объекта раздела и настроить их отображение	30
Результат выполнения примера	31
Добавить действие раздела для единичной записи	32
1. В пользовательском пакете создать схему замещающей модели представления раздела Заказы	32
2. В коллекцию локализуемых строк схемы замещающей модели представления раздела добавить строку с заголовком пункта меню Действия	33
3. В коллекцию методов схемы замещающей модели представления раздела добавить реализацию методов	35
Результат выполнения примера	33
Добавить действие раздела для нескольких записей	36
1. В пользовательском пакете создать схему замещающей модели представления раздела Заказы	36
2. В коллекцию локализуемых строк схемы замещающей модели представления раздела добавить строку с заголовком пункта меню Действия	36
3. В коллекцию методов схемы замещающей модели представления раздела добавить реализацию методов	39
Результат выполнения примера	37
Обработка выбора нескольких записей. Примеры	39
Пример 1	39
Пример 2	42
Настроить внешний вид реестра	46
1. В пользовательском пакете создать схему замещающей модели представления раздела Заказы	46
2. Переопределить метод prepareResponseCollectionItem	47
Результат выполнения примера	48
Добавить в раздел блок быстрых фильтров	48
1. В пользовательском пакете создать схему замещающей модели представления раздела Договоры (Contracts)	49

2. В структуру схемы добавить локализуемые строки	49
3. В коллекцию методов схемы замещающей модели представления раздела добавить реализацию метода <code>initFixedFiltersConfig()</code>	50
Результат выполнения примера	51
Добавить редактируемый реестр в раздел	51
1. Добавить пользовательскую колонку в объект [Продукт в заказе] ([Product in order])	52
2. Создать схему замещающей модели представления для страницы подбора продуктов	52
3. Настроить отображение колонок на странице подбора продуктов	54
Результат выполнения примера	54
Кнопка	54
DOM-модель стандартных кнопок страницы	55
Особенности кнопки выбора цвета	56
Добавить к кнопке всплывающую подсказку	56
Добавить кнопку в раздел	57
1. Создать схему замещающей модели представления страницы раздела Контрагенты	57
2. В коллекцию локализуемых строк схемы замещающей модели представления раздела добавить строку с заголовком кнопки	57
3. В коллекцию методов модели представления раздела добавить реализацию методов	58
4. В массив <code>diff</code> добавить конфигурационный объект с настройками расположения кнопки на странице	58
Результат выполнения примера	60
Добавить кнопку в совмещенном режиме	60
Описание примера	61
Исходный код	61
Алгоритм реализации примера	61
Добавить кнопку в режиме добавления новой записи	66
Описание примера	66
Исходный код	66
Алгоритм реализации примера	67
Добавить кнопку выбора цвета	70
Описание примера	70
Исходный код	70
Алгоритм реализации примера	70
Добавить к кнопке всплывающую подсказку	74
Исходный код	74
Пример 1	74
Свойство <code>diff</code> объекта кнопки	83
Свойства	83
Поле	86
Типы полей	86

Действия над полями	89
Добавить новое поле	93
Исходный код	93
Пример 1	93
Пример 2	97
Добавить поле с изображением	98
Описание примера	98
Исходный код	98
Алгоритм реализации примера	99
Добавить вычисляемые поля	107
Описание примера	107
Исходный код	107
Алгоритм реализации примера	107
Добавить мультивалютное поле	111
Описание примера	111
Исходный код	111
Алгоритм реализации примера	111
Добавить валидацию к полю	118
Исходный код	118
Пример 1	118
Пример 2	121
Установить для поля значение по умолчанию	124
Исходный код	124
Пример установки для поля значения по умолчанию через свойства колонки объекта	125
Пример установки значения по умолчанию в исходном коде страницы записи	127
Сделать поле обязательным для заполнения по условию	129
Описание примера	129
Исходный код	129
Алгоритм реализации примера	129
Добавить разницу дат в полях страницы	132
Отфильтровать справочное поле	134
Описание примера	134
Исходный код	135
Алгоритм реализации примера	135
Применить фильтрацию к справочным полям	139
Описание примера	140
Исходный код	140
Алгоритм реализации примера	140
Заблокировать поля	142

Описание примера	142
Исходный код	143
Алгоритм реализации примера	143
Заблокировать поле по условию	145
Описание примера	145
Исходный код	145
Алгоритм реализации примера	145
Скрыть поле по условию	148
Описание примера	148
Исходный код	148
Алгоритм реализации примера	149
Добавить автонумерацию к полю	153
Описание примера	153
Исходный код	153
Алгоритм реализации кейса на стороне клиента	153
Алгоритм реализации кейса на стороне сервера	156
Страница записи	162
Настройка логики страницы записи	163
Добавить действие на страницу записи	163
Добавить действие на страницу записи	166
Описание примера	166
Исходный код	166
Алгоритм реализации примера	166
Отобразить часовой пояс контакта	171
Описание примера	171
Исходный код	171
Алгоритм реализации примера	171
Деталь	176
Множественное добавление записей на деталь	181
Удалить деталь	181
Настроить деталь с полями	182
1. Создать пользовательскую деталь	182
2. Настроить пользовательскую деталь	186
3. Добавить деталь на страницу записи раздела	188
4. Добавить пользовательские стили детали	189
5. Добавить валидацию к полю детали	192
6. Сделать виртуальными поля детали	194
Скрыть пункты меню детали с реестром	195
Создать схему замещающей модели представления реестра детали	195

Результат выполнения примера	197
Реализовать множественное добавление записей на деталь	197
1. Создать схему замещающей модели представления детали	198
2. Реализуйте бизнес-логику детали	199
Результат выполнения примера	201
Реализовать деталь типа [Файлы и ссылки]	202
1. Создать пользовательский раздел	203
2. Создать пользовательскую деталь	204
3. Настроить пользовательскую деталь	205
4. Добавить деталь в раздел	206
5. Добавить пользовательские стили детали	207
Результат выполнения примера	211
Схема BaseDetailV2	212
Сообщения	212
Атрибуты	212
Методы	213
Массив модификаций	215
Схема BaseGridDetailV2	215
Сообщения	216
Атрибуты	216
Миксины	217
Методы	218
Массив модификаций	221
Миксин GridUtilitiesV2	222
Методы	223
Модуль ConfigurationGrid	225
Методы	226
Модуль ConfigurationGridGenerator	227
Методы	227
Модуль ConfigurationGridUtilities	227
Свойства	227
Методы	228
Мини-карточка	229
Схема модели представления мини-карточки	230
Операции с мини-карточками	231
Создать пользовательскую мини-карточку	232
1. Создать схему модели представления мини-карточки	232
2. Отобразить поля основного объекта	233
3. Добавить функциональную кнопку в мини-карточку	234

4. Выполнить стилизацию мини-карточки	238
5. Зарегистрировать мини-карточку в базе данных	242
6. Добавить системную настройку	243
Результат выполнения примера	243
Создать мини-карточку добавления	244
1. Создать схему модели представления мини-карточки	244
2. Отобразить поля основного объекта	245
3. Зарегистрировать мини-карточку в базе данных	247
4. Добавить системную настройку	247
Результат выполнения примера	248
Добавить мини-карточку к произвольному модулю	249
1. Создать схему модуля	249
2. Создать представление и модель представления модуля	250
3. Добавить стили модуля	252
4. Создать контейнер отображения представления	253
Результат выполнения примера	255
Модальное окно	256
Коммуникационная панель	256
Добавить всплывающую кнопку для сообщений из внутренней ленты обращений	258
Описание примера	258
Исходный код	258
Алгоритм реализации примера	259
Скрыть область ленты в едином окне	262
Командная строка	264
Инструментальная панель действий	265
Добавить инструментальную панель действий на страницу	266
Добавить новый канал на инструментальную панель действий	266
Добавить пользовательское действие верификации	267
Добавить инструментальную панель действий	269
Описание примера	269
Исходный код	269
Алгоритм реализации примера	269
Добавить новый канал на инструментальную панель действий	272
Описание примера	272
Исходный код	272
Алгоритм выполнения примера	273
Добавить мультиязычные шаблоны email-сообщений	279
Описание примера	280
Исходный код	280

Предварительные настройки	280
Алгоритм реализации примера	281
Дашборды	286
Структура данных для хранения информации по итогам	287
Реализация функциональности в режиме просмотра итогов	288
Реализация функциональности в режиме настройки итогов	289
Базовые классы, реализующие функциональность дашборда	289
Виды дашбордов	290
Изменить расчеты в воронке продаж	297
Последовательность действий для изменения расчетов в воронке:	297
Пример изменения отображения расчетов в воронке продаж в срезе “по количеству”	297
Подключить дополнительную фильтрацию к воронке	302
Описание примера	302
Исходный код	303
Алгоритм реализации примера	303
Добавить пользовательский дашборд	305
Описание примера	305
Исходный код	305
Алгоритм реализации примера	305
Схема BaseWidgetDesigner	312
Методы	312
Перечисление DashboardEnums	313
Свойства	313
Хронология	313
Таблицы базы данных	313
Добавление вкладки Хронология в раздел	316
Пример использования базовой плитки	317
Алгоритм реализации примера	317
Создать плитки вкладки [Хронология], связанные с пользовательским разделом	319
Настройка плитки	319
Алгоритм создания преднастроенной плитки	320
Алгоритм реализации примера	321
Профиль связанной сущности	330
Классы, реализующие функциональность	331
Устройство BaseProfileSchema	333
Схема профиля BaseMultipleProfileSchema	333
Пример конфигурации схемы профиля	333
Реализация примера	334
Пример встраивания профиля в страницу записи	335

Реализация примера	335
Схема профиля BaseMultipleProfileSchema	336
Пример модуля клиентского профиля BaseMultipleProfileSchema	343
Реализация примера	343
Пример встраиваемого клиентского профиля BaseRelatedProfileSchema	347
Реализация примера	347
Пример встраивания модуля клиентского профиля на страницу записи	349
Реализация примера	349

Общие принципы работы



Элементы управления — это объекты, используемые для организации интерфейса между пользователем и приложением. Как правило, они отображаются в навигационных панелях, диалоговых окнах и на панелях инструментов. К ним относятся кнопки, признаки (флажки), поля и т. д.

Все элементы управления наследуются от класса `Terrasoft.Component` (полное имя — `Terrasoft.controls.Component`), который, в свою очередь, наследуется от `Terrasoft.BaseObject`. Благодаря тому, что в родительском классе (`Component`) объявлен миксин `Bindable`, имеется возможность связывания свойств элемента управления с требуемыми свойствами, методами или атрибутами модели представления.

В элементе управления объявляются события, необходимые для корректной работы. Также каждый элемент содержит в себе события, унаследованные от класса `Terrasoft.Component`:

- `added` — срабатывает после того, как компонент был добавлен в контейнер.
- `afterrender` — срабатывает после того, как компонент был отрендерен и его HTML-представление попало в DOM.
- `afterrerender` — срабатывает после того, как компонент отрабатывает ререндеринг и его HTML представление обновилось в DOM.
- `beforererender` — срабатывает перед тем, как компонент был отрендерен и его HTML представление попало в DOM.
- `destroy` — срабатывает перед окончательным уничтожением элемента управления.
- `destroyed` — срабатывает после удаления компонента.
- `init` — срабатывает, когда инициализация компонента завершена.

Подробнее о событиях класса `Terrasoft.Component` можно узнать из [библиотеки классов клиентской части ядра платформы](#).

Также элемент управления может подписываться на события браузера и определять свои собственные события.

Элемент управления определяется в массиве модификаций `diff` модуля, в котором его необходимо расположить.

Массив модификаций `diff` модуля

```
// Массив модификаций diff модуля.
diff: [{
  // Операция вставки.
  "operation": "insert",
  // Имя элемента-родителя, в который осуществится вставка элемента управления.
  "parentName": "CardContentContainer",
  // Имя свойства элемента-родителя, с которым производится операция.
```

```

"propertyName": "items",
// Имя элемента управления.
"name": "ExampleButton",
// Значения элемента управления.
"values": {
  // Тип элемента управления.
  "itemType": "Terrasoft.ViewItemType.BUTTON",
  // Заголовок элемента управления.
  "caption": "ExampleButton",
  // Связывание события элемента управления с функцией.
  "click": {"bindTo": "onExampleButtonClick"},
  // Стил элемент управления.
  "style": Terrasoft.controls.ButtonEnums.style.GREEN
}
}]

```

Внешний вид элемента управления определяется шаблоном (`template <tpl>`). Во время рендеринга элемента управления в представление страницы генерация представления элемента выполняется согласно заданному шаблону.

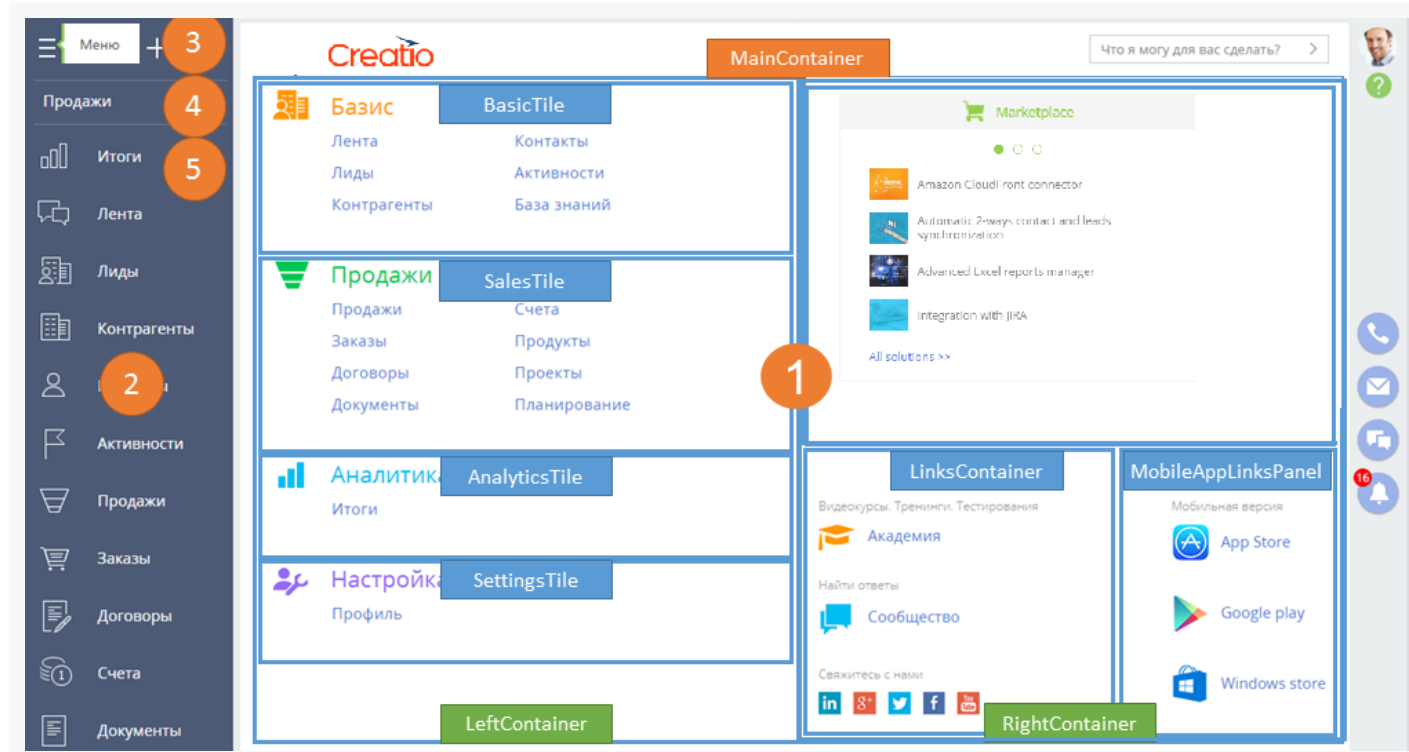
Элемент управления не должен иметь никакой бизнес логики — она определяется в модуле, в который добавляется элемент управления.

Элемент управления имеет атрибуты `styles` , `selectors` , которые определены в классе-родителе `Terrasoft.Component` . Эти атрибуты предоставляют возможность гибкой настройки стилей.

Главное меню

Основы

Главное меню приложения отображается в рабочей области (1) пользовательского интерфейса после загрузки приложения. Также главное меню можно вызвать, нажав на кнопку [*Меню*], расположенную в верхнем меню (3) боковой панели (2).



Команды главного меню, при помощи которых происходит переход в соответствующий раздел приложения, также отображены на боковой панели (2). При этом команды перехода в разделы отображаются в зависимости от рабочего места, выбранного при помощи меню на боковой панели (4).

Главному меню соответствует базовая схема бизнес-объекта `ApplicationMainMenu` и схема главного меню продукта, унаследованная от схемы главного меню для базового продукта `SimpleIntro`. Например, для продукта `SalesEnterprise` схема главного меню продукта имеет название `EnterpriseIntro`.

Состав элементов пользовательского интерфейса главного меню зависит от продукта. Все элементы размещены в соответствующих контейнерах, которые конфигурируются в базовой либо унаследованной схеме главного меню. К основным контейнерам продукта `SalesEnterprise` относятся:

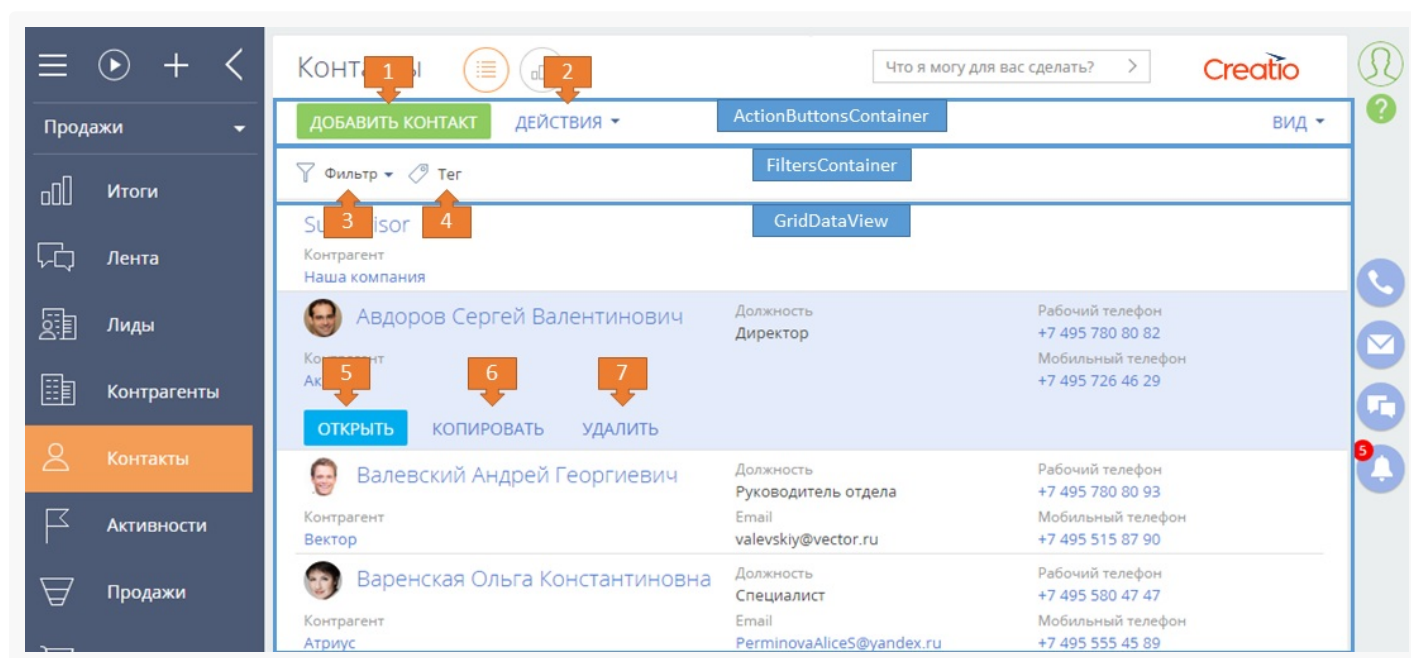
- Основной контейнер меню (`MainContainer`), содержащий в себе все элементы главного меню.
- Контейнер разделов и настроек (`LeftContainer`), который содержит секции команд перехода в разделы и секцию настроек.
- Контейнер ресурсов (`RightContainer`), содержащий секции ссылок на разные ресурсы.
- Контейнер базовой функциональности (`BasicTile`) — содержит команды перехода в разделы, общие для всех продуктов.
- Контейнер продаж (`SalesTile`) — содержит команды перехода в разделы продуктов семейства `Sales`.
- Контейнер аналитики (`AnalyticsTile`) — содержит команду перехода в раздел [*Итоги*].
- Контейнер настроек (`SettingsTile`) — содержит команды перехода в разделы настроек.
- Контейнер видео (`VideoPanel`) — содержит проигрыватель и название видеоролика.
- Контейнер ссылок (`LinksContainer`) — содержит ссылки на обучающие и социальные ресурсы.
- Контейнер ссылок мобильного приложения (`MobileAppLinksPanel`) — содержит ссылки на мобильное приложение в основных online-магазинах.

Раздел

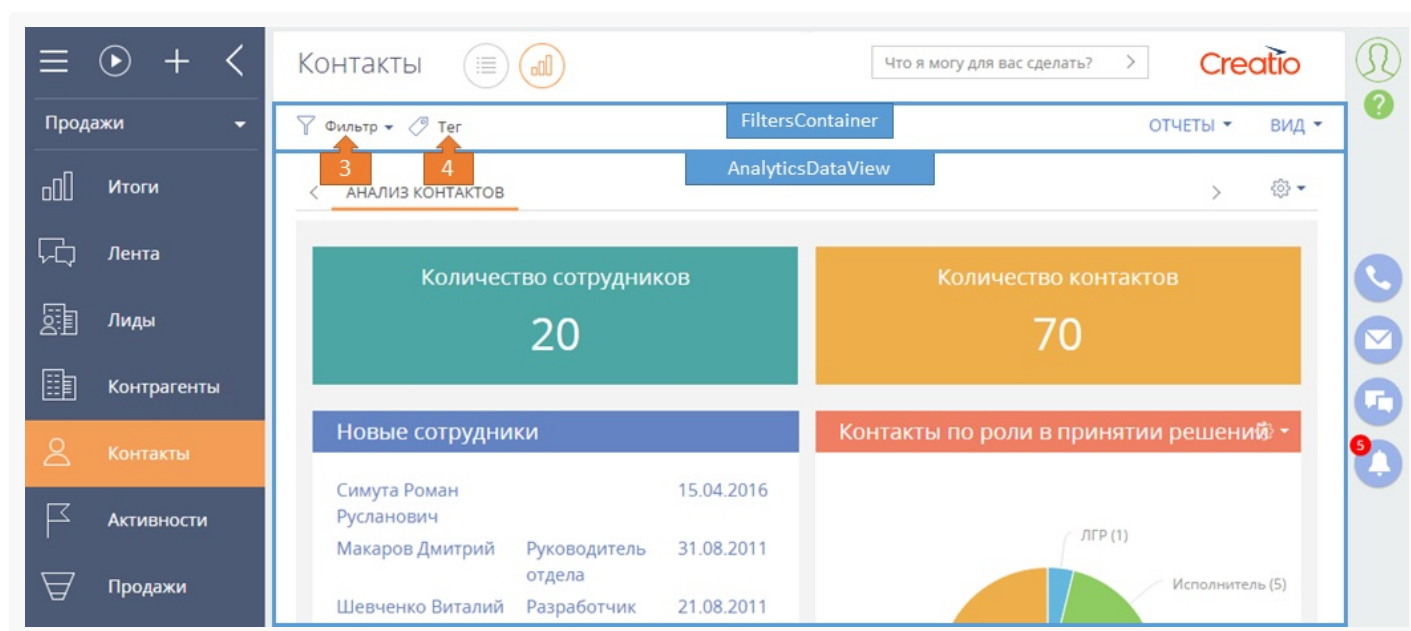
Оснoвы

Интерфейс элемента "Раздел"

Элемент "Раздел" отображается в рабочей области пользовательского интерфейса после выбора соответствующего пункта меню в боковой панели либо на главной странице приложения.



Как правило, раздел имеет два представления: отображение данных в реестре раздела и отображение аналитики раздела.



Каждому разделу соответствует определенная схема. Например, конфигурирование раздела [*Контакты*] выполняется в схеме `ContactSectionV2`. Все схемы разделов унаследованы от родительской схемы `BaseSectionV2`.

Элементы пользовательского интерфейса приложения, относящиеся к разделу, размещены в соответствующих контейнерах, которые конфигурируются в базовой или унаследованной схеме раздела. К основным контейнерам раздела относятся:

- контейнер кнопок действий (`ActionButtonsContainer`), в котором размещены кнопка действия раздела (1) и кнопка с выпадающим списком действий (2);
- контейнер фильтров (`FiltersContainer`), в котором размещаются фильтры (3) и теги (4);
- контейнер отображения реестра раздела (`GridDataView`), в активной записи которого размещаются кнопки действий редактирования (5), копирования (6) и удаления (7) текущей записи;
- контейнер отображения аналитики раздела (`AnalyticsDataView`).

Расположение и состав основных контейнеров раздела, в зависимости от режима отображения данных, показаны на рисунках.

К основным элементам интерфейса и функциональным элементам раздела относятся: реестр, аналитика раздела, действия, фильтры и теги.

Реестр раздела — компонент, в котором в плиточном или списочном представлении отображается список записей, добавленных в раздел. Реестр раздела отображается в контейнере `GridDataView`.

Аналитика раздела — используется для визуализации статистических данных с помощью графиков, единичных показателей или списков. Блоки итогов и пользовательские дашборды отображаются в контейнере отображения аналитики раздела `AnalyticsDataView`.

Действия раздела — функциональные элементы раздела, которые представляют собой набор операций над активной записью реестра. Действия могут быть вызваны при помощи кнопок разного вида, размещенных в контейнере `ActionsButtons` и контейнере активной записи.

Фильтр — инструмент для поиска и сегментации записей системы по заданным условиям. Различают быстрый, стандартный, расширенный фильтры, а также группы фильтрации. Кнопки [*Фильтр*] размещаются в контейнере фильтров `FiltersContainer`.

Тег — специальная метка, которую можно использовать для сегментации записей вручную. Как и фильтр, размещен в контейнере фильтров.

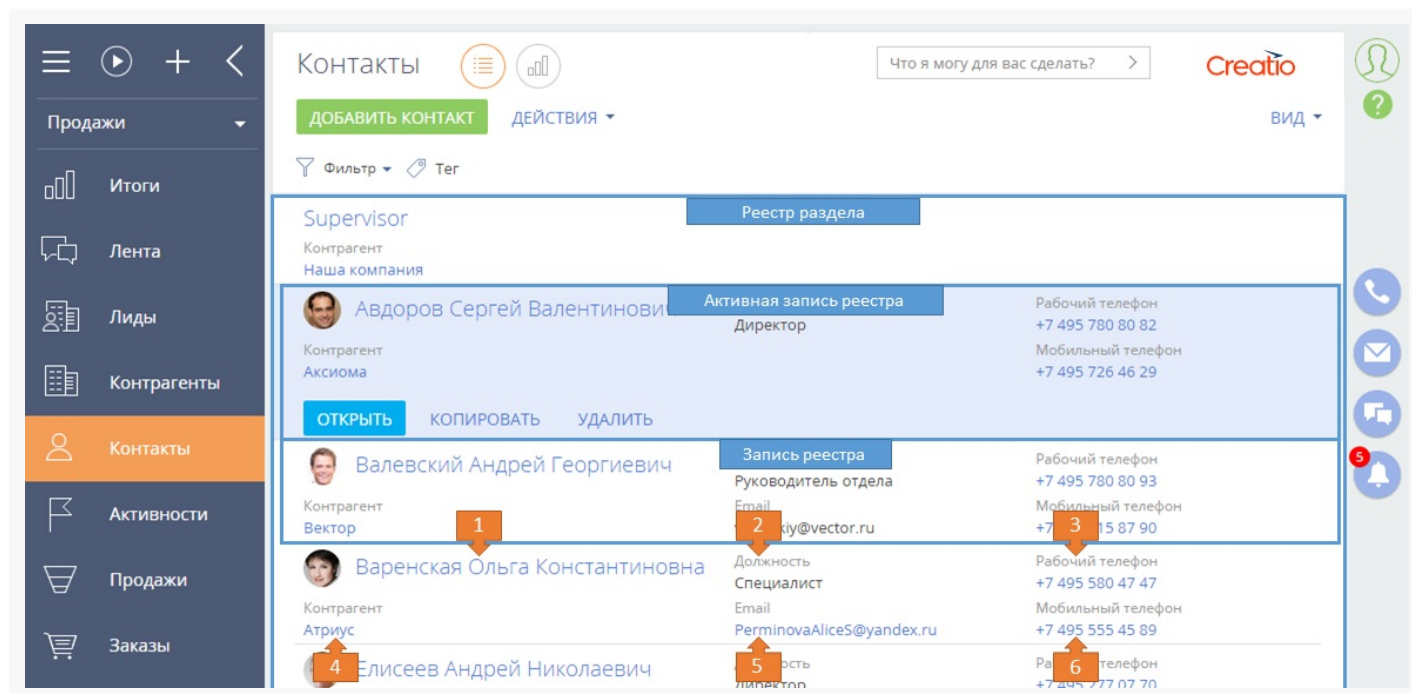
Реестр раздела

Реестр раздела представляет собой список записей, каждая из которых может отображаться в одном из двух представлений.

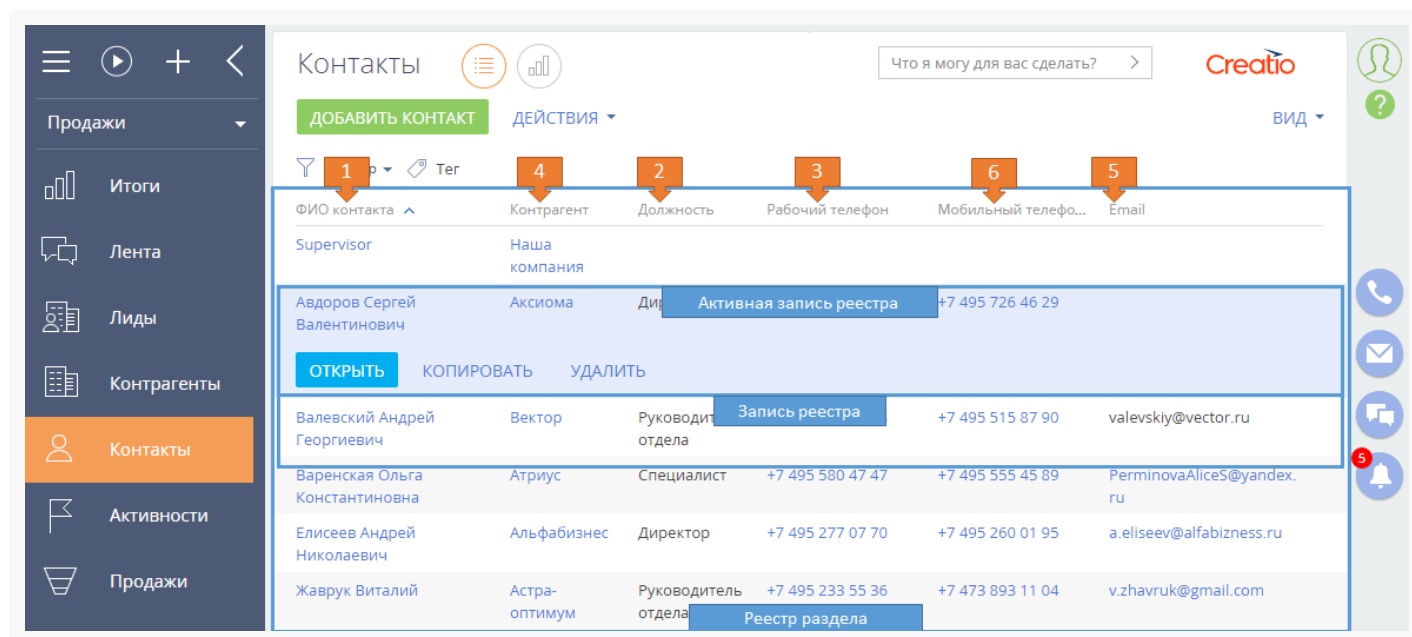
Плиточное представление отображает поля каждой записи в несколько строк. Это представление для реестра установлено по умолчанию. Так, например, для раздела [*Контакты*] в каждой записи отображены следующие поля:

- ФИО (1);
- Должность (2);
- Рабочий телефон (3);
- Контрагент (4);

- Email (5);
- Мобильный телефон (6).



Списочное представление отображает записи в виде простой таблицы, в которой каждой записи соответствует одна строка. При этом последовательность расположения полей в списочном представлении может не совпадать с последовательностью расположения полей в плиточном представлении.



Чтобы избежать избыточности представляемых данных, реестр раздела отображает как правило только самые значимые колонки таблицы. Все данные отображаются и редактируются на [странице записи](#)

[раздела.](#)

Каждому разделу соответствует своя схема бизнес-объекта, описывающая структуру таблицы базы данных, в которой и хранятся данные записей, а также описывающая определенные инструкции по обработке этих данных. Из этих данных и формируется реестр раздела. Условно каждая строка таблицы соответствует записи раздела. Так, например, разделу [*Контакты*] соответствует схема бизнес-объекта `Contact`, которая содержит перечень и свойства колонок таблицы `[Contact]`.

Схема объекта `Contact` в дизайнера элементов конфигурации

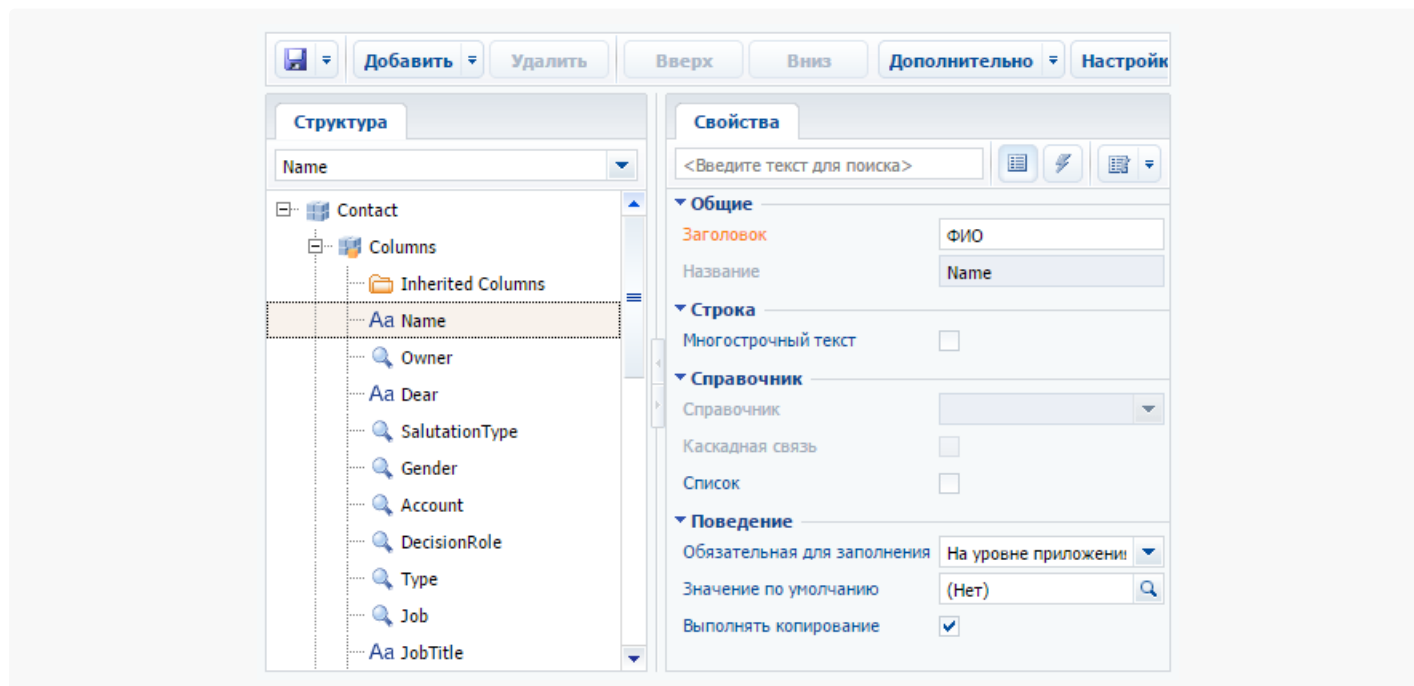


Таблица `[Contact]`

Name	OwnerId	Dear	SalutationTypeId	GenderId	AccountId	DecisionRoleId
Supervisor	NULL		7426FFB3-56E6-DF1...	EEAC42EE-6...	E308B781-3C...	NULL
Авдоров Сергей Валентино...	DAD159F3-6...	Сергей Ва...	7426FFB3-56E6-DF1...	EEAC42EE-6...	4BCC8FCE-9B...	ED46BB3A-57E6-D.
Валевский Андрей Георгие...	370C072F-35...	Андрей	7426FFB3-56E6-DF1...	EEAC42EE-6...	7999AF10-394...	D2AB0E4E-57E6-D.
Варенская Ольга Констант...	D73F2E3A-F4...	Ольга Ко...	7526FFB3-56E6-DF1...	FC2483F8-65...	9348A4CA-B4...	91E32C57-57E6-DF.
Елисеев Андрей Николаевич	DAD159F3-6...	Андрей Н...	7426FFB3-56E6-DF1...	EEAC42EE-6...	64573E47-A4...	NULL
Жаврук Виталий	DAD159F3-6...	Виталий В...	7426FFB3-56E6-DF1...	EEAC42EE-6...	C1469C4D-4D...	D2AB0E4E-57E6-D.
Захарцев Александр Серге...	DAD159F3-6...	Александ...	7426FFB3-56E6-DF1...	EEAC42EE-6...	EEFCD4D8-B6...	ED46BB3A-57E6-D.
Золотов Ярослав Викторо...	C7399B13-F8...	Ярослав ...	7426FFB3-56E6-DF1...	EEAC42EE-6...	63A7961D-CE...	NULL
Карасев Евгений Леонидо...	DAD159F3-6...		7426FFB3-56E6-DF1...	EEAC42EE-6...	4CA7AA49-06...	NULL
Кирдин Анатолий Алексан...	DAD159F3-6...	Анатолий ...	7426FFB3-56E6-DF1...	EEAC42EE-6...	BC3BCE26-18...	D2AB0E4E-57E6-D.
Комаров Александр Андре...	DAD159F3-6...	Александ...	7426FFB3-56E6-DF1...	EEAC42EE-6...	4BCC8FCE-9B...	NULL
Латышев Александр Иван...	42008077-16...	Александ...	7426FFB3-56E6-DF1...	EEAC42EE-6...	3D5C13B7-D2...	NULL
Левченков Сергей	C7399B13-F8...		NULL	NULL	29F8FAF4-146...	NULL

Положение и состав отображаемых полей в записях реестра раздела для существующих колонок схемы бизнес-объекта можно изменить при помощи мастера раздела или мастера настройки списка, доступных из меню кнопки [*Вид*].

Если же необходимо добавить пользовательскую колонку в схему бизнес-объекта и отобразить ее в реестре, то это можно выполнить двумя способами.

Первый способ — использовать мастер разделов. При этом в текущем пользовательском пакете будет

создана схема замещающей модели представления схемы `Contact`, который унаследует все колонки базового объекта `Contact` из пакета `Base`, и к которому будет добавлена новая пользовательская колонка.

Второй способ — используя дизайнер элементов конфигурации, создать в пользовательском пакете схему замещающего объекта `Contact`, который унаследует все колонки базового объекта `Contact` из пакета `Base`. Добавить к замещающему объекту необходимые колонки и настроить их свойства. Далее, используя мастер разделов или страницу настройки колонок, настроить отображение добавленных колонок в реестре.

Настроить внешний вид реестра

В Creatio существует возможность **настраивать внешний вид реестра**, выделяя в нем определенные записи при выполнении заданного условия. Такая настройка помогает выделять записи, на которые необходимо обратить особое внимание.

За отображение строки реестра отвечает свойство `customStyle` записи реестра.

Свойство `customStyle` — это объект, свойства которого являются аналогами CSS-свойств и формируют стиль отображения записи реестра. Например:

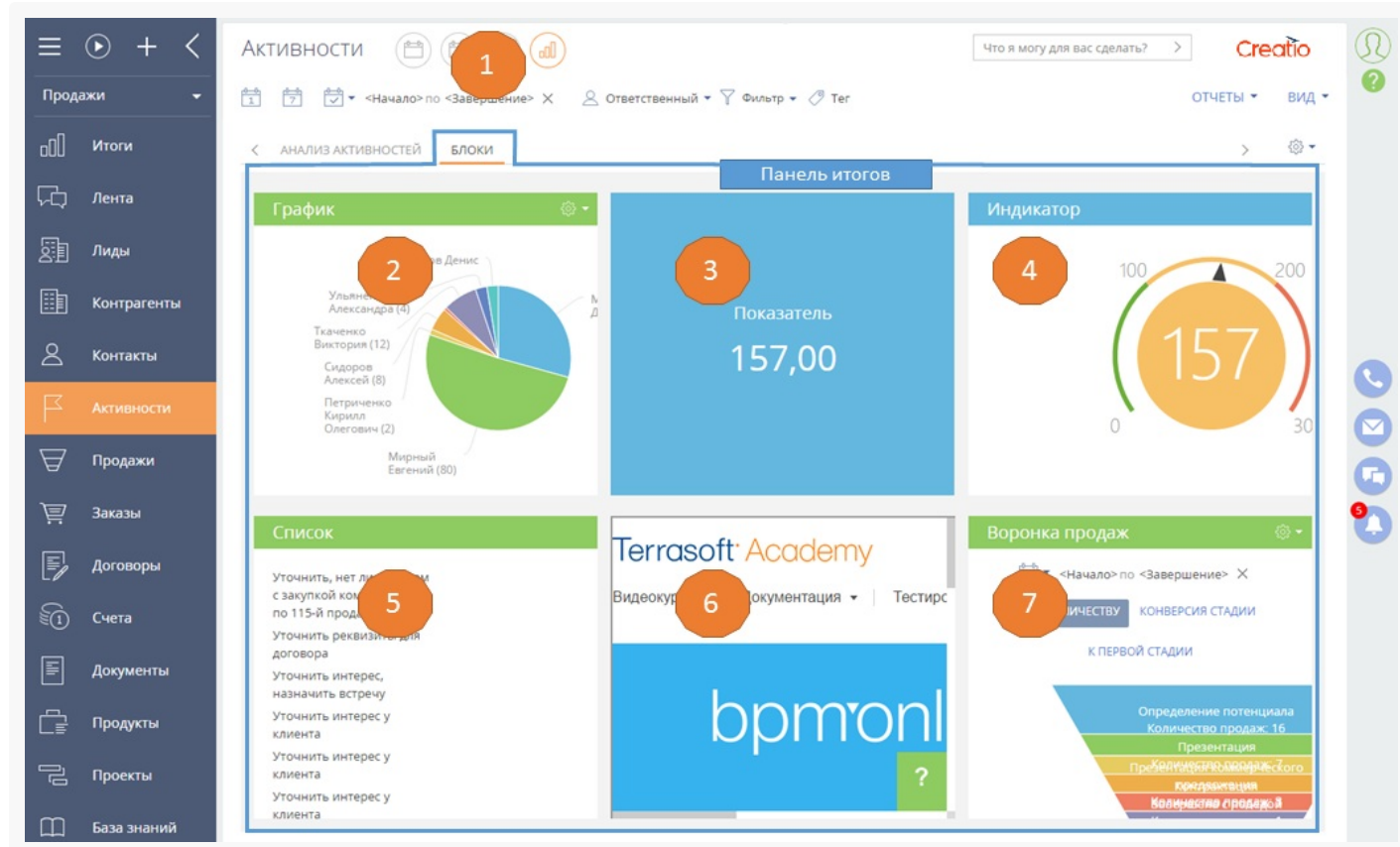
```
item.customStyle = {
    // Цвет текста – белый.
    "color": "white",
    // Цвет фона – оранжевый.
    "background": "orange"
};
```

Для настройки отображения отдельных записей реестра выполните следующие действия:

1. В схеме замещающей модели представления раздела переопределите базовый метод `prepareResponseCollectionItem(item)`, который модифицирует строку данных перед загрузкой в реестр.
2. Реализуйте в методе `prepareResponseCollectionItem(item)` присвоение определенного значения свойству `customStyle` для необходимых записей реестра.

Аналитика раздела

Для анализа статистических данных разделов используются элементы аналитики Creatio. Для работы с аналитикой раздела необходимо перейти в представление аналитики необходимого раздела, нажав на кнопку [*Итоги*] (1). Если подразумевается работа с данными всех разделов приложения, то необходимо перейти в раздел [*Итоги*].



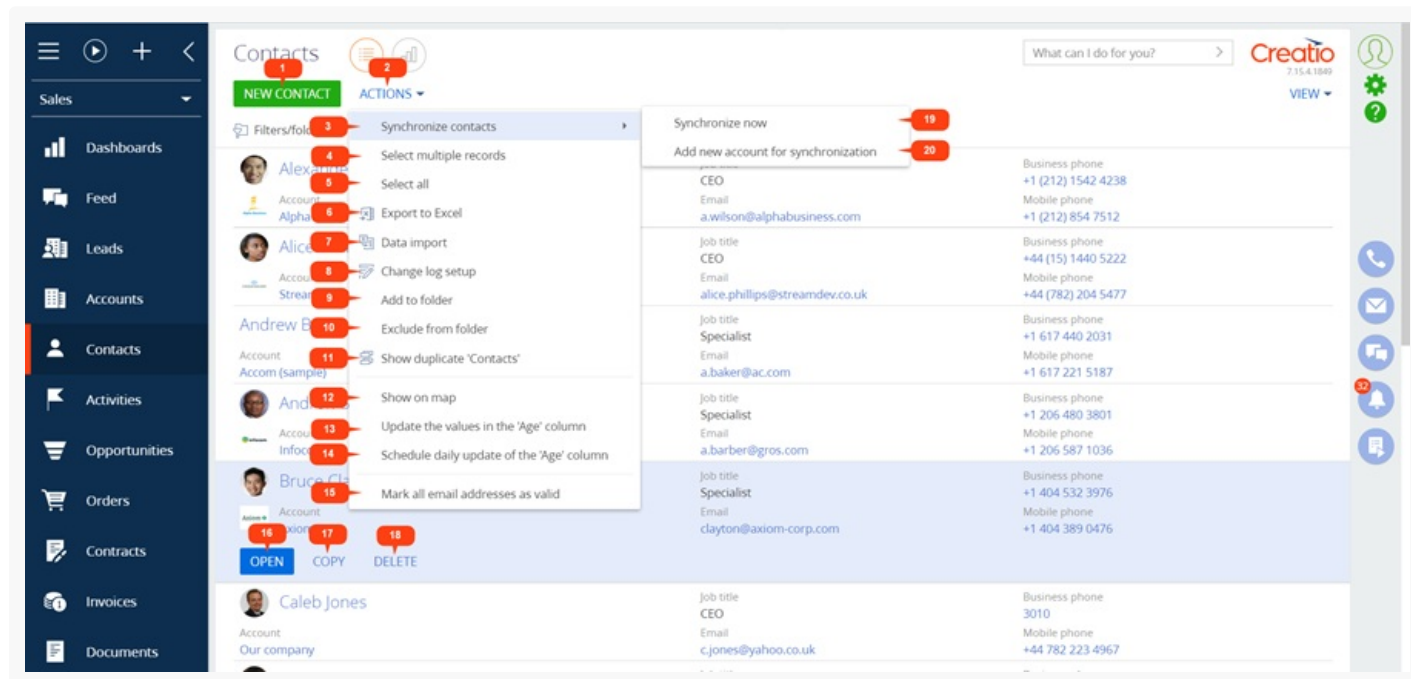
Информация элементов аналитики отображается в специальных блоках, которые называются блоками итогов. Область раздела, на которой размещены блоки итогов, называется [панелью итогов](#).

В приложении Creatio используются следующие блоки итогов:

- График (2) — в наглядной форме отображает множественные данные из системы. Например, с помощью графика можно отобразить распределение активностей по пользователям системы. График может отображать информацию в виде диаграмм разных типов, либо в виде реестра данных. Подробнее о настройке графиков можно узнать из [статьи](#).
- Показатель (3) — отображает информацию, полученную в результате запроса к данным системы, например, общее количество активностей. Подробнее настройки показателей описываются в [статье](#).
- Шкала (4) — отображает информацию, полученную в результате запроса к данным системы относительно нормативной шкалы. Основные сведения и настройка индикаторов изложены в [статье](#).
- Список (5) — отображает информацию из системы в виде списка. Подробности настройки списка изложены в [статье](#).
- Web-страница (6) — предназначен для отображения веб-страниц. Это может быть, например, страница поисковой системы, онлайн-калькулятор валют или корпоративный сайт. Настройка блока итогов веб-страниц описана в [статье](#).
- Воронка продаж (7) — используется для анализа динамики продвижения продаж по стадиям. Подробности о настройке воронки продаж описаны в [статье](#).
- Виджет — отображает дополнительные пользовательские виджеты, например, виджет курсов валют, виджет погоды. Подробнее о пользовательских виджетах можно узнать из [статьи](#).

Действия раздела

Действиями раздела называют функциональные элементы, которые представляют собой набор операций с одной или несколькими записями реестра. Действия могут быть вызваны при помощи кнопок разного вида, размещенных как в контейнере действий текущего раздела, так и в контейнере активной записи.



Действия раздела доступны:

- по кнопкам непосредственного вызова действий (1), (16)...(18).
- по кнопкам с выпадающим меню (2), (3).
- из пунктов меню вызова действий (4)...(15), (19), (20).

Различают стандартные и дополнительные действия разделов.

К **стандартным** относят следующие действия раздела:

- [*Добавить контакт*] ([*New contact*]) (1) — вызывает всплывающее окно для добавления и сохранения новой записи раздела.
- [*Открыть*] ([*Open*]) (16) — открывает страницу выбранной записи раздела.
- [*Копировать*] ([*Copy*]) (17) — открывает страницу записи раздела, копирует в нее данные выбранной записи и при сохранении создает новую запись.
- [*Удалить*] ([*Delete*]) (18) — удаляет активную запись.
- [*Выбрать несколько записей*] ([*Select multiple records*]) (4) — выполнение множественного выбора записей реестра.
- [*Выбрать все*] ([*Select all*]) (5) — выполнение выбора всех записей реестра.
- [*Экспорт в Excel*] ([*Export to Excel*]) (6) — экспортирует все записи реестра текущего раздела в файл в формате `*.xlsx`.

- [*Импорт данных*] ([*Data import*]) (7) — импортирует в Creatio данные из файла в формате *.xlsx. Дополнительная информация о процессе импорта данных содержится в [статье](#).
- [*Настроить журнал изменений*] ([*Change log setup*]) (8) — открывает страницу управления логированием и выбора колонок раздела, которые будут логироваться при изменении записи.

Дополнительные действия реализуют функциональность, в зависимости от бизнес-логики раздела. Дополнительными для раздела [*Контакты*] ([*Contacts*]) являются следующие действия:

- [*Запустить синхронизацию*] ([*Synchronize now*]) (19) — [выполнение синхронизации Creatio с контактами Google](#). Синхронизация контактов из Creatio в Google выполняется только для записей, отмеченных личным тегом, указанным в настройках синхронизации.
- [*Добавить аккаунт для синхронизации*] ([*Add new account for synchronization*]) (20) — [выполнение синхронизации с Google для Creatio Cloud](#).
- [*Поместить в группу*] ([*Add to folder*]) (9) — вызывает всплывающее окно для выбора группы, в которую необходимо поместить выбранную запись.
- [*Исключить из группы*] ([*Exclude from folder*]) (10) — исключить активную запись из всех групп, в которую она входит.
- [*Перейти к дублиям раздела 'Контакты'*] ([*Show duplicate 'Contacts'*]) (11) — открывает дополнительную страницу, которая содержит все предполагаемые дубли контактов. Записи добавляются на эту страницу автоматически после выполнения поиска дублей.
- [*Показать на карте*] ([*Show on map*]) (12) — позволяет отобразить на карте местонахождение выбранных контактов. По действию открывается окно с картой, на которой отмечены выбранные в реестре контакты. Если для всех выбранных контактов адрес не заполнен, то действие не будет выполнено. Если для некоторых контактов адрес не заполнен или заполнен некорректно, то в окне отобразится соответствующая информация.
- [*Обновить возраст*] ([*Update the values in the 'Age' column*]) (13) — обновление значения в колонке [*Возраст*] ([*Age*]) на странице активной записи контакта.
- [*Настроить время обновления возраста*] ([*Schedule daily update of the 'Age' column*]) (14) — вызывает всплывающее окно для настройки времени ежедневного обновления информации о возрасте контакта.
- [*Снять признак 'Неактуальный' у email-адресов*] ([*Mark all email addresses as valid*]) (15) — устанавливает признак [*Является актуальным*] email-адресу выбранной записи контакта.

В приложении Creatio существует возможность создавать пользовательские действия.

Список действий раздела представляет собой экземпляр класса `Terrasoft.BaseViewModelCollection`. Каждый элемент списка действий – это модель представления.

Настройка действия выполняется в конфигурационном объекте, в котором можно как явно задавать свойства модели представления действий, так и использовать базовый механизм привязки.

Базовое наполнение меню [*Действия*] страницы раздела реализовано в базовом классе раздела `BaseSectionV2`. Список действий раздела возвращает защищенный виртуальный метод `getSectionActions()` из схемы `BaseSectionV2`.

Добавление отдельного действия в коллекцию выполняется вызовом метода `addItem()`. В качестве параметра ему передается callback-метод `getButtonItem()`. Метод создает экземпляр модели представления действия по конфигурационному объекту, который передан ему в качестве параметра.

Базовая реализация добавления действия

```
/**
 * Возвращает коллекцию действий раздела в режиме отображения реестра.
 * @protected
 * @virtual
 * @return {Terrasoft.BaseViewModelCollection} Коллекция действий раздела.
 */
getSectionActions: function() {
    // Коллекция действий. Экземпляр Terrasoft.BaseViewModelCollection.
    var actionMenuItems = this.Ext.create("Terrasoft.BaseViewModelCollection");
    // Добавления действия в коллекцию. В качестве callback- метода передается метод,
    // инстанцирующий экземпляр модели действия пр переданному конфигурационному
    // объекту.
    actionMenuItems.addItem(
        this.getButtonItem({
            // Конфигурационный объект настройки действия.
            ...
        })
    );
    return actionMenuItems;
}
```

Свойства конфигурационного объекта действия раздела, который передается параметром в метод `getButtonItem()` представлены в таблице.

Свойство	Описание
Type	Тип элемента меню [<i>Действия</i>]. С помощью этого свойства можно добавить в меню действия горизонтальную линию для отделения блоков меню. Для этого в качестве значения свойства нужно указать строку <code>Terrasoft.MenuSeparator</code> . Если значение свойства не указано, то по умолчанию будет добавлен пункт меню.
Caption	Заголовок пункта меню [<i>Действия</i>]. Для задания заголовков рекомендуется использовать локализуемые строки схемы.
Click	В этом свойстве выполняется привязка метода-обработчика действия по имени метода.
Enabled	Логическое свойство, регулирующее доступность пункта меню.
Visible	Логическое свойство, которое регулирует видимость пункта меню.

Общий алгоритм добавления пользовательского действия в раздел

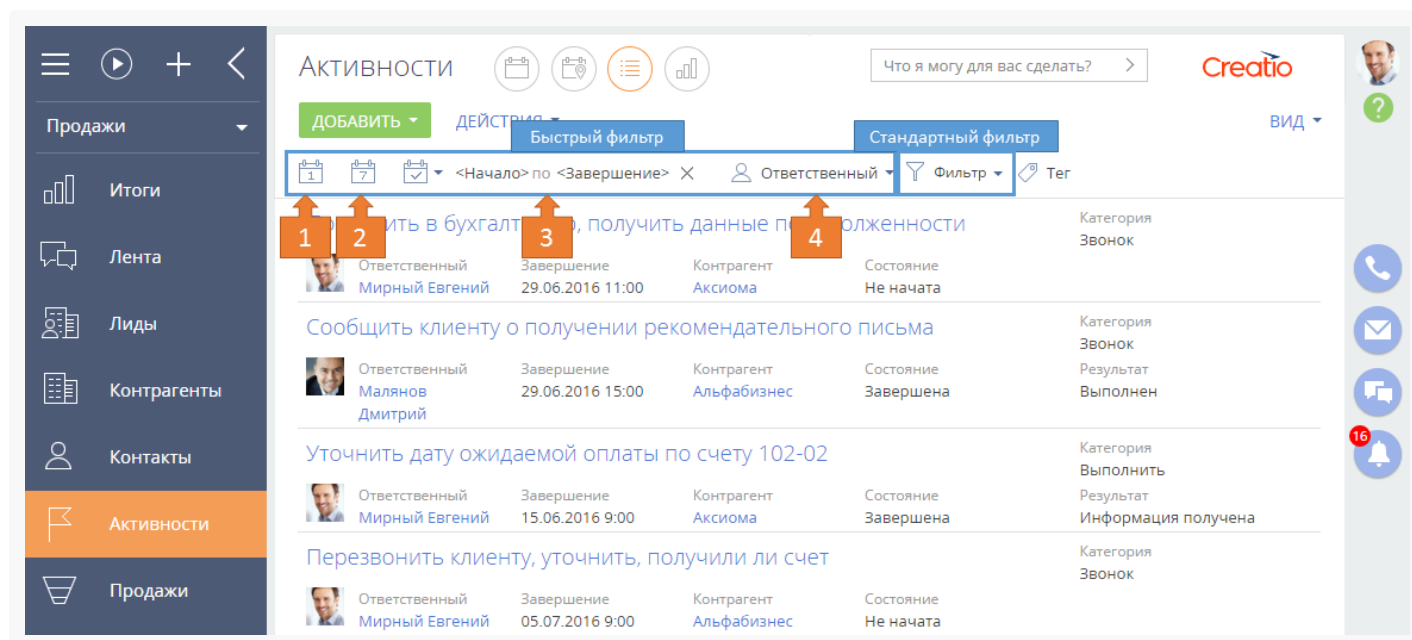
1. Создать замещающую схему существующего раздела или схему нового раздела.
2. Переопределить метод `getSectionActions()`.
3. Добавить действие в коллекцию действий методом `addItem()`.
4. В callback-метод `getButtonItem()` передать конфигурационный объект с настройками добавляемого действия.

При замещении базовых разделов в методе `getSectionActions()` замещающего модуля, сначала нужно вызвать родительскую реализацию этого метода для инициализации действий родительского раздела. Для этого необходимо выполнить метод `this.callParent(arguments)`, который возвратит коллекцию действий базового раздела.

Фильтры

Фильтры предназначены для поиска и фильтрации записей в разделах. В Creatio предусмотрены быстрый, стандартный и расширенный фильтры, а также группы.

Элементы управления фильтрами отображаются над реестром разделов системы. Управление быстрым фильтром осуществляется непосредственно из панели инструментов [*Быстрый фильтр*], а управление стандартным, расширенным фильтрами и группами — из меню [*Фильтр*].



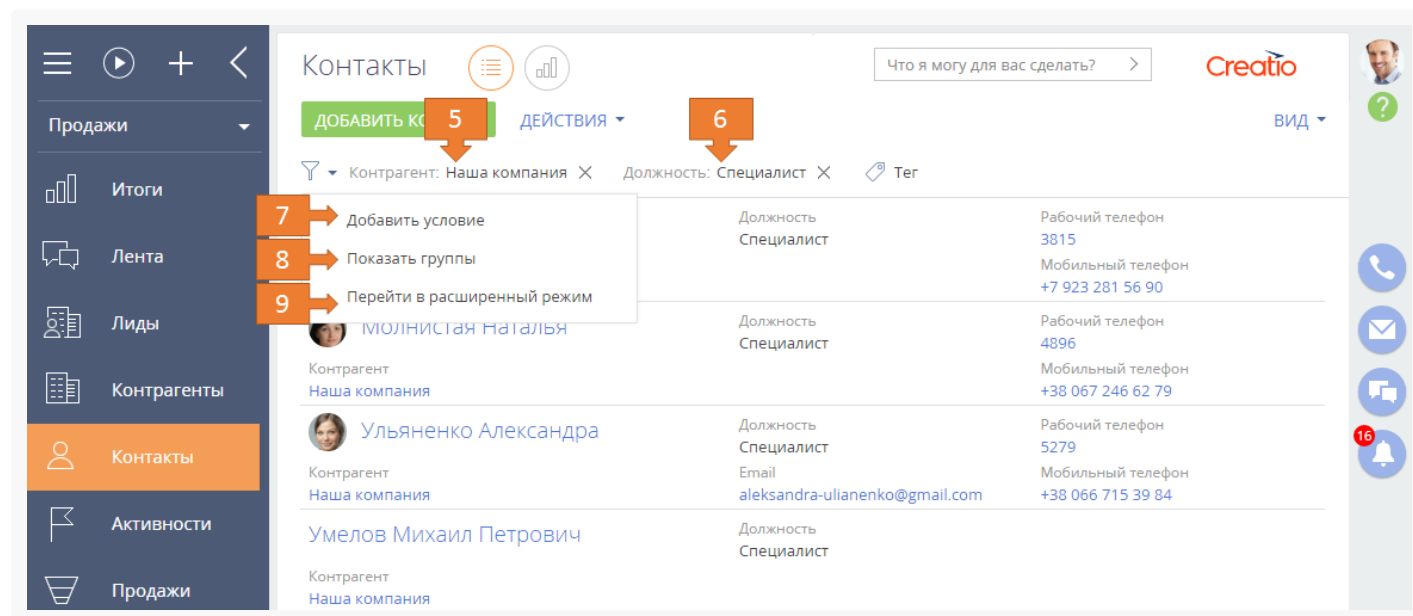
Быстрый фильтр — фильтр, который используется в реестре некоторых разделов Creatio для поиска данных по наиболее часто используемым параметрам (обычно — по периоду и ответственному). Например, быстрый фильтр присутствует в разделе [*Активности*], поскольку чаще всего нужно просматривать активности одного сотрудника за указанный период времени. Для этого предусмотрены следующие быстрые фильтры:

- Сегодня (1) — отображает записи текущего дня.
- Текущая неделя (2) — отображает записи текущей недели.
- Выбрать период (3) — отображает записи стандартного периода, например, "Вчера", "Текущая неделя"

и т.п. Также можно установить произвольный период, указав даты его начала и завершения во встроенном календаре.

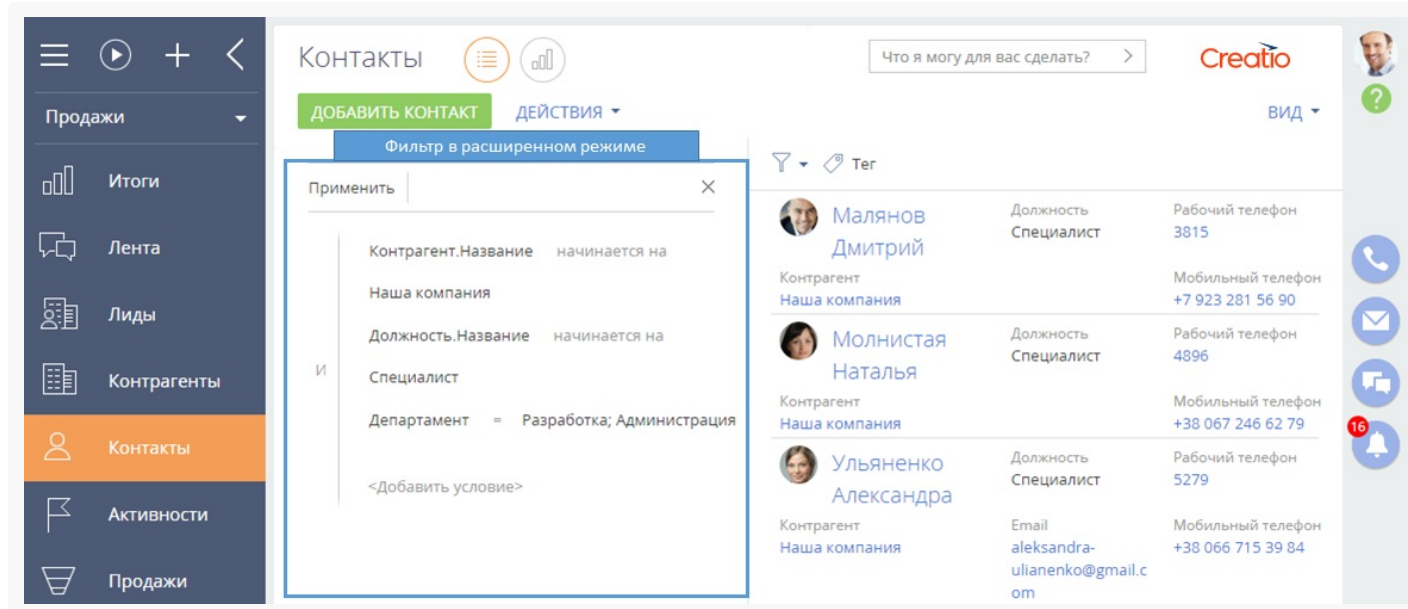
- Выбрать ответственного (4) — используется для отображения активностей одного или нескольких сотрудников.

Стандартный фильтр — фильтр, который используется для быстрого поиска записей по значениям одной или нескольких колонок текущего раздела. Доступен в реестре большинства разделов. Например, если необходимо найти всех специалистов компании в разделе, то нужно установить фильтры полям [*Контрагент*] (5) и [*Должность*] (6).



Установка стандартных фильтров в разделе выполняется при помощи команды [*Добавить условие*] (7) меню [*Фильтр*].

Расширенный фильтр — фильтр, состоящий из нескольких параметров и сложных условий поиска. Например, если необходимо из всех специалистов компании отобразить только тех, кто работает в департаментах "Разработка" и "Администрация".

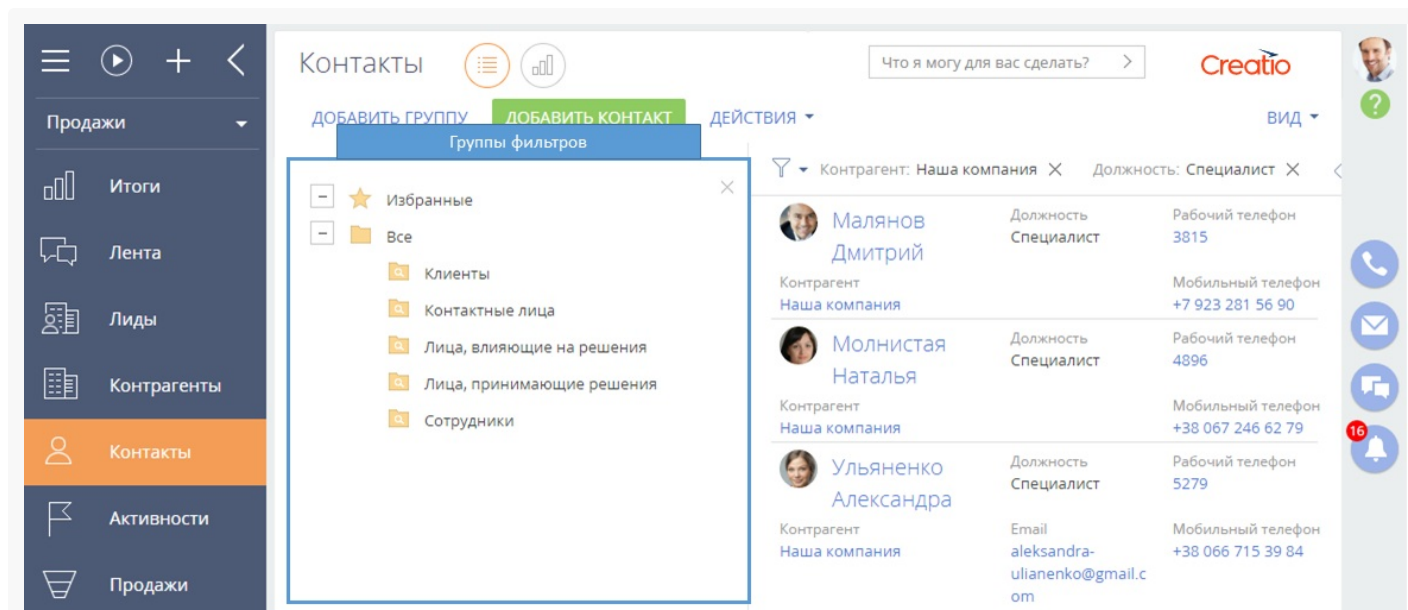


Для установки [расширенного фильтра](#) необходимо использовать команду [*Перейти в расширенный режим*] (8).

Группа — инструмент фильтрации записей путем объединения их по определенным условиям. При выборе группы в разделе будут отображены только те записи, которые отвечают условиям фильтрации группы.

Включение записи в группу, как и исключение из нее, не может быть выполнено вручную. Запись будет автоматически отображена в группе, если она отвечает условиям фильтрации группы. Если запись больше не соответствует условиям фильтра группы, она исключается из группы автоматически.

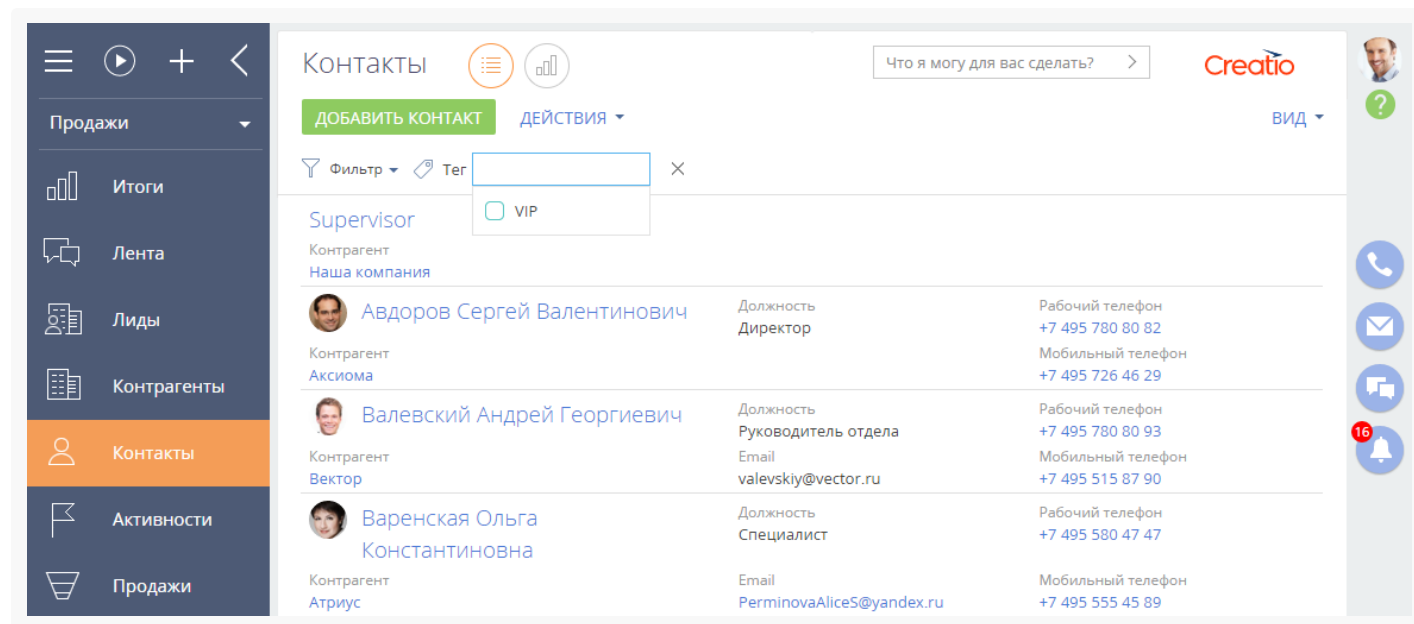
Для отображения в режиме фильтрации данных по группам необходимо выполнить команду [*Показать группы*] (8). При этом отобразятся уже существующие в системе группы для активного раздела. При необходимости можно [создать нужную структуру групп](#) и определить правила их наполнения.



В приложении Creatio существует возможность создавать пользовательские фильтры в разделе.

Теги

Тег — специальная метка, которую можно использовать для сегментации записей вручную. Например, тегировав записи в разделе [*Контакты*], можно выделить VIP-клиентов или определить черный список клиентов.



Тегирование записей выполняется вручную. Для каждого раздела Sales Creatio используется отдельный перечень тегов.

В Sales Creatio используются такие виды тегов:

- **Личные** — теги, которые может видеть и использовать только тот сотрудник, который их создал. Любой пользователь системы может создать необходимое количество личных тегов. Ни администраторы системы, ни руководители не смогут увидеть личные теги сотрудников. Личные теги в Creatio отображаются зеленым цветом.
- **Корпоративные** — теги, которые отображаются для всех сотрудников компании. Любой сотрудник может установить или снять корпоративный тег. Создавать новые корпоративные теги могут все сотрудники/роли, которым предоставлено право на операцию [*Управление корпоративными тегами*]. Корпоративные теги в Creatio отображаются голубым цветом.
- **Публичные** — теги, которые отображаются для всех сотрудников компании, а также для пользователей портала самообслуживания. Любой сотрудник может установить или снять публичный тег. Создавать новые публичные теги могут все сотрудники/роли, которым предоставлено право на операцию [*Управление публичными тегами*]. Публичные теги в Creatio отображаются красным цветом.

Создать новый раздел

Одной из типовых задач разработки в Creatio является добавление нового раздела. Для реализации этой задачи используется [мастер разделов](#). С помощью мастера можно настроить основные свойства раздела и его страниц, бизнес-правила и кейсы.

Результатом выполненных настроек будут схемы объектов и страниц раздела, добавленные в текущий

пользовательский пакет.

Схемы объектов, создаваемые мастером разделов

Правила именования	Назначение	Родитель
[Имя объекта раздела]	Основной объект раздела	Базовый объект (BaseEntity)
[Имя объекта раздела]Folder	Группа объекта. Служебный объект для корректной группировки записей в разделе. Формирует общую структуру дерева групп раздела.	Базовая группа (BaseFolder)
[Имя объекта раздела]InFolder	Объект в группе. Служебный объект для корректной работы группировки записей в разделе. Определяет связи между записями раздела и группами, в которые они входят.	Базовый элемент в группе (BaseItemInFolder)
[Имя объекта раздела]File	Объект для детали [<i>Файлы и ссылки</i>].	Файл (File)
[Имя объекта раздела]Tag	Тег раздела.	Базовый тег (BaseTag)
[Имя объекта раздела]InTag	Тег в объекте раздела.	Базовый тег в базовом объекте (BaseEntityInTag)

Клиентские схемы, создаваемые мастером раздела

Правила именования	Назначение	Родитель
[Имя объекта раздела]Section	Схема раздела	Базовая схема раздела (BaseSectionV2)
[Имя объекта раздела]Page	Схема страницы записи раздела	Базовая схема страницы записи раздела (BaseModulePageV2)

Мастер разделов и пакет [*Custom*]

В процессе работы мастер разделов или мастер деталей не только создает различные схемы, но и привязывает данные к текущему пакету. Для пакета [*Custom*] не предусмотрено использование стандартного механизма импорта пакетов. Поэтому если текущим пакетом является пакет [*Custom*], то

перенести привязанные данные в другой пользовательский пакет можно только с помощью запросов к базе данных. Мы настоятельно не рекомендуем использовать этот способ, поскольку изменения могут повлиять на структуру базы данных, что приведет к неработоспособности приложения. Пакет [*Custom*] невозможно добавить в систему контроля версий. Поэтому пакет [*Custom*] не рекомендуется использовать в качестве текущего пользовательского пакета.

На заметку. Чтобы [изменить текущий пакет](#), необходимо использовать системную настройку [Текущий пакет] (`CurrentPackageId`). Перед запуском мастера разделов рекомендуется проверить значение этой системной настройки.

Последовательность создания раздела

1. С помощью мастера разделов создайте раздел и добавьте его в нужное рабочее место.
2. Добавьте необходимые колонки в схему объекта раздела и настройте их отображение в реестре, на странице записи и на деталях.

Удалить раздел

На заметку. Если на основании объекта, который требуется удалить, создан раздел, то этот раздел удаляется в первую очередь.

Чтобы удалить пользовательский раздел Creatio, необходимо иметь доступ к конфигурации системы и базе данных.

Важно. Прежде чем удалить пользовательский раздел, необходимо снять блокировку с соответствующего этому разделу файла в хранилище версий SVN.

В первую очередь потребуется удалить записи из базы данных.

Скрипт удаления раздела

```
DECLARE @UIId UNIQUEIDENTIFIER
DECLARE @ModuleEntityUID UNIQUEIDENTIFIER;
DECLARE @ModuleID UNIQUEIDENTIFIER;
DECLARE @Name NVARCHAR(max) = 'ToDelete';
select @UIId = UIId from SysSchema where Name Like @Name
select @ModuleEntityUID = Id from SysModuleEntity where
SysEntitySchemaUID = @UIId
select @ModuleID = Id from SysModule where SysModuleEntityId = @ModuleEntityUID;
delete from SysModuleInWorkplace where SysModuleId = @ModuleID;
delete from SysModule where Id = @ModuleID;
delete from SysModuleEdit where SysModuleEntityId = @ModuleEntityUID;
```



```
delete from SysModuleEntity where Id = @ModuleEntityUID;
delete from SysDetail where EntitySchemaUid = @UID;
delete from SysLookup where SysEntitySchemaUid = @UID;
delete from [Lookup] where SysEntitySchemaUid = @UID;
```

Обратите внимание, что значение "ToDelete" нужно заменить на название схемы пользовательского раздела. После очистки базы данных необходимо удалить пользовательские схемы в разделе [*Управление конфигурацией*] ([*Advanced settings*]) в следующем порядке:

1. ToDeleteFile
2. ToDeleteInFolder
3. ToDeleteInTag
4. ToDeleteTag
5. ToDeleteFolder
6. ToDelete

Создать новый раздел



Пример. В приложение добавить рабочее место Автосалон (Car showroom). В созданное рабочее место добавить пользовательский раздел Грузовые автомобили (Trucks). Схема объекта грузового автомобиля должна содержать обязательные для заполнения колонки:

- Название (Name) — строка.
- Владелец (Owner) — справочник Контакт (Contact).
- Организация (Organization) — справочник Контрагент (Account).
- Цена (Price) — строка.

1. С помощью мастера разделов создать раздел и добавить его в нужное рабочее место

Процесс создания рабочего места подробно описан в статье "[Настройка рабочих мест](#)". Для нового рабочего места укажите [*Название*] ([*Name*]) — "Автосалон" ("Car showroom") и добавьте группу пользователей, для которой будет доступно создаваемое рабочее место.

Работа с мастером разделов подробно описана в статье "[Мастер разделов](#)". Для создания нового раздела достаточно использовать первый шаг мастера. При этом в основной объект раздела будут добавлены колонка [*Название*] ([*Name*]) с типом "строка" и колонки, унаследованные от базового объекта. Для первоначальной настройки раздела на первом шаге мастера укажите:

- [*Код раздела*] ([*Code*]) — "UsrTruck";
- [*Заголовок*] ([*Title*]) — "Грузовые автомобили" ("Trucks").

Важно. Название объекта раздела, вводимое в поле [*Код раздела*] в мастере разделов, не должно содержать префиксов "Base", "Sys" или "Vw". Также оно не должно содержать суффиксов "InFolder", "Lcz", "Lookup" и "Settings". В противном случае для такого объекта будет невозможно настроить импорт данных из Excel.

Если работа в системе ведется с настройками по умолчанию, то при вводе значения в поле [*Код раздела*] возникнет сообщение о том, что значение должно начинаться с префикса "Usr". Значение префикса задается в системной настройке [*Префикс названия объекта*] (SchemaNamePrefix). При необходимости можно сменить значение префикса на пользовательское. Использовать пустую строку в качестве префикса не рекомендуется из-за возможных совпадений названий с другими элементами базовой конфигурации.

В результате в пользовательском пакете будут созданы все необходимые для работы раздела схемы.

Schemas: All ▾ External Assemblies: All ▾ SQL Scripts: All ▾ Data: All ▾ Package Dependencies ▾ Add ▾ Edit Delete ?			
▲ Name ¹	Package	▲ Title ²	Database Update Required
☆ UsrTruck	sdkCreateNewSection	Truck	☒
☆ UsrTruck1Page	sdkCreateNewSection	Card schema: "Truck"	☐
☆ UsrTruck1Section	sdkCreateNewSection	Section schema: "Trucks"	☐
☆ UsrTruckFile	sdkCreateNewSection	Truck attachment	☒
☆ UsrTruckFolder	sdkCreateNewSection	Truck folder	☒
☆ UsrTruckInFolder	sdkCreateNewSection	Truck in Folder	☒
☆ UsrTruckInTag	sdkCreateNewSection	Truck section record tag	☒
☆ UsrTruckTag	sdkCreateNewSection	Truck section tag	☒

2. Добавить необходимые колонки в схему объекта раздела и настроить их отображение

Существует два способа добавления новых колонок в схему объекта:

1. При помощи мастера создать новую колонку и сразу же добавить ее на страницу записи. При этом колонка будет автоматически добавлена в схему основного объекта раздела. Подробно о настройке полей страницы записи раздела читайте в статье ["Как настроить поля страницы"](#).
2. Добавить колонку в схему основного объекта раздела с помощью дизайнера объекта в разделе [*Конфигурация*]. Затем с помощью мастера разделов добавить колонки на страницу. Подробнее о возможностях раздела [*Конфигурация*] можно узнать из статьи [Принципы разработки в Creatio IDE](#).

Поскольку мастер раздела задействован в любом случае, удобнее использовать первый способ.

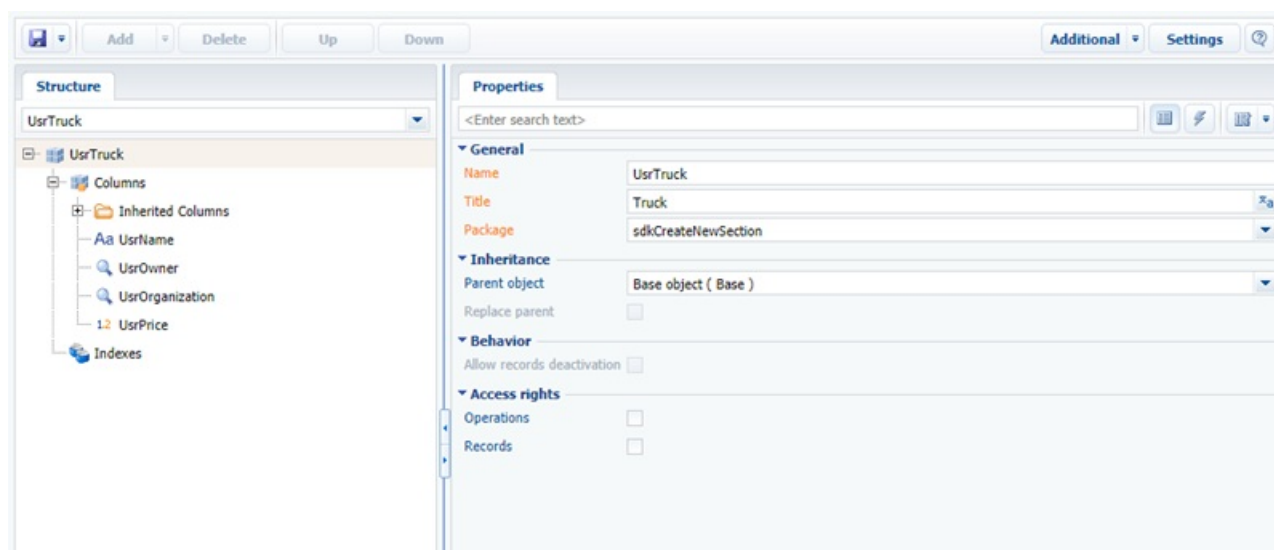
Колонка [*Название*] ([*Name*]) создается и добавляется на страницы раздела мастером раздела автоматически.

Для остальных колонок заполните следующие свойства:

Колонка	Тип	Заголовок [Title]	Название в базе данных [Name in DB]
Владелец	Справочник [Контакт] ([Contact])	Владелец (Owner)	UsrOwner
Организация	Справочник [Контрагент] ([Account])	Организация (Organization)	UsrOrganization
Цена	Дробное число (Decimal)	Цена, USD (Price, USD)	UsrPrice

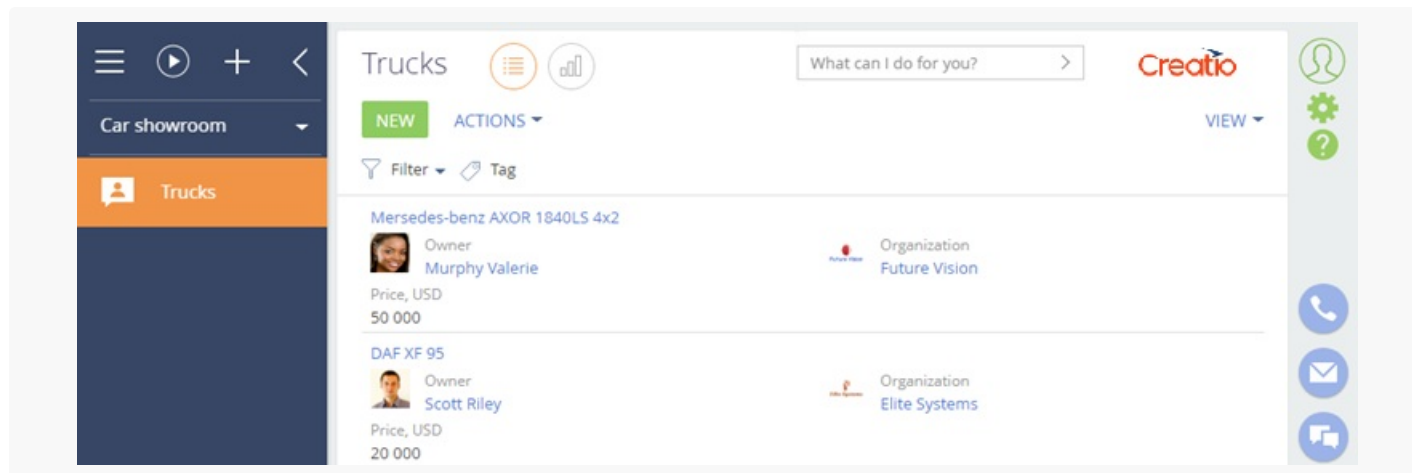
Для всех колонок установите признак [Является обязательным] ([Is required]).

После сохранения изменений в мастере разделов в схему объекта `UsrTruck` будут добавлены новые колонки, а в массив модификаций `diff` схемы страницы записи `UsrTruckPage` — соответствующие им конфигурационные объекты.



Настройте отображение колонок в реестре раздела. Страница настройки колонок реестра открывается по команде [Настроить колонки] ([Select fields to display]) меню кнопки [Вид] ([View]). Подробнее о настройке колонок раздела можно узнать из статьи "[Настройка колонок реестра](#)".

Результат выполнения примера



Добавить действие раздела для единичной записи

 Средний

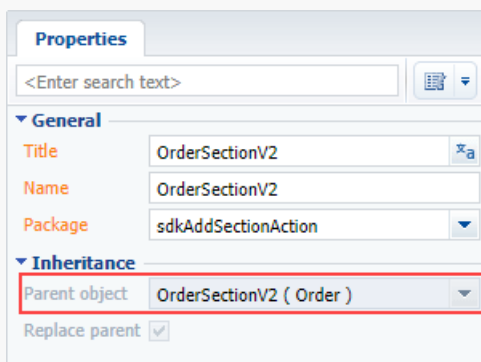
Пример. Для реестра раздела Заказы реализовать действие, которое в информационном окне будет отображать дату создания заказа. Действие должно быть доступно только для заказов, находящихся на стадии Исполнение.

Важно. Пример актуален для кастомизации линейки продуктов Sales Creatio.

Обращение к выделенной записи выполняется через атрибут модели представления раздела `ActiveRow`, который возвращает значение первичной колонки выделенной записи. Далее это значение может использоваться для получения значений, загруженных в реестр полей выбранного объекта, например, из коллекции данных списочного реестра, которая хранится в свойстве `GridData` модели представления реестра.

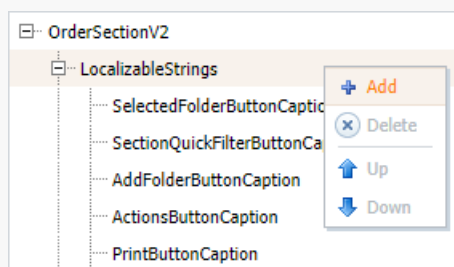
1. В пользовательском пакете создать схему замещающей модели представления раздела [Заказы]

Создайте схему замещающей модели представления, в которой в качестве родительского объекта укажите схему `OrderSectionV2`. Процесс создания схемы замещающей модели представления описан в статье ["Создать клиентскую схему"](#).



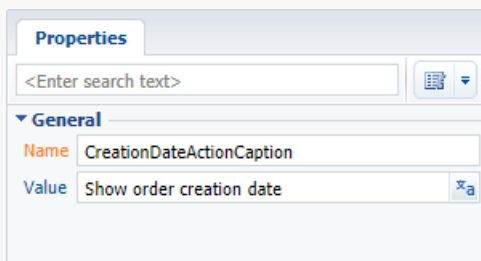
2. В коллекцию локализуемых строк схемы замещающей модели представления раздела добавить строку с заголовком пункта меню [Действия]

Создайте новую локализуемую строку.



Для созданной строки укажите:

- [*Название*] ([*Name*]) — "CreationDateActionCaption";
- [*Значение*] ([*Value*]) — "Показать дату создания заказа" ("Show order creation date").



3. В коллекцию методов схемы замещающей модели представления раздела добавить реализацию методов

- `isRunning()` — проверяет, находится ли выбранный в реестре заказ на стадии [*Исполнение*].
- `isCustomActionEnabled()` — определяет доступность добавленного пункта меню.

- `showOrderInfo()` — метод-обработчик действия, который отображает в информационном окне планируемую дату завершения выбранного заказа.
- `getSectionActions()` — переопределенный метод родительской схемы, возвращающий коллекцию действий раздела.

Исходный код схемы замещающей модели представления:

OrderSectionV2.js

```
define("OrderSectionV2", ["OrderConfigurationConstants"],
    function(OrderConfigurationConstants) {
        return {
            // Название схемы объекта раздела.
            entitySchemaName: "Order",
            // Методы модели представления раздела.
            methods: {
                // Проверяет, на какой стадии находится заданный заказ.
                // activeRowId — значение первичной колонки выделенной записи реестра.
                isRunning: function(activeRowId) {
                    // Получение коллекции данных списочного представления реестра раздела.
                    var gridData = this.get("GridData");
                    // Получение модели выбранного заказа по заданному значению первичной колонки
                    var selectedOrder = gridData.get(activeRowId);
                    // Получение свойства модели — статуса выбранного заказа.
                    var selectedOrderStatus = selectedOrder.get("Status");
                    // Метод возвращает true, если статус заказа [Исполнение]. Иначе возвращает
                    return selectedOrderStatus.value === OrderConfigurationConstants.Order.Order
                },
                // Определяет, будет ли доступен пункт меню.
                isCustomActionEnabled: function() {
                    // Попытка получения идентификатора активной (выбранной в реестре) записи.
                    var activeRowId = this.get("ActiveRow");
                    // Если идентификатор определен и статус выбранного заказа
                    // [Исполнение], то возвращается true, иначе — false.
                    return activeRowId ? this.isRunning(activeRowId) : false;
                },
                // Метод-обработчик действия. Отображает в информационном окне дату создания за
                showOrderInfo: function() {
                    var activeRowId = this.get("ActiveRow");
                    var gridData = this.get("GridData");
                    // Получение даты создания заказа. Колонка должна быть добавлена в реестр.
                    var dueDate = gridData.get(activeRowId).get("Date");
                    // Отображение информационного окна.
                    this.showInformationDialog(dueDate);
                },
                // Переопределение базового виртуального метода, возвращающего коллекцию действи
                getSectionActions: function() {
                    // Вызывается родительская реализация метода для получения коллекции
```

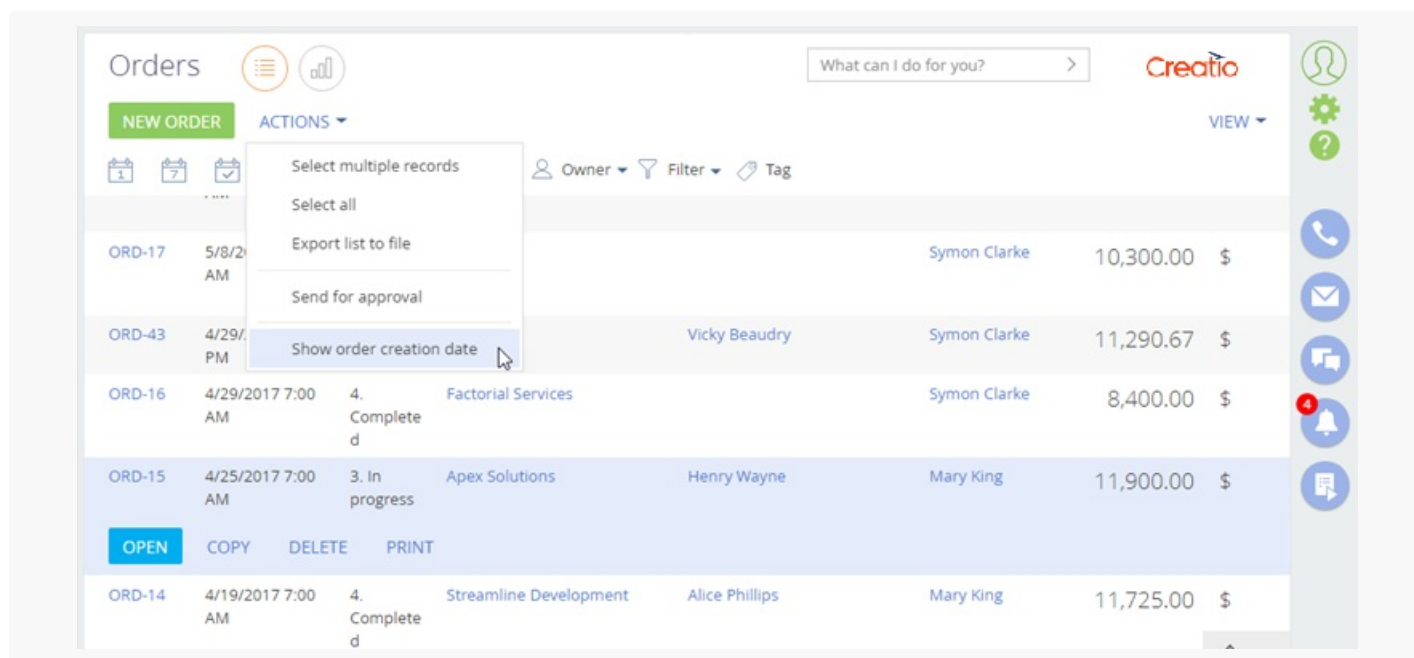
```

// проинициализированных действий раздела.
var actionMenuItems = this.callParent(arguments);
// Добавление линии-разделителя.
actionMenuItems.addItem(this.getButtonItem({
    Type: "Terrasoft.MenuSeparator",
    Caption: ""
}));
// Добавление пункта меню в список действий раздела.
actionMenuItems.addItem(this.getButtonItem({
    // Привязка заголовка пункта меню к локализуемой строке схемы.
    "Caption": {bindTo: "Resources.Strings.CreationDateActionCaption"},
    // Привязка метода-обработчика действия.
    "Click": {bindTo: "showOrderInfo"},
    // Привязка свойства доступности пункта меню к значению, которое возвра
    "Enabled": {bindTo: "isCustomActionEnabled"}
}));
// Возврат дополненной коллекции действий раздела.
return actionMenuItems;
}
}
};
});

```

Результат выполнения примера

После сохранения схемы и обновления страницы приложения с очисткой кэша в разделе [Заказы] ([Orders]) появится новое действие, которое будет активно при выборе заказа, находящегося на стадии [Исполнение] ([In progress]).



Добавить действие раздела для нескольких записей

 Средний

По умолчанию для реестра используется режим выбора одной записи. Для выбора в реестре нескольких активных записей в меню кнопки [Действия] нажмите [*Выбрать несколько записей*]. После этого изменится визуальное представление реестра — появятся элементы для выбора записей. Для отмены режима выбора нескольких записей в меню кнопки [Действия] нажмите [*Отменить множественный выбор*].

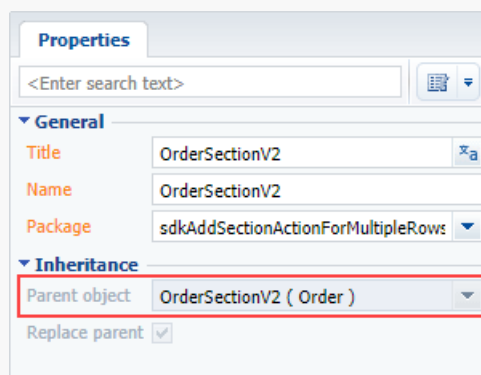
Пример. Для реестра раздела Заказы реализовать действие, которое в информационном окне будет отображать названия контрагентов у нескольких выбранных в реестре заказов.

Важно. Пример актуален для кастомизации линейки продуктов Sales Creatio.

Значение первичных колонок выбранных записей хранится в свойстве `SelectedRows` модели представления раздела. Эти значения могут быть использованы для получения значений, загруженных в реестр полей выбранных объектов, например, из коллекции данных списочного реестра, которая хранится в свойстве `GridData` модели представления реестра.

1. В пользовательском пакете создать схему замещающей модели представления раздела [Заказы]

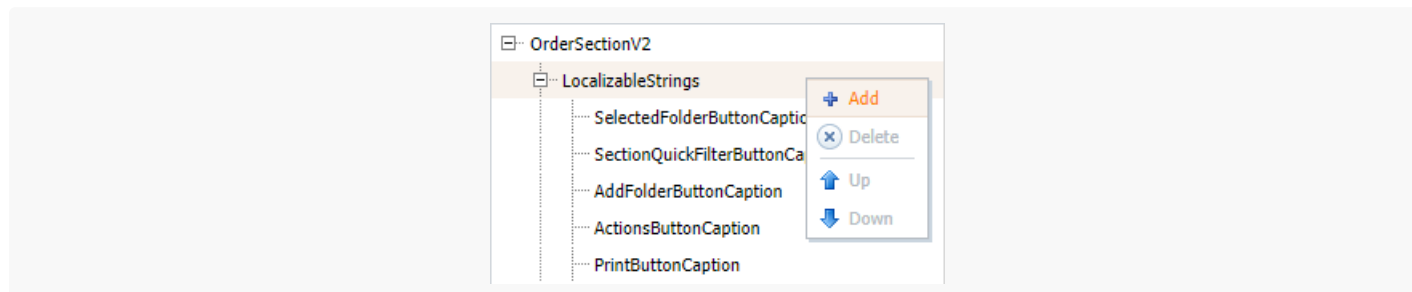
Создайте схему замещающей модели представления раздела, в которой в качестве родительского объекта укажите схему `OrderSectionV2`. Процесс создания схему замещающей модели представления описан в статье ["Создать клиентскую схему"](#).



2. В коллекцию локализуемых строк схему замещающей модели представления раздела добавить строку с

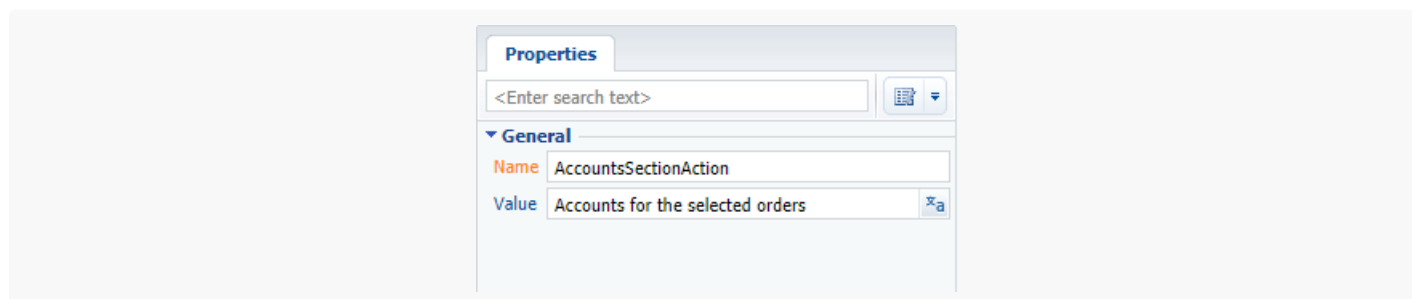
заголовком пункта меню [Действия]

Создайте новую локализуемую строку.



Для созданной строки укажите:

- [*Название*] ([*Name*]) — "AccountsSectionAction";
- [*Значение*] ([*Value*]) — "Отобразить контрагентов по выбранным заказам" ("Accounts for the selected orders").



3. В коллекцию методов схемы замещающей модели представления раздела добавить реализацию методов

- `isCustomActionEnabled()` — определяет доступность добавленного пункта меню.
- `showOrdersInfo()` — метод-обработчик действия, который отображает в информационном окне список контрагентов выбранных заказов.
- `getSectionActions()` — переопределенный метод родительской схемы, возвращающий коллекцию действий раздела.

OrderSectionV2.js

```
define("OrderSectionV2", ["OrderConfigurationConstants"],
    function(OrderConfigurationConstants) {
        return {
            // Название схемы раздела.
            entitySchemaName: "Order",
            // Методы модели представления раздела.
            methods: {
```

```

// Определяет, будет ли доступен пункт меню.
isCustomActionEnabled: function() {
    // Попытка получить массив идентификаторов выбранных записей.
    var selectedRows = this.get("SelectedRows");
    // Если массив содержит элементы (выбрана хотя бы одна запись в реестре),
    // то возвращает true, иначе – false.
    return selectedRows ? (selectedRows.length > 0) : false;
},
// Метод-обработчик действия. Отображает в информационном окне список контрагентов.
showOrdersInfo: function() {
    // Получение массива идентификаторов выбранных записей.
    var selectedRows = this.get("SelectedRows");
    // Получение коллекции данных записей реестра.
    var gridData = this.get("GridData");
    // Переменная для хранения модели объекта выбранного заказа.
    var selectedOrder = null;
    // Переменная для хранения названия контрагента выбранного заказа.
    var selectedOrderAccount = "";
    // Переменная для формирования текста информационного окна.
    var infoText = "";
    // Обработка массива идентификаторов выбранных записей реестра.
    selectedRows.forEach(function(selectedRowId) {
        // Получение модели объекта выбранного заказа.
        selectedOrder = gridData.get(selectedRowId);
        // Получение названия контрагента выбранного заказа. Колонка должна быть до
        selectedOrderAccount = selectedOrder.get("Account").displayValue;
        // Добавление названия контрагента в текст информационного окна.
        infoText += "\n" + selectedOrderAccount;
    });
    // Отображение информационного окна.
    this.showInformationDialog(infoText);
},
// Переопределение базового виртуального метода, возвращающего коллекцию действий ра
getSectionActions: function() {
    // Вызывается родительская реализация метода,
    // возвращающая коллекцию проинициализированных действий раздела.
    var actionMenuItems = this.callParent(arguments);
    // Добавление линии-разделителя.
    actionMenuItems.addItem(this.getButtonItem({
        Type: "Terrasoft.MenuSeparator",
        Caption: ""
    }));
    // Добавление пункта меню в список действий раздела.
    actionMenuItems.addItem(this.getButtonItem({
        // Привязка заголовка пункта меню к локализуемой строке схемы.
        "Caption": {bindTo: "Resources.Strings.AccountsSectionAction"},
        // Привязка метода-обработчика действия.
        "Click": {bindTo: "showOrdersInfo"},
        // Привязка свойства доступности пункта меню к значению,

```

```

        // которое возвращает метод isCustomActionEnabled.
        "Enabled": {bindTo: "isCustomActionEnabled"},
        // Поддержка режима множественного выбора.
        "IsEnabledForSelectedAll": true
    }));
    // Возврат дополненной коллекции действий раздела.
    return actionMenuItems;
}
}
};
});

```

Результат выполнения примера

После сохранения схемы и обновления страницы приложения с очисткой кэша в разделе [Заказы] ([Orders]) появится новое действие, которое будет активно при выборе заказов в режиме выбора нескольких записей реестра.

Account	Contact	Owner	Total	C...			
IT-box	Ray Crowden	Mary King	11,927.16	\$			
Console Solutions	Hillam Jazlyn	John Best	8,909.93	\$			
IT-Plus	Samuel Melendrez	John Best	5,389.81	\$			
Axiom		John Best	5,600.00	\$			
ORD-19	5/9/2017 7:00 AM	4. Completed	Alpha Business	John Best	22,950.00	\$	
ORD-18	5/8/2017 7:00 AM	4. Completed	Durable Industries	Virginia L. Quinn	Symon Clarke	340.00	\$
ORD-17	5/8/2017 7:00 AM	4. Completed	Novelty	Symon Clarke	10,300.00	\$	
ORD-43	4/29/2017 2:00 PM	1. Draft	Nuclon	Vicky Beaudry	Symon Clarke	11,290.67	\$

Обработка выбора нескольких записей. Примеры

Средний

Пример 1

Пример. Для реестра раздела Активности реализовать действие, которое установит состояние

Выполнено для нескольких выбранных в реестре активностей.

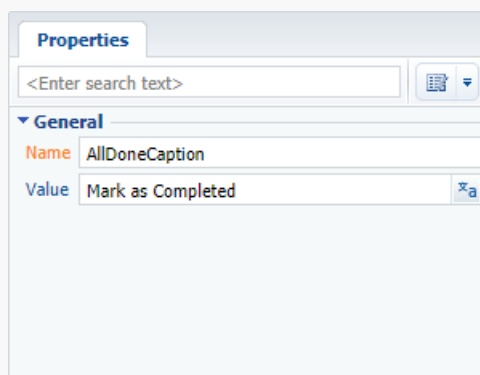
1. В пользовательском пакете создать схему замещающей модели представления раздела [*Активности*]

Процесс создания схему замещающей модели представления детально описан в статье "[Создать клиентскую схему](#)".

2. В коллекцию локализуемых строк схему замещающей модели представления раздела добавить строку с заголовком пункта меню [*Действия*]

Для созданной строки укажите:

- [*Название*] ([*Name*]) — "AllDoneCaption";
- [*Значение*] ([*Value*]) — "Отметить как 'Завершены'" ("Mark as Completed").



3. В коллекцию методов схему замещающей модели представления раздела добавить реализацию методов

- `isCustomActionEnabled()` — метод, который определяет доступность добавленного пункта меню.
- `setAllDone()` — метод-обработчик действия, который устанавливает состояние [*Выполнено*] для нескольких выбранных в реестре активностей.
- `getSectionActions()` — переопределенный метод родительской схемы, возвращающий коллекцию действий раздела.

Исходный код замещающей схемы:

ActivitySectionV2.js

```
define("ActivitySectionV2", ["ConfigurationConstants"],
    function(ConfigurationConstants) {
        return {
            // Название схемы раздела.
```

```

entitySchemaName: "Activity",
// Методы модели представления раздела.
methods: {
    // Определяет, будет ли доступен пункт меню.
    isCustomActionEnabled: function() {
        // Попытка получить массив идентификаторов выбранных записей.
        var selectedRows = this.get("SelectedRows");
        // Если массив содержит элементы (выбрана хотя бы одна запись в реестре),
        // то возвращается true, иначе – false.
        return selectedRows ? (selectedRows.length > 0) : false;
    },
    // Метод-обработчик действия. Устанавливает для выбранных записей статус [Выполнено]
    setAllDone: function() {
        // Получение массива идентификаторов выбранных записей.
        var selectedRows = this.get("SelectedRows");
        // Обработка запускается в случае, если выбрана хотя бы одна запись.
        if (selectedRows.length > 0) {
            // Создание экземпляра класса пакетных запросов.
            var batchQuery = this.Ext.create("Terrasoft.BatchQuery");
            // Обновление каждой из выбранных записей.
            selectedRows.forEach(function(selectedRowId) {
                // Создание экземпляра класса UpdateQuery с корневой схемой Activity
                var update = this.Ext.create("Terrasoft.UpdateQuery", {
                    rootSchemaName: "Activity"
                });
                // Применение фильтра для определения записи для обновления.
                update.enablePrimaryColumnFilter(selectedRowId);
                // Для колонки Status устанавливается значение "Выполнено" с помощью
                // конфигурационной константы ConfigurationConstants.Activity.Status
                update.setParameterValue("Status", ConfigurationConstants.Activity.Status
                    this.Terrasoft.DataValueType.GUID);
                // Добавление запроса на обновление записи в пакетный запрос.
                batchQuery.add(update);
            }, this);
            // Выполнение пакетного запроса к серверу.
            batchQuery.execute(function() {
                // Обновление реестра.
                this.reloadGridData();
            }, this);
        }
    },
    // Переопределение базового виртуального метода, возвращающего коллекцию действий
    getSectionActions: function() {
        // Вызывается родительская реализация метода,
        // возвращающая коллекцию проинициализированных действий раздела.
        var actionMenuItems = this.callParent(arguments);
        // Добавление линии-разделителя.
        actionMenuItems.addItem(this.getButtonMenuItem({
            Type: "Terrasoft.MenuSeparator",

```

```

        Caption: ""
    }));
    // Добавление пункта меню в список действий раздела.
    actionMenuItems.addItem(this.getButtonItem({
        // Привязка заголовка пункта меню к локализуемой строке схемы.
        "Caption": { bindTo: "Resources.Strings.AllDoneCaption" },
        // Привязка метода-обработчика действия.
        "Click": { bindTo: "setAllDone" },
        // Привязка свойства доступности пункта меню к значению, которое возвращает
        "Enabled": { bindTo: "isCustomActionEnabled" },
        // Поддержка режима множественного выбора.
        "IsEnabledForSelectedAll": true
    }));
    // Возврат дополненной коллекции действий раздела.
    return actionMenuItems;
}

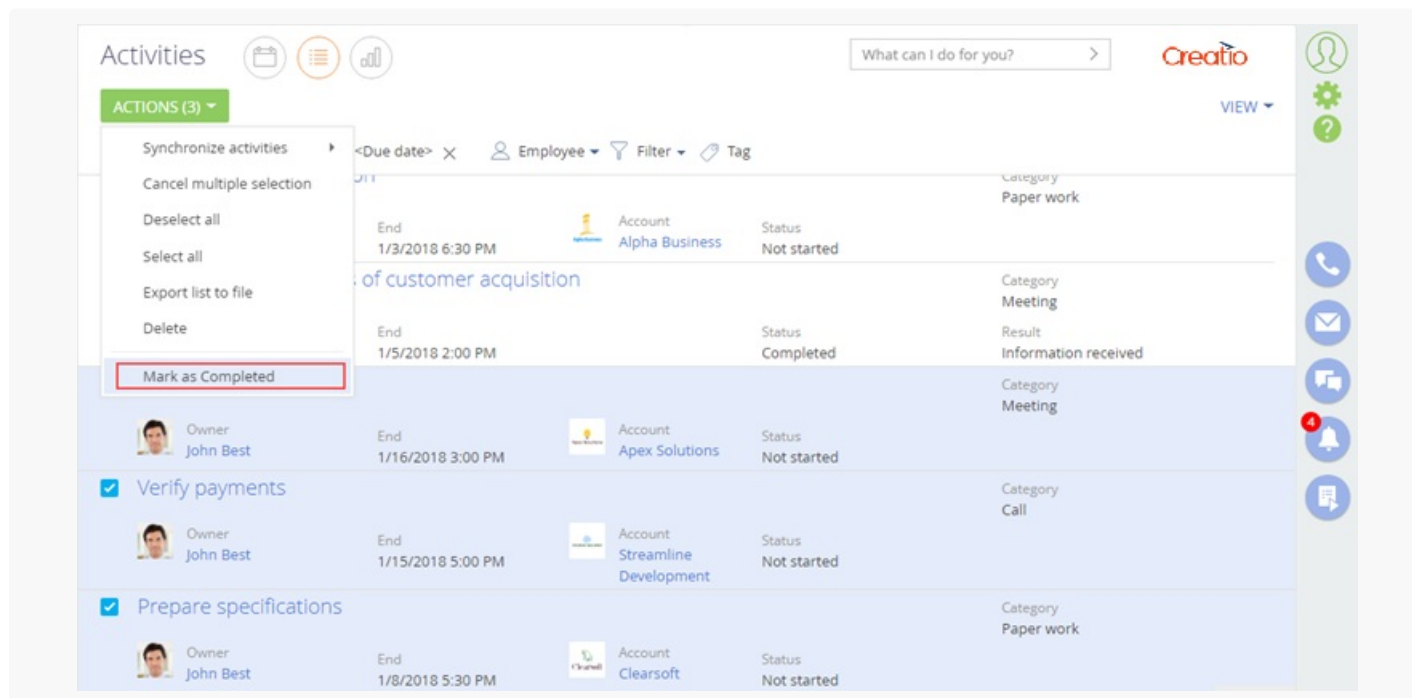
}

});
});

```

Результат выполнения примера

После сохранения схемы и обновления страницы приложения с очисткой кэша в разделе [*Активности*] ([*Activities*]) с помощью нового действия [*Отметить как "Завершены"*] ([*Mark as Completed*]) можно будет применить статус [*Завершено*] ([*Completed*]) сразу к нескольким выбранным активностям.



Пример 2

Пример. Для реестра раздела Активности реализовать действие, которое вызовет окно выбора ответственного и установит выбранное значение для нескольких выбранных в реестре активностей.

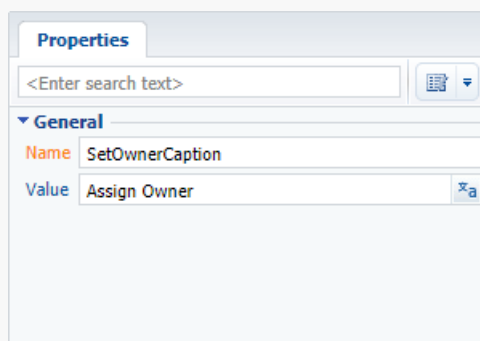
1. В пользовательском пакете создать схему замещающей модели представления раздела [*Активности*]

Процесс создания схему замещающей модели представления детально описан в статье "[Создать клиентскую схему](#)".

2. В коллекцию локализуемых строк схему замещающей модели представления раздела добавить строку с заголовком пункта меню [*Действия*]

Для созданной строки укажите:

- [*Название*] ([*Name*]) — "SetOwnerCaption";
- [*Значение*] ([*Value*]) — "Назначить ответственного" ("Assign Owner").



3. В коллекцию методов модели представления раздела добавить реализацию методов

- `isCustomActionEnabled()` — метод, который определяет доступность добавленного пункта меню.
- `setOwner()` — метод-обработчик действия, который вызывает открытие справочника [*Контакты*].
- `lookupCallback()` — callback-метод, который устанавливает выбранный в справочнике контакт в качестве ответственного для выбранных в реестре записей.
- `getSectionActions()` — переопределенный метод родительской схемы, возвращающий коллекцию действий раздела.

Исходный код замещающей схемы:

ActivitySectionV2.js

```

define("ActivitySectionV2", ["ConfigurationConstants"],
    function(ConfigurationConstants) {
        return {
            // Название схемы раздела.
            entitySchemaName: "Activity",
            // Методы модели представления раздела.
            methods: {
                // Определяет, будет ли доступен пункт меню.
                isCustomActionEnabled: function() {
                    // Попытка получить массив идентификаторов выбранных записей.
                    var selectedRows = this.get("SelectedRows");
                    // Если массив содержит элементы (выбрана хотя бы одна запись в реестре),
                    // то возвращается true, иначе – false.
                    return selectedRows ? (selectedRows.length > 0) : false;
                },
                // Метод-обработчик действия. Вызывает открытие справочника [Контакты].
                setOwner: function() {
                    // Определение конфигурации справочника.
                    var config = {
                        // Схема [Contact].
                        entitySchemaName: "Contact",
                        // Множественный выбор отключен.
                        multiSelect: false,
                        // Отображаемая колонка – [Name].
                        columns: ["Name"]
                    };
                    // Открытие справочника с определенной конфигурацией и callback-функцией, к
                    // после нажатия кнопки [Выбрать].
                    this.openLookup(config, this.lookupCallback, this);
                },
                // Выполняет установку выбранного в справочнике контакта в качестве ответственно
                // для выбранных в реестре записей.
                lookupCallback: function(args) {
                    // Идентификатор выбранной из справочник записи.
                    var activeRowId;
                    // Получение выбранных в справочнике записей.
                    var lookupSelectedRows = args.selectedRows.getItems();
                    if (lookupSelectedRows && lookupSelectedRows.length > 0) {
                        // Получение Id первой выбранной в справочнике записи.
                        activeRowId = lookupSelectedRows[0].Id;
                    }
                    // Получение массива идентификаторов выбранных записей.
                    var selectedRows = this.get("SelectedRows");
                    // Обработка запускается в случае, если выбрана хотя бы одна запись в реестр
                    // в справочнике.
                    if ((selectedRows.length > 0) && activeRowId) {
                        // Создание экземпляра класса пакетных запросов.

```



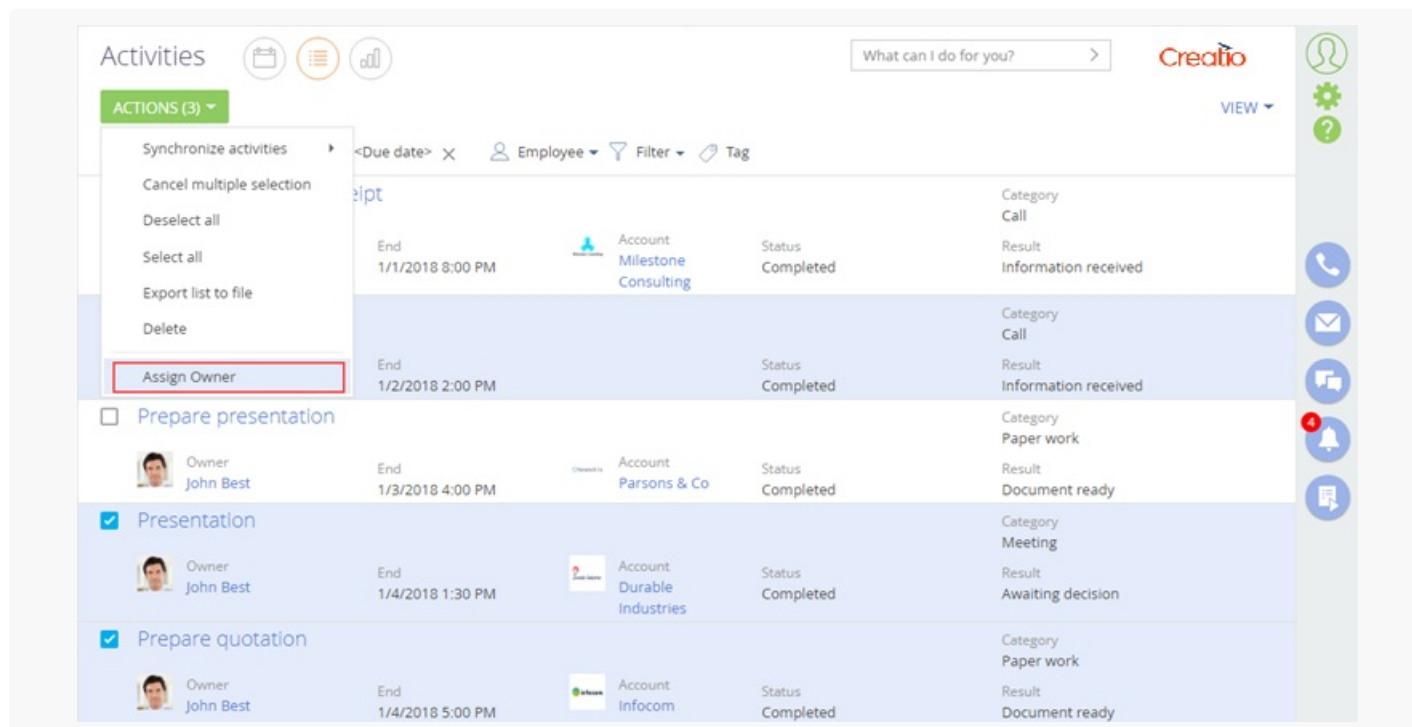
```

var batchQuery = this.Ext.create("Terrasoft.BatchQuery");
// Обновление каждой из выбранных записей.
selectedRows.forEach(function(selectedRowId) {
    // Создание экземпляра класса UpdateQuery с корневой схемой Activity
    var update = this.Ext.create("Terrasoft.UpdateQuery", {
        rootSchemaName: "Activity"
    });
    // Применение фильтра для определения обновляемой записи.
    update.enablePrimaryColumnFilter(selectedRowId);
    // Для колонки [Owner] устанавливается значение, равное идентификатору
    // выбранного из справочника контакта.
    update.setParameterValue("Owner", activeRowId, this.Terrasoft.DataView);
    // Добавление запроса на обновление записи в пакетный запрос.
    batchQuery.add(update);
}, this);
// Выполнение пакетного запроса к серверу.
batchQuery.execute(function() {
    // Обновление реестра.
    this.reloadGridData();
}, this);
}
},
// Переопределение базового виртуального метода, возвращающего коллекцию действий
getSectionActions: function() {
    // Вызывается родительская реализация метода,
    // возвращающая коллекцию проинициализированных действий раздела.
    var actionMenuItems = this.callParent(arguments);
    // Добавление линии-разделителя.
    actionMenuItems.addItem(this.getButtonMenuItem({
        Type: "Terrasoft.MenuSeparator",
        Caption: ""
    }));
    // Добавление пункта меню в список действий раздела.
    actionMenuItems.addItem(this.getButtonMenuItem({
        // Привязка заголовка пункта меню к локализуемой строке схемы.
        "Caption": { bindTo: "Resources.Strings.SetOwnerCaption" },
        // Привязка метода-обработчика действия.
        "Click": { bindTo: "setOwner" },
        // Привязка свойства доступности пункта меню к значению, которое возвращает
        "Enabled": { bindTo: "isCustomActionEnabled" },
        // Поддержка режима множественного выбора.
        "IsEnabledForSelectedAll": true
    }));
    // Возврат дополненной коллекции действий раздела.
    return actionMenuItems;
}
}
};
});

```

Результат выполнения примера

После сохранения схемы и обновления страницы приложения с очисткой кэша в разделе [**Активности**] ([**Activities**]) с помощью нового действия [**Назначить ответственного**] ([**Assign Owner**]) можно будет изменить ответственного сразу у нескольких выбранных активностей.



Настроить внешний вид реестра

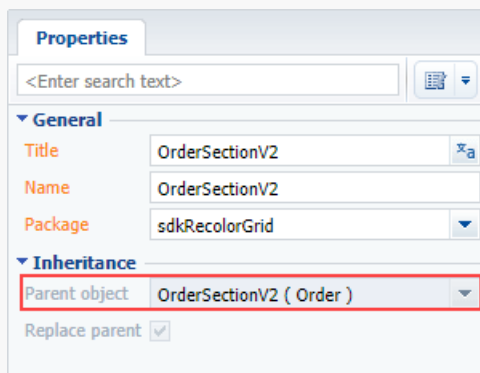
 Средний

Пример. Для раздела Заказы реализовать выделение тех записей реестра, которые находятся на стадии Исполнение.

Важно. Пример актуален для кастомизации линейки продуктов Sales Creatio.

1. В пользовательском пакете создать схему замещающей модели представления раздела [**Заказы**]

Создайте схему замещающей модели представления, в которой в качестве родительского объекта укажите `OrderSectionV2`. Процесс создания схему замещающей модели представления описан в статье "[Создать клиентскую схему](#)".



2. Переопределить метод prepareResponseCollectionItem

В коллекцию методов созданной схемы добавьте метод `prepareResponseCollectionItem()`. Он переопределяет базовый метод, модифицирует строку данных перед загрузкой в реестр, а также добавляет настраиваемые стили к определенным строкам реестра.

OrderSectionV2.js

```
define("OrderSectionV2", ["OrderConfigurationConstants"], function(OrderConfigurationConstants) {
    return {
        // Название схемы раздела.
        entitySchemaName: "Order",
        // Методы модели представления раздела.
        methods: {
            // Переопределение базового метода, который модифицирует строку данных перед загрузкой в реестр.
            prepareResponseCollectionItem: function(item) {
                // Вызов базового метода.
                this.callParent(arguments);
                item.customStyle = null;
                // Определение статуса заказа.
                var running = item.get("Status");
                //Если состояние заказа "Выполнение", меняется стиль записи.
                if (running.value === OrderConfigurationConstants.Order.OrderStatus.Running) {
                    item.customStyle = {
                        // Цвет текста – белый.
                        "color": "white",
                        // Цвет фона – зеленый.
                        "background": "#8ecb60"
                    };
                }
            }
        }
    };
});
```

Результат выполнения примера

После сохранения схемы и обновления страницы приложения с очисткой кэша в разделе [*Заказы*] заказы, находящиеся на стадии [*Исполнение*], будут выделены в реестре зеленым цветом.

Order ID	Date/Time	Status	Company	Owner	Contact	Amount	Currency
ORD-32	4/16/2017 3:00 AM	1. Draft	Alpha Business	Jordan Anderson	John Best	18,840.00	\$
ORD-29	4/13/2017 3:00 AM	4. Completed	Alpha Business		Mary King	4,800.00	\$
ORD-35	4/13/2017 3:00 AM	3. In progress	Infocom	Barber Andrew	Mary King	9,415.00	\$
ORD-30	4/13/2017 3:00 AM	3. In progress	Apex Solutions	Henry Wayne	John Best	1,140.00	\$
ORD-34	4/13/2017 3:00 AM	3. In progress	Build Technologies		Mary King	4,400.00	\$
ORD-42	4/12/2017 2:00 PM	3. In progress	Planet Soft	Scot Grammer	Mary King	7,122.19	\$
ORD-11	4/12/2017 2:00 AM	4. Completed	Alpha Business	Alexander Wilson	John Best	10,000.00	\$
ORD-27	4/11/2017 4:00 AM	2. Confirmation	Alpha Business	Jordan Anderson	John Best	9,000.00	\$

Добавить в раздел блок быстрых фильтров

 Средний

Фильтры предназначены для поиска и фильтрации записей в разделах. В Creadio для этих целей предусмотрены быстрый, стандартный и расширенный фильтры, а также группы. Подробнее о фильтрах можно узнать из статьи "[Фильтры](#)".

Для добавления блока быстрых фильтров в раздел необходимо в схеме замещающей модели представления раздела переопределить метод `initFixedFiltersConfig()`, в котором создать конфигурационный объект `fixedFilterConfig` с такими свойствами:

- `entitySchema` — схема объекта,
- `filters` — массив добавляемых фильтров.

Затем необходимо присвоить ссылку на созданный конфигурационный объект атрибуту `fixedFiltersConfig` схеме замещающей модели представления

```
this.set("FixedFilterConfig", fixedFilterConfig);
```

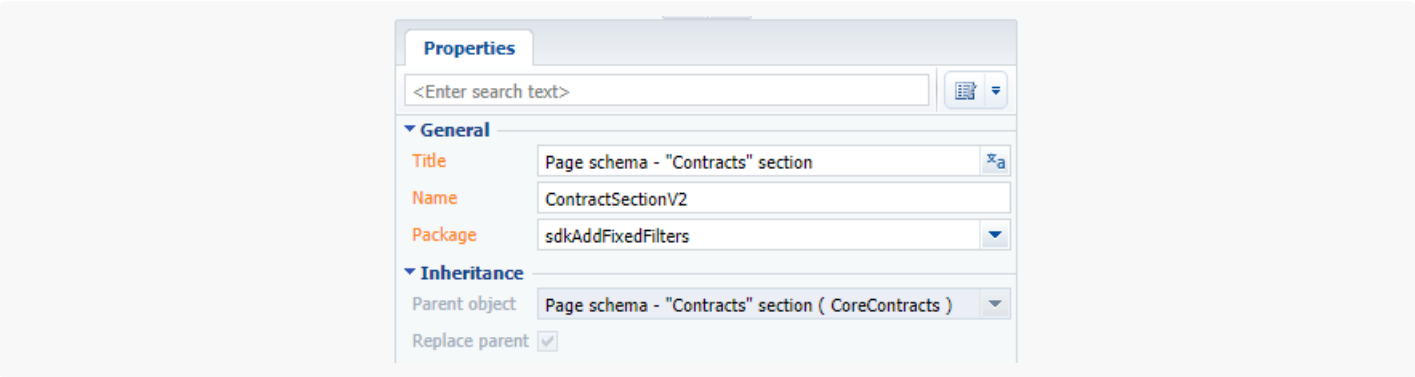
Пример. В раздел Договоры (Contracts) добавить блок быстрых фильтров. Фильтрацию осуществлять по дате начала договора и по ответственному.

1. В пользовательском пакете создать схему замещающей модели представления раздела [Договоры] ([Contracts])

Создайте схему замещающей модели представления, в свойствах которой укажите:

- [Родительский объект] ([Parent object]) — "Схема страницы раздела — "Договоры"" ("Page schema — "Contracts" section");
- [Название] ([Name]) — ContractSectionV2.

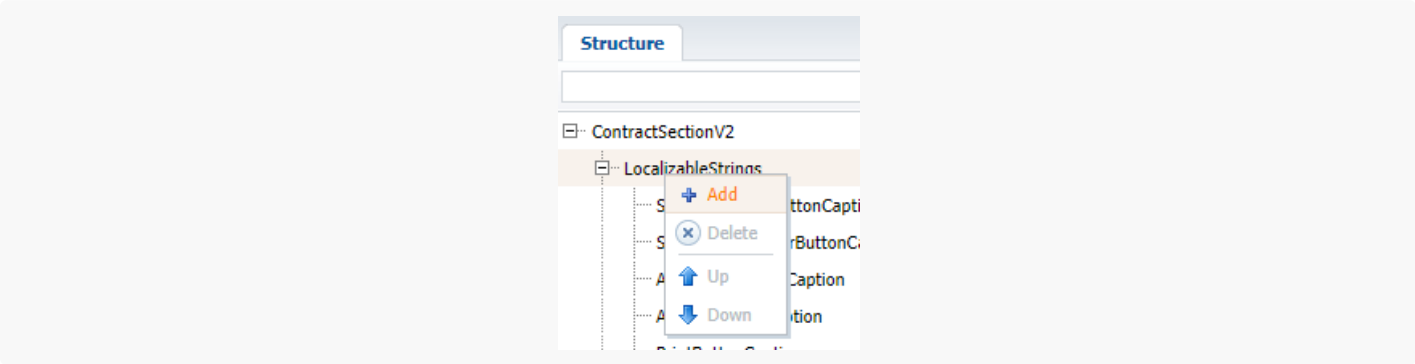
Процесс создания схемы замещающей модели представления описан в статье "[Создать клиентскую схему](#)".



2. В структуру схемы добавить локализуемые строки

Создайте две новые локализуемые строки со следующими свойствами:

Название (Name)	Значение (Value)
OwnerFilterCaption	Ответственный (Owner)
PeriodFilterCaption	Период (Period)



3. В коллекцию методов схемы замещающей модели

представления раздела добавить реализацию метода `initFixedFiltersConfig()`

В методе `initFixedFiltersConfig()` создайте конфигурационный объект с массивом фильтров `PeriodFilter` и `OwnerFilter`, присвойте ссылку на данный объект атрибуту `fixedFiltersConfig` модели представления.

Исходный код замещающей схемы:

ContractSectionV2.js

```
define("ContractSectionV2", ["BaseFiltersGenerateModule"], function(BaseFiltersGenerateModule) {
    return {
        // Название схемы раздела
        entitySchemaName: "Contract",
        // Методы модели представления раздела.
        methods: {
            // Инициализирует фиксированные фильтры.
            initFixedFiltersConfig: function() {
                // Создание конфигурационного объекта.
                var fixedFilterConfig = {
                    // В качестве схемы объекта для фиксированных фильтров указывается схема обь
                    entitySchema: this.entitySchema,
                    // Массив фильтров.
                    filters: [
                        // Фильтр периода.
                        {
                            // Название фильтра.
                            name: "PeriodFilter",
                            // Заголовок фильтра.
                            caption: this.get("Resources.Strings.PeriodFilterCaption"),
                            // Тип данных – дата.
                            dataValueType: this.Terrasoft.DataValueType.DATE,
                            // Дата начала периода фильтрации.
                            startDate: {
                                // Фильтруются данные из колонки [Date].
                                columnName: "StartDate",
                                // Значение по умолчанию – начало текущей недели.
                                defValue: this.Terrasoft.startOfWeek(new Date())
                            },
                        },
                        // Дата завершения периода фильтрации – завершение текущей недели.
                        dueDate: {
                            columnName: "StartDate",
                            defValue: this.Terrasoft.endOfWeek(new Date())
                        }
                    ],
                },
                // Фильтр ответственного.
                {
```

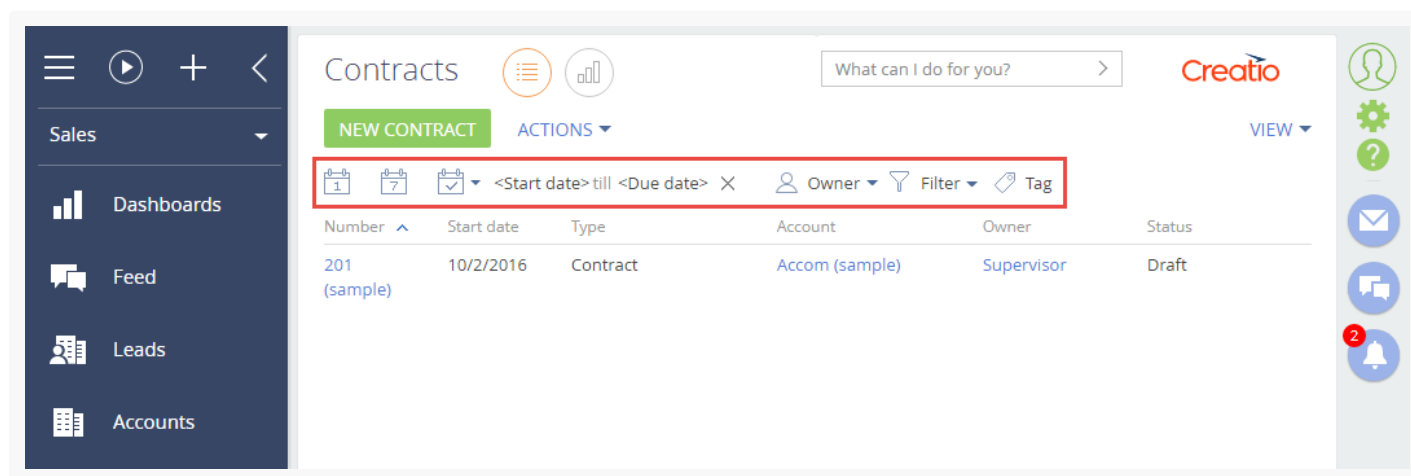
```

        // Название фильтра.
        name: "Owner",
        // Заголовок фильтра.
        caption: this.get("Resources.Strings.OwnerFilterCaption"),
        // Фильтрация данных из колонки [Owner].
        columnName: "Owner",
        // Значение по умолчанию – контакт текущего пользователя.
        // Берется из системной настройки.
        defValue: this.Terrasoft.SysValue.CURRENT_USER_CONTACT,
        // Тип данных – справочник.
        dataValueType: this.Terrasoft.DataValueType.LOOKUP,
        // Фильтр.
        filter: BaseFiltersGenerateModule.OwnerFilter
    }
}
};
// Атрибуту [FixedFilterConfig] присваивается ссылка на созданный конфигурационн
this.set("FixedFilterConfig", fixedFilterConfig);
}
}
});
});

```

Результат выполнения примера

После сохранения схемы и перезагрузки системы в разделе [*Договоры*] ([*Contracts*]) появится блок фиксированных фильтров, который позволит отбирать договоры как по дате их начала, так и по ответственному сотруднику.



Добавить редактируемый реестр в раздел



На [странице подбора продуктов](#) продуктов Sales Creatio доступна функциональность редактируемого реестра. По умолчанию редактируемый реестр добавлен в колонки [*Количество*], [*Единица измерения*] и [*Цена*]. Также существует возможность добавить редактируемый реестр в другие колонки.

Пример. На странице добавления продукта в разделе [*Заказы*] ([*Orders*]) необходимо добавить редактируемый реестр в колонку [*Скидка, %*] ([*Discount, %*]). Также добавить пользовательскую колонку [*Пользовательская цена*] ([*Custom price*]), в которую добавить редактируемый реестр.

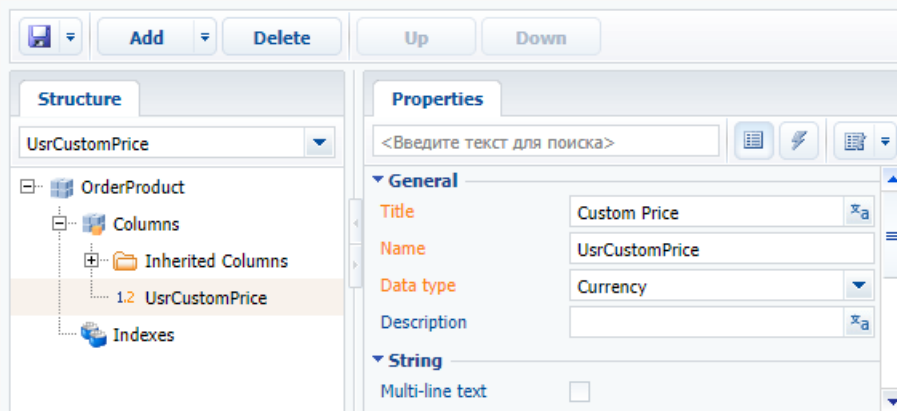
Пример можно выполнить также для страницы подбора продукта в разделе [*Счета*] ([*Invoices*]).

1. Добавить пользовательскую колонку в объект [Продукт в заказе] ([Product in order])

Для этого нужно создать схему замещающей модели представления [*Продукт в заказе*] ([*Product in order*]) и добавить в него колонку. Создание схему замещающей модели представления описано в статье "[Создать схему объекта](#)".

Для добавленной колонки необходимо установить следующие свойства:

- [Заголовок] ([Title]) — "Пользовательская цена" ("Custom Price").
- [Название] ([Name]) — "UsrCustomPrice".
- [Тип данных] ([Data type]) — "Валюта" ("Currency").



После внесения изменений схему необходимо опубликовать.

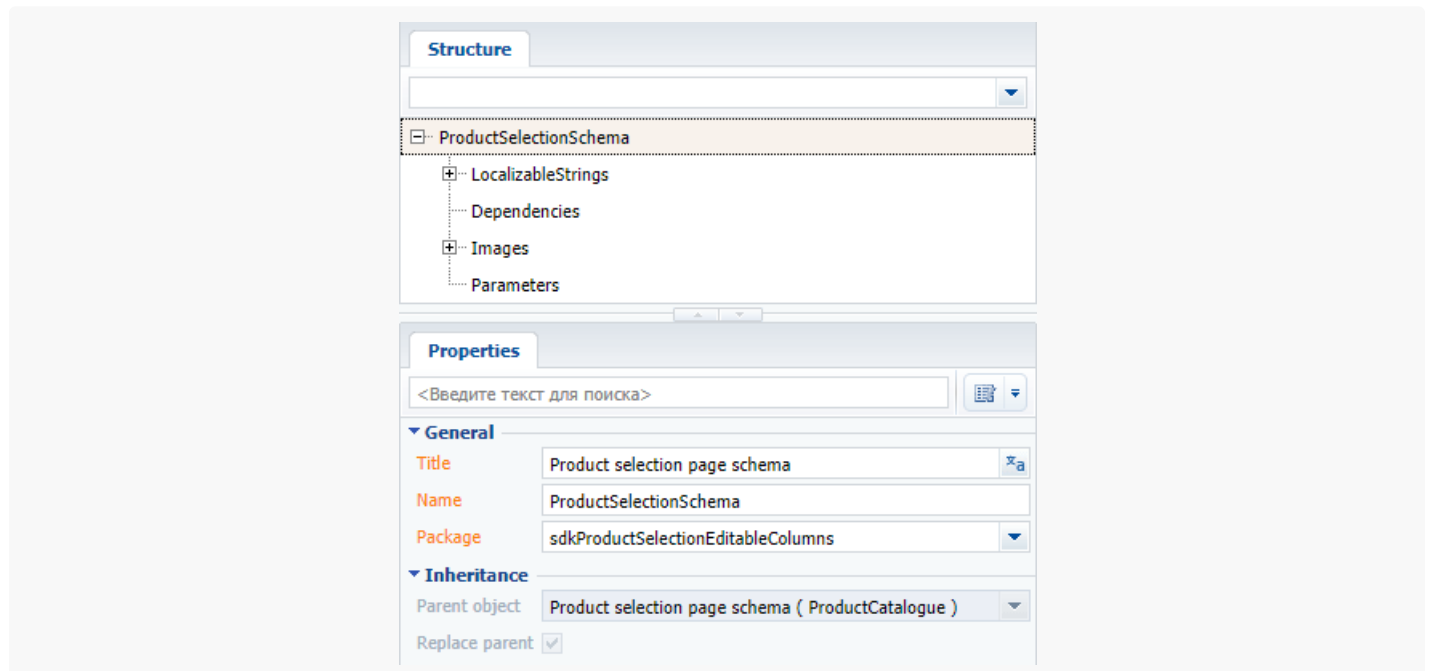
На заметку. Для реализации примера в разделе [*Счета*] ([*Invoices*]) приведенные выше действия необходимо выполнить для объекта [*Продукт в счете*] ([*Product in invoice*]).

2. Создать схему замещающей модели представления для

страницы подбора продуктов

Последовательность создания схемы замещающей модели представления описана в статье "[Создать клиентскую схему](#)". Для созданного модуля должны быть установлены следующие свойства:

- [Заголовок] ([Title]) — [Схема страницы подбора продуктов] ([Product selection page schema]).
- [Название] ([Name]) — "ProductSelectionSchema".
- [Родительский объект] ([Parent object]) — [Схема страницы подбора продуктов] ([Product selection page schema]).



Далее на вкладку [Исходный код] ([Source Code]) дизайнера схемы нужно добавить исходный код, приведенный ниже:

ProductSelectionSchema.js

```
define("ProductSelectionSchema", [],
    function() {
        return {
            methods: {
                getEditableColumns: function() {
                    // Получение массива редактируемых колонок.
                    var columns = this.callParent(arguments);
                    // Добавление колонки [Скидка, %] в массив редактируемых колонок.
                    columns.push("DiscountPercent");
                    // Добавление пользовательской колонки.
                    columns.push("UsrCustomPrice");
                    return columns;
                },
                setColumnHandlers: function(item) {
```

```

        this.callParent(arguments);
        // Привязка обработчика события изменения пользовательской колонки.
        item.on("change:UsrCustomPrice", this.onCustomPriceChanged, this);
    },
    // Метод-обработчик, который будет вызываться при изменении значения поля.
    onCustomPriceChanged: function(item, value) {
        window.console.log("Changed: ", item, value);
    }
}
};
});

```

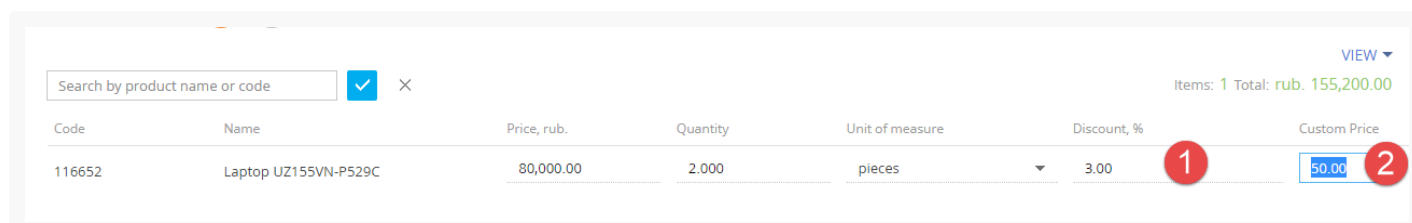
После внесения изменений схему необходимо сохранить.

3. Настроить отображение колонок на странице подбора продуктов

Чтобы требуемые колонки появились на странице подбора продуктов необходимо настроить их отображение (см. ["Настройка колонок реестра"](#)). В режиме настройки плиточного представления реестра нужно добавить колонки [Скидка, %] ([Discount, %]) и [Пользовательская цена] ([Custom price]).

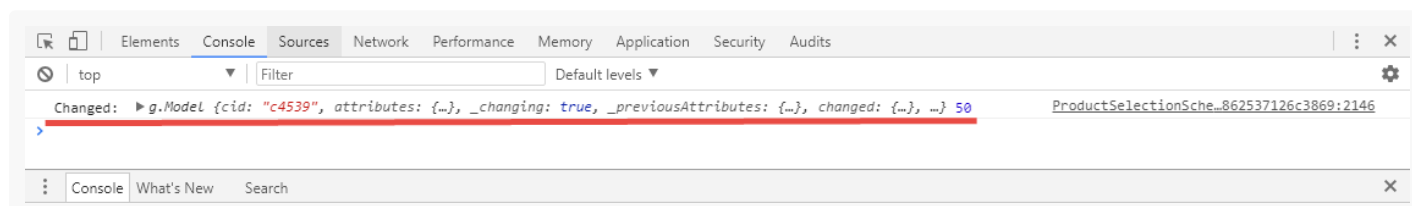
Результат выполнения примера

В результате выполнения примера на странице подбора продуктов появятся две редактируемые колонки.



Code	Name	Price, rub.	Quantity	Unit of measure	Discount, %	Custom Price
116652	Laptop UZ155VN-P529C	80,000.00	2.000	pieces	3.00	50.00

При этом после изменения значения колонки [Пользовательская цена] ([Custom price]) в консоли браузера выводится соответствующее сообщение.



Кнопка

Для того, чтобы добавить на страницу записи пользовательскую кнопку, в модели представления необходимо изменить два свойства:

- массив конфигурационных объектов `diff`. В него необходимо добавить конфигурационный объект для настройки визуального расположения элемента управления на странице записи.
- коллекцию `methods`. В нее необходимо добавить реализацию метода-обработчика, который будет вызываться при нажатии на кнопку, а также другие вспомогательные методы, необходимые для функционирования элемента управления. Это могут быть методы, которые будут регулировать видимость или доступность элемента управления в зависимости от условий.

Важно.

Для отображения кнопки на странице в совмещенном режиме редактирования записи (с открытым вертикальным реестром) необходимо внести изменения как в схему модели представления раздела, так и в схему модели представления страницы записи.

Чтобы кнопка отображалась на странице в обычном режиме редактирования записи (режиме добавления данных), изменения вносятся только в схему модели представления страницы.

На заметку.

В обычный режим отображения страницы записи можно перейти не только при создании записи, но и если обновить страницу, находящуюся в совмещенном режиме отображения (с вертикальным реестром).

DOM-модель стандартных кнопок страницы

Для расположения стандартных функциональных кнопок страниц записей Creatio используется иерархическая структура html-контейнеров.

`CombinedModeActionButtonsCardContainer` — контейнер верхнего уровня в совмещенном режиме редактирования записи. Внутри него располагаются еще два контейнера:

- `CombinedModeActionButtonsCardLeftContainer`, в котором размещены стандартные кнопки [*Заккрыть*], [*Сохранить*], [*Отмена*], [*Действия*] и [*Теги*];
- `CombinedModeActionButtonsCardRightContainer`, в котором расположены кнопки [*Печать*] и [*Вид*].

Аналогично, для страницы записи в обычном режиме контейнер верхнего уровня — `ActionButtonsContainer`. Внутри него располагаются еще два контейнера:

- `Leftcontainer`, в котором размещены стандартные кнопки [*Заккрыть*], [*Сохранить*], [*Отмена*], [*Действия*] и [*Теги*];
- `RightContainer`, в котором расположены кнопки [*Печать*] и [*Вид*].

В зависимости от того, где именно нужно расположить кнопку, при настройке визуализации кнопки в массиве модификаций `diff` указывается соответствующий контейнер.

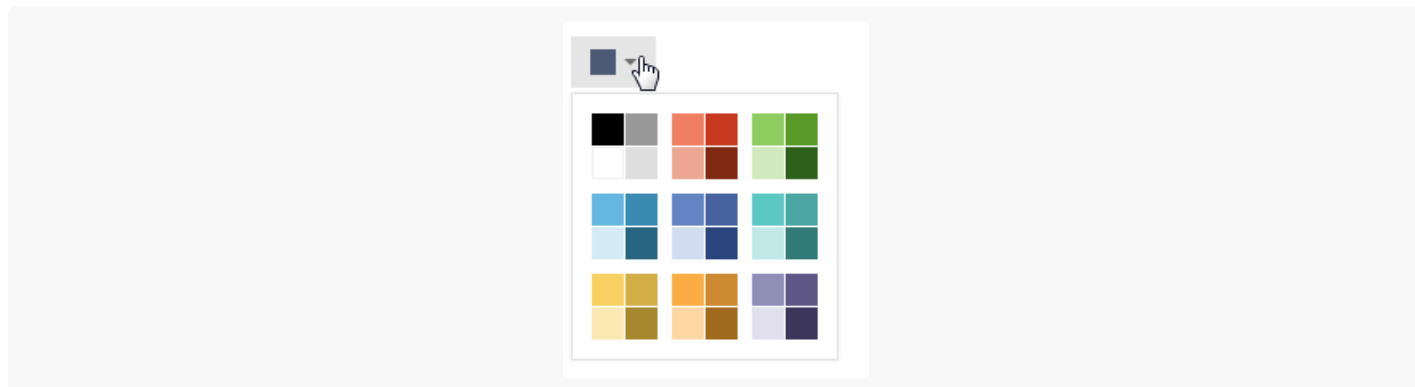
На заметку.

Здесь используются мета-имена html-контейнеров. Эти имена указываются при настройке визуализации элемента управления в конфигурационном объекте массива `diff`. Фактические идентификаторы соответствующих html-элементов страницы формируются системой автоматически на основании таких мета-имен.

Особенности кнопки выбора цвета

Один из элементов управления, который предоставляет Creatio — кнопка выбора цвета (рис. 1).

Рис. 1. — Кнопка выбора цвета



Алгоритм добавления кнопки выбора цвета на страницу записи объекта:

1. Добавить в объект колонку с типом данных [*Строка (50 символов)*], которая будет хранить информацию о выбранном цвете.
2. Добавить в массив `diff` описание элемента с типом [*COLOR_BUTTON*]. Установить для свойства `value` этого элемента привязку к добавленной в предыдущем шаге колонке.
3. При необходимости добавить подпись для кнопки с помощью элемента управления [*LABEL*].

Добавить к кнопке всплывающую подсказку

В Creatio существует возможность добавлять к элементам системы всплывающие подсказки — текстовые сообщения, которые предоставляют пользователю дополнительную информацию о функциональности элемента или правилах его заполнения.

Все всплывающие подсказки можно разделить на 3 основные группы:

1. Всплывающая подсказка к полю (при наличии такой подсказки заголовок поля помечается маленьким зеленым треугольником)  *Ответственный*. Подсказка появляется при наведении курсора на треугольник либо при нажатии на заголовок поля.
2. Всплывающая подсказка к другим элементам управления (кнопкам, индикаторам заполненности, изображениям). Подсказка появляется при наведении курсора на элемент управления.
3. Информационная кнопка . Подсказка появляется при наведении курсора на информационную кнопку.

Общий алгоритм добавления всплывающих подсказок к стандартным элементам управления:

1. Создать схему замещающей модели представления страницы или раздела.
2. Добавить текст подсказки в коллекцию локализуемых строк схемы.
3. Описать необходимые модификации элементов схемы в массиве `diff`.

Добавить кнопку в раздел

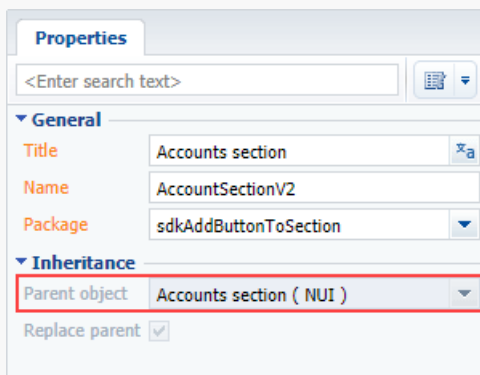
 Средний

Пример. В раздел [*Контрагенты*] добавить кнопку, которая будет открывать страницу основного контакта у выбранного в реестре контрагента.

Обращение к выделенной записи выполняется через атрибут модели представления раздела `ActiveRow`, который возвращает значение первичной колонки выделенной записи. Далее это значение может использоваться для получения значений, загруженных в реестр полей выбранного объекта, например, из коллекции данных списочного реестра, которая хранится в свойстве `GridData` модели представления реестра.

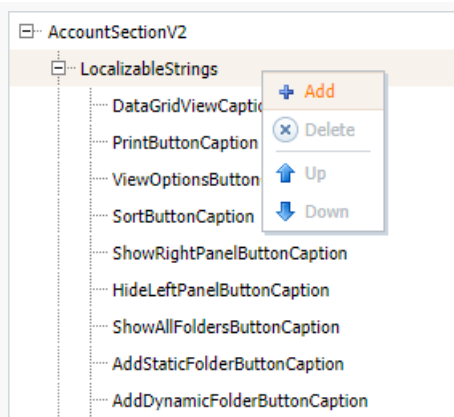
1. Создать схему замещающей модели представления страницы раздела [*Контрагенты*]

Создайте схему замещающей модели представления, в которой в качестве родительского объекта укажите схему [*Раздел контрагенты*] ([*Accounts section*]) . Процесс создания схемы замещающей модели представления описан в статье "[Создать клиентскую схему](#)".



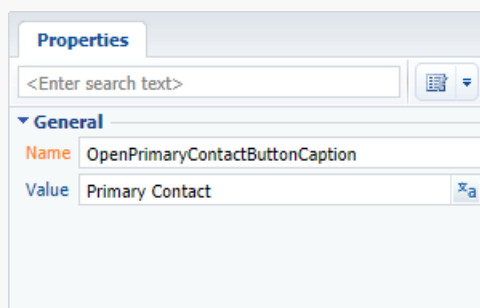
2. В коллекцию локализуемых строк схемы замещающей модели представления раздела добавить строку с заголовком кнопки

Создайте новую локализуемую строку.



Для созданной строки укажите:

- [*Название*] ([*Name*]) — "OpenPrimaryContactButtonCaption";
- [*Значение*] ([*Value*]) — "Основной контакт" ("Primary Contact").



3. В коллекцию методов модели представления раздела добавить реализацию методов

- `isAccountPrimaryContactSet()` — проверяет, заполнено ли поле [*Основной контакт*] страницы.
- `onOpenPrimaryContactClick()` — метод-обработчик нажатия кнопки. Выполняет переход на страницу основного контакта.

4. В массив `diff` добавить конфигурационный объект с настройками расположения кнопки на странице

В массив `diff` добавьте объект с необходимыми настройками расположения кнопки на странице.

AccountSectionV2.js

```
define("AccountSectionV2", [], function() {
    return {
        // Название схемы объекта раздела.
        entitySchemaName: "Account",
```

```

// Методы модели представления раздела.
methods: {
    // Метод-обработчик нажатия кнопки.
    onOpenPrimaryContactClick: function() {
        // Получение идентификатора выбранной записи.
        var activeRow = this.get("ActiveRow");
        if (!activeRow) {
            return;
        }
        // Определение идентификатора основного контакта.
        var primaryId = this.get("GridData").get(activeRow).get("PrimaryContact").value;
        if (!primaryId) {
            return;
        }
        // Формирование строки адреса.
        var requestUrl = "CardModuleV2/ContactPageV2/edit/" + primaryId;
        // Публикация сообщения о пополнении истории навигации по страницам
        // и переход на страницу основного контакта.
        this.sandbox.publish("PushHistoryState", {
            hash: requestUrl
        });
    },
    // Проверяет, заполнено ли поле [Основной контакт] выбранного элемента.
    isAccountPrimaryContactSet: function() {
        var activeRow = this.get("ActiveRow");
        if (!activeRow) {
            return false;
        }
        var pc = this.get("GridData").get(activeRow).get("PrimaryContact");
        return (pc || pc !== "") ? true : false;
    }
},
//Отображение кнопки в разделе.
diff: /**SCHEMA_DIFF*/[
    // Метаданные для добавления в раздел пользовательской кнопки.
    {
        // Выполняется операция добавления компонента на страницу.
        "operation": "insert",
        // Мета-имя родительского контейнера, в который добавляется кнопка.
        "parentName": "ActionButtonsContainer",
        // Кнопка добавляется в коллекцию компонентов
        // родительского элемента.
        "propertyName": "items",
        // Мета-имя добавляемой кнопки.
        "name": "MainContactSectionButton",
        // Свойства, передаваемые в конструктор компонента.
        "values": {
            // Тип добавляемого компонента – кнопка.
            itemType: Terrasoft.ViewItemType.BUTTON,

```

```

// Привязка заголовка кнопки к локализуемой строке схемы.
caption: { bindTo: "Resources.Strings.OpenPrimaryContactButtonCaption" }
// Привязка метода-обработчика нажатия кнопки.
click: { bindTo: "onOpenPrimaryContactClick" },
// Привязка свойства доступности кнопки.
enabled: { bindTo: "isAccountPrimaryContactSet" },
// Настройка расположения кнопки.
"layout": {
    "column": 1,
    "row": 6,
    "colSpan": 1
}
}
}
]/**SCHEMA_DIFF*/
};
});

```

Результат выполнения примера

После сохранения схемы и обновления страницы приложения с очисткой кэша в разделе [*Контрагенты*] ([*Accounts*]) появится кнопка [*Основной контакт*] ([*Primary Contact*]). Кнопка активируется после выбора контрагента, у которого указан основной контакт.

The screenshot shows the 'Accounts' application interface. At the top, there is a search bar with the text 'What can I do for you?' and the 'Creatio' logo. Below the search bar, there are buttons for 'NEW ACCOUNT', 'ACTIONS', and 'PRIMARY CONTACT' (which is highlighted with a red box). There is also a 'VIEW' dropdown menu. Below these buttons, there is a table of accounts. The table has columns for 'Web', 'Primary phone', 'City', 'Country', and 'Type'. The accounts listed are 'XT Group', 'Global Venture', 'Feature IT', and 'Fast Works'. The 'Feature IT' row is highlighted in blue. At the bottom of the table, there are buttons for 'OPEN', 'COPY', 'DELETE', and 'PRINT'. On the right side of the interface, there is a vertical sidebar with icons for a user profile, settings, help, and other functions.

Account Name	Web	Primary phone	City	Country	Type
XT Group Primary contact: Jason Robinson	www.xtg.au 148 Bunda St, Canberra ACT 2601	+61 2 6247 5656 Canberra	Australia	Customer	
Global Venture Primary contact: Zane Rogers	www.globalventure.com.ny 412 Pine Street, New York, NY	+1 212 721 1810 New York	United States	Customer	
Feature IT Primary contact: Tony Campbell	www.feature-it.com 85 46th Street	+1 212 735 2537 New York	United States	Partner	
Fast Works Primary contact: Kate Roberts	www.fast-works.com New York	+1 212 775 9836 New York	United States	Customer	

Добавить кнопку в совмещенном режиме



Описание примера

На страницу контрагента, отображаемую в совмещенном режиме, необходимо добавить кнопку, которая будет открывать страницу основного контакта. При этом, если основной контакт не указан, кнопка должна быть заблокирована.

Исходный код

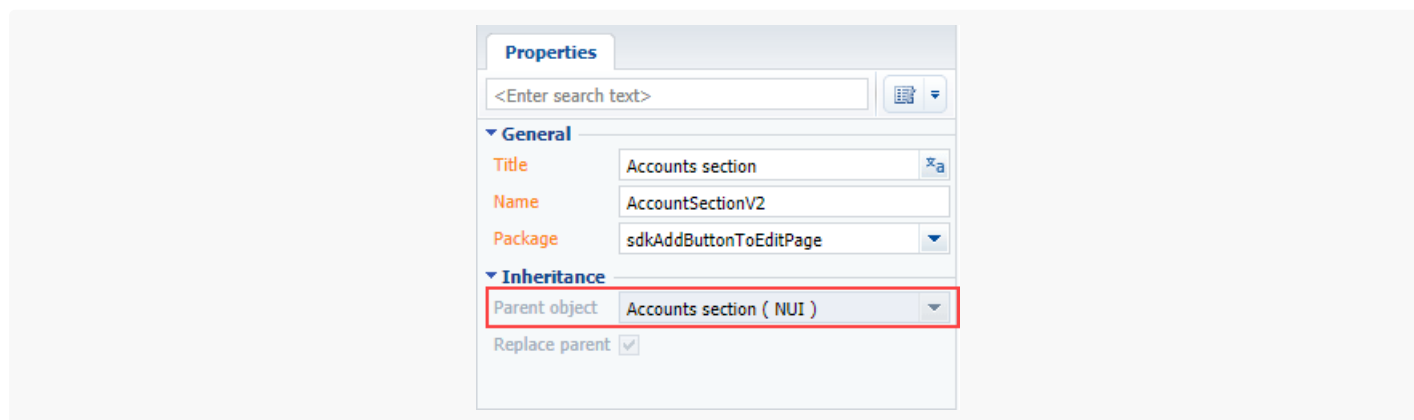
Пакет с реализацией примера можно скачать по [ссылке](#).

Алгоритм реализации примера

1. Создать схему замещающей модели представления раздела [*Контрагенты*]

Создайте схему замещающей модели представления, в которой в качестве родительского объекта укажите схему [*Раздел контрагенты*] ([*Accounts section*]) (рис. 1). Процесс создания схемы замещающей модели представления описан в статье "[Создать клиентскую схему](#)".

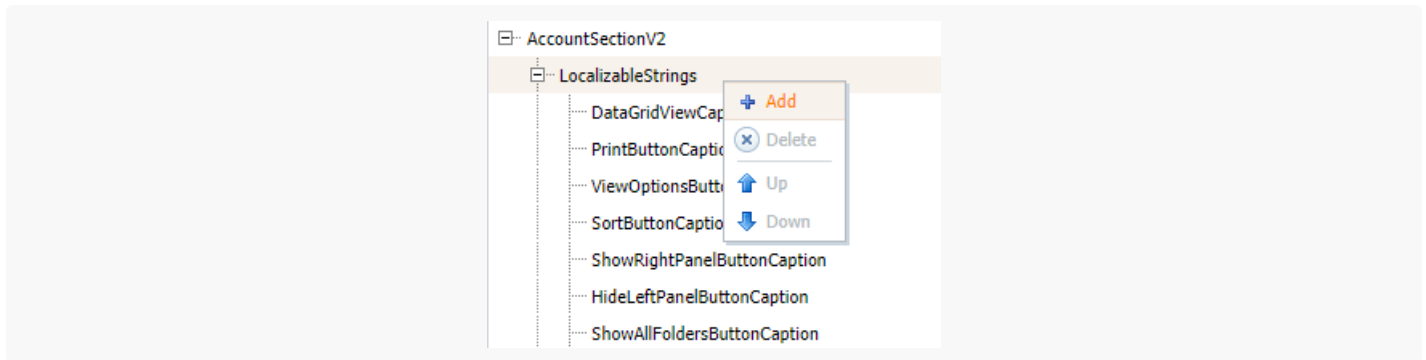
Рис. 1. — Свойства схему замещающей модели представления раздела



2. В коллекцию локализуемых строк схему замещающей модели представления раздела добавить строку с заголовком кнопки

Создайте новую локализуемую строку (рис. 2).

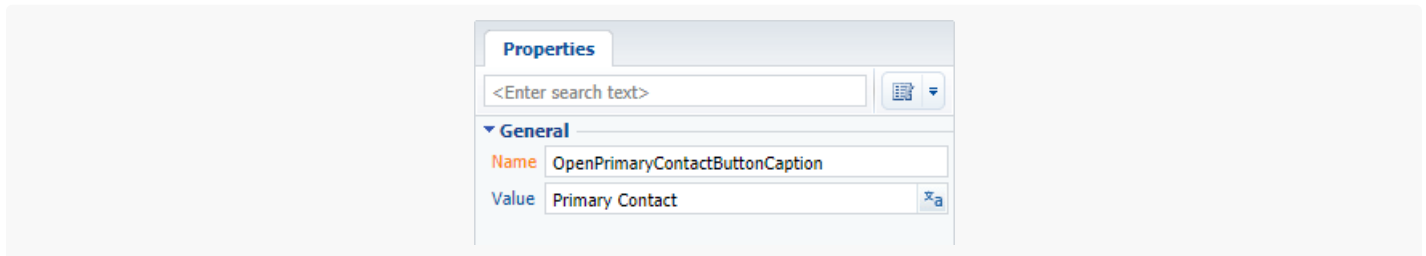
Рис. 2. — Добавление в схему локализуемой строки



Для созданной строки укажите (рис. 3):

- [*Название*] ([*Name*]) — "OpenPrimaryContactButtonCaption";
- [*Значение*] ([*Value*]) — "Основной контакт" ("Primary Contact").

Рис. 3. — Свойства пользовательской локализуемой строки



3. Добавить атрибут доступности кнопки

В коллекцию атрибутов схемы модели представления раздела добавьте атрибут `ButtonEnabled` типа `Terrasoft.DataValueType.BOOLEAN`, сохраняющий состояние доступности кнопки.

4. В коллекцию методов модели представления раздела добавить реализацию методов

- `onOpenPrimaryContactClick()` — выполняет переход на страницу основного контакта.
- `onCardRendered()` — выполняется после отрисовки страницы контрагента в совмещенном режиме. Используется для подписки на события загрузки данных в реестр и событие изменения активной записи реестра.
- `isPrimaryContactExist()` — определяет наличие основного контакта для активной записи реестра.
- `setButtonEnabled()` — устанавливает значение атрибута `ButtonEnabled` в зависимости от того, определен ли основной контакт контрагента.

5. Добавить кнопку на страницу записи в совмещенном режиме

В массив `diff` схемы раздела добавьте конфигурационный объект с настройками расположения и стиля кнопки.

Исходный код схемы замещающей модели представления:

```

define("AccountSectionV2", [], function() {
    return {
        // Название схемы объекта раздела.
        entitySchemaName: "Account",
        attributes: {
            // Атрибут для хранения состояния доступности кнопки.
            "ButtonEnabled": {
                "dataValueType": Terrasoft.DataValueType.BOOLEAN,
                "type": Terrasoft.ViewModelColumnType.VIRTUAL_COLUMN,
                "value": false
            }
        },
        // Методы модели представления раздела.
        methods: {
            // Метод-обработчик нажатия кнопки.
            onOpenPrimaryContactClick: function() {
                // Определение активной записи вертикального реестра.
                var activeRow = this.get("ActiveRow");
                if (activeRow) {
                    // Определение идентификатора основного контакта.
                    var primaryId = this.get("GridData").get(activeRow).get("PrimaryContact").value;
                    if (primaryId) {
                        // Формирование строки адреса.
                        var requestUrl = "CardModuleV2/ContactPageV2/edit/" + primaryId;
                        // Публикация сообщения о пополнении истории навигации по страницам
                        // и переход на страницу основного контакта.
                        this.sandbox.publish("PushHistoryState", {
                            hash: requestUrl
                        });
                    }
                }
            },
            // Выполняется после отрисовки страницы контрагента.
            onCardRendered: function() {
                this.callParent();
                // Данные реестра.
                var gridData = this.get("GridData");
                var activeRow = this.get("ActiveRow");
                if (activeRow) {
                    {
                        this.setButtonEnabled(activeRow, this);
                    }
                }
                // После закрытия страницы основного контакта теряется активная запись. Ее нужно
                // и для нее проверить наличие основного контакта.
                else {
                    var historyState = this.sandbox.publish("GetHistoryState");
                    var hash = historyState.hash;
                }
            }
        }
    };
});

```

```

        if (hash && hash.valuePairs)
        {
            activeRow = hash.valuePairs[0].name;
            // Восстановление активной записи.
            this.set("ActiveRow", activeRow);
            // Сохранение контекста в локальную переменную.
            var self = this;
            // Подписка на событие полной загрузки данных в вертикальный реестр.
            gridData.on("dataloaded", function() {
                self.setButtonEnabled(activeRow, self);
            });
        }
    }
    // Подписка на событие изменения активной записи реестра.
    gridData.on("itemchanged", function() {
        this.setButtonEnabled(activeRow, this);
    }, this);
},
// Определяет наличие основного контакта для активной записи реестра.
isPrimaryContactExist: function(id) {
    var pc = this.get("GridData").get(id).get("PrimaryContact");
    return (pc || pc !== "") ? true : false;
},
// Устанавливает значение атрибута ButtonEnabled в зависимости от того, определен ли
setButtonEnabled: function(activeRow, context) {
    if (context.isPrimaryContactExist(activeRow)) {
        context.set("ButtonEnabled", true);
    }
    else {
        context.set("ButtonEnabled", false);
    }
}
},
//Настройка визуализации кнопки на странице записи.
diff: [
    // Метаданные для добавления на страницу пользовательской кнопки.
    {
        // Выполняется операция добавления элемента на страницу.
        "operation": "insert",
        // Мета-имя родительского контейнера, в который добавляется кнопка.
        "parentName": "CombinedModeActionButtonsCardLeftContainer",
        // Кнопка добавляется в коллекцию компонентов
        // родительского элемента.
        "propertyName": "items",
        // Мета-имя добавляемой кнопки.
        "name": "MainContactButton",
        // Свойства, передаваемые в конструктор компонента.
        "values": {
            // Тип добавляемого элемента – кнопка.

```

```

        "itemType": Terrasoft.ViewItemType.BUTTON,
        // Привязка заголовка кнопки к локализуемой строке схемы.
        "caption": {bindTo: "Resources.Strings.OpenPrimaryContactButtonCaption"},
        // Привязка метода-обработчика нажатия кнопки.
        "click": {bindTo: "onOpenPrimaryContactClick"},
        // Стиль отображения кнопки.
        "style": Terrasoft.controls.ButtonEnums.style.GREEN,
        // Привязка свойства доступности кнопки.
        "enabled": {bindTo: "ButtonEnabled"}
    }
}
];
};
});

```

После сохранения схемы и обновления страницы приложения, на странице контрагента появится кнопка [Основной контакт] ([*Primary contact*]) (рис. 4). При этом, если основной контакт не указан, кнопка будет неактивна (рис. 5).

Рис. 4. — Результат выполнения примера. Основной контакт указан — кнопка активна

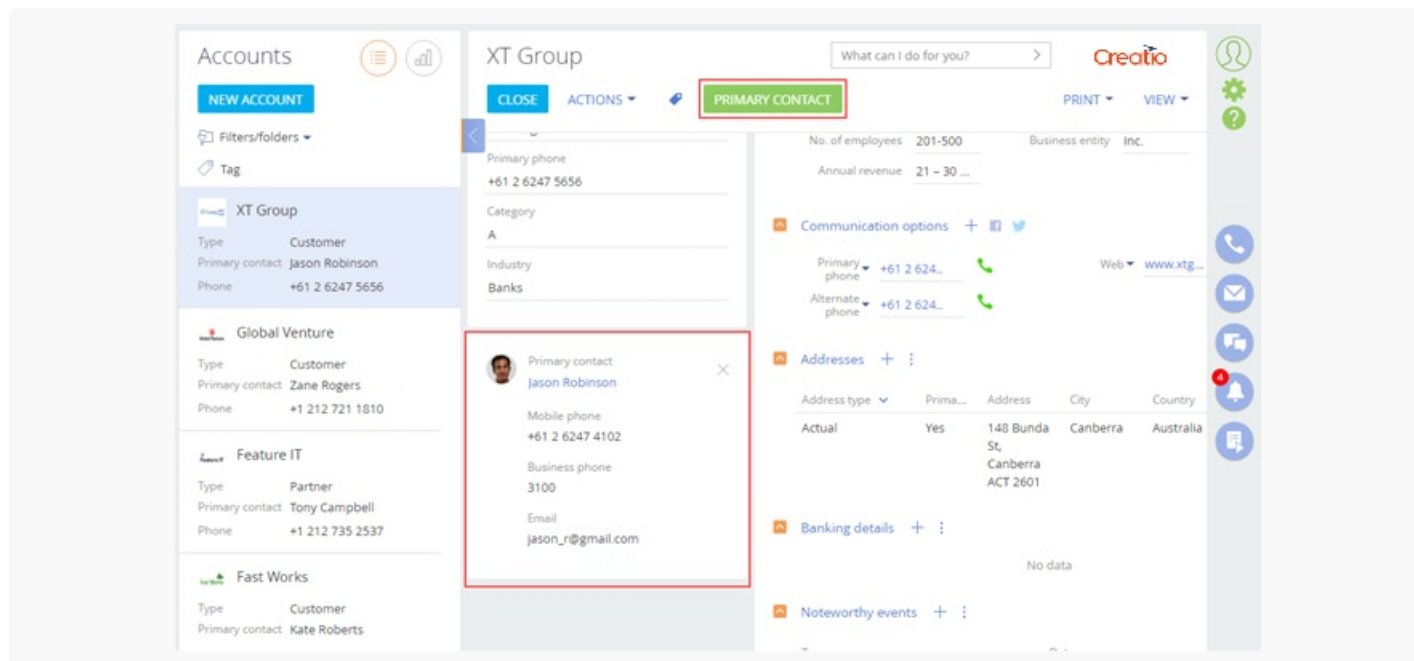
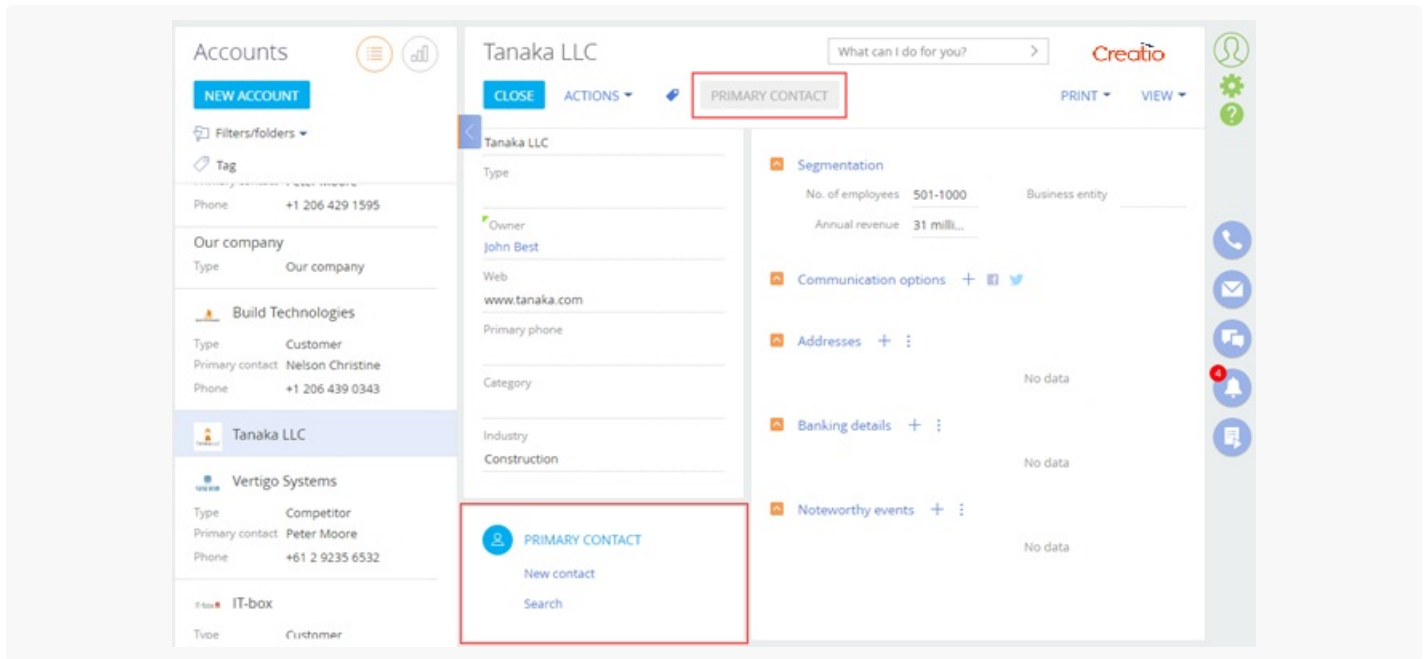


Рис. 5. — Результат выполнения примера. Основной контакт не указан — кнопка неактивна



Добавить кнопку в режиме добавления новой записи

 Средний

Описание примера

На страницу добавления нового контрагента необходимо добавить кнопку, которая будет открывать страницу основного контакта.

Важно.

По умолчанию для добавления нового контрагента используется миникарточка. Чтобы для добавления контрагента использовать страницу добавления, необходимо установить значение `false` в поле [Значение по умолчанию] системной настройки [Использовать миникарточку добавления контрагента] (`HasAccountMiniPageAddMode`). Для того, чтобы изменение системной настройки вступило в силу, следует выполнить выход и вход пользователя.

Для выполнения этого примера нужно использовать страницу добавления.

На заметку.

В обычный режим отображения страницы записи можно перейти не только при создании записи, но и если обновить страницу, находящуюся в совмещенном режиме отображения, например, с помощью клавиши F5. При этом вертикальный реестр должен быть скрыт.

Исходный код

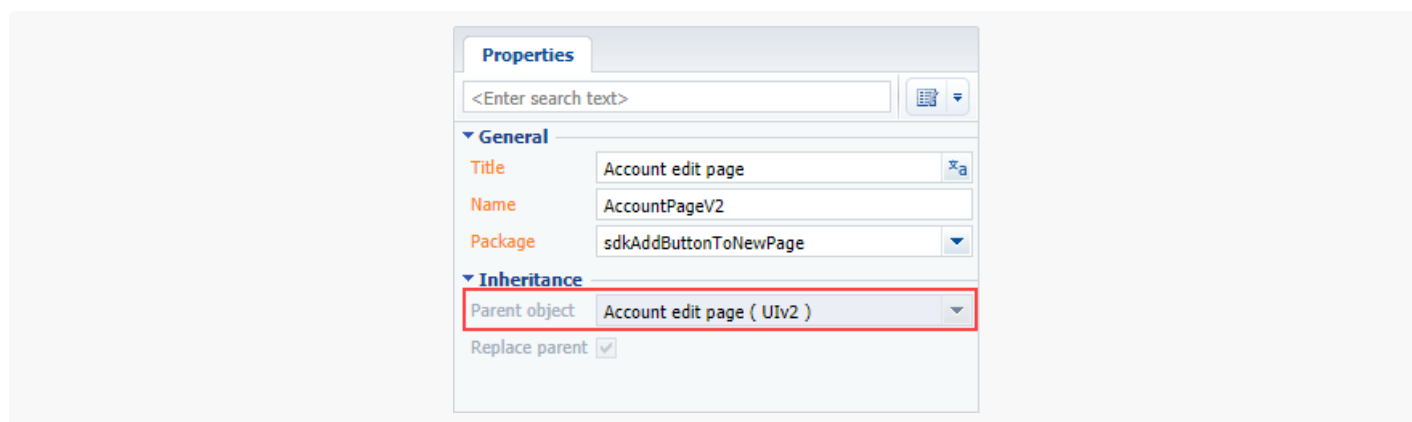
Пакет с реализацией примера можно скачать по [ссылке](#).

Алгоритм реализации примера

1. Создать схему замещающей модели представления страницы контрагента

Создайте схему замещающей модели представления, в котором в качестве родительского объекта укажите схему [*Страница редактирования контрагента*] ([*Account edit page*], AccountPageV2) (рис. 1). Процесс создания схемы замещающей модели представления описан в статье "[Создать клиентскую схему](#)".

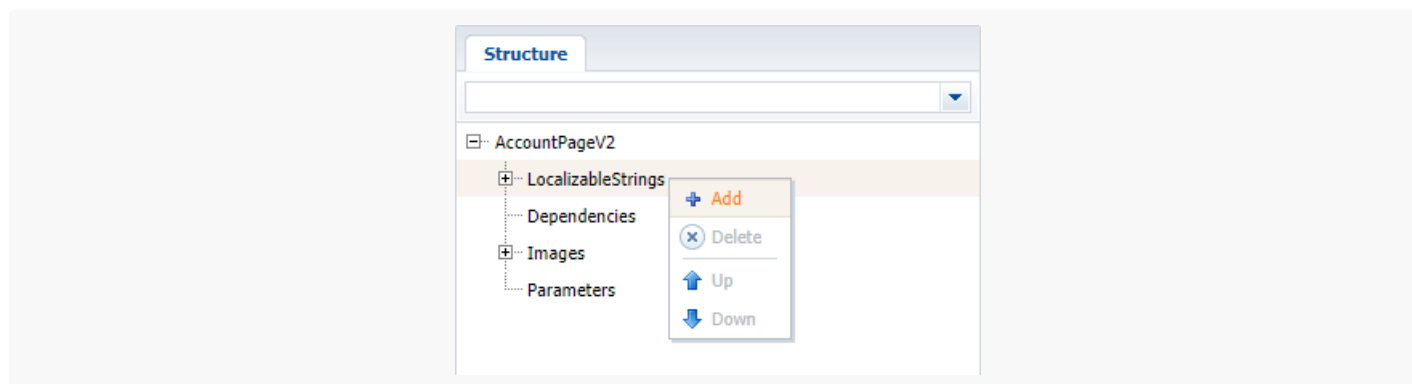
Рис. 1. — Свойства схему замещающей модели представления страницы записи



2. В коллекцию локализуемых строк схему замещающей модели представления страницы добавить строку с заголовком кнопки

Создайте новую локализуемую строку (рис. 2).

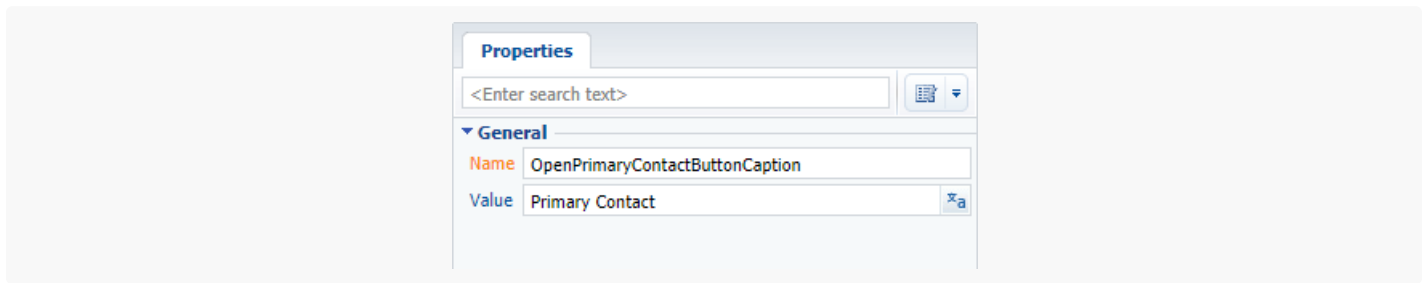
Рис. 2. — Добавление в схему локализуемой строки



Для созданной строки укажите (рис. 3):

- [*Название*] ([*Name*]) — "OpenPrimaryContactButtonCaption";
- [*Значение*] ([*Value*]) — "Основной контакт" ("Primary Contact").

Рис. 3. — Свойства пользовательской локализуемой строки



3. В коллекцию методов модели представления страницы добавить реализацию методов

- `isAccountPrimaryContactSet()` — проверяет, заполнено ли поле [*Основной контакт*] страницы.
- `onOpenPrimaryContactClick()` — метод-обработчик нажатия кнопки, который выполняет переход на страницу основного контакта.

4. Добавить кнопку на страницу записи

В массив `diff` добавьте объект с необходимыми настройками расположения кнопки на странице контрагента.

Исходный код схемы замещающей модели представления:

```
define("AccountPageV2", [], function() {
    return {
        // Название схемы объекта страницы записи.
        entitySchemaName: "Account",
        // Методы модели представления страницы записи.
        methods: {
            // Проверяет, заполнено ли поле [Основной контакт] страницы.
            isAccountPrimaryContactSet: function() {
                return this.get("PrimaryContact") ? true : false;
            },
            // Метод-обработчик нажатия кнопки.
            onOpenPrimaryContactClick: function() {
                var primaryContactObject = this.get("PrimaryContact");
                if (primaryContactObject) {
                    // Определение идентификатора основного контакта.
                    var primaryContactId = primaryContactObject.value;
                    // Формирование строки адреса.
                    var requestUrl = "CardModuleV2/ContactPageV2/edit/" + primaryContactId;
                    // Публикация сообщения и переход на страницу основного контакта.
                    this.sandbox.publish("PushHistoryState", {
                        hash: requestUrl
                    });
                }
            }
        }
    }
});
```



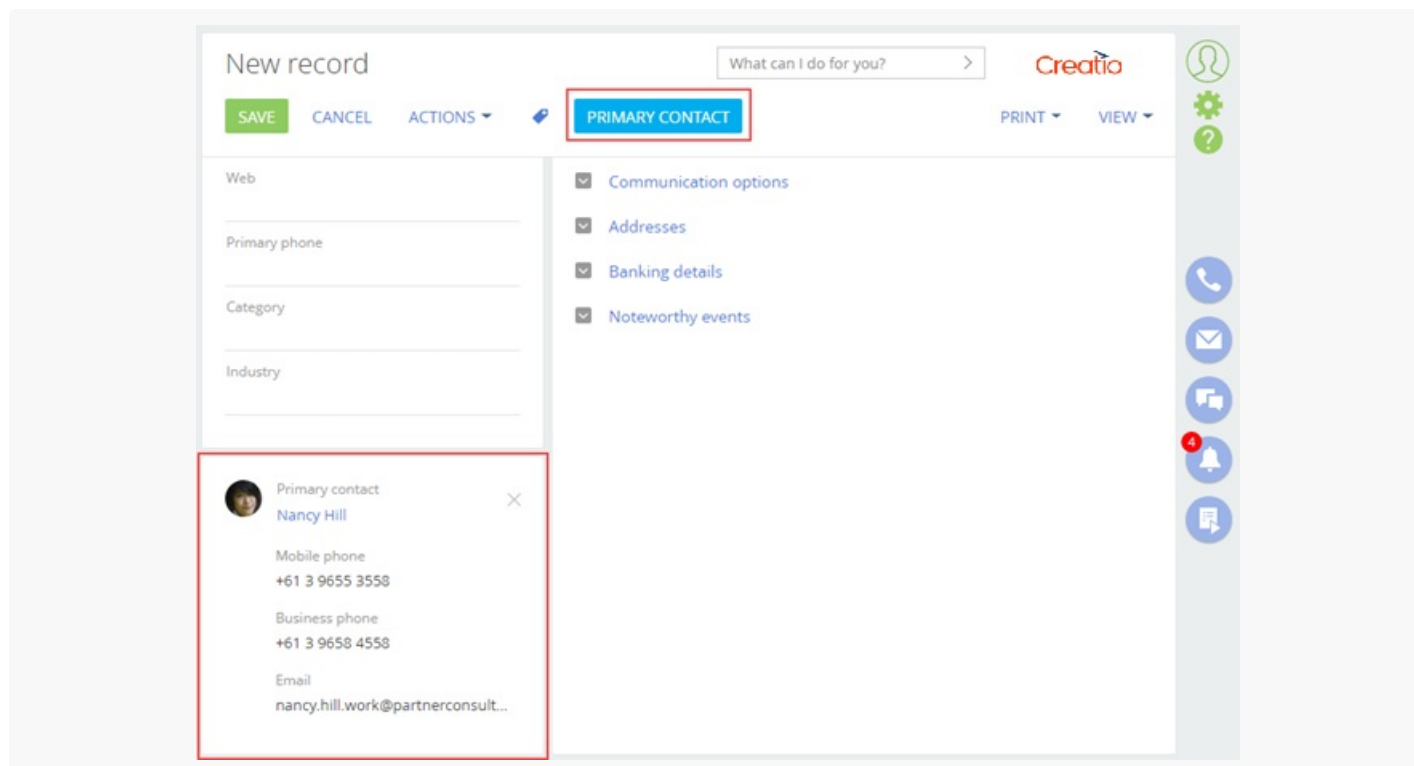
```

    },
    //Отображение кнопки на странице записи.
    diff: [
        // Метаданные для добавления на страницу пользовательской кнопки.
        {
            // Выполняется операция добавления компонента на страницу.
            "operation": "insert",
            // Мета-имя родительского контейнера, в который добавляется кнопка.
            "parentName": "LeftContainer",
            // Кнопка добавляется в коллекцию компонентов
            // родительского элемента.
            "propertyName": "items",
            // Мета-имя добавляемой кнопки.
            "name": "PrimaryContactButton",
            // Свойства, передаваемые в конструктор компонента.
            "values": {
                // Тип добавляемого элемента – кнопка.
                itemType: Terrasoft.ViewItemType.BUTTON,
                // Привязка заголовка кнопки к локализуемой строке схемы.
                caption: {bindTo: "Resources.Strings.OpenPrimaryContactButtonCaption"},
                // Привязка метода-обработчика нажатия кнопки.
                click: {bindTo: "onOpenPrimaryContactClick"},
                // Привязка свойства доступности кнопки.
                enabled: {bindTo: "isAccountPrimaryContactSet"},
                // Стиль отображения кнопки.
                "style": Terrasoft.controls.ButtonEnums.style.BLUE
            }
        }
    ]
};
});

```

После сохранения схемы и обновления страницы приложения, на странице создания нового контрагента появится кнопка [*Основной контакт*] ([*Primary contact*]), которая будет активизироваться после заполнения у контрагента поля [*Основной контакт*] ([*Primary contact*]) (рис. 4).

Рис. 4. — Демонстрация результата выполнения примера



Добавить кнопку выбора цвета

 Средний

Описание примера

Добавить на страницу продукта кнопку выбора цвета.

Исходный код

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

Алгоритм реализации примера

1. Создать схему замещающего объекта "Продукт" и добавить в него колонку [*UsrColor*]

Создайте схему замещающего объекта [*Продукт*] ([*Product*]) (рис. 2). Процесс создания схему замещающего объекта описан в статье "[Создать схему объекта](#)".

Рис. 2. — Свойства схему замещающего объекта

Properties

<Enter search text>

General

Name: Product

Title: Product

Package: sdkAddColorButton

Inheritance

Parent object: Product (Base)

Replace parent: ☒

Behavior

Allow records deactivation: ☐

Access rights

Operations: ☐

Records: ☒

Создайте новую колонку (рис. 3) и укажите для нее следующие свойства (рис. 4):

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

Рис. 3. — Добавление новой колонки

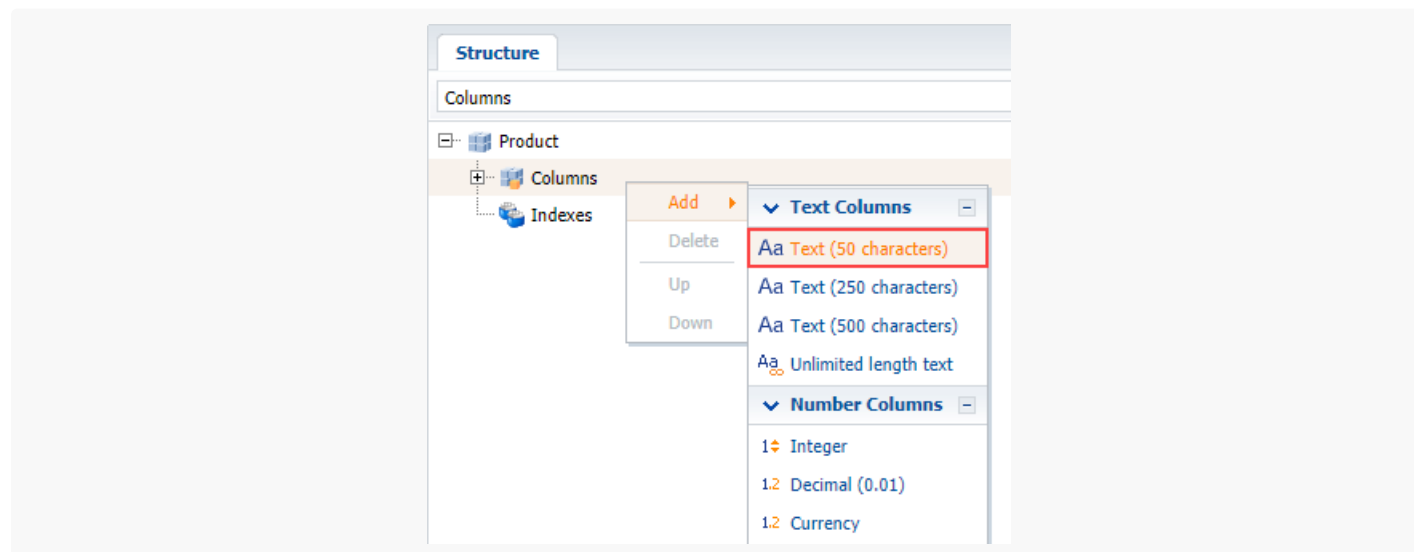
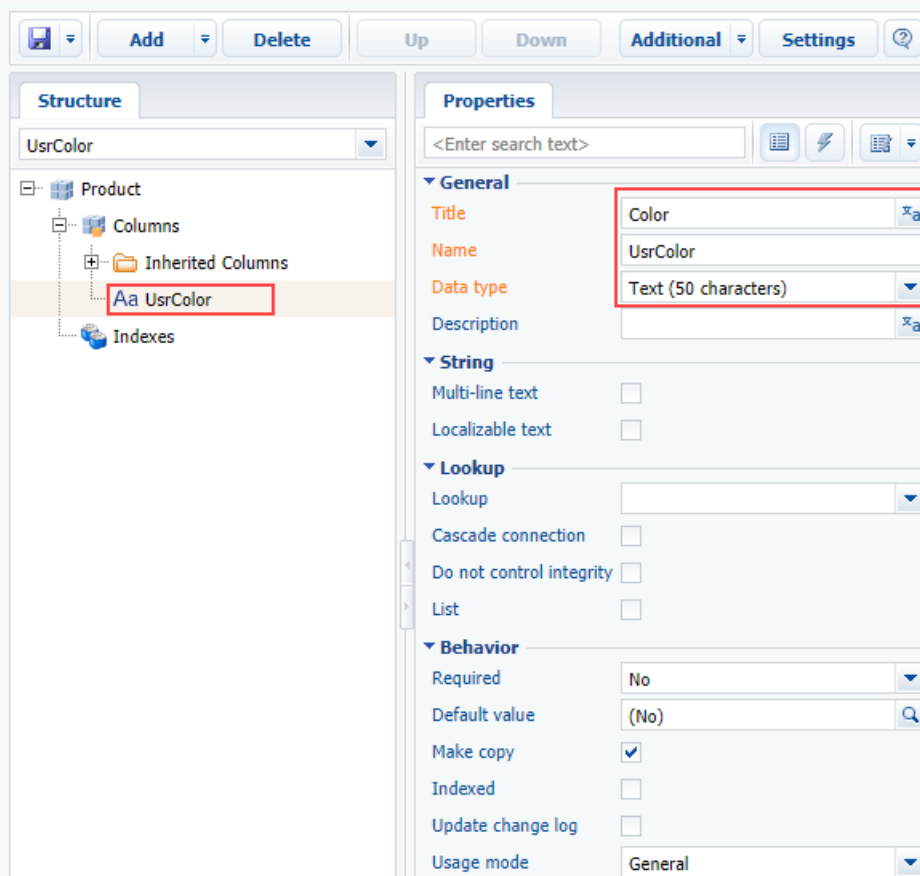


Рис. 4. — Свойства добавленной колонки

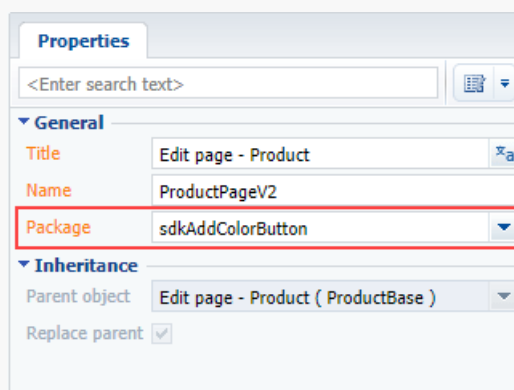


После установки всех свойств сохраните и опубликуйте схему объекта.

2. В пользовательском пакете создать схему замещающей модели представления страницы продукта

Создайте схему замещающей модели представления, в которой в качестве родительского объекта укажите схему [Страница редактирования продукта] ([*Edit page - Product*], *ProductPageV2*) (рис. 5). Процесс создания схемы замещающей модели представления описан в статье "[Создать клиентскую схему](#)".

Рис. 5. — Свойства схему замещающей модели представления страницы продукта

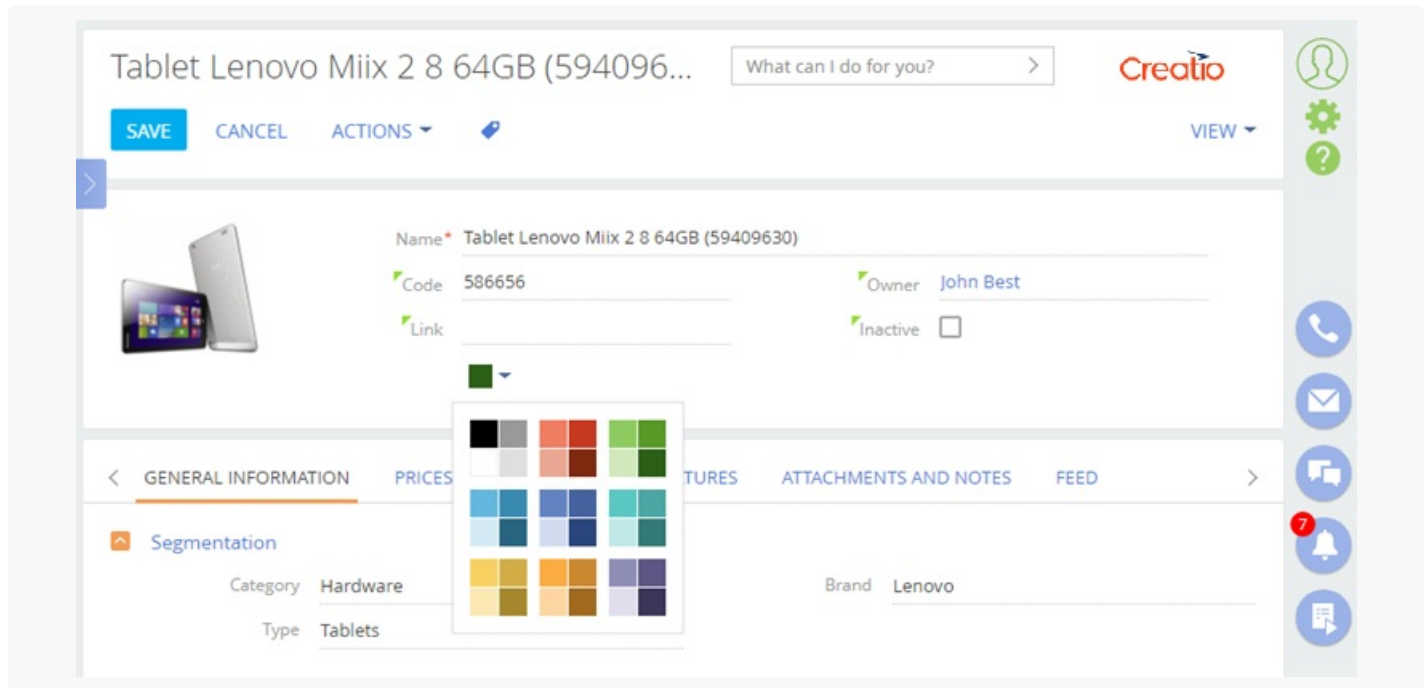


Исходный код замещающей схемы:

```
define("ProductPageV2", [], function() {
    return {
        // Название схемы объекта страницы записи.
        entitySchemaName: "Product",
        diff: /**SCHEMA_DIFF*/[
            // Кнопка выбора цвета.
            {
                // Операция добавления.
                "operation": "insert",
                // Мета-имя родительского контейнера, в который добавляется компонент.
                "parentName": "ProductGeneralInfoBlock",
                // Кнопка добавляется в коллекцию компонентов
                // родительского контейнера.
                "propertyName": "items",
                // Мета-имя компонента схемы, над которым производится действие.
                "name": "ColorButton",
                // Свойства, передаваемые в конструктор компонента.
                "values": {
                    // Тип элемента – кнопка выбора цвета.
                    "itemType": this.Terrasoft.ViewItemType.COLOR_BUTTON,
                    // Привязка значения элемента управления к колонке модели представления.
                    "value": { "bindTo": "UsrColor" },
                    // Расположение кнопки.
                    "layout": { "column": 5, "row": 6, "colSpan": 12 }
                }
            }
        ]/**SCHEMA_DIFF*/
    };
});
```

После сохранения схемы и обновления страницы приложения на странице продукта появится кнопка выбора цвета (рис. 6).

Рис. 6. — Результат выполнения примера



Добавить к кнопке всплывающую подсказку

 Сложный

Исходный код

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

Пример 1

Описание примера

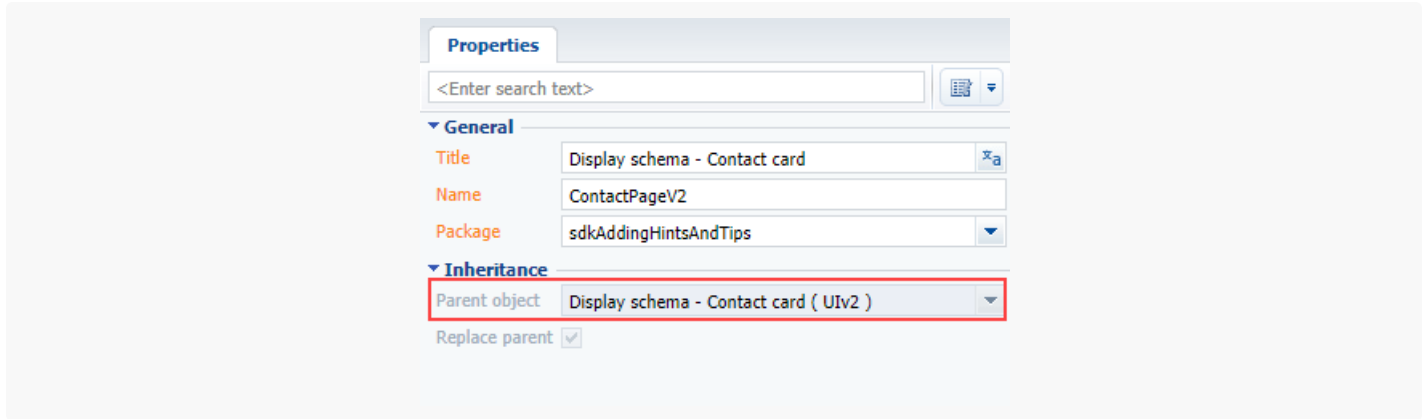
Добавить подсказку к кнопке [Сохранить] ([Save]) страницы контакта.

Алгоритм реализации примера

1. Создать схему замещающей модели представления страницы контакта

В пользовательском пакете создайте схему замещающей модели представления, в котором в качестве родительского объекта укажите [Схема отображения карточки контакта] ([Display schema — Contact card], ContactPageV2) (рис. 1). Процесс создания схемы замещающей модели представления описан в статье "[Создать клиентскую схему](#)".

Рис. 1. — Свойства схему замещающей модели представления страницы контакта

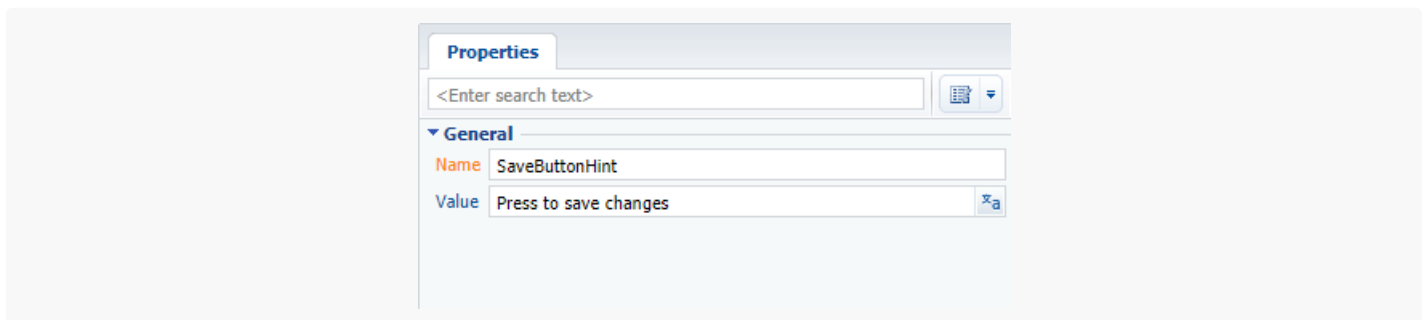


2. Добавить локализуемую строку с текстом подсказки

В коллекцию локализуемых строк схему замещающей модели представления страницы записи добавьте локализуемую строку с текстом подсказки. Свойства для созданной строки (рис. 2):

- [Название] ([Name]) — "SaveButtonHint";
- [Значение] ([Value]) — "Нажмите, чтобы сохранить изменения" ("Press to save changes").

Рис. 2. — Свойства пользовательской локализуемой строки



3. Добавить в массив diff конфигурационный объект кнопки

Существует несколько способов добавления всплывающей подсказки к элементу управления.

Способ 1

Добавьте в свойство `values` элемента управления свойство `hint`, которое, в свою очередь, должно содержать текст всплывающей подсказки.

Исходный код схемы замещающей модели представления страницы контакта при добавлении подсказки первым способом:

```
define("ContactPageV2", [],
function () {
    return {
        // Название схемы объекта страницы записи.
        entitySchemaName: "Contact",
        //Настройка визуализации всплывающей подсказки.
```

```
diff: /**SCHEMA_DIFF*/[
    // Метаданные для добавления к кнопке всплывающей подсказки.
    {
        // Выполняется операция изменения существующего элемента.
        "operation": "merge",
        "parentName": "LeftContainer",
        "propertyName": "items",
        "name": "SaveButton",
        "values": {
            // Всплывающая подсказка для кнопки.
            "hint": { "bindTo": "Resources.Strings.SaveButtonHint" }
        }
    }
]/**SCHEMA_DIFF*/
};
});
```

Способ 2

Добавьте в свойство `values` элемента управления массив `tips`. Затем, используя операцию `insert`, добавьте в массив `tips` конфигурационный объект подсказки. В свойстве `values` этого объекта обязательно укажите свойство `content` — текст всплывающей подсказки. При таком способе можно более индивидуально настроить всплывающую подсказку — изменить стиль отображения, привязать видимость подсказки к какому-либо событию модели представления, добавить элементы управления и т.д.

Важно.

Указанный способ работает для `itemType`:

- `Terrasoft.ViewItemType.BUTTON`,
- `Terrasoft.ViewItemType.LABEL`,
- `Terrasoft.ViewItemType.COLOR_BUTTON`,
- `Terrasoft.ViewItemType.HYPERLINK`,
- `Terrasoft.ViewItemType.INFORMATION_BUTTON`,
- для элементов, у которых указано свойство `generator`.

Исходный код схемы замещающей модели представления страницы контакта при добавлении подсказки вторым способом:

```
define("ContactPageV2", [],
function () {
    return {
        // Название схемы объекта страницы записи.
```



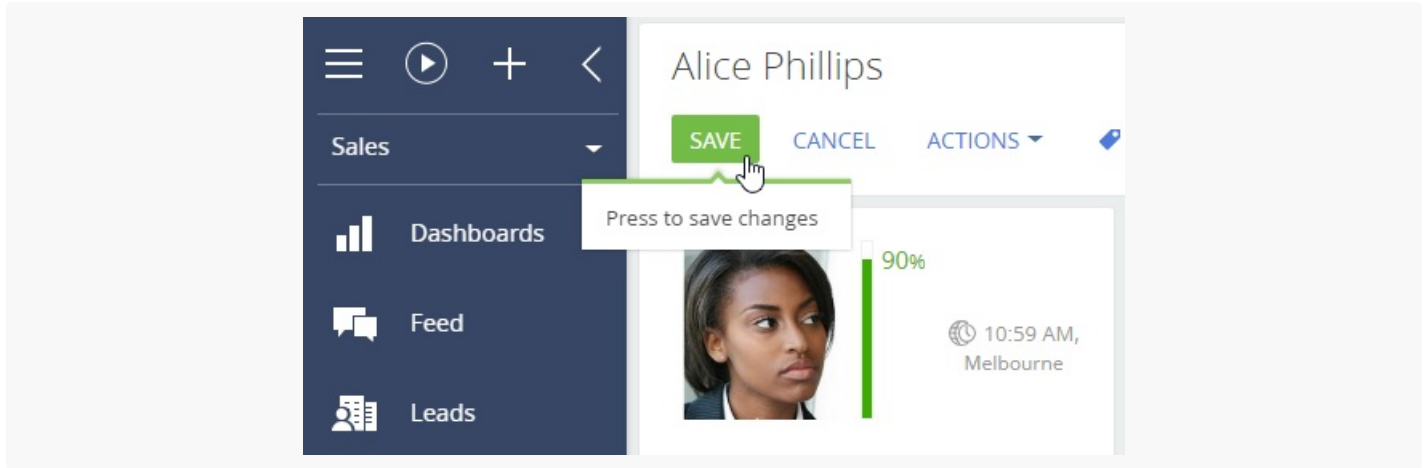
```

entitySchemaName: "Contact",
//Настройка визуализации всплывающей подсказки.
diff: /**SCHEMA_DIFF*/[
    // Метаданные для добавления к кнопке всплывающей подсказки.
    {
        // Выполняется операция изменения существующего элемента.
        "operation": "merge",
        "parentName": "LeftContainer",
        "propertyName": "items",
        "name": "SaveButton",
        "values": {
            // Массив подсказок для кнопки.
            "tips": []
        }
    },
    // Конфигурационный объект простой подсказки.
    {
        // Выполняется операция добавления нового элемента.
        "operation": "insert",
        "parentName": "SaveButton",
        "propertyName": "tips",
        "name": "CustomShowedTip",
        "values": {
            // Текст подсказки.
            "content": {"bindTo": "Resources.Strings.SaveButtonHint"}
            // Здесь можно дополнительно настроить другие параметры отображения и
            // работы всплывающей подсказки.
        }
    },
]/**SCHEMA_DIFF*/
};
});

```

После сохранения схемы на странице контакта у кнопки [*Сохранить*] ([*Save*]) появиться всплывающая подсказка (рис. 3).

Рис. 3. — Демонстрация результата



Пример 2

Описание примера

Добавить подсказку к полю [*Тип*] страницы контакта.

Алгоритм реализации примера

1. Создать схему замещающей модели представления страницы контакта

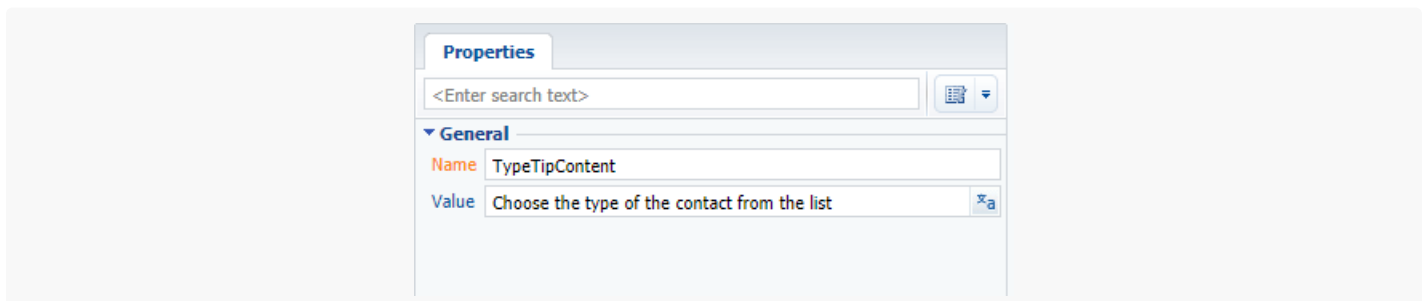
В пользовательском пакете создайте схему замещающей модели представления, в которой в качестве родительского объекта укажите [*Схема отображения карточки контакта*] ([*Display schema — Contact card*], `ContactPageV2`) (рис. 1).

2. Добавить локализуемую строку с текстом подсказки

В коллекцию локализуемых строк схемы замещающей модели представления страницы записи добавьте локализуемую строку с текстом подсказки. Свойства для созданной строки (рис. 4):

- [Название] ([Name]) — "TypeTipContent";
- [Значение] ([Value]) — "Выберите тип контакта из списка" ("Choose the type of contact from the list").

Рис. 4. — Свойства пользовательской локализуемой строки



3. Добавить в массив diff конфигурационный объект поля

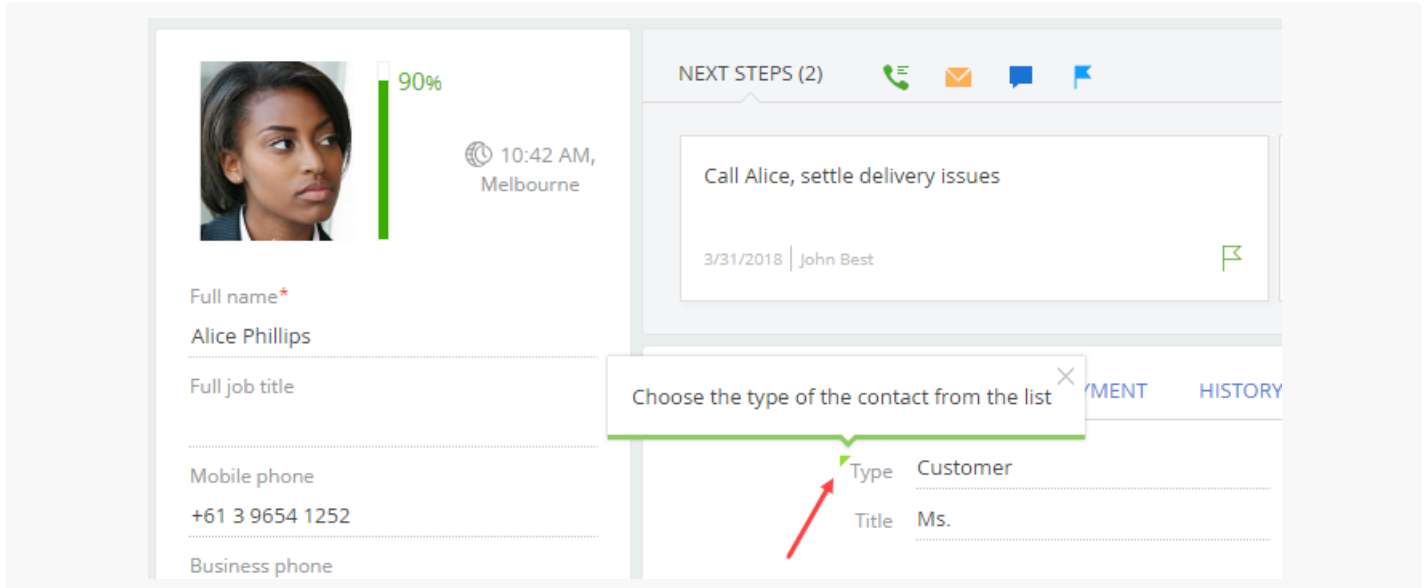
Для добавления к полю всплывающей подсказки добавьте в свойство `values` поля свойство `tip`, которое, в свою очередь, должно содержать свойство `content`. Значение свойства `content` будет являться текстом всплывающей подсказки.

Ниже приведен исходный код схемы замещающей модели представления страницы.

```
define("ContactPageV2", [],
function () {
    return {
        // Название схемы объекта страницы записи.
        entitySchemaName: "Contact",
        //Настройка визуализации всплывающей подсказки.
        diff: /**SCHEMA_DIFF*/[
            // Метаданные для добавления к полю всплывающей подсказки.
            {
                // Выполняется операция изменения существующего элемента.
                "operation": "merge",
                "name": "Type",
                "parentName": "ContactGeneralInfoBlock",
                "propertyName": "items",
                "values": {
                    // Свойство поля, отвечающее за отображение подсказки.
                    "tip": {
                        // Текст подсказки.
                        "content": { "bindTo": "Resources.Strings.TypeTipContent" },
                        // Режим отображения подсказки.
                        // По умолчанию режим WIDE - толщина зеленой полосы,
                        // которая отображается в подсказке.
                        "displayMode": Terrasoft.controls.TipEnums.displayMode.WIDE
                    }
                }
            }
        ]/**SCHEMA_DIFF*/
    };
});
```

После сохранения схемы на странице контакта поле [*Тип*] ([*Type*]) будет отображаться со всплывающей подсказкой (рис. 5).

Рис. 5. — Демонстрация результата



Пример 3

Описание примера

Добавить информационную кнопку к полю [ФИО] ([Full name]) страницы контакта.

Алгоритм реализации примера

1. Создать схему замещающей модели представления страницы контакта

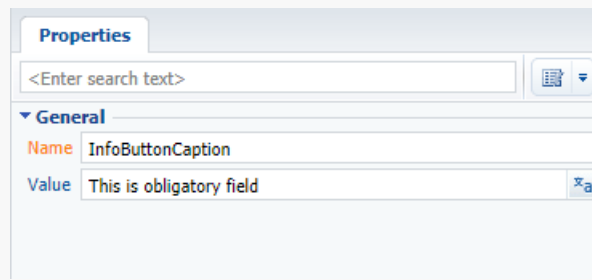
В пользовательском пакете создайте схему замещающей модели представления, в которой в качестве родительского объекта укажите [Схема отображения карточки контакта] ([Display schema — Contact card], `ContactPageV2`) (рис. 1).

2. Добавить локализуемую строку с текстом подсказки

В коллекцию локализуемых строк схему замещающей модели представления страницы записи добавьте локализуемую строку с текстом подсказки. Свойства для созданной строки (рис. 6):

- [Название] ([Name]) — "InfoButtonCaption";
- [Значение] ([Value]) — "Это обязательное поле" ("This is obligatory field").

Рис. 6. — Свойства пользовательской локализуемой строки



3. Добавить в массив diff конфигурационный объект кнопки

Добавьте в массив `diff` новый элемент с типом `Terrasoft.ViewItemType.INFORMATION_BUTTON` и свойством `content`. Значение свойства `content` является текстом всплывающей подсказки.

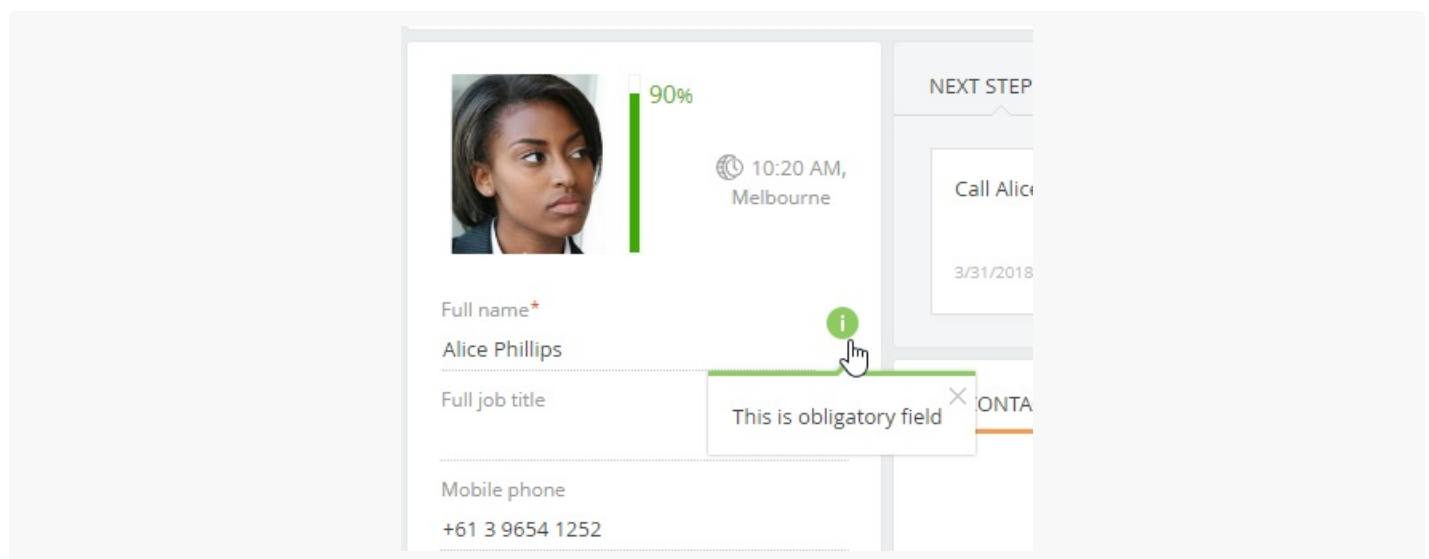
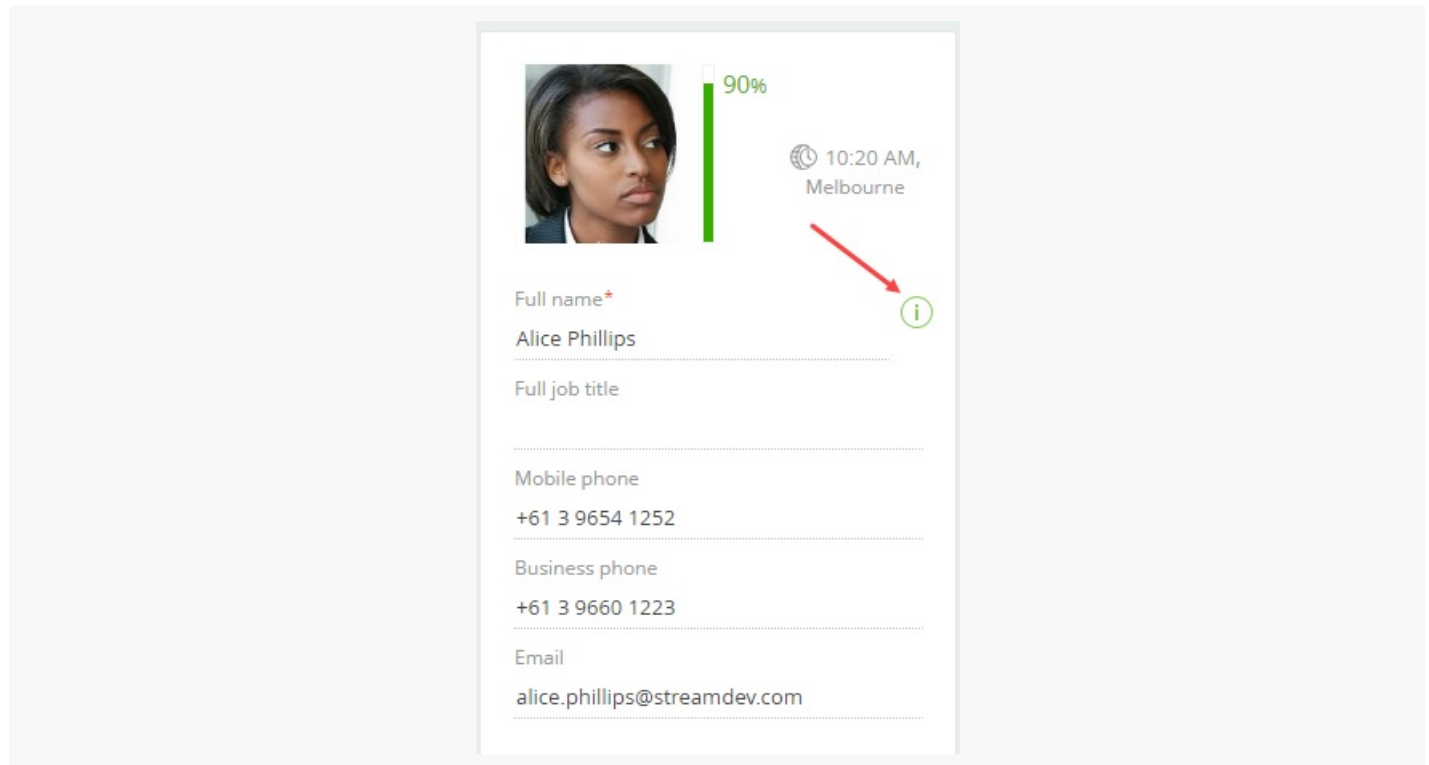
Исходный код схем замещающей модели представления страницы записи:

```
define("ContactPageV2", [],
function () {
    return {
        // Название схемы объекта страницы записи.
        entitySchemaName: "Contact",
        //Настройка визуализации всплывающей подсказки.
        diff: /**SCHEMA_DIFF*/[
            // Метаданные для добавления к кнопке всплывающей подсказки.
            {
                // Выполняется операция изменения существующего элемента.
                "operation": "merge",
                "parentName": "ProfileContainer",
                "propertyName": "items",
                "name": "AccountName",
                "values": {
                    "layout": { "column": 0, "row": 1, "colSpan": 22, "rowSpan": 1 }
                }
            },
            {
                // Выполняется операция добавления нового элемента.
                "operation": "insert",
                "parentName": "ProfileContainer",
                "propertyName": "items",
                "name": "SimpleInfoButton",
                "values": {
                    "layout": { "column": 22, "row": 1, "colSpan": 1, "rowSpan": 1 },
                    "itemType": Terrasoft.ViewItemType.INFORMATION_BUTTON,
                    "content": { "bindTo": "Resources.Strings.InfoButtonCaption" }
                }
            }
        ]/**SCHEMA_DIFF*/
    };
});
```

});

После сохранения схемы на странице контакта поле [*Контрагент*] будет отображаться со всплывающей подсказкой (рис. 7).

Рис. 7. — Демонстрация результата

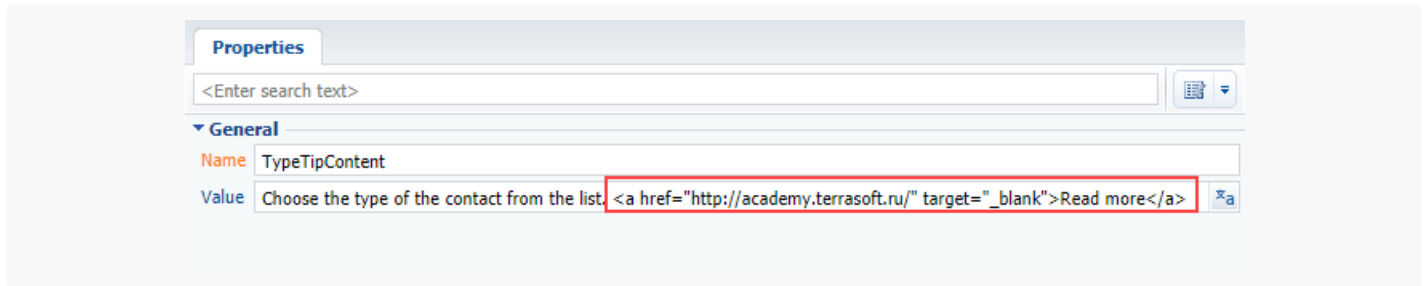


Пример 4. Добавление во всплывающую подсказку ссылки на веб-ресурс

В подсказки можно добавлять ссылки на веб-ресурсы или контекстную справку. Для этого добавьте

html-код ссылки непосредственно в локализуемую строку текста подсказки (рис. 8).

Рис. 8. — Пример определения всплывающей подсказки со ссылкой

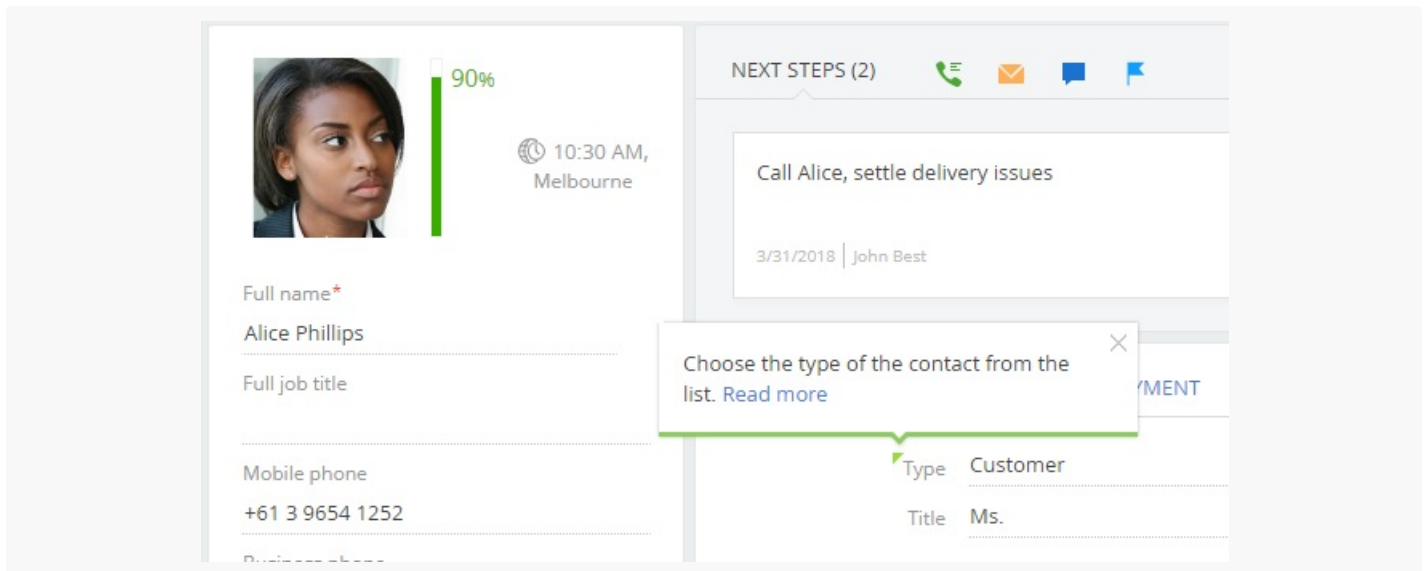


Пример добавления прямой ссылки на веб-ресурс:

```
<a href="http://academy.terrasoft.ru/" target="_blank">Узнать больше</a>
```

В результате всплывающая подсказка будет выглядеть как на рисунке 9.

Рис. 9. — Пример отображения всплывающей подсказки со ссылкой



Свойство diff объекта кнопки JS

 Легкий

Для настройки визуального расположения пользовательской кнопки на странице записи, в массив `diff` модели представления необходимо добавить конфигурационный объект.

Свойства

operation

Тип операции с элементом управления (`insert` , `move` , `remove` , `merge` , `set`). Задается строкой с названием соответствующей операции. Для добавления нового элемента управления указывается

значение `insert` .

parentName

Мета-имя родительского элемента управления, в который помещается пользовательский элемент. Если это функциональная кнопка, то в качестве родительских контейнеров могут выступать `LeftContainer` и `RightContainer` .

propertyName

Для пользовательского элемента управления указывается значение `items` .

name

Мета-имя добавляемого элемента управления.

values

Конфигурационный объект с настройками дополнительных свойств элемента управления.

[Свойства конфигурационного объекта](#)

itemType	Тип элемента. Задается значением перечисления <code>Terrasoft.ViewItemType</code> . Для кнопки используется значение <code>BUTTON</code> .
caption	Заголовок кнопки. Рекомендуется задавать значения заголовков через привязку к локализуемой строке схемы.
click	Привязка метода-обработчика кнопки.
layout	Объект настроек расположения элемента управления в сетке.
enabled	Регулирует доступность (активность) кнопки.
visible	Регулирует видимость кнопки.
style	Стиль компонента. Свойство должно содержать значение перечисления <code>Terrasoft.controls.ButtonEnums.style</code>. Возможные значения (<code>Terrasoft.controls.ButtonEnums.style</code>) DEFAULT Стиль по умолчанию. GREEN Цвет кнопки — зеленый. RED Цвет кнопки — красный. BLUE Цвет кнопки — синий. GREY Прозрачная кнопка. Значение оставлено из предыдущих версий Creatio. TRANSPARENT Прозрачная кнопка.

Поле



Добавление поля на страницу записи может быть реализовано двумя способами:

1. С помощью мастера разделов (см. статьи "[Мастер разделов](#)", "[Как настроить поля страницы](#)").

В результате работы мастера разделов в пользовательском пакете будут созданы схема замещающего базового объекта (например, объекта [*Активность*]) и схема замещающей модели представления базовой страницы записи (например, `ActivityPageV2`). В схему замещающей модели представления будет добавлена новая колонка. В схему замещающей модели представления страницы записи в массив `diff` будет добавлен конфигурационный объект с настройками расположения на странице нового поля.

На заметку.

Если с помощью мастера создается новый раздел, то в текущем пользовательском пакете будут созданы не замещающие, а новые схемы.

Дополнительную бизнес-логику страницы записи или разработку новых элементов клиентского интерфейса необходимо реализовывать в созданной схеме замещающей модели представления страницы записи.

2. Путем создания схемы замещающего базового объекта и схемы замещающей модели представления страницы средствами разработки.

Типы полей

Поле с изображением

Особенности добавления на страницу записи поля с изображением (фото контакта, товара, логотип контрагента и т.д.):

1. Колонка объекта, которая используется для поля с изображением, должна иметь тип — "Ссылка на изображение". При этом ее следует указать в качестве системной колонки [*Изображение*] объекта.
2. В коллекцию изображений схемы страницы записи необходимо добавить изображение по умолчанию.
3. В массив `diff` схемы страницы записи поле с изображением добавляется с использованием вспомогательного контейнера-обертки с CSS-классом `image-edit-container`.
4. Конфигурационный объект с настройками поля с изображением в свойстве `values` должен содержать свойства:
 - `getSrcMethod` — метод, который получает изображение по ссылке;
 - `onPhotoChange` — метод, который вызывается при изменении изображения;
 - `readonly` — свойство, которое определяет возможность редактирования (изменения, удаления) изображения;
 - `generator` — генератор элемента управления. Для поля с изображением необходимо указать

```
ImageCustomGeneratorV2.generateCustomImageControl ;
```

- `beforeFileSelected` — метод, который вызывается перед вызовом диалогового окна выбора изображения.

5. В коллекцию методов схемы страницы записи необходимо добавить:

- метод, который получает изображение по ссылке;
- метод, который вызывается при изменении изображения;
- метод, который сохраняет ссылку на измененное изображение в колонке объекта;
- метод, который вызывается перед вызовом диалогового окна выбора изображения.

Вычисляемое поле

Вычисляемое поле — это элемент управления страницы, значение которого формируется в зависимости от состояния или значений других элементов этой страницы.

В Creatio работа вычисляемых полей основана на базовом клиентском механизме Creatio — подписке на события изменения атрибутов схемы модели представления страницы. Так, для любого атрибута можно задать конфигурационный объект, в котором указать имена колонок схемы объекта, при изменении которых должно изменяться значение данной колонки, а также указать имя метода-обработчика события.

Общий алгоритм добавления вычисляемого поля

1. В схему объекта страницы добавить колонку для хранения значения вычисляемого поля.
2. В модели представления страницы настроить зависимость атрибута, указав имена колонок, от которых он зависит, и задать имя обработчика.
3. В коллекцию методов модели представления добавить реализацию метода-обработчика.
4. Настроить визуальное отображение вычисляемого поля в свойстве `diff` модели представления.

Настройка зависимостей вычисляемого поля

В свойство `attributes` модели представления необходимо добавить атрибут, для которого устанавливается зависимость.

Для этого атрибута объявить свойство `dependencies`, которое представляет собой массив конфигурационных объектов, каждый из которых содержит в себе такие свойства:

- `columns` — массив имен колонок, от значений которых зависит значение текущей колонки;
- `methodName` — имя метода-обработчика.

Если в модели представления изменится значение хотя бы одной из перечисленных колонок, будет вызван метод-обработчик события, имя которого указывается в свойстве `methodName`.

Реализация метода обработчика должна быть добавлена в коллекцию методов модели представления.

Мультивалютное поле

Одной из распространенных задач конфигурирования системы является добавление на страницу элемента управления [*Мультивалютное поле*]. Этот элемент предоставляет пользователю возможность не просто вводить денежную сумму, но и указывать ее валюту. Также он позволяет фиксировать эквивалент суммы в базовой валюте, которая задана в настройках системы. При изменении валюты существующая сумма автоматически пересчитывается с учетом обменных курсов валют. На рисунке 1 представлено мультивалютное поле в интерфейсе системы.

Рис. 1. — Мультивалютное поле

Для добавления мультивалютного поля на страницу записи необходимо:

1. Добавить в схему объекта четыре колонки:

- колонку — справочник валюты;
- колонку курса валюты;
- колонку для хранения общей суммы в выбранной валюте;
- колонку для хранения суммы в базовой валюте.

На заметку.

В схеме объекта может быть определена только одна колонка — для хранения общей суммы в выбранной валюте. Остальные колонки могут являться виртуальными, если бизнес-задача не предполагает хранения в базе данных их значений. Они могут быть определены как атрибуты в схеме модели представления.

2. Указать в объявлении класса модели представления в качестве зависимостей три модуля:

- `MoneyModule` ,
- `MultiCurrencyEdit` ,
- `MultiCurrencyEditUtilities` .

3. Подключить к модели представления миксин `Terrasoft.MultiCurrencyEditUtilities` и инициализировать его в переопределенном методе `init()` .

4. Добавить конфигурационный объект с настройками мультивалютного поля в массив `diff` схемы модели представления страницы записи. В нем в свойстве `values` кроме общих для всех элементов управления свойств должны содержаться свойства:

- `primaryAmount` — наименование колонки, которая содержит сумму в базовой валюте;

- `currency` — наименование колонки, которая ссылается на справочник валют;
- `rate` — наименование колонки, которая содержит курс валюты;
- `generator` — генератор элемента управления. Для мультивалютного поля необходимо указать `MultiCurrencyEditViewGenerator.generate`.

5. Добавить логику пересчета суммы в зависимости от валюты. Для этого применить механизм вычисляемых полей, подробно описанный в статье [Добавить вычисляемые поля](#).

Действия над полями

Добавить валидацию к полю

Валидация — это проверка значений заполненных полей на соответствие определенным требованиям. Валидация значений полей страницы Creatio реализуется на уровне колонок моделей представления страниц. Логика проверки значения поля выполняется в пользовательском методе-валидаторе.

Валидатор — это метод модели представления, в котором выполняется анализ значения колонки модели представления на соответствие бизнес-требованиям. Особенностью этого метода является то, что он должен возвращать объект с результатами валидации, который имеет свойство:

- `invalidMessage` — строка с сообщением, которое будет отображено под полем при попытке ввести в него некорректное значение и в информационном окне при попытке сохранить страницу с полем, которое не прошло валидацию.

Если валидация значения прошла успешно, то метод-валидатор возвращает объект с пустой строкой.

Чтобы валидация поля начала работать, нужно выполнить привязку соответствующей колонки модели представления конкретному валидатору. Для этого в модели представления необходимо переопределить базовый метод `setValidationConfig()`, в котором вызвать метод `addColumnValidator()`.

Метод `addColumnValidator()` принимает два параметра:

- имя колонки модели представления, к которой привязывается валидатор;
- имя метода-валидатора значения колонки.

Важно.

Если валидация поля реализуется в схеме замещающей модели представления базовой страницы, то для того, чтобы были корректно проинициализированы валидаторы полей базовой страницы, перед вызовом метода `addColumnValidator()` необходимо добавить вызов родительской реализации метода `setValidationConfig()`.

Последовательность действий для добавления проверки корректности значения поля:

1. Добавить в коллекцию методов модели представления метод-валидатор, проверяющий значение поля.
2. Переопределить метод `setValidationConfig()`, в котором выполнить привязку валидатора к соответствующей колонке модели представления.

Установить для поля значение по умолчанию

В Creatio реализована возможность определения значений по умолчанию для элементов управления страницы записи.

Установить значение по умолчанию можно двумя способами:

1. Задать на уровне колонок бизнес-объектов. При создании нового объекта некоторые поля страницы должны заполняться определенными, известными изначально значениями. В таких случаях для колонок объекта, которые соответствуют этим полям, необходимо в дизайнера объектов указать эти значения в качестве значений по умолчанию.

Виды значений по умолчанию:

Название	Описание
<i>Установить константу</i>	Строка, число, значение из справочника, логическое значение.
<i>Установить из системной настройки</i>	Полный список настроек доступен из конфигурации в разделе [<i>Системные настройки</i>], и он может быть дополнен пользовательской системной настройкой.
<i>Установить из системной переменной</i>	Системные переменные Creatio — это глобальные переменные, хранящие информацию о настройках системы. В отличие от системных настроек, значения которых могут отличаться для разных пользователей, значения системных переменных являются одинаковыми для всех пользователей. Полный перечень системных переменных реализован на уровне ядра и не может быть изменен пользователем: • Новый идентификатор; • Новый последовательный идентификатор; • Текущий пользователь; • Контакт текущего пользователя; • Контрагент текущего пользователя; • Текущее значение даты и времени; • Текущее значение даты; • Текущее значение времени.
<i>Не устанавливать значение по умолчанию</i>	

2. Задать в исходном коде страницы записи. В некоторых случаях невозможно установить значение по умолчанию с помощью свойств колонки объекта. Например, это могут быть вычисляемые значения, которые рассчитываются по значениям других колонок объекта и т.д.. В таком случае, задать значение

по умолчанию можно только программными средствами.

Применить фильтрацию к справочным полям

В Creatio существует два способа применения фильтрации к справочным полям страниц записи:

1. Использование бизнес-правила [*FILTRATION*].
2. Явное указание фильтров в описании колонки в свойстве модели `attributes`.

Использование бизнес-правила [*FILTRATION*] целесообразно в случае, когда к полю необходимо применить простой фильтр — по определенному значению или атрибуту. Бизнес-правила подробно описаны в статье "[Бизнес-правила и их применение](#)". Подробный пример по использованию бизнес-правила [*FILTRATION*] приведен в статье "[Пример применения правила FILTRATION](#)".

Если необходимо применение произвольной фильтрации, сортировки и добавления дополнительных колонок в запрос при отображении выпадающего списка, используйте явное описание фильтров в свойстве модели представления `attributes`.

Настройка фильтров справочного поля в свойстве модели `attributes`:

1. В свойство `attributes` модели представления добавьте новый атрибут с именем колонки объекта, для которой устанавливаются фильтры.
2. Для этой колонки объявите свойство `lookupListConfig` — конфигурационный объект, содержащий в себе такие свойства (опционально):
 - `columns` — массив имен колонок, которые будут добавлены к запросу дополнительно к Id и первичной для отображения колонке;
 - `orders` — массив конфигурационных объектов, которые определяют сортировку данных при отображении;
 - `filter` — метод, возвращающий объект класса `Terrasoft.BaseFilter` или его наследника, который будет применен к запросу;
 - или `filters` — массив фильтров (методов, возвращающих коллекции класса `Terrasoft.FilterGroup`).

Фильтры в коллекцию добавляются с помощью метода `add()`, который имеет следующие параметры:

Название	Тип данных	Описание
key	String	Ключ.
item	Mixed	Элемент.
index	Number	Индекс для вставки. Если не указан, то не учитывается.

В качестве параметра `item` выступает объект класса `Terrasoft.BaseFilter` или его наследника. Методы создания фильтров с описаниями приведены в статье "[Работа с фильтрами](#)".

Важно.

По умолчанию фильтры в коллекции объединяются с использованием логической операции AND. Если необходимо применить операцию OR, ее нужно указать явно в свойстве `logicalOperation` объекта `Terrasoft.FilterGroup`.

Заблокировать поля

При разработке пользовательской функциональности Creatio может возникнуть необходимость в одновременной блокировке всех полей и деталей на странице при выполнении некоторого условия. Механизм блокировки полей страницы записи позволяет существенно ускорить процесс решения данной задачи, не прибегая к написанию большого количества бизнес-правил.

Подробно механизм блокировки полей страницы записи описан в статье [Механизм блокировки полей страницы записи](#).

Важно.

Механизм блокировки реализован в версиях приложения 7.11.1 и выше.

Добавить автонумерацию к полю

Вы можете добавить автонумерацию для определенного поля страницы записи. Например, автонумерация реализована в разделах [*Документы*], [*Счета*] и [*Договоры*], где при создании записи для нее автоматически генерируется номер в заданном формате.

Для реализации задачи автонумерации возможно два варианта решения:

- реализовать автонумерацию на стороне клиента;
- реализовать автонумерацию на сервере.

Чтобы реализовать автонумерацию **на стороне клиента**, необходимо переопределить базовый виртуальный метод `onEntityInitialized()` и в нем вызвать метод `getIncrementCode()` базовой схемы страницы записи `BasePageV2`.

Метод `getIncrementCode()` принимает два параметра:

- `callback` — функция, которая будет вызвана при получении ответа от сервиса. Ответ необходимо передать в соответствующую колонку (атрибут);
- `scope` — контекст, в котором будет вызвана функция `callback` (необязательный параметр).

Для реализации автонумерации **на стороне сервера** дополнительно к объекту, к колонке которого будет применяться автонумерация, необходимо добавить обработчик события [*Перед добавлением записи*]. В бизнес-процессе необходимо установить параметры генерации номера, а именно:

- Указать схему объекта, для которого будет осуществляться генерация.
- Вызвать действие [Сгенерировать номер по порядку].
- Передать сгенерированное значение в необходимую колонку объекта.

На заметку.

Это не единственный вариант реализации автонумерации на стороне сервера. Ее можно реализовать пользовательскими методами, например, создав [пользовательский сервис](#).

Для использования механизма автонумерации, независимо от выбранного варианта решения, необходимо добавить две системные настройки:

- `[Entity]CodeMask` — маска номера объекта,
- `[Entity]LastNumber` — текущий номер объекта,

где `Entity` — это наименование того объекта, к колонке которого будет применяться автонумерация. Например, `InvoiceCodeMask` (Маска номера счета) и `InvoiceLastNumber` (Текущий номер счета).

Добавить новое поле

 Средний

Исходный код

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

Пример 1

Описание примера

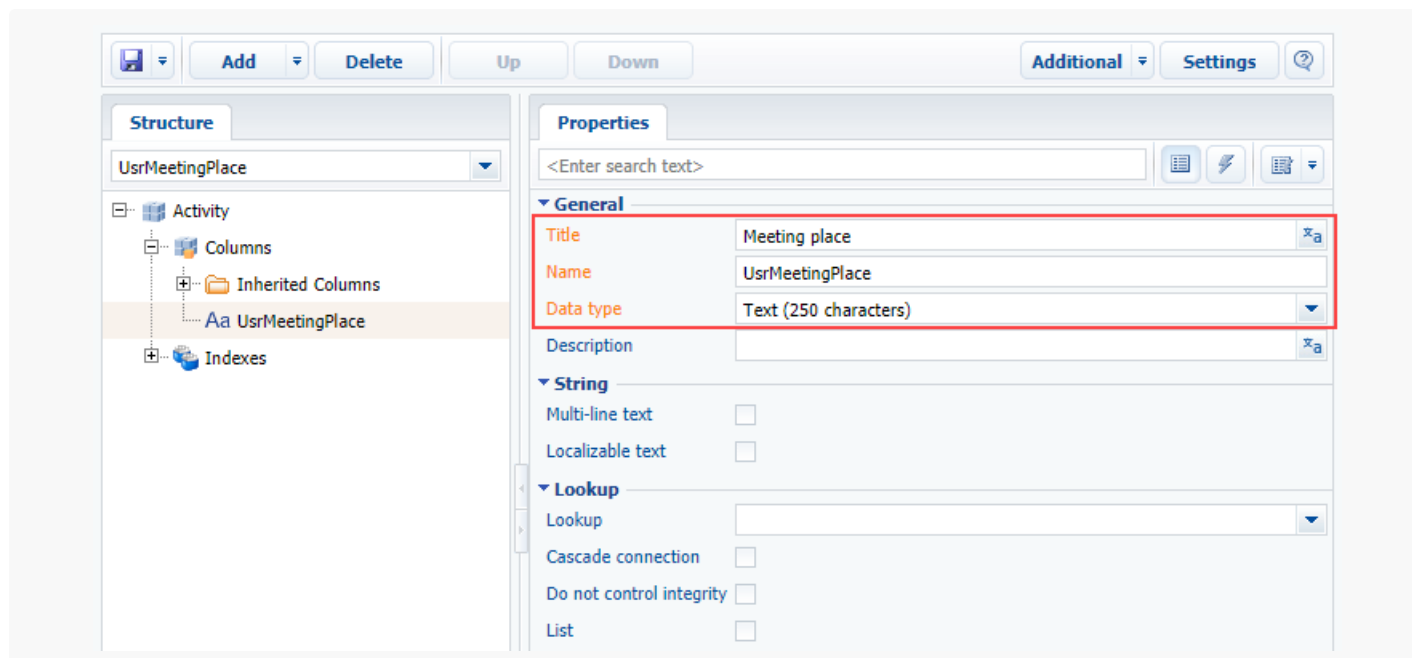
Добавить вручную на страницу активности новое поле [*Место встречи*].

Алгоритм реализации примера

1. Создать схему замещающего объекта и добавить в него новую колонку

Создайте схему замещающего объекта [*Активность*] ([*Activity*]) и добавьте в него новую колонку [*Место встречи*] ([*Meeting place*]) типа "строка" (рис. 1). Процесс создания схему замещающего объекта описан в статье "[Создать схему объекта](#)".

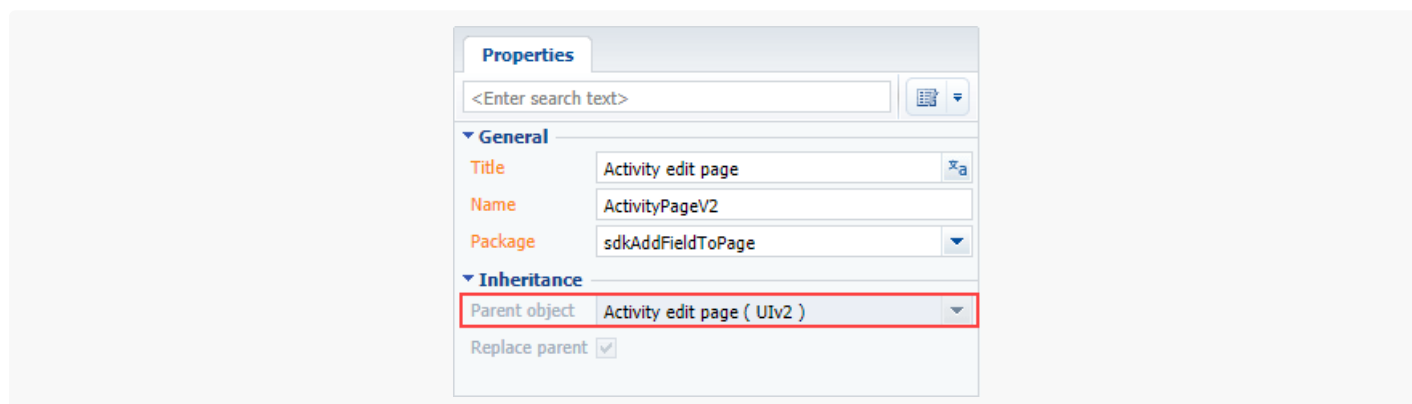
Рис. 1. — Добавление пользовательской колонки в схему замещающего объекта



2. Создать схему замещающей модели представления страницы активности

Создайте схему замещающей модели представления, в которой в качестве родительского объекта укажите схему [*Страница редактирования активности*] ([*Activity edit page*]) (рис. 2). Процесс создания схемы замещающей модели представления страницы описан в статье "[Создать клиентскую схему](#)".

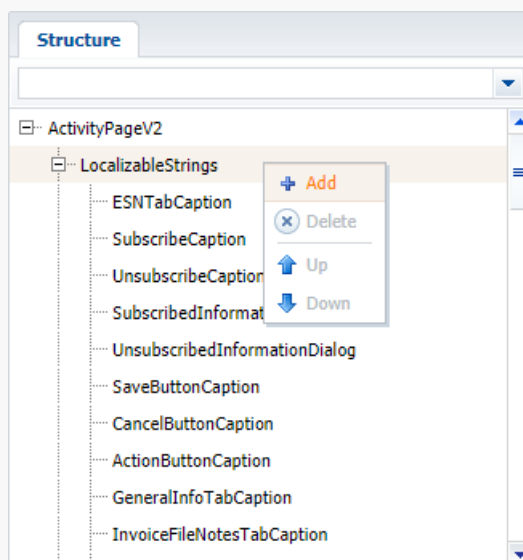
Рис. 2. — Свойства схемы замещающей модели представления страницы записи



3. Добавить локализуемую строку с заголовком поля

В коллекцию локализуемых строк замещающей схемы страницы добавьте строку, содержащую заголовок добавляемого поля (рис. 3).

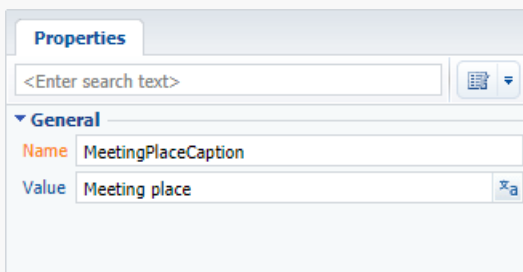
Рис. 3. — Добавление в схему локализуемой строки



Для созданной строки укажите (рис. 4):

- [*Название*] ([*Name*]) — "MeetingPlaceCaption";
- [*Значение*] ([*Value*]) — "Место встречи" ("Meeting place").

Рис. 4. — Свойства пользовательской локализуемой строки



4. Добавить новое поле на страницу активности

Добавьте в массив `diff` конфигурационный объект с настройками расположения поля [*Место встречи*] ([*Meeting place*]) на странице.

Свойства массива `diff` подробно описаны в статье "[Массив diff](#)".

Исходный код схемы замещающей модели представления:

```
define("ActivityPageV2", [], function() {
    return {
        // Название схемы объекта страницы записи.
        entitySchemaName: "Activity",
        // Отображение нового поля на странице записи.
        diff: /**SCHEMA_DIFF*/[
            // Метаданные для добавления поля [Место встречи].
            {
```

```

// Операция добавления компонента на страницу.
"operation": "insert",
// Мета-имя родительского контейнера, в который добавляется поле.
"parentName": "Header",
// Поле добавляется в коллекцию компонентов
// родительского элемента.
"propertyName": "items",
// Имя колонки схемы, к которой привязан компонент.
"name": "UsrMeetingPlace",
"values": {
  // Заголовок поля.
  "caption": {"bindTo": "Resources.Strings.MeetingPlaceCaption"},
  // Расположение поля.
  "layout": {
    // Номер столбца.
    "column": 0,
    // Номер строки.
    "row": 5,
    // Диапазон занимаемых столбцов.
    "colSpan": 12
  }
}
}
]/**SCHEMA_DIFF*/
});
});

```

После сохранения схемы и обновления страницы приложения с очисткой кэша на странице активности появится новое поле (рис. 5).

Рис. 5. — Демонстрация результата выполнения примера

The screenshot shows the 'Prepare quotation' form in the Creatio application. The form has a header with the title 'Prepare quotation', a search bar 'What can I do for you?', and the Creatio logo. Below the header, there are buttons for 'CLOSE', 'ACTIONS', and 'VIEW'. The main content area contains several fields for scheduling and assignment:

- Subject**: Prepare quotation
- Start**: 1/4/2018, 4:00 PM
- Due**: 1/4/2018, 6:00 PM
- Status**: Completed
- Owner**: Murphy Valerie
- Reporter**: John Best
- Priority**: Medium
- Category**: Paper work
- Show in calendar**: ☒
- Meeting place**: (highlighted with a red rectangle)

Пример 2

Описание примера

Добавить вручную в профиль контакта на странице контакта поле [Страна]. Отличие данного примера в том, что колонка [Страна] уже присутствует в схеме объекта.

Алгоритм реализации примера

1. Создать схему замещающей модели представления страницы контакта

Создайте схему замещающей модели представления, в которой в качестве родительского объекта укажите схему [Схема отображения карточки контакта] ([Display schema - Contact card], `ContactPageV2`). Процесс создания схемы замещающей модели представления страницы описан в статье "[Создать клиентскую схему](#)".

2. Добавить поле [Страна] в профиль контакта

Добавьте в массив `diff` конфигурационный объект с настройками свойства поля. При этом в качестве родительского элемента схемы, в котором будет размещено поле, укажите элемент `ProfileContainer`.

Исходный код схем замещающей модели представления:

```
define("ContactPageV2", [], function() {
    return {
        // Название схемы объекта страницы записи.
        entitySchemaName: "Contact",
        diff: [
            // Метаданные для добавления поля [Страна].
            {
                // Операция добавления компонента на страницу.
                "operation": "insert",
                // Мета-имя родительского контейнера, в который добавляется поле.
                "parentName": "ProfileContainer",
                // Поле добавляется в коллекцию компонентов
                // родительского элемента.
                "propertyName": "items",
                // Имя колонки схемы, к которой привязан компонент.
                "name": "Country",
                "values": {
                    // Тип поля – справочник.
                    "contentType": Terrasoft.ContentType.LOOKUP,
                    // Расположение поля.
                    "layout": {
                        // Номер столбца.
                        "column": 0,
                        // Номер строки.
```

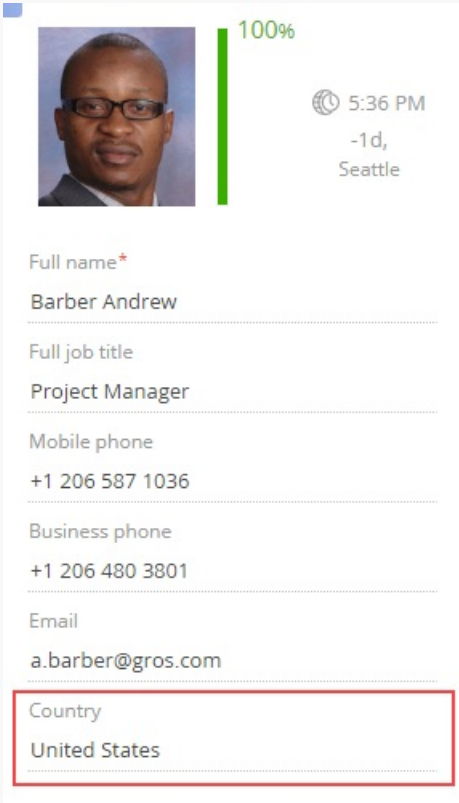
```

        "row": 6,
        // Диапазон занимаемых столбцов.
        "colSpan": 24
      }
    }
  ]
};
});

```

После сохранения схемы и обновления страницы приложения в профиле контакта на странице контакта появится новое поле (рис. 6).

Рис. 6. — Демонстрация результата выполнения примера



100%

5:36 PM
-1d,
Seattle

Full name*

Barber Andrew

Full job title

Project Manager

Mobile phone

+1 206 587 1036

Business phone

+1 206 480 3801

Email

a.barber@gros.com

Country

United States

Добавить поле с изображением

 Сложный

Описание примера

Добавить поле с логотипом на страницу статьи базы знаний.

Исходный код

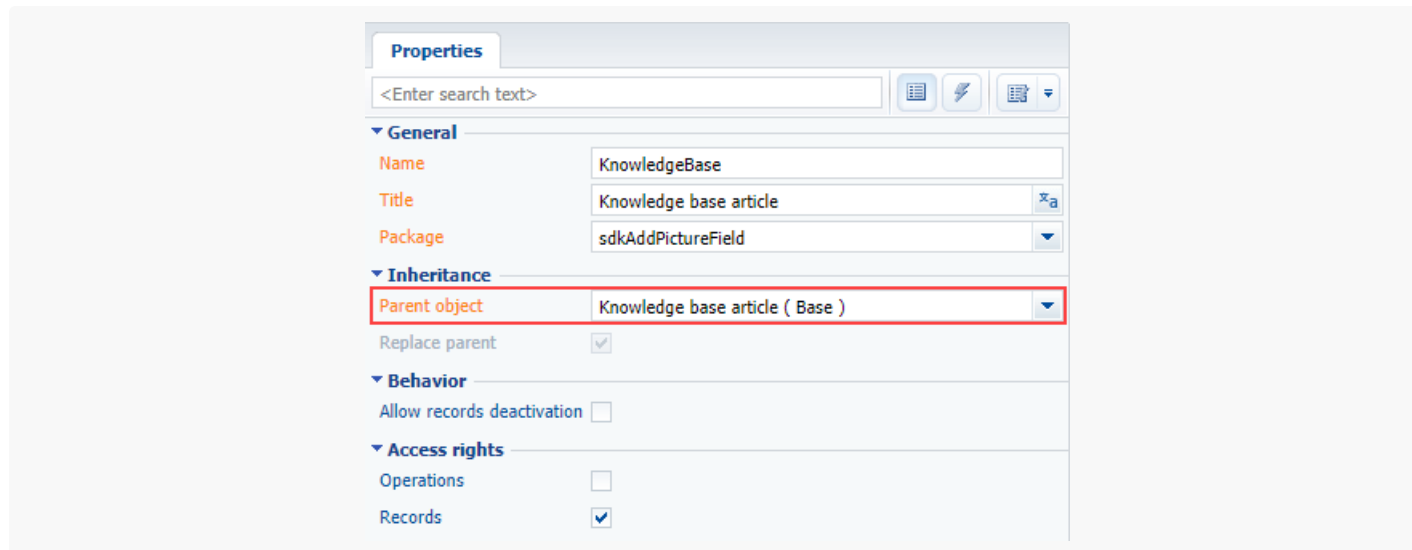
Пакет с реализацией примера можно скачать по [ссылке](#).

Алгоритм реализации примера

1. Создать схему замещающего объекта [*Статья базы знаний*]

Создайте схему замещающего объекта [*Статья базы знаний*] ([*Knowledge base article*]) (рис. 1). Процесс создания схемы замещающего объекта описан в статье "[Создать схему объекта](#)".

Рис. 1. — Свойства схемы замещающего объекта



2. Добавить в схему замещающего объекта новую колонку

Для созданной колонки укажите (рис. 2):

- [*Заголовок*] ([*Title*]) — "Логотип статьи базы знаний" ("Knowledge base article logo");
- [*Название*] ([*Name*]) — "UsrLogo";
- [*Тип данных*] ([*Data type*]) — "Ссылка на изображение" ("Image Link").

Рис. 2. — Добавление пользовательской колонки в схему замещающего объекта

Properties

<Enter search text>

General

Title Knowledge base article logo

Name UsrLogo

Data type Image Link

Description

String

Multi-line text ☐

Localizable text ☐

Lookup

Lookup Image

Cascade connection ☐

Do not control integrity ☐

List ☐

Behavior

Required No

Default value (No)

Make copy ☒

Indexed ☒

Update change log ☐

Usage mode General

Обязательно укажите созданную колонку в качестве системной колонки [Изображение] ([Image]) объекта (рис. 3).

Рис. 3. — Установка созданной колонки в качестве системной

Properties

<Enter search text>

General

Name: KnowledgeBase

Title: Knowledge base article

Change Log Object Name:

Permission Object Name:

Localization Object Name:

Description:

Package: sdkAddPictureField

Inheritance

Behavior

Access rights

System Columns

Id: Id

Displayed value: Name

Image: Knowledge base article logo

Sorting in Lists:

Parent in Hierarchy:

Owner:

History Columns

На заметку. Для отображения всех свойств объекта перейдите в режим расширенного отображения свойств объекта.

После установки всех свойств сохраните и опубликуйте схему замещающего объекта.

3. Создать схему замещающей модели представления страницы записи

Создайте схему замещающей модели представления, в которой в качестве родительского объекта укажите схему [*Knowledge base edit page*] (KnowledgeBasePageV2) (рис. 4). Процесс создания схему замещающей модели представления страницы описан в статье "[Создать клиентскую схему](#)".

Рис. 4. — Свойства схему замещающей модели представления страницы записи

Properties

<Enter search text>

General

Title: Knowledge base edit page

Name: KnowledgeBasePageV2

Package: sdkAddPictureField

Inheritance

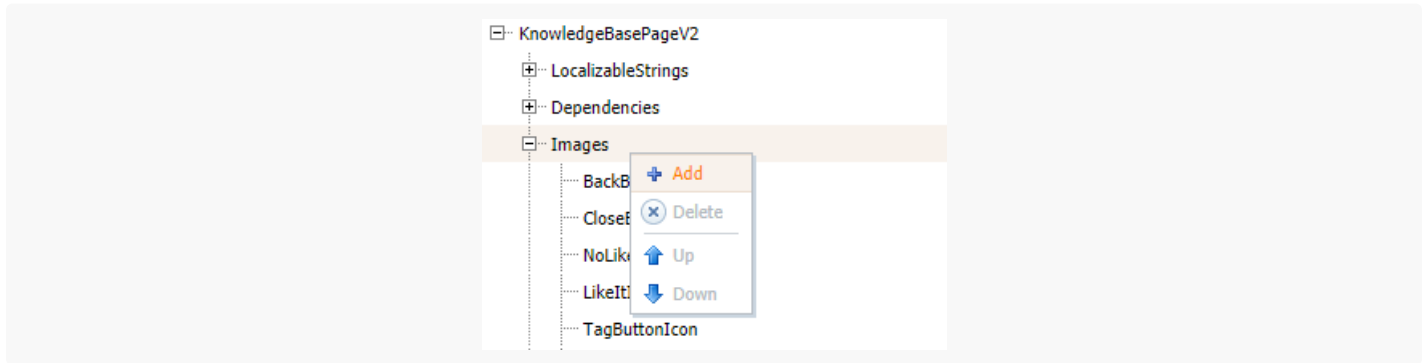
Parent object: Knowledge base edit page (UIv2)

Replace parent: ☒

4. Добавить изображение по умолчанию в ресурсы [*Images*] схемы страницы записи

В коллекцию изображений схемы замещающей модели представления страницы добавьте изображение по умолчанию (рис. 5).

Рис. 5. — Добавление в ресурсы схемы изображения по умолчанию



Для созданного изображения укажите (рис. 6):

- [*Название*] ([*Name*]) — "DefaultLogo";
- [*Изображение*] ([*Image*]) — файл с изображением по умолчанию (рис. 7).

Рис. 6. — Свойства ресурса схемы

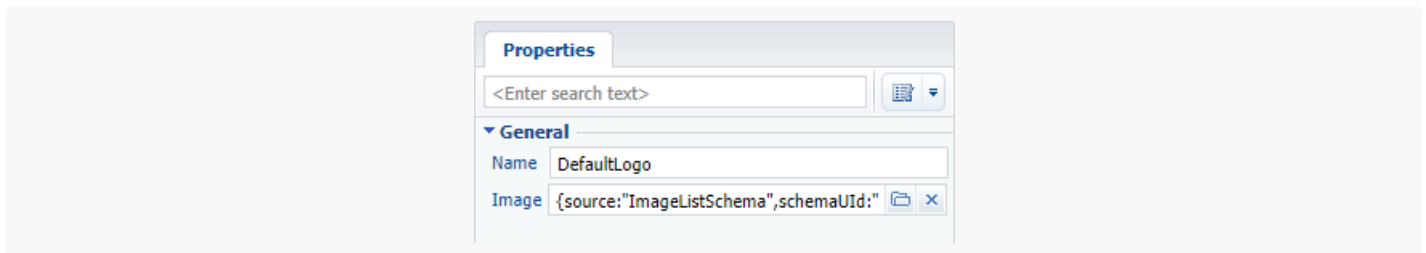


Рис. 7. — Изображение по умолчанию для статьи базы знаний



5. Настроить визуальное отображение поля с логотипом на странице записи

Поле с логотипом должно быть размещено в верхней части страницы контрагента. В базовой реализации данной схемы поля расположены таким образом, что добавление логотипа может нарушить интерфейс страницы. Поэтому, кроме размещения нового поля, необходимо дополнительно поменять расположение существующих полей.

В свойстве `diff` модели представления (см. исходный код ниже) в массив добавьте конфигурационный объект поля с логотипом с необходимыми параметрами, а также опишите модификации для полей верхней части страницы: [*Name*], [*ModifiedBy*] и [*Type*].

Поле с изображением добавляется на страницу с использованием вспомогательного контейнера-обертки `PhotoContainer` с классом [*"image-edit-container"*].

6. В коллекцию методов модели представления страницы добавить реализацию методов:

- `getPhotoSrcMethod()` — получает изображение по ссылке;
- `beforePhotoFileSelected()` — вызывается перед открытием диалогового окна выбора изображения;
- `onPhotoChange()` — вызывается при изменении изображения;
- `onPhotoUploaded()` — сохраняет ссылку на измененное изображение в колонке объекта.

Исходный код схемы замещающей модели представления:

```
define("KnowledgeBasePageV2", ["KnowledgeBasePageV2Resources", "ConfigurationConstants"],
function(resources, ConfigurationConstants) {
    return {
        // Название схемы объекта страницы записи.
        entitySchemaName: "KnowledgeBase",
        // Методы модели представления страницы записи.
        methods: {
            // Вызывается перед открытием диалогового окна выбора изображения.
            beforePhotoFileSelected: function() {
                return true;
            },
            // Получает изображение по ссылке.
            getPhotoSrcMethod: function() {
                // Получение ссылки на изображение из колонки объекта.
                var imageColumnValue = this.get("UsrLogo");
                // Если ссылка установлена, то метод возвращает url файла с изображением.
                if (imageColumnValue) {
                    return this.getSchemaImageUrl(imageColumnValue);
                }
                // Если ссылка не установлена, то возвращает изображение по умолчанию.
                return this.Terrasoft.ImageUrlBuilder.getUrl(this.get("Resources.Images.DefaultImage"));
            },
            // Обрабатывает изменение изображения.
            // photo — файл с изображением.
            onPhotoChange: function(photo) {
                if (!photo) {
                    this.set("UsrLogo", null);
                    return;
                }
                // Выполняется загрузка файла в базу данных. По окончании загрузки вызывается
```

```

        this.Terrasoft.ImageApi.upload({
            file: photo,
            onComplete: this.onPhotoUploaded,
            onError: this.Terrasoft.emptyFn,
            scope: this
        });
    },
    // Сохраняет ссылку на измененное изображение.
    // imageId – Id сохраненного файла из базы данных.
    onPhotoUploaded: function(imageId) {
        var imageData = {
            value: imageId,
            displayValue: "Image"
        };
        // Колонке изображения присваивается ссылка на изображение.
        this.set("UsrLogo", imageData);
    }
},
//
diff: /**SCHEMA_DIFF*/[
    // Контейнер-обертка, в который будет размещен компонент.
    {
        // Операция добавления.
        "operation": "insert",
        // Мета-имя родительского контейнера, в который добавляется компонент.
        "parentName": "Header",
        // Изображение добавляется в коллекцию компонентов
        // родительского контейнера.
        "propertyName": "items",
        // Мета-имя компонента схемы, над которым производится действие.
        "name": "PhotoContainer",
        // Свойства, передаваемые в конструктор компонента.
        "values": {
            // Тип элемента – контейнер.
            "itemType": Terrasoft.ViewItemType.CONTAINER,
            // Имя CSS класса.
            "wrapClass": ["image-edit-container"],
            // Размещение в родительском контейнере.
            "layout": { "column": 0, "row": 0, "rowSpan": 3, "colSpan": 3 },
            // Массив дочерних элементов.
            "items": []
        }
    },
    // Поле [UsrLogo] – поле с логотипом контрагента.
    {
        "operation": "insert",
        "parentName": "PhotoContainer",
        "propertyName": "items",

```

```

    "name": "UsrLogo",
    "values": {
        // Метод, который получает изображение по ссылке.
        "getSrcMethod": "getPhotoSrcMethod",
        // Метод, который вызывается при изменении изображения.
        "onPhotoChange": "onPhotoChange",
        // Метод, который вызывается перед вызовом диалогового окна выбора изобра
        "beforeFileSelected": "beforePhotoFileSelected",
        // Свойство, которое определяет возможность редактирования (изменения, у
        "readonly": false,
        // View-генератор элемента управления.
        "generator": "ImageCustomGeneratorV2.generateCustomImageControl"
    }
},
// Изменение расположения поля [Name].
{
    // Операция слияния.
    "operation": "merge",
    "name": "Name",
    "parentName": "Header",
    "propertyName": "items",
    "values": {
        "bindTo": "Name",
        "layout": {
            "column": 3,
            "row": 0,
            "colSpan": 20
        }
    }
},
// Изменение расположения поля [ModifiedBy].
{
    "operation": "merge",
    "name": "ModifiedBy",
    "parentName": "Header",
    "propertyName": "items",
    "values": {
        "bindTo": "ModifiedBy",
        "layout": {
            "column": 3,
            "row": 2,
            "colSpan": 20
        }
    }
},
// Изменение расположения поля [Type].
{
    "operation": "merge",
    "name": "Type",

```

```

        "parentName": "Header",
        "propertyName": "items",
        "values": {
            "bindTo": "Type",
            "layout": {
                "column": 3,
                "row": 1,
                "colSpan": 20
            },
            "contentType": Terrasoft.ContentType.ENUM
        }
    }
}
]/**SCHEMA_DIFF*/
});
});

```

После сохранения схемы и обновления страницы приложения на странице статьи базы знаний появится изображение логотипа по умолчанию (рис. 8). При наведении курсора мыши на изображение отобразится меню действий, с помощью которого можно удалить изображение либо установить новое для определенной статьи базы знаний (рис. 9).

Рис. 8. — Логотип по умолчанию

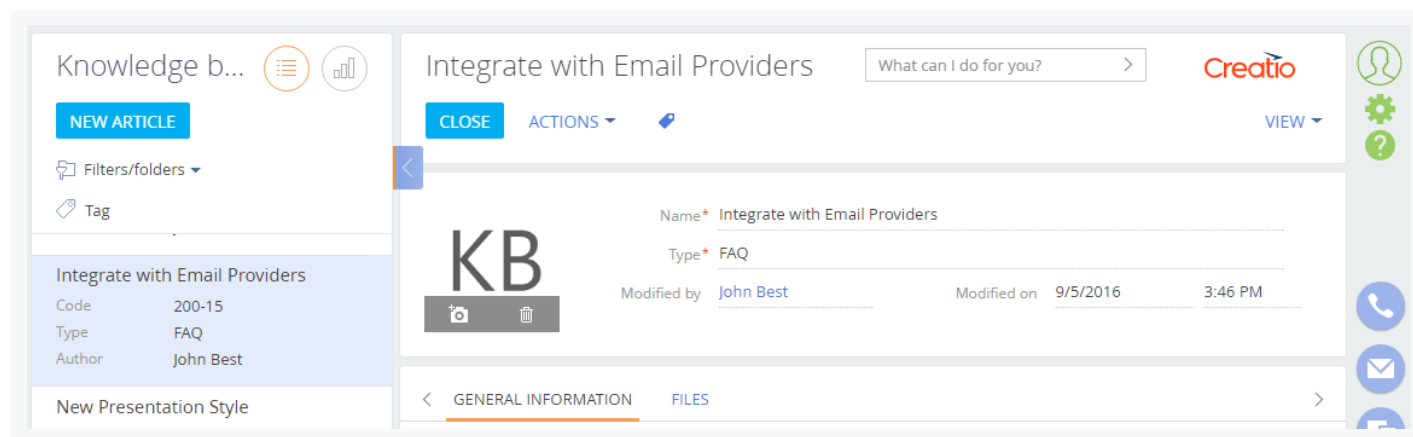
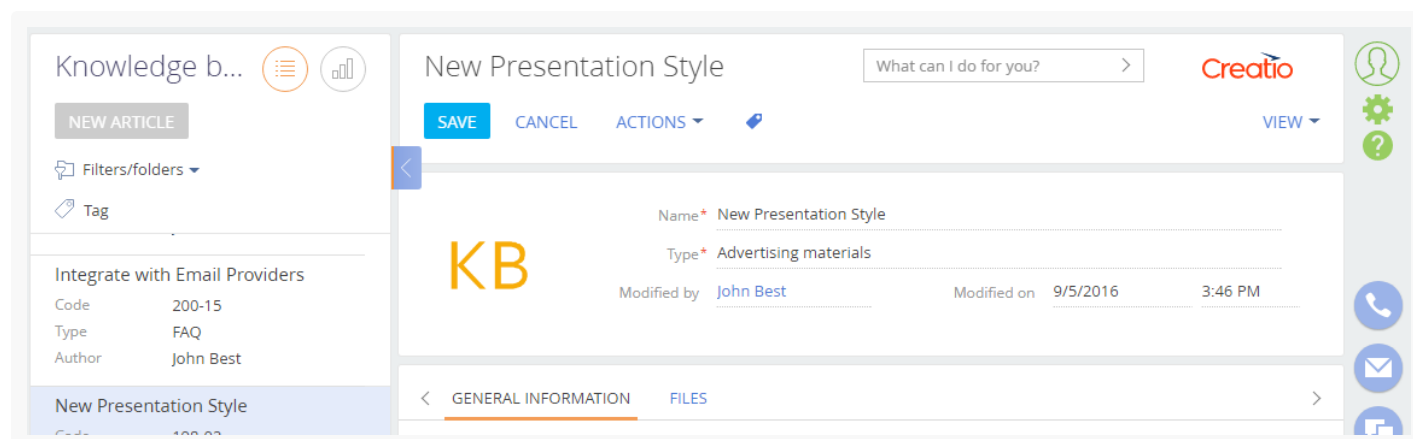


Рис. 9. — Пользовательский логотип



Добавить вычисляемые поля



Описание примера

Добавить на страницу заказа поле [*Остаток для оплаты*] ([*Payment balance*]), в котором будет отображаться разница между суммой заказа и суммой оплаты.

На заметку.

Добавление поля на страницу записи может быть реализовано двумя способами — с помощью мастера разделов и вручную. Процесс добавления поля на страницу записи описан в статье "[Добавление нового поля](#)".

Исходный код

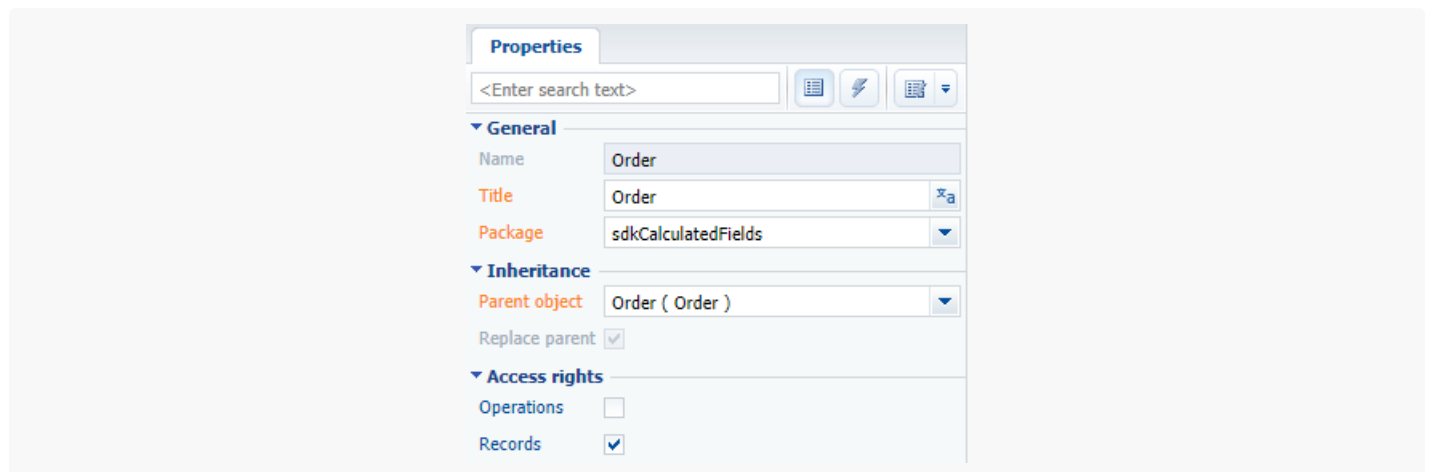
Пакет с реализацией примера можно скачать по [ссылке](#).

Алгоритм реализации примера

1. Создать схему замещающего объекта

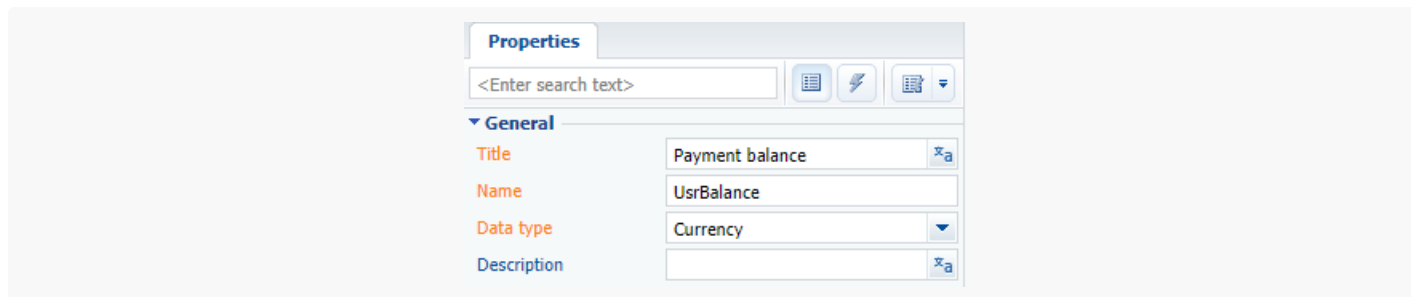
Выберите пользовательский пакет и на вкладке [*Схемы*] ([*Schemas*]) выполните пункт меню [*Добавить*] ([*Add*]) — [*Замещающий объект*] ([*Replacing Object*]). Для нового объекта укажите в качестве родительского объект [*Заказ*] ([*Order*]) (рис. 1).

Рис. 1. — Свойства схемы замещающего объекта [*Заказ*]



Добавьте в схему замещающего объекта новую колонку [*Остаток для оплаты*] ([*Payment balance*]) с типом данных [*Деньги*] (рис. 2).

Рис. 2. — Добавление пользовательской колонки в схему замещающего объекта

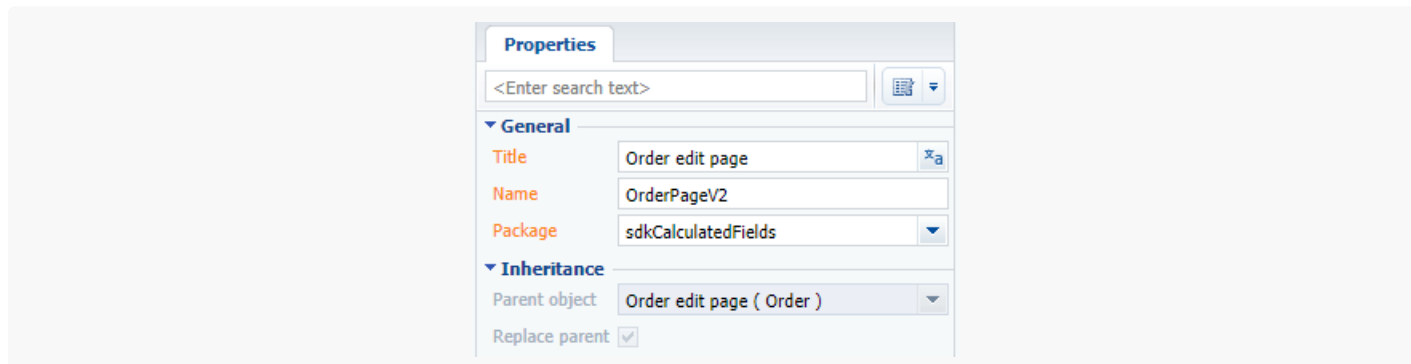


Опубликуйте объект.

2. Создать схему замещающей модели представления страницы заказа

Создайте схему замещающей модели представления, в которой в качестве родительского объекта укажите схему [Страница редактирования заказа] ([*Order Edit Page*], `OrderPageV2`) (рис. 3). Процесс создания схемы замещающей модели представления описан в статье "[Разработка конфигурационных элементов](#)".

Рис. 3. — Свойства схемы замещающей модели представления страницы записи



3. Настроить визуальное отображение поля [*Остаток для оплаты*]

В свойстве `diff` модели представления опишите конфигурационный объект с необходимыми параметрами. Исходный код схемы страницы представлен ниже.

4. В схему модели представления добавить атрибут `UsrBalance`

В исходном коде схемы модели представления страницы добавьте атрибут `UsrBalance` в коллекцию атрибутов свойства `attributes`. В конфигурационном объекте атрибута `UsrBalance` нужно указать зависимость от колонок [*Amount*] и [*PaymentAmount*], а также метод-обработчик — `calculateBalance()`, который будет рассчитывать значение для колонки [*UsrBalance*].

5. В коллекцию `methods` модели представления добавить необходимые методы

В коллекцию `methods` модели представления добавьте метод-обработчик события изменения колонок

[Amount] и [PaymentAmount] — `calculateBalance()`. Этот метод используется в атрибуте `UsrBalance`.

Важно.

В методе-обработчике нужно учитывать тип данных, которые должны вернуться в расчетное поле в результате. Например, тип данных [*Дробное число (0.01)*] предполагает число с двумя знаками после запятой. В таком случае перед записью результата в поле объекта необходимо преобразовать его с помощью функции `toFixed()`. Код примера в таком случае мог бы выглядеть следующим образом:

```
// Расчет разницы между значениями в колонках [Amount] и [PaymentAmount].
var result = amount - paymentAmount;
// Результат расчета присваивается в качестве значения колонке [UsrBalance].
this.set("UsrBalance", result.toFixed(2));
```

Переопределите базовый виртуальный метод `onEntityInitialized()`. Метод `onEntityInitialized()` срабатывает после окончания инициализации схемы объекта страницы записи. Добавление в данный метод вызов метода-обработчика `calculateBalance()`, обеспечит расчет суммы остатка для оплаты в момент открытия страницы записи, а не только после изменений в колонках-зависимостях.

Полностью исходный код модуля представлен ниже.

```
define("OrderPageV2", [], function() {
  return {
    // Название схемы объекта страницы записи.
    entitySchemaName: "Order",
    details: /**SCHEMA_DETAILS*/{}/**SCHEMA_DETAILS*/,
    // Свойство attributes модели представления.
    attributes: {
      // Название атрибута модели представления.
      "UsrBalance": {
        // Тип данных колонки модели представления.
        dataValueType: Terrasoft.DataValueType.FLOAT,
        // Массив конфигурационных объектов, определяющих зависимости колонки [UsrBalance]
        dependencies: [
          {
            // Значение колонки [UsrBalance] зависит от значений колонок [Amount]
            // и [PaymentAmount].
            columns: ["Amount", "PaymentAmount"],
            // Метод-обработчик, который вызывается при изменении значения одной из
            // и [PaymentAmount].
            methodName: "calculateBalance"
          }
        ]
      }
    }
  }
})
```

```

    },
    // Коллекция методов модели представления страницы записи.
    methods: {
        // Переопределение базового метода Terrasoft.BasePageV2.onEntityInitialized, который
        // срабатывает после окончания инициализации схемы объекта страницы записи.
        onEntityInitialized: function() {
            // Вызывается родительская реализация метода.
            this.callParent(arguments);
            // Вызов метода-обработчика, который рассчитывает значение колонки [UsrBalance].
            this.calculateBalance();
        },
        // Метод-обработчик, который рассчитывает значение колонки [UsrBalance].
        calculateBalance: function() {
            // Проверка, проинициализированы ли в момент открытия страницы записи
            // колонки [Amount] и [PaymentAmount]. Если нет, то для них устанавливаются нуле
            var amount = this.get("Amount");
            if (!amount) {
                amount = 0;
            }
            var paymentAmount = this.get("PaymentAmount");
            if (!paymentAmount) {
                paymentAmount = 0;
            }
            // Расчет разницы между значениями в колонках [Amount] и [PaymentAmount].
            var result = amount - paymentAmount;
            // Результат расчета присваивается в качестве значения колонке [UsrBalance].
            this.set("UsrBalance", result);
        }
    },
    // Визуальное отображение поля [Остаток для оплаты] на странице записи.
    diff: /**SCHEMA_DIFF*/[
        {
            "operation": "insert",
            "parentName": "Header",
            "propertyName": "items",
            "name": "UsrBalance",
            "values": {
                "bindTo": "UsrBalance",
                "layout": {"column": 12, "row": 2, "colSpan": 12}
            }
        }
    ]/**SCHEMA_DIFF*/
};
});

```

После сохранения схемы и обновления веб-страницы системы с очисткой кэша на странице заказа появится дополнительное поле [*Остаток для оплаты*] ([*Payment balance*]), которое будет рассчитываться в зависимости от значений в полях [*Итого*] ([*Total*]) и [*Сумма оплаты*] ([*Payment*

amount])

 (рис. 4).

Рис. 4. — Демонстрация результата выполнения кейса

ORD-1 (sample)

What can I do for you? >

Creatio

SAVE CANCEL ACTIONS >

PRINT > VIEW >

Customer * Accom (sample)

Status 1. Draft

Total, \$ 3,492.00

Payment amount, \$ 0.00

Payment balance 3,492.00

Добавить мультивалютное поле

 Сложный

Описание примера

Необходимо добавить мультивалютное поле [*Итого*] на страницу проекта.

Исходный код

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

Алгоритм реализации примера

1. Добавить в схему замещающего объекта необходимые колонки

В пользовательском пакете создайте схему замещающего объекта[*Проект*] ([*Project*]) (рис. 2).

Подробно создание схему замещающего объекта и добавление колонок описано в статье "[Добавление нового поля](#)".

Рис. 2. — Свойства схемы замещающего объекта

The screenshot shows the 'Properties' window for a column. The 'General' section contains the following fields:

- Name:** Project
- Title:** Project
- Package:** sdkMultiCurrencyEdit

The 'Inheritance' section contains the following fields:

- Parent object:** Project (Project)
- Replace parent:** ☒

В схему замещающего объекта добавьте четыре колонки, свойства которых приведены на рис. 3 — рис. 6. Свойства колонок в дизайнера объектов отображены в расширенном режиме.

Рис. 3. — Свойства колонки [*UsrCurrency*]

The screenshot shows the 'Properties' window for a column. The 'General' section contains the following fields:

- Title:** Currency
- Name:** UsrCurrency
- Data type:** Lookup
- Description:**

The 'String' section contains the following fields:

- Multi-line text:** ☐
- Localizable text:** ☐

The 'Lookup' section contains the following fields:

- Lookup:** Currency

Рис. 4. — Свойства колонки [*UsrAmount*]

The screenshot shows the 'Properties' window for a column. The 'General' section contains the following fields:

- Title:** Amount
- Name:** UsrAmount
- Data type:** Currency
- Description:**

Рис. 5. — Свойства колонки [*UsrPrimaryAmount*]

The screenshot shows the 'Properties' window for a column. The 'General' section contains the following fields:

- Title:** Amount, base currency
- Name:** UsrPrimaryAmount
- Data type:** Currency
- Description:**

Рис. 6. — Свойства колонки [*UsrCurrencyRate*]

Properties	
<Enter search text>	
▼ General	
Title	Exchange rate
Name	UsrCurrencyRate
Data type	Decimal (0.0001)
Description	

Для колонки [*UsrCurrency*] добавьте значение по умолчанию - системная настройка [*Базовая валюта*] (рис. 7).

Рис. 7. — Значение по умолчанию для колонки [*UsrCurrency*]

Properties	
<Enter search text>	
▼ General	
Title	Currency
Name	UsrCurrency
Data type	Lookup
Description	
String	
Multi-line text	☐
Localizable text	☐
Lookup	
Lookup	Currency
ascade connection	☐
do not control integrity	☐
st	☐
Behavior	
required	No
Default value	(System Setting)
Make copy	☒
Indexed	☒
Update change log	☐
Usage mode	General

Default value

☐ None

☐ Set Constant

☒ Select from System Settings

Name: Base currency

☐ Select from System Variables

Name:

OK Cancel

2. В пользовательском пакете создать схему замещающей модели представления страницы проекта

Создайте схему замещающей модели представления, в которой в качестве родительского объекта укажите схему [*Страница редактирования проекта*] ([*Project edit page*], *ProjectPageV2*) (рис. 8). Процесс создания схемы замещающей модели представления описан в статье "[Создать клиентскую схему](#)".

Рис. 8. — Свойства схему замещающей модели представления страницы [*Проекты*]

В объявлении класса модели представления в качестве зависимостей добавьте модули `MoneyModule`, `MultiCurrencyEdit`, `MultiCurrencyEditUtilities` (см. исходный код ниже).

3. Добавить необходимые атрибуты

В свойстве `attributes` схему замещающей модели представления страницы определите атрибуты `UsrCurrency`, `UsrCurrencyRate`, `UsrAmount`, `UsrPrimaryAmount`, соответствующие добавленным колонкам схемы объекта.

Мультивалютный модуль работает только с колонкой `Currency`, поэтому дополнительно создайте атрибут `Currency`, в котором объявите виртуальную колонку. Эту колонку свяжите с созданной ранее колонкой `UsrCurrency` при помощи метода-обработчика.

Для корректной работы мультивалютного модуля добавьте атрибуты `CurrencyRateList` (коллекция курсов валют) и `CurrencyButtonMenuList` (коллекция для кнопки выбора валюты) (см. исходный код ниже).

4. Подключить к модели представления миксин `Terrasoft.MultiCurrencyEditUtilities`

Объявите миксин `Terrasoft.MultiCurrencyEditUtilities` в свойстве `mixins` схемы замещающей модели представления страницы. Выполните его инициализацию в переопределенном методе `init()` схемы модели представления (см. исходный код ниже).

5. Реализовать логику пересчета суммы в зависимости от валюты.

В коллекцию `methods` схемы модели представления страницы добавьте методы-обработчики зависимостей атрибутов (см. исходный код ниже).

6. Добавить мультивалютное поле на страницу

В массив `diff` схемы замещающей модели представления страницы добавьте конфигурационный объект с настройками мультивалютного поля.

Исходный код схемы замещающей модели представления:

```
// В качестве зависимостей нужно указать модули
```

```

// MoneyModule, MultiCurrencyEdit и MultiCurrencyEditUtilities.
define("ProjectPageV2", ["MoneyModule", "MultiCurrencyEdit", "MultiCurrencyEditUtilities"],
    function(MoneyModule, MultiCurrencyEdit, MultiCurrencyEditUtilities) {
        return {
            // Название схемы объекта страницы записи.
            entitySchemaName: "Project",
            // Атрибуты модели представления.
            attributes: {
                // Валюта.
                "UsrCurrency": {
                    // Тип данных атрибута – справочник.
                    "dataValueType": this.Terrasoft.DataValueType.LOOKUP,
                    // Конфигурация справочника валют.
                    "lookupListConfig": {
                        "columns": ["Division", "Symbol"]
                    }
                },
                // Курс.
                "UsrCurrencyRate": {
                    "dataValueType": this.Terrasoft.DataValueType.FLOAT,
                    // Зависимости атрибута.
                    "dependencies": [
                        {
                            // Колонки, от которых зависит атрибут.
                            "columns": ["UsrCurrency"],
                            // Метод-обработчик.
                            "methodName": "setCurrencyRate"
                        }
                    ]
                },
                // Сумма.
                "UsrAmount": {
                    "dataValueType": this.Terrasoft.DataValueType.FLOAT,
                    "dependencies": [
                        {
                            "columns": ["UsrCurrencyRate", "UsrCurrency"],
                            "methodName": "recalculateAmount"
                        }
                    ]
                },
                // Сумма в базовой валюте.
                "UsrPrimaryAmount": {
                    "dependencies": [
                        {
                            "columns": ["UsrAmount"],
                            "methodName": "recalculatePrimaryAmount"
                        }
                    ]
                }
            },
        },
    )

```

```

// Валюта – виртуальная колонка для совместимости с модулем MultiCurrencyEditUtili
"Currency": {
    "type": this.Terrasoft.ViewModelColumnType.VIRTUAL_COLUMN,
    "dataValueType": this.Terrasoft.DataValueType.LOOKUP,
    "lookupListConfig": {
        "columns": ["Division"]
    },
    "dependencies": [
        {
            "columns": ["Currency"],
            "methodName": "onVirtualCurrencyChange"
        }
    ]
},
// Коллекция курсов валют
"CurrencyRateList": {
    dataValueType: this.Terrasoft.DataValueType.COLLECTION,
    value: this.Ext.create("Terrasoft.Collection")
},
// Коллекция для кнопки выбора валюты
"CurrencyButtonMenuList": {
    dataValueType: this.Terrasoft.DataValueType.COLLECTION,
    value: this.Ext.create("Terrasoft.BaseViewModelCollection")
}
},
// Миксины модели представления.
mixins: {
    // Миксин управления мультивалютностью на странице записи.
    MultiCurrencyEditUtilities: "Terrasoft.MultiCurrencyEditUtilities"
},
// Методы модели представления страницы.
methods: {
    // Переопределение базового метода Terrasoft.BasePageV2.onEntityInitialized().
    onEntityInitialized: function() {
        // Вызов родительской реализации метода onEntityInitialized.
        this.callParent(arguments);
        this.set("Currency", this.get("UsrCurrency"), {silent: true});
        // Инициализация миксина управления мультивалютностью.
        this.mixins.MultiCurrencyEditUtilities.init.call(this);
    },
    // Устанавливает курс валюты.
    setCurrencyRate: function() {
        //Загружает курс валют на дату начала проекта.
        MoneyModule.LoadCurrencyRate.call(this, "UsrCurrency", "UsrCurrencyRate", th
    },
    // Пересчитывает сумму.
    recalculateAmount: function() {
        var currency = this.get("UsrCurrency");

```



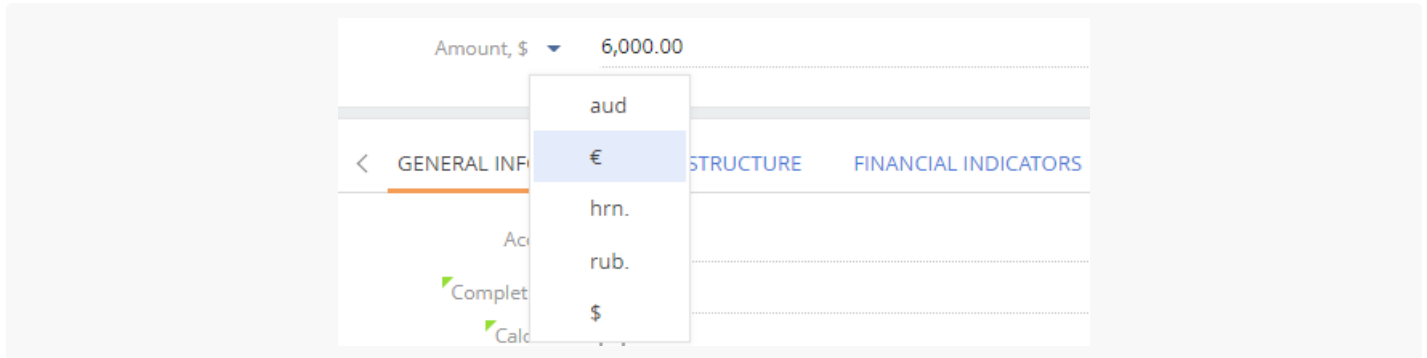
```

        var division = currency ? currency.Division : null;
        MoneyModule.RecalcCurrencyValue.call(this, "UsrCurrencyRate", "UsrAmount", "
    },
    // Пересчитывает сумму в базовой валюте.
    recalculatePrimaryAmount: function() {
        var currency = this.get("UsrCurrency");
        var division = currency ? currency.Division : null;
        MoneyModule.RecalcBaseValue.call(this, "UsrCurrencyRate", "UsrAmount", "UsrF
    },
    // Обработчик изменения виртуальной колонки валюты.
    onVirtualCurrencyChange: function() {
        var currency = this.get("Currency");
        this.set("UsrCurrency", currency);
    }
},
// Настройка визуализации мультивалютного поля на странице записи.
diff: /**SCHEMA_DIFF*/[
    // Метаданные для добавления поля [Сумма].
    {
        // Операция добавления.
        "operation": "insert",
        // Мета-имя родительского контейнера, в который добавляется компонент.
        "parentName": "Header",
        // Поле добавляется в коллекцию компонентов
        // родительского контейнера.
        "propertyName": "items",
        // Мета-имя компонента схемы, над которым производится действие.
        "name": "UsrAmount",
        // Свойства, передаваемые в конструктор компонента.
        "values": {
            // Имя колонки модели представления, к которой выполняется привязка.
            "bindTo": "UsrAmount",
            // Расположение элемента в контейнере.
            "layout": { "column": 0, "row": 2, "colSpan": 12 },
            // Наименование колонки, которая содержит сумму в базовой валюте.
            "primaryAmount": "UsrPrimaryAmount",
            // Наименование колонки, которая содержит валюту суммы.
            "currency": "UsrCurrency",
            // Наименование колонки, которая содержит курс валюты.
            "rate": "UsrCurrencyRate",
            // Свойство определяющее доступность для редактирования поля суммы в баз
            "primaryAmountEnabled": false,
            // Генератор представления элемента управления.
            "generator": "MultiCurrencyEditViewGenerator.generate"
        }
    }
]/**SCHEMA_DIFF*/
});
});

```

После сохранения схемы и обновления страницы приложения на странице проекта появится мультивалютное поле [*Итого*] ([*Amount*]) (рис. 1). Значение поля будет автоматически пересчитано после выбора валюты из выпадающего списка (рис. 9).

Рис. 9. — Выпадающий список валют



Добавить валидацию к полю

 Средний

Исходный код

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

Пример 1

Описание примера

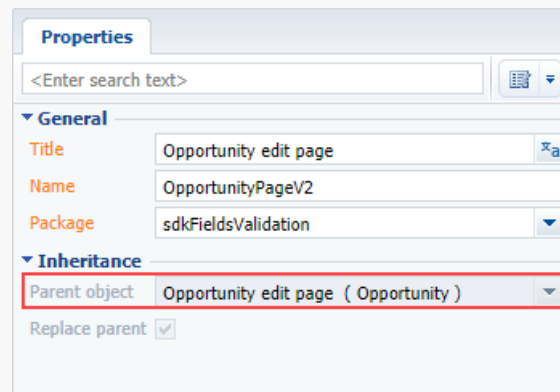
Настроить валидацию на странице продажи следующим образом: значение в поле [*Дата создания*] должно быть меньше значения в поле [*Дата закрытия*].

Алгоритм реализации примера

1. Создать схемы замещающей модели представления страницы продажи

Создайте схему замещающей модели представления, в которой в качестве родительского объекта укажите схему [*Страница редактирования продажи*] ([*Opportunity edit page*], `OpportunityPageV2`) (рис. 1). Процесс создания схемы замещающей модели представления описан в статье "[Создать клиентскую схему](#)".

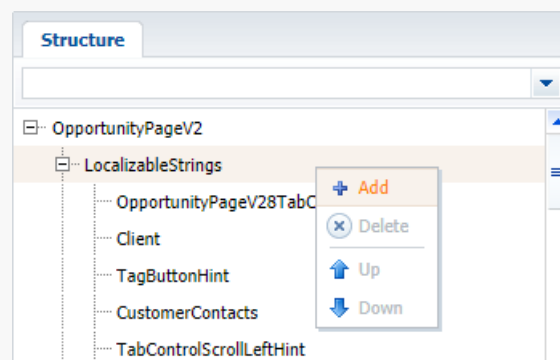
Рис. 1. — Свойства схемы замещающей модели представления страницы записи



2. В коллекцию локализуемых строк схемы замещающей модели представления страницы добавить строку ошибки

Создайте новую локализуемую строку (рис. 2).

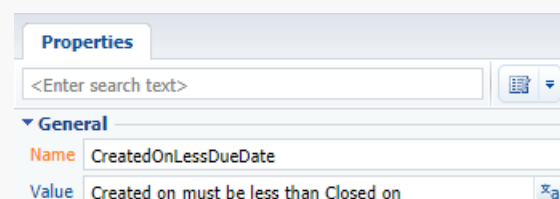
Рис. 2. — Добавление в схему локализуемой строки



Для созданной строки укажите (рис. 3):

- [*Название*] ([*Name*]) — "CreatedOnLessDueDate";
- [*Значение*] ([*Value*]) — "Дата создания должна быть меньше даты закрытия" ("Created on must be less than Closed on").

Рис. 3. — Свойства пользовательской локализуемой строки



3. В коллекцию methods схемы замещающей модели представления добавить реализацию методов

- `dueDateValidator()` — метод-валидатор, который определяет выполняется ли условие.
- `setValidationConfig()` — переопределенный базовый метод, в котором метод-валидатор привязан к колонкам [*DueDate*] и [*CreatedOn*].

Исходный код схемы замещающей модели представления:

```
define("OpportunityPageV2", [], function() {
    return {
        // Название схемы объекта страницы записи.
        entitySchemaName: "Opportunity",
        methods: {
            // Метод-валидатор значения колонок [DueDate] и [CreatedOn].
            dueDateValidator: function() {
                // Переменная для хранения сообщения об ошибке валидации.
                var invalidMessage = "";
                // Проверка значений колонок [DueDate] и [CreatedOn].
                if (this.get("DueDate") < this.get("CreatedOn")) {
                    // Если значение колонки [DueDate] меньше значения колонки [CreatedOn],
                    // в переменную invalidMessage помещается значение локализуемой строки с соо-
                    // об ошибке валидации.
                    invalidMessage = this.get("Resources.Strings.CreatedOnLessDueDate");
                }
                // Объект, свойство которого содержит сообщение об ошибке валидации.
                // Если валидация прошла успешно, в объекте возвращается пустая строка.
                return {
                    // Сообщение об ошибке валидации.
                    invalidMessage: invalidMessage
                };
            },
            // Переопределение базового метода, инициализирующего пользовательские валидаторы.
            setValidationConfig: function() {
                // Вызывает инициализацию валидаторов родительской модели представления.
                this.callParent(arguments);
                // Для колонки [DueDate] добавляется метод-валидатор dueDateValidator().
                this.addColumnValidator("DueDate", this.dueDateValidator);
                //Для колонки [CreatedOn] добавляется метод-валидатор dueDateValidator().
                this.addColumnValidator("CreatedOn", this.dueDateValidator);
            }
        }
    };
});
```

После сохранения схемы и обновления страницы приложения на странице продажи при вводе даты закрытия или даты создания, не соответствующих условию валидации (дата создания должна быть меньше даты закрытия), будет появляться строка с соответствующим сообщением (рис. 4), а при попытке сохранить такую продажу появится информационное окно (рис. 5).

Рис. 4. — Демонстрация результатов. Сообщение при выборе неправильной даты

The screenshot shows the 'OPPORTUNITY DETAILS' form for a customer named 'Vertigo Systems'. The form includes fields for 'New customer' (selected), 'Existing customer', 'Name' (022 / Vertigo Systems / Sale of Goods), 'Opportunity amount' (1,288.30), 'Probability, %' (5), 'Category' (Small business), 'Type', 'Division' (Direct sales team), 'Owner' (Supervisor), and 'Source'. The 'Created on' field is set to 7/19/2018, and the 'Closed on' field is set to 6/13/2018. Both fields have a blue validation message: 'Created on must be less than Closed on'. The 'Closed on' field also has a calendar icon.

Рис. 5. — Демонстрация результатов. Сообщение при сохранении

The screenshot shows the same 'OPPORTUNITY DETAILS' form, but with a modal dialog box displayed in the center. The dialog box contains the text: 'Field "Closed on": Created on must be less than Closed on' and an 'OK' button. The background form is slightly dimmed.

Пример 2

Описание примера

Настроить валидацию поля [*Рабочий телефон*] страницы контакта следующим образом: номер

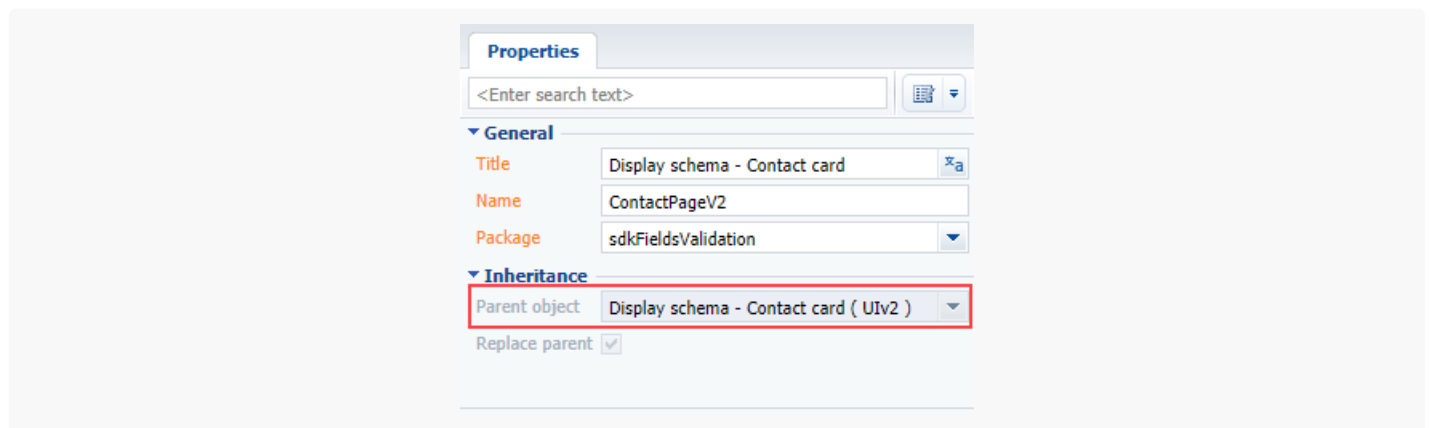
телефона должен соответствовать маске [+44 XXX XXX XXXX], иначе появляется сообщение "Введите номер в формате: +44 XXX XXX XXXX".

Алгоритм реализации примера

1. Создать схему замещающей модели представления

Создайте схему замещающей модели представления, в которой в качестве родительского объекта укажите схему [*Схема отображения карточки контакта*] ([*Display schema - Contact card*], `ContactPageV2`) (рис. 6). Процесс создания схемы замещающей модели представления страницы описан в статье "[Создать клиентскую схему](#)".

Рис. 6. — Свойства схемы замещающей модели представления страницы записи



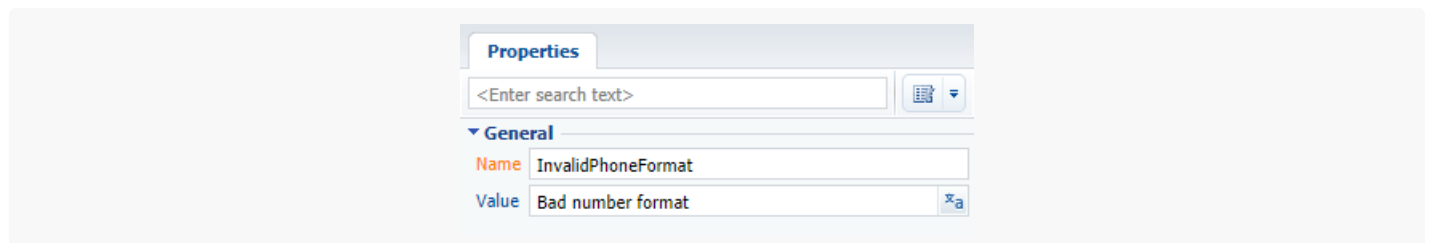
2. В коллекцию локализуемых строк схемы замещающей модели представления страницы добавить строку ошибки

Создайте новую локализуемую строку (рис. 2).

Для созданной строки укажите (рис. 7):

- [*Название*] ([*Name*]) — "InvalidPhoneFormat";
- [*Значение*] ([*Value*]) — "Введите номер в формате: +44 XXX XXX XXXX" ("Enter the number in format +44 XXX XXX XXXX").

Рис. 7. — Свойства пользовательской локализуемой строки



3. В коллекцию methods модели представления добавить реализацию методов

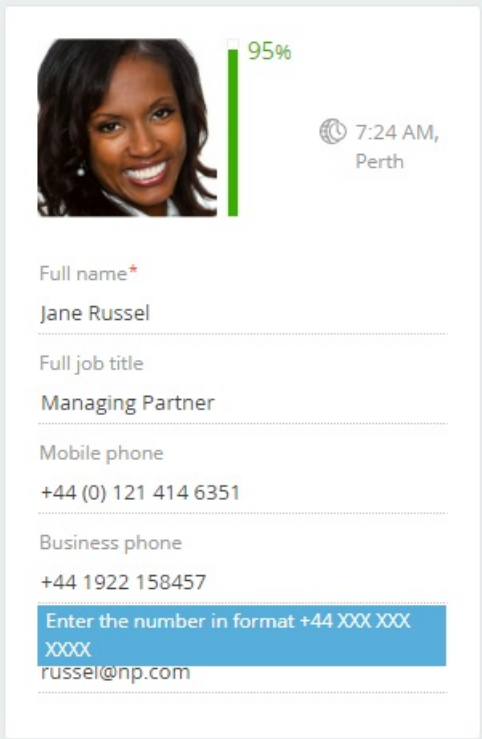
- `phoneValidator()` — метод-валидатор, который определяет выполняется ли условие.
- `setValidationConfig()` — переопределенный базовый метод, в котором метод-валидатор привязан к колонке [*Phone*].

Исходный код схемы замещающей модели представления:

```
define("ContactPageV2", ["ConfigurationConstants"], function(ConfigurationConstants) {
    return {
        entitySchemaName: "Contact",
        methods: {
            // Переопределение базового метода, инициализирующего пользовательские валидаторы.
            setValidationConfig: function() {
                // Вызывает инициализацию валидаторов родительской модели представления.
                this.callParent(arguments);
                // Для колонки [Phone] добавляется метод-валидатор phoneValidator().
                this.addColumnValidator("Phone", this.phoneValidator);
            },
            phoneValidator: function(value) {
                // Переменная для хранения сообщения об ошибке валидации.
                var invalidMessage = "";
                // Переменная для хранения результата проверки номера.
                var isValid = true;
                // Переменная для хранения номера телефона.
                var number = value || this.get("Phone");
                // Определение правильности формата номера с помощью регулярного выражения.
                isValid = (Ext.isEmpty(number) ||
                    new RegExp("^\\+44\\s[0-9]{3}\\s[0-9]{3}\\s[0-9]{4}$").test(number));
                // Если формат номера неправильный, то заполняется сообщение об ошибке.
                if (!isValid) {
                    invalidMessage = this.get("Resources.Strings.InvalidPhoneFormat");
                }
                // Объект, свойство которого содержит сообщение об ошибке валидации.
                // Если валидация прошла успешно, в объекте возвращается пустая строка.
                return {
                    invalidMessage: invalidMessage
                };
            }
        }
    };
});
```

После сохранения схемы и обновления страницы приложения на странице контакта при редактировании номера рабочего телефона будет происходить проверка правильности формата номера и если формат неверный, то будет появляться строка с соответствующим сообщением (рис. 8, 9).

Рис. 8. — Демонстрация результата. Сообщение при вводе неправильного номера



95%

7:24 AM, Perth

Full name*

Jane Russel

Full job title

Managing Partner

Mobile phone

+44 (0) 121 414 6351

Business phone

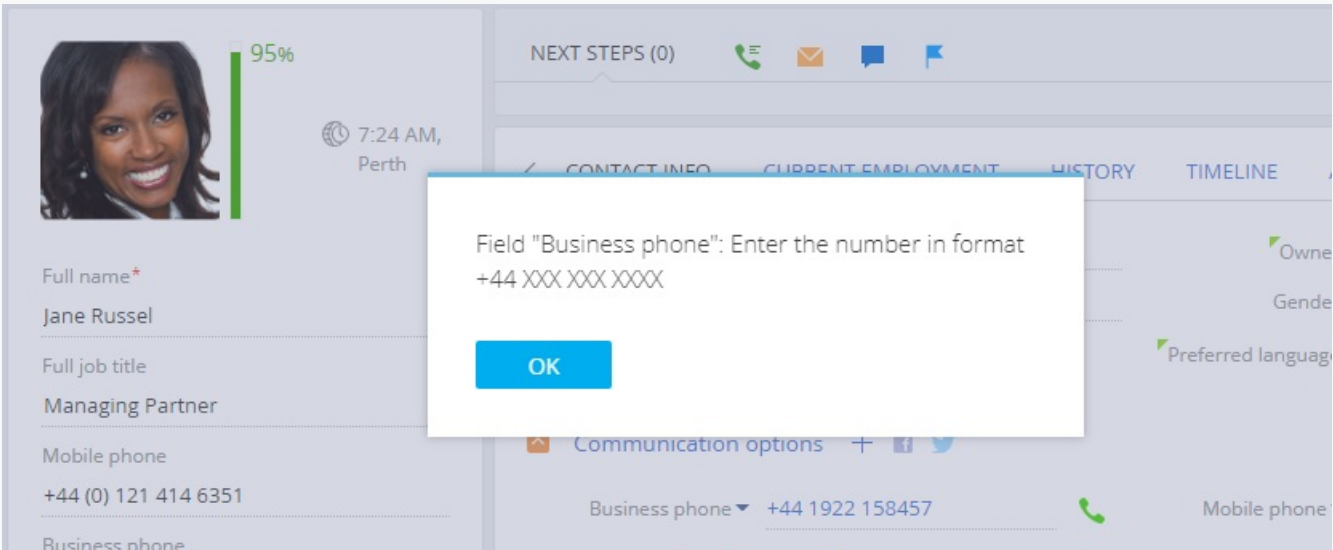
+44 1922 158457

Enter the number in format +44 XXX XXX XXXX

XXXX

russel@np.com

Рис. 9. — Демонстрация результата. Сообщение при сохранении



95%

7:24 AM, Perth

Full name*

Jane Russel

Full job title

Managing Partner

Mobile phone

+44 (0) 121 414 6351

Business phone

+44 1922 158457

Field "Business phone": Enter the number in format +44 XXX XXX XXXX

OK

Установить для поля значение по умолчанию

Средний

Исходный код

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

Пример установки для поля значения по умолчанию через свойства колонки объекта

Описание примера

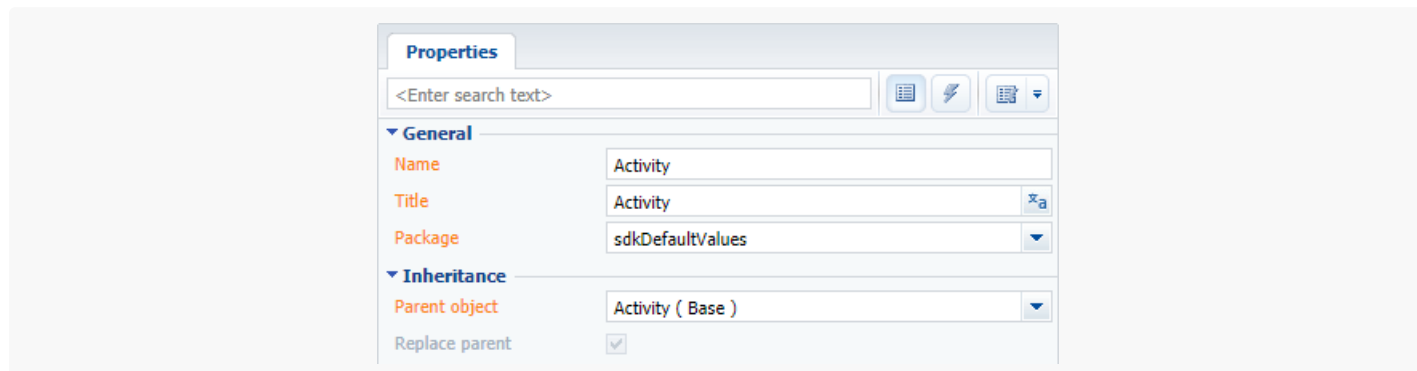
При создании новой активности по умолчанию должен быть установлен признак [*Отображать в расписании*] (*Show in calendar*).

Алгоритм реализации примера

1. Создать в пользовательском пакете схему замещающего объекта [*Активность*] (*Activity*)

Создайте схему замещающего объекта [*Активность*] (*Activity*) (рис. 1). Процесс создания схемы замещающего объекта описан в статье "[Создать схему объекта](#)".

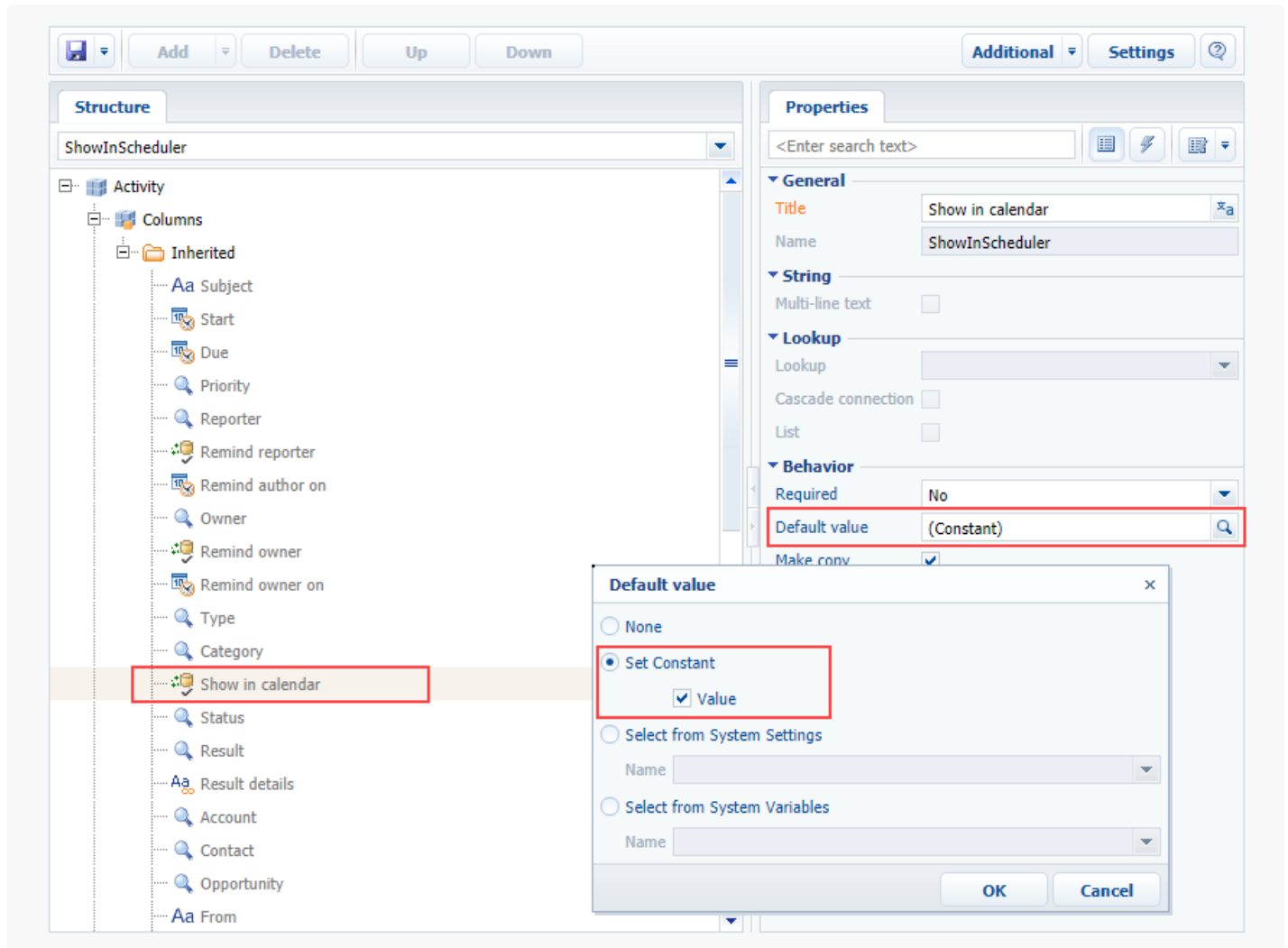
Рис. 1. — Свойства схемы замещающего объекта [*Активность*] (*Activity*)



2. Установить для колонки [*Отображать в расписании*] (*Show in calendar*) значение по умолчанию

Выберите из списка унаследованных колонок объекта колонку [*Отображать в расписании*] (*Show in calendar*) и отредактируйте ее свойство [*Значение по умолчанию*] (*Default value*) как показано на рисунке 2. Для реализации примера выберите в качестве значения по умолчанию константное значение.

Рис. 2. — Установка значения по умолчанию для колонки [*Отображать в расписании*] (*Show in calendar*)



После публикации схемы и обновления страницы приложения с очисткой кэша на странице активности при создании новой активности поле [*Отображать в расписании*] ([*Show in calendar*]) будет автоматически отмечено (рис. 3).

Рис. 3. — Демонстрация установки значения по умолчанию

New task

What can I do for you? >

CREATIO

CLOSE ACTIONS +

VIEW

Subject* New task

Start* 4/22/2018 2:15 AM Owner* Supervisor

Due* 4/22/2018 2:45 AM Reporter* Supervisor

Status* Not started Priority* Medium

Category* To do

Show in calendar ☒

Пример установки значения по умолчанию в исходном коде страницы записи

Описание примера

При создании нового проекта значение поля [*Крайний срок*] на странице проекта по умолчанию должно принимать значение поля [*Начало*] плюс 10 дней.

Алгоритм реализации примера

1. В пользовательском пакете создать схему замещающей модели представления страницу проекта

Создайте схему замещающей модели представления, в которой в качестве родительского объекта укажите схему [*Страница редактирования проекта*] ([*Project edit page*], ProjectPageV2) (рис. 4). Процесс создания схемы замещающей модели представления описан в статье "[Создать клиентскую схему](#)".

Рис. 4. — Свойства схему замещающей модели представления страницы записи

Properties

<Enter search text>

General

Title Project edit page

Name ProjectPageV2

Package sdkDefaultValues

Inheritance

Parent object Project edit page (Project)

Replace parent ☒

2. В коллекцию методов схемы замещающей модели представления страницы добавить реализацию методов

- `setDeadline()` – метод-обработчик. Рассчитывает значение поля [*Крайний срок*] ([*Deadline*]).
- `onEntityInitialized()` — переопределенный базовый виртуальный метод. Срабатывает после окончания инициализации схемы объекта. В него добавьте вызов метода-обработчика для установки значения поля [*Крайний срок*] ([*Deadline*]) в момент открытия страницы записи.

Исходный код схемы замещающей модели представления:

```
define("ProjectPageV2", [], function() {
    return {
        // Название схемы объекта страницы записи.
        entitySchemaName: "Project",
        methods: {
            // Переопределение базового метода Terrasoft.BasePageV2.onEntityInitialized().
            // Срабатывает после окончания инициализации схемы объекта страницы записи.
            onEntityInitialized: function() {
                // Вызывается родительская реализация метода.
                this.callParent(arguments);
                // Вызов метода-обработчика, который рассчитывает значение колонки [Крайний срок]
                this.setDeadline();
            },
            // Метод-обработчик. Рассчитывает значение колонки [Крайний срок].
            setDeadline: function() {
                // Значение колонки [Крайний срок].
                var deadline = this.get("Deadline");
                // Установлен ли режим новой записи?
                var newmode = this.isNewMode();
                // Если значение не установлено и режим новой записи установлен.
                if (!deadline && newmode) {
                    // Получение значения колонки [Начало]
                    var newDate = new Date(this.get("StartDate"));
                    newDate.setDate(newDate.getDate() + 10);
                    // Установка значения колонки [Крайний срок].
                    this.set("Deadline", newDate);
                }
            }
        }
    };
});
```

После сохранения схемы и обновления страницы приложения при создании нового проекта в поле [*Крайний срок*] ([*Deadline*]) будет установлена дата, равная дате в поле [*Начало*] ([*Start*]) плюс 10 дней (рис. 5).

Рис. 5. — Демонстрация установки вычисляемого значения по умолчанию

New record

What can I do for you? >

Creatio

SAVE CANCEL ACTIONS VIEW

Name*

Status* Planned

Owner* Supervisor

Amount

< GENERAL INFORMATION STRUCTURE FINANCIAL INDICATORS HISTORY ATTACHMENTS AND NOTES >

Account

Contact

Completion %

Type*

Calculate automatically ☐

Start 4/22/2018

End

Duration 0 min

Deadline 5/2/2018

Сделать поле обязательным для заполнения по условию

Средний

На заметку.

В Creatio реализована возможность настраивать бизнес-правила не только средствами разработки, но и с помощью мастера разделов. Подробнее об этом читайте в статье ["Настройка бизнес-правил"](#).

Описание примера

Настроить поля на странице контакта таким образом, чтобы поле [*Рабочий телефон*] ([*Business phone*]) было обязательным для заполнения при условии, что в поле [*Тип контакта*] ([*Type*]) выбрано значение "Клиент" ("Customer").

Исходный код

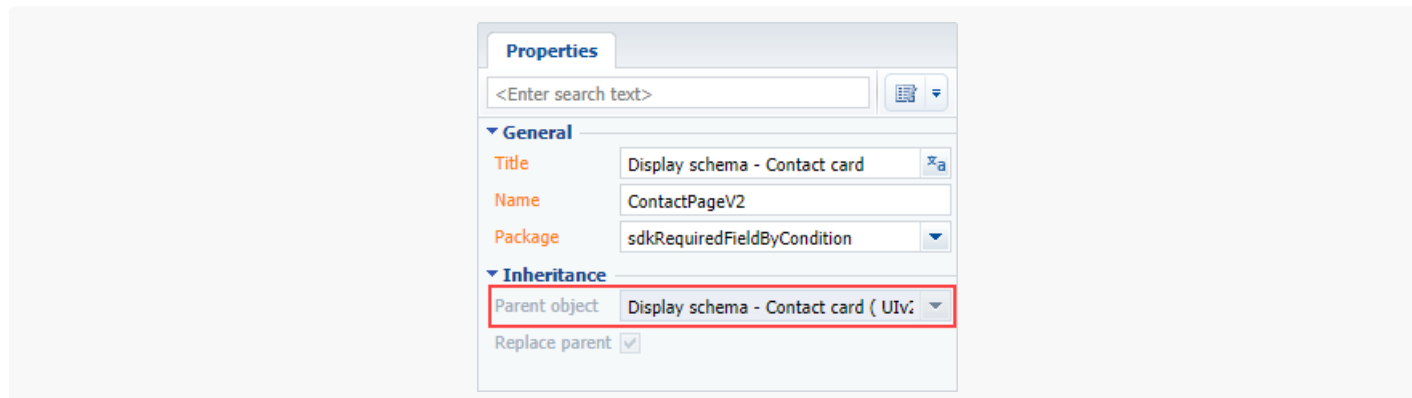
Пакет с реализацией примера можно скачать по [ссылке](#).

Алгоритм реализации примера

1. Создать схему замещающей модели представления страницы контакта

Создайте схему замещающей модели представления, в которой в качестве родительского объекта укажите схему [*Схема отображения карточки контакта*] ([*Display schema — Contact card*]) (рис. 1). Процесс создания схемы замещающей модели представления описан в статье "[Создать клиентскую схему](#)".

Рис. 1. — Свойства схему замещающей модели представления страницы записи



2. В свойство rules модели представления страницы добавить правило

Добавьте в свойство `rules` схемс замещающей модели представления страницы правило с типом `BINDPARAMETER` для колонки `Phone`. Свойству `property` правила установите значение `BusinessRuleModule.enums.Property.REQUIRED`. В массив `conditions` добавьте условие выполнения правила — значение колонки `Type` модели должно быть равно конфигурационной константе `ConfigurationConstants.ContactType.Client`.

На заметку.

Конфигурационная константа `ConfigurationConstants.ContactType.Client` содержит идентификатор записи "Клиент" справочника [*Типы контактов*].

Исходный код схемс замещающей модели представления:

```
// В список зависимостей модуля добавить модули BusinessRuleModule и ConfigurationConstants.
define("ContactPageV2", ["BusinessRuleModule", "ConfigurationConstants"],
function(BusinessRuleModule, ConfigurationConstants) {
    return {
        // Название схемы объекта страницы записи.
        entitySchemaName: "Contact",
        // Правила модели представления страницы записи.
        rules: {
            // Набор правил для колонки [Рабочий телефон] модели представления.
            "Phone": {
```

```

// Зависимость обязательности поля [Рабочий телефон] от значения в поле
"BindParameterRequiredAccountByType": {
  // Тип правила BINDPARAMETER.
  "ruleType": BusinessRuleModule.enums.RuleType.BINDPARAMETER,
  // Правило регулирует свойство REQUIRED.
  "property": BusinessRuleModule.enums.Property.REQUIRED,
  // Массив условий, при выполнении которых срабатывает правило.
  // Определяет равно ли значение в колонке [Тип] значению "Клиент".
  "conditions": [{
    // Выражение левой части условия.
    "leftExpression": {
      // Тип выражения – атрибут (колонка) модели представления.
      "type": BusinessRuleModule.enums.ValueType.ATTRIBUTE,
      // Название колонки модели представления, значение которой с
      "attribute": "Type"
    },
    // Тип операции сравнения.
    "comparisonType": Terrasoft.ComparisonType.EQUAL,
    // Выражение правой части условия.
    "rightExpression": {
      // Тип выражения – константное значение.
      "type": BusinessRuleModule.enums.ValueType.CONSTANT,
      // Значение, с которым сравнивается выражение левой части.
      "value": ConfigurationConstants.ContactType.Client
    }
  }]
}
}
}
}
};
});

```

После сохранения схемы и обновления веб-страницы приложения на странице контакта поле [*Рабочий телефон*] ([*Business phone*]) будет обязательным для заполнения, если тип контакта — "Клиент" ("Customer") (рис. 2, 3).

Рис. 2. — Результат выполнения примера. Поле [*Рабочий телефон*] ([*Business phone*]) — необязательно

100%

9:40 PM
-1d,
Atlanta

Full name*
Clayton Bruce

Full job title
Purchase Manager

Mobile phone
+1 404 389 0476

Business phone
+1 404 532 3976

Email
clayton@axiom-corp.com

NEXT STEPS (1)

Prepare quotation
1/11/2018 | John Best

< CONTACT INFO CURRENT EMPLOYMENT HISTORY

Type **Contact person**

Title Mr.

Communication options + f t

Mobile phone +1 404 389 0476

Business phone +1 404 532 3976

Рис. 3. — Результат выполнения примера. Поле [Рабочий телефон] ([Business phone]) — обязательно

100%

9:40 PM
-1d,
Atlanta

Full name*
Clayton Bruce

Full job title
Purchase Manager

Mobile phone
+1 404 389 0476

Business phone*
+1 404 532 3976

Email
clayton@axiom-corp.com

NEXT STEPS (1)

Prepare quotation
1/11/2018 | John Best

< CONTACT INFO CURRENT EMPLOYMENT HISTORY

Type **Customer**

Title Mr.

Communication options + f t

Mobile phone +1 404 389 0476

Business phone +1 404 532 3976

Добавить разницу дат в полях страницы

Для работы с датами в front-end части приложения используются возможности стандартного JavaScript-объекта `Date`. Например, для получения дня месяца для определенной даты в соответствии с локальным временем пользователя используется метод `Date.prototype.getDate()`. А для установки дня месяца относительно текущего месяца — метод `Date.prototype.setDate()`. Все свойства и методы объекта `Date` приведены в [документации](#).

Например, при создании нового договора нужно, чтобы в поле [*Дата завершения*] ([*End Date*]) отображалась дата на 5 дней больше, чем в поле [*Дата начала*] ([*Start Day*]). Для этого необходимо:

1. Создать схему замещающей модели представления страницы записи раздела [*Договоры*] `ContractPageV2` ([*Section edit page schema - "Contracts"*]). Как создать схему замещающей модели представления описано в статье ["Создать клиентскую схему"](#).
2. В созданную схему замещающей модели представления добавить следующий код:

```
define("ContractPageV2", [], function() {
    return {
        entitySchemaName: "Contract",
        methods: {
            // Установка даты выполняется после инициализации объекта.
            onEntityInitialized: function() {
                // Проверка режима новой записи
                if ((this.isAddMode() && this.Ext.isEmpty(this.get("EndDate")))) {
                    // Вызов вспомогательного метода.
                    this.setEndDate(this.get("StartDate"), 5);
                }
                // Вызов базовой функциональности.
                this.callParent(arguments);
            },
            // Вспомогательный метод для установки даты.
            setEndDate: function(date, dateOffsetInDays) {
                var offsetDate = new Date();
                offsetDate.setDate(date.getDate() + dateOffsetInDays);
                this.set("EndDate", offsetDate);
            }
        }
    };
});
```

3. Сохранить изменения.

4. Обновить страницу браузера.

В результате при добавлении нового договора будет автоматически установлена дата его завершения, отличающаяся от даты начала на 5 дней (рис. 1).

Рис. 1. — Результат выполнения кейса

На заметку.

Чтобы дата в поле [Дата завершения] ([End Date]) пересчитывалась автоматически при изменении пользователем поля [Дата начала] ([Start Day]), необходимо воспользоваться функциональностью вычисляемых полей. Как это сделать подробно описано в статье "[Добавление вычисляемых полей](#)".

Отфильтровать справочное поле

Средний

На заметку.

В Creatio реализована возможность настраивать бизнес-правила не только средствами разработки, но и с помощью мастера разделов. Подробнее об этом можно прочитать в статье "[Настройка бизнес-правил](#)".

Описание примера

Добавить на страницу контакта поля [Страна], [Область/Штат] и [Город]. При выборе определенной страны список областей должен быть отфильтрован по выбранной стране. При выборе определенной области список городов должен отображать города только из выбранной области. При выборе определенного города поля страны и области должны заполняться соответствующими городу значениями.

На заметку.

В базовой схеме страницы контакта уже определено правило фильтрации городов в зависимости от страны. Поэтому, если не добавлять поле [Страна], то невозможно будет выбрать город из

страны, отличной от указанной для контакта.

Исходный код

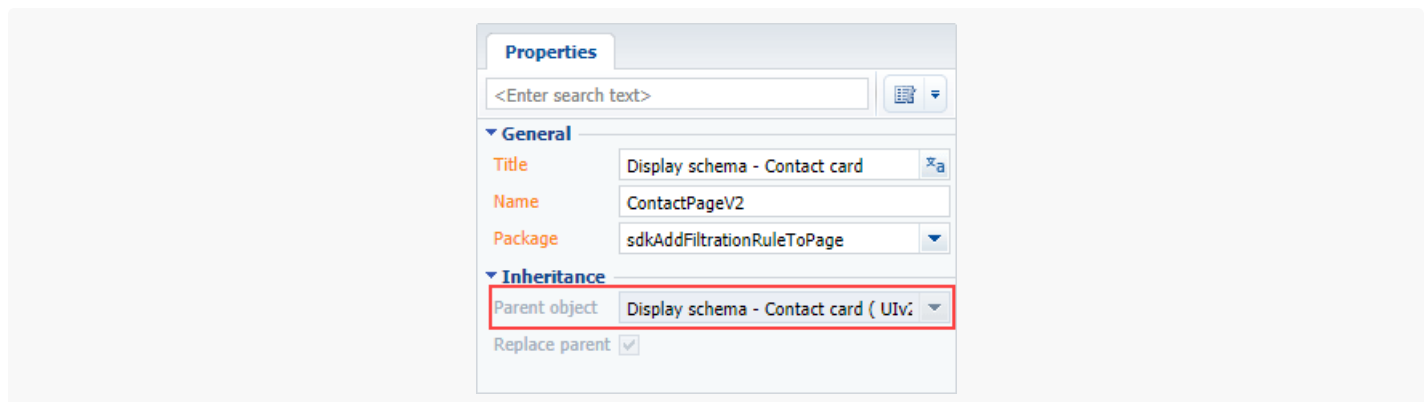
Пакет с реализацией примера можно скачать по [ссылке](#).

Алгоритм реализации примера

1. Создать схему замещающей модели представления страницы контакта

Создайте схему замещающей модели представления, в котором в качестве родительского объекта укажите схему [*Схема отображения карточки контакта*] ([*Display schema — Contact card*]) (рис. 1). Процесс создания схему замещающей модели представления описан в статье "[Создать клиентскую схему](#)".

Рис. 1. — Свойства схему замещающей модели представления страницы записи



2. Добавить поля [Страна], [Область/штат] и [Город] на страницу записи

Добавьте в массив `diff` три конфигурационных объекта с настройками свойств соответствующих полей.

3. Добавить правила с типом FILTRATION для колонок [Город] и [Область/штат]

Добавьте в свойство `rules` модели представления страницы два правила с типом `BusinessRuleModule.enums.RuleType.FILTRATION` для колонок [*City*] и [*Region*]. Для выполнения обратной фильтрации, т. е. автозаполнения полей [*Страна*] и [*Область/Штат*] соответствующими значениями в зависимости от выбранного города необходимо установить значение `true` для свойства `autocomplete`.

Исходный код схемы замещающей модели представления:

```

// В список зависимостей модуля добавить модуль BusinessRuleModule.
define("ContactPageV2", ["BusinessRuleModule"],
    function(BusinessRuleModule) {
        return {
            // Название схемы объекта страницы записи.
            entitySchemaName: "Contact",
            // Свойство, содержащее коллекцию бизнес-правил схемы модели представления страниц
            rules: {
                // Набор правил для колонки [City] модели представления.
                "City": {
                    // Правило фильтрации колонки [City] по значению колонки [Region].
                    "FiltrationCityByRegion": {
                        // Тип правила FILTRATION.
                        "ruleType": BusinessRuleModule.enums.RuleType.FILTRATION,
                        // Будет выполняться обратная фильтрация.
                        "autocomplete": true,
                        // Будет выполняться очистка значения при изменении значения колонки
                        "autoClean": true,
                        // Путь к колонке для фильтрации в справочной схеме [City],
                        // на которую ссылается колонка [City] модели представления
                        // страницы записи.
                        "baseAttributePatch": "Region",
                        // Тип операции сравнения в фильтре.
                        "comparisonType": Terrasoft.ComparisonType.EQUAL,
                        // В качестве значения при сравнении выступает колонка (атрибут)
                        // модели представления.
                        "type": BusinessRuleModule.enums.ValueType.ATTRIBUTE,
                        // Имя колонки модели представления страницы записи,
                        // по значению которой будет выполняться фильтрация.
                        "attribute": "Region"
                    }
                },
                // Набор правил для колонки [Region] модели представления.
                "Region": {
                    "FiltrationRegionByCountry": {
                        "ruleType": BusinessRuleModule.enums.RuleType.FILTRATION,
                        "autocomplete": true,
                        "autoClean": true,
                        "baseAttributePatch": "Country",
                        "comparisonType": Terrasoft.ComparisonType.EQUAL,
                        "type": BusinessRuleModule.enums.ValueType.ATTRIBUTE,
                        "attribute": "Country"
                    }
                }
            },
            // Настройка визуализации полей [Страна], [Область/штат] и [Город] на странице
            diff: [

```

```

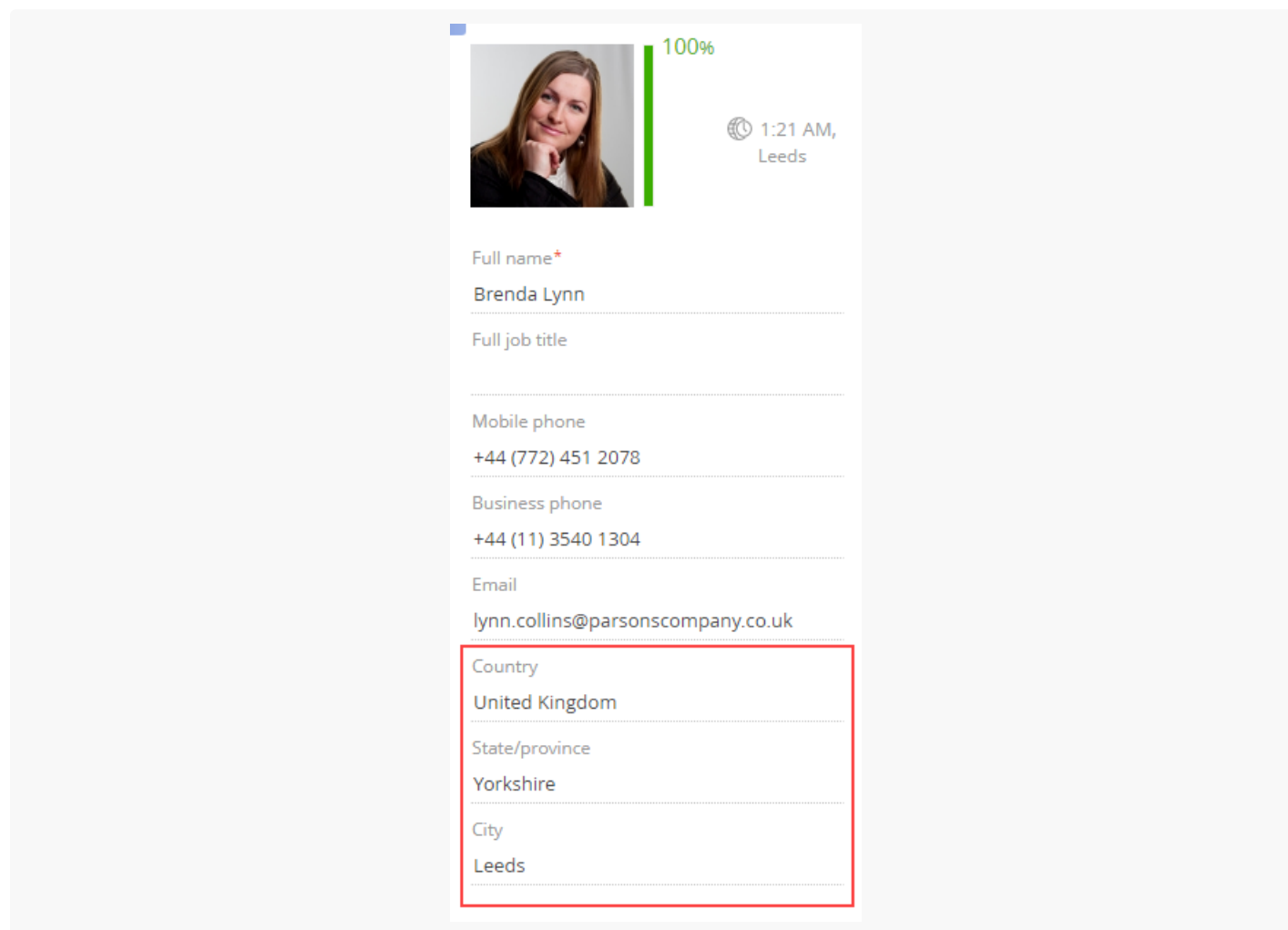
// Метаданные для добавления поля [Страна].
{
    "operation": "insert",
    "parentName": "ProfileContainer",
    "propertyName": "items",
    "name": "Country",
    "values": {
        "contentType": Terrasoft.ContentType.LOOKUP,
        "layout": {
            "column": 0,
            "row": 6,
            "colSpan": 24
        }
    }
},
// Метаданные для добавления поля [Область/штат].
{
    "operation": "insert",
    "parentName": "ProfileContainer",
    "propertyName": "items",
    "name": "Region",
    "values": {
        "contentType": Terrasoft.ContentType.LOOKUP,
        "layout": {
            "column": 0,
            "row": 7,
            "colSpan": 24
        }
    }
},
// Метаданные для добавления поля [Город].
{
    "operation": "insert",
    "parentName": "ProfileContainer",
    "propertyName": "items",
    "name": "City",
    "values": {
        "contentType": Terrasoft.ContentType.LOOKUP,
        "layout": {
            "column": 0,
            "row": 8,
            "colSpan": 24
        }
    }
}
]
};
});

```

4. Сохранить созданную схему замещающей модели представления страницы

После сохранения схемы и обновления веб-страницы приложения в профиле контакта появятся три новых поля (рис. 2). В каждом из них содержимое будет фильтроваться в зависимости от значения, выбранного в другом (рис. 3). При этом фильтрация также работает в окне выбора значения из справочника (рис. 4).

Рис. 2. — Новые поля в профиле контакта



The screenshot shows a contact profile for Brenda Lynn. At the top, there is a profile picture, a green progress bar at 100%, and a clock icon indicating the last update at 1:21 AM in Leeds. Below the picture are several form fields:

- Full name*: Brenda Lynn
- Full job title: (empty)
- Mobile phone: +44 (772) 451 2078
- Business phone: +44 (11) 3540 1304
- Email: lynn.collins@parsonsccompany.co.uk
- A red box highlights a group of location fields:
 - Country: United Kingdom
 - State/province: Yorkshire
 - City: Leeds

Рис. 3. — Фильтрация содержимого полей

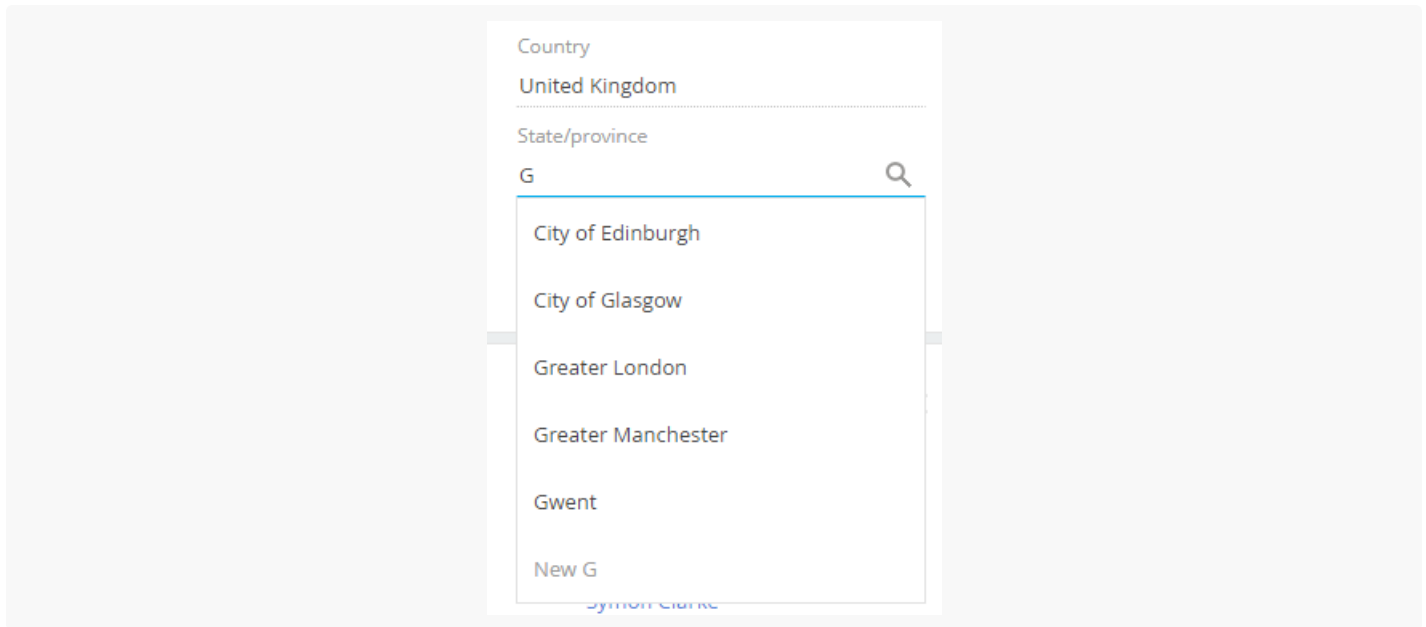
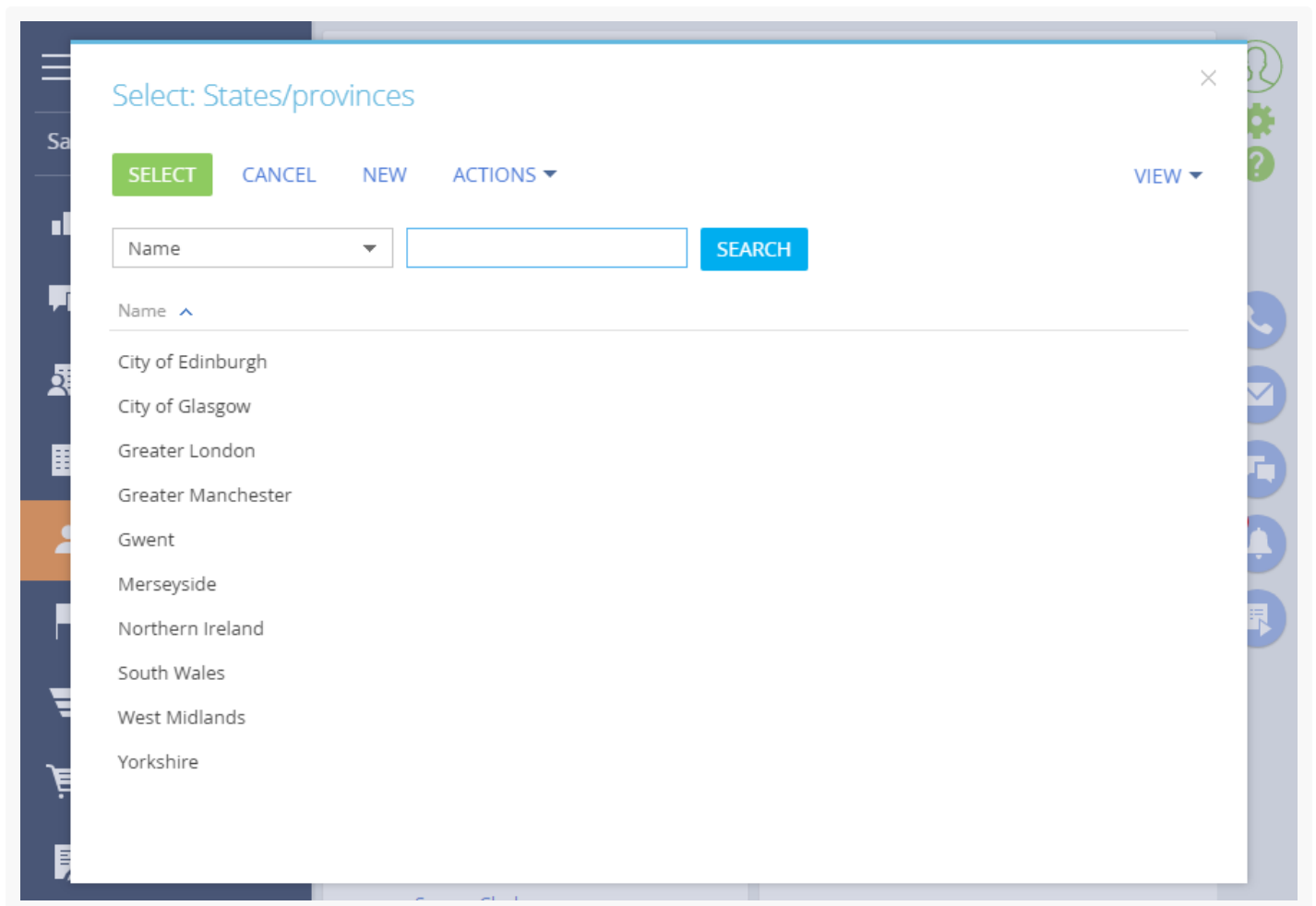


Рис. 4. — Отфильтрованные области в окне выбора



Применить фильтрацию к справочным

ПОЛЯМ

 Средний

Описание примера

При добавлении значения в поле [*Ответственный*] страницы контрагента отображать только те значения из справочника контактов, для которых выполняются следующие условия:

- с данным контактом есть связанный пользователь системы;
- данный пользователь системы активен.

Исходный код

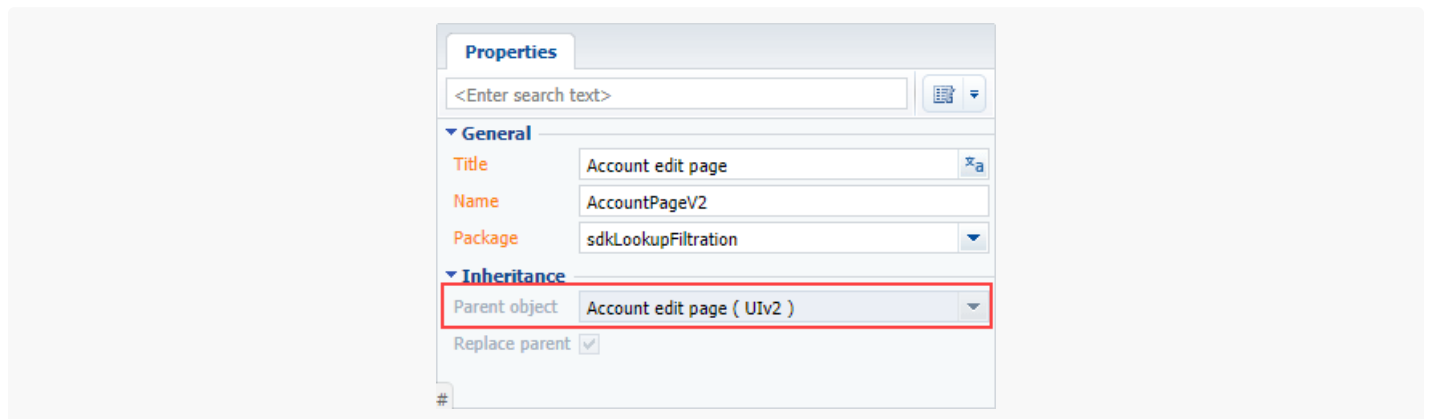
Пакет с реализацией примера можно скачать по [ссылке](#).

Алгоритм реализации примера

1. Создать схему замещающей модели представления страницы контрагента

Создайте схему замещающей модели представления, в которой в качестве родительского объекта укажите схему [*Страница редактирования контрагента*] ([*Account edit page*], `AccountPageV2`) (рис. 1). Процесс создания схемы замещающей модели представления описан в статье "[Создать клиентскую схему](#)".

Рис. 1. — Свойства схемы замещающей модели представления страницы записи



2. В свойство `attributes` модели представления добавить атрибут с фильтрацией

В конфигурационном объекте атрибута [*Owner*] укажите тип данных колонки —

`Terrasoft.DataValueType.LOOKUP` и опишите конфигурационный объект поля-справочника `lookupListConfig`.

В `lookupListConfig` добавьте свойство `filters`, представляющее собой массив. В массив добавьте

функцию, возвращающую коллекцию фильтров.

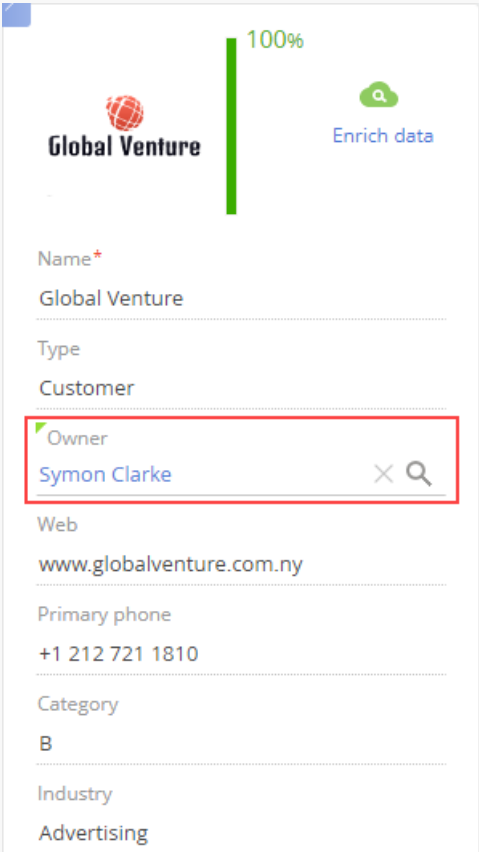
Исходный код схемы замещающей модели представления:

```
define("AccountPageV2", [], function() {
    return {
        // Название схемы объекта страницы записи.
        "entitySchemaName": "Account",
        // Атрибуты схемы.
        "attributes": {
            // Колонка модели представления.
            "Owner": {
                // Тип данных атрибута.
                "dataValueType": Terrasoft.DataValueType.LOOKUP,
                // Конфигурационный объект атрибута типа LOOKUP.
                "lookupListConfig": {
                    // Массив фильтров, применяемых к запросу для формирования данных поля-справочника.
                    "filters": [
                        function() {
                            var filterGroup = Ext.create("Terrasoft.FilterGroup");
                            // Добавление фильтра "IsUser" в результирующую коллекцию фильтров.
                            // Выбирает все записи из корневой схемы Contact, к которой присоеди
                            // колонка Id из схемы SysAdminUnit, для которых Id не равен null.
                            filterGroup.add("IsUser",
                                Terrasoft.createColumnIsNotNullFilter("[SysAdminUnit:Contact].Id"));
                            // Добавление фильтра "IsActive" в результирующую коллекцию фильтров.
                            // Выбирает все записи из корневой схемы Contact, к которой присоеди
                            // колонка Active из схемы SysAdminUnit, для которых Active=true.
                            filterGroup.add("IsActive",
                                Terrasoft.createColumnFilterWithParameter(
                                    Terrasoft.ComparisonType.EQUAL,
                                    "[SysAdminUnit:Contact].Active",
                                    true));
                            return filterGroup;
                        }
                    ]
                }
            }
        }
    };
});
```

После сохранения схемы и обновления страницы приложения на странице контрагента при добавлении значения в поле [*Ответственный*] ([*Owner*]) страницы контрагента будут отображаться только те значения из справочника контактов, которые соответствуют условиям примера (рис. 2, рис. 3). Т.е.:

- с данным контактом есть связанный пользователь системы;
- данный пользователь системы активен.

Рис. 2. — Профиль контрагента с ответственным



Global Venture

100%

Enrich data

Name*

Global Venture

Type

Customer

Owner

Symon Clarke

Web

www.globalventure.com.ny

Primary phone

+1 212 721 1810

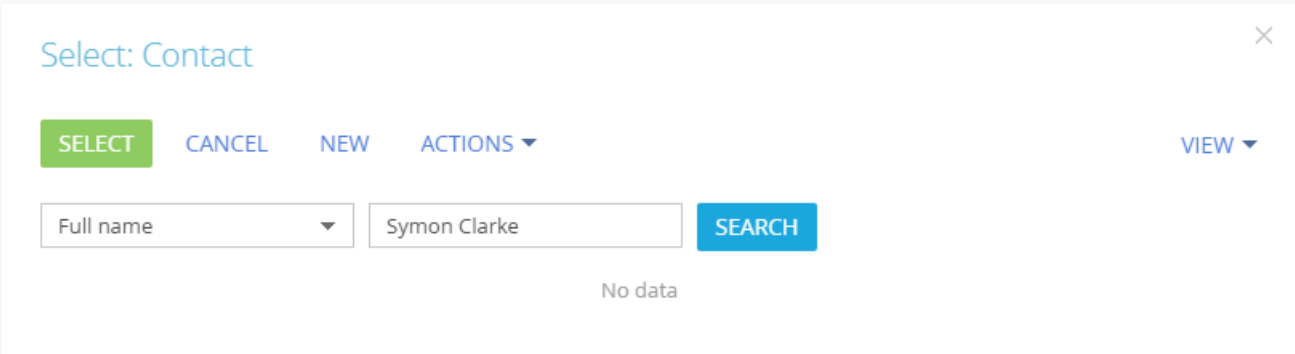
Category

B

Industry

Advertising

Рис. 3. — Ответственный отсутствует в отфильтрованном справочнике контактов



Select: Contact

SELECT CANCEL NEW ACTIONS

VIEW

Full name

Symon Clarke

SEARCH

No data

Заблокировать поля

Средний

Описание примера

Блокировать все поля на странице счета если счет находится в состоянии [*Оплачен полностью*] ([*Paid*]). Поле [*Состояние оплаты*] ([*Payment status*]) и деталь [*Активности*] ([*Activities*]) должны оставаться доступными для редактирования.

Важно.

Если для поля существует привязка для свойства `enabled` в элементе массива `diff` или в бизнес-правиле `BINDPARAMETER`, то механизм не будет блокировать такое поле.

Исходный код

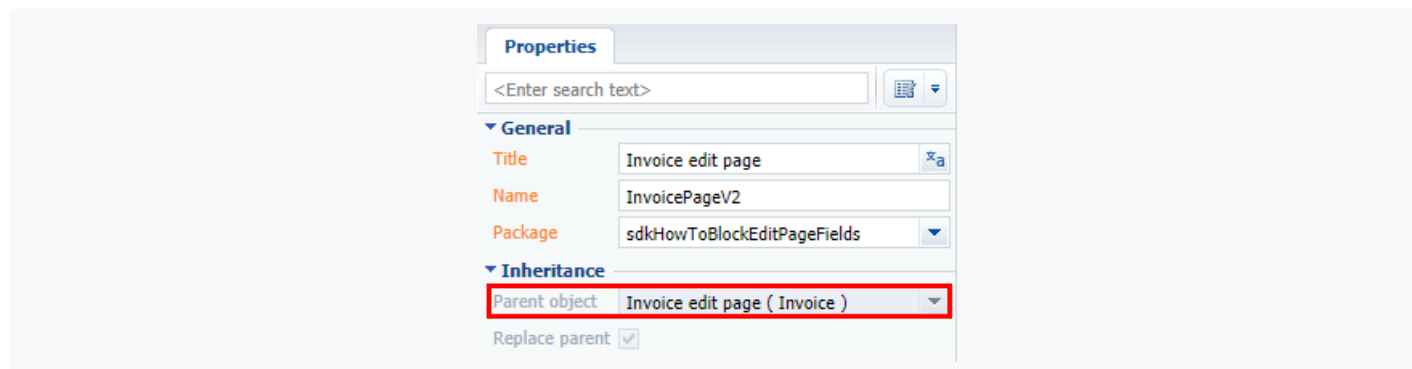
Пакет с реализацией примера можно скачать по [ссылке](#).

Алгоритм реализации примера

1. В пользовательском пакете создать схему замещающей модели представления страницы счета

Создайте схему замещающей модели представления, в которой в качестве родительского объекта укажите схему [*Страница редактирования счета*] ([*Invoice edit page*], `InvoicePageV2`) (рис. 1). Процесс создания схемы замещающей модели представления описан в статье "[Создать клиентскую схему](#)".

Рис. 1. — Свойства схемы замещающей модели представления страницы записи



2. Добавить исходный код схемы

Определите атрибут `IsModelItemsEnabled` и реализуйте методы, необходимые для блокирования и исключения блокирования полей страницы счета. В качестве контейнера механизма блокировки используется контейнер `CardContentWrapper` страницы записи.

Исходный код схемы замещающей модели представления:

```
define("InvoicePageV2", ["InvoiceConfigurationConstants"], function(InvoiceConfigurationConstant
    return {
        entitySchemaName: "Invoice",
        attributes: {
            "IsModelItemsEnabled": {
                dataValueType: Terrasoft.DataValueType.BOOLEAN,
                value: true,
```

```

        dependencies: [{
            columns: ["PaymentStatus"],
            methodName: "setCardLockoutStatus"
        }]
    },
    methods: {
        getDisableExclusionsColumnTags: function() {
            return ["PaymentStatus"];
        },

        getDisableExclusionsDetailSchemaNames: function() {
            return ["ActivityDetailV2"];
        },
        setCardLockoutStatus: function() {
            var state = this.get("PaymentStatus");
            if (state.value === InvoiceConfigurationConstants.Invoice.PaymentStatus.Paid) {
                this.set("IsModelItemsEnabled", false);
            } else {
                this.set("IsModelItemsEnabled", true);
            }
        },
        onEntityInitialized: function() {
            this.callParent(arguments);
            this.setCardLockoutStatus();
        }
    },
    details: /**SCHEMA_DETAILS*/{}/**SCHEMA_DETAILS*/,
    diff: /**SCHEMA_DIFF*/[
        {
            "operation": "merge",
            "name": "CardContentWrapper",
            "values": {
                "generator": "DisableControlsGenerator.generatePartial"
            }
        }
    ]/**SCHEMA_DIFF*/
    };
});

```

После сохранения схемы и обновления страницы приложения с очисткой кэша, при установке счета в состояние [*Оплачен полностью*] ([*Paid*]) большинство полей страницы счета станут заблокированными (рис. 2). Незаблокированными останутся поля и детали, указанные в исключениях для блокировки. Также незаблокированными останутся поля, для которых явно указано значение `true` для свойства `enabled`.

Рис. 2. — Результат выполнения примера. Состояние счета [*Оплачен полностью*] — блокируется большинство полей

The screenshot shows the Creatio CRM interface for an invoice (INV-2). The left sidebar contains navigation links: Sales, Dashboards, Feed, Leads, Accounts, Contacts, Activities, Opportunities, and Orders. The main area displays the invoice details under the 'GENERAL INFORMATION' tab. Fields include:

- Number: INV-2
- Date: 10/10/2017 (locked with a red bar and lock icon)
- Owner: Supervisor
- Order: (empty)
- Customer: John Smith
- Supplier: Our Company
- Amount: 500.00
- Payment status: Paid
- Paid on: 10/10/2017
- Payment amount: 500.00

On the right side, there are icons for user profile, settings, help, and communication tools.

Заблокировать поле по условию

Средний

На заметку.

В Creatio реализована возможность настраивать бизнес-правила не только средствами разработки, но и с помощью мастера разделов. Подробнее об этом можно прочитать в статье ["Настройка бизнес-правил"](#).

Описание примера

Настроить поля на странице контакта таким образом, чтобы поле [*Рабочий телефон*] ([*Business phone*]) было доступным только при условии, что заполнено поле [*Мобильный телефон*] ([*Mobile phone*]).

Исходный код

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

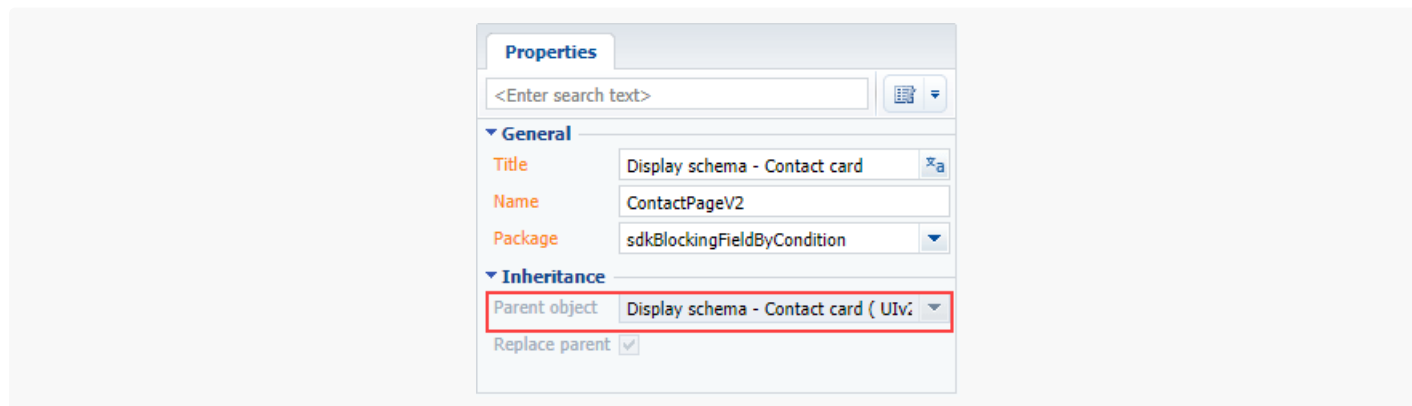
Алгоритм реализации примера

1. Создать схему замещающей модели представления страницы

КОНТАКТА

Создайте схему замещающей модели представления, в которой в качестве родительского объекта укажите схему [*Схема отображения карточки контакта*] ([*Display schema — Contact card*]) (рис. 1). Процесс создания схемы замещающей модели представления описан в статье "[Создать клиентскую схему](#)".

Рис. 1. — Свойства схемы замещающей модели представления страницы записи



2. В свойство rules модели представления страницы добавить правило

Добавьте в свойство `rules` модели представления страницы правило с типом `BINDPARAMETER` для колонки `Phone` ([*Рабочий телефон*]). Свойству `property` правила установите значение `BusinessRuleModule.enums.Property.ENABLED`. В массив `conditions` добавьте условие выполнения правила — значение колонки `MobilePhone` ([*Мобильный телефон*]) модели не должно быть пустым.

Исходный код схемы замещающей модели представления:

```
// В список зависимостей модуля добавить модуль BusinessRuleModule.
define("ContactPageV2", ["BusinessRuleModule"], function(BusinessRuleModule) {
    return {
        // Название схемы объекта страницы записи.
        entitySchemaName: "Contact",
        // Правила модели представления страницы записи.
        rules: {
            // Набор правил для колонки [Рабочий телефон] модели представления.
            "Phone": {
                // Зависимость доступности поля [Рабочий телефон] от значения поля [Мобильный телефон].
                "BindParameterEnabledPhoneByMobile": {
                    // Тип правила BINDPARAMETER.
                    "ruleType": BusinessRuleModule.enums.RuleType.BINDPARAMETER,
                    // Правило регулирует свойство ENABLED.
                    "property": BusinessRuleModule.enums.Property.ENABLED,
                    // Массив условий, при выполнении которых срабатывает правило.
                    // Определяет установлено ли значение в поле [Мобильный телефон].
                    "conditions": [{
```

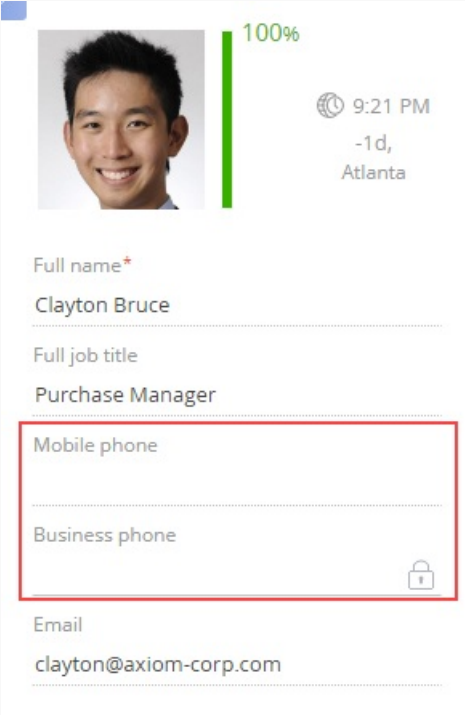
```

// Выражение левой части условия.
"leftExpression": {
    // Тип выражения – атрибут (колонка) модели представления.
    "type": BusinessRuleModule.enums.ValueType.ATTRIBUTE,
    // Название колонки модели представления, значение которой срав-
    "attribute": "MobilePhone"
},
// Тип операции сравнения – не равно.
"comparisonType": Terrasoft.ComparisonType.NOT_EQUAL,
// Выражение правой части условия.
"rightExpression": {
    // Тип выражения – константное значение.
    "type": BusinessRuleModule.enums.ValueType.CONSTANT,
    // Значение, с которым сравнивается выражение левой части.
    "value": ""
}
}]
}
}
};
});

```

После сохранения схемы и обновления веб-страницы приложения на странице контакта поле [*Рабочий телефон*] ([*Business phone*]) будет недоступно для редактирования до тех пор, пока в поле [*Мобильный телефон*] ([*Mobile phone*]) не будут введены данные (рис. 2).

Рис. 2. — Демонстрация результата выполнения примера



100%

9:21 PM
-1d,
Atlanta

Full name*

Clayton Bruce

Full job title

Purchase Manager

Mobile phone

Business phone

Email

clayton@axiom-corp.com

Скрыть поле по условию

 Средний

На заметку.

В Creatio реализована возможность настраивать бизнес-правила не только средствами разработки, но и с помощью мастера разделов. Подробнее об этом читайте в статье ["Настройка бизнес-правил"](#).

Описание примера

Добавить на страницу активности новое поле [Место встречи] ([Meeting place]), которое будет отображаться только для активностей категории "Встреча" ("Meeting").

На заметку.

Добавление поля на страницу записи может быть реализовано двумя способами — с помощью Мастера разделов и вручную.

Процесс добавления поля на страницу записи описан в статье ["Добавление нового поля"](#).

Исходный код

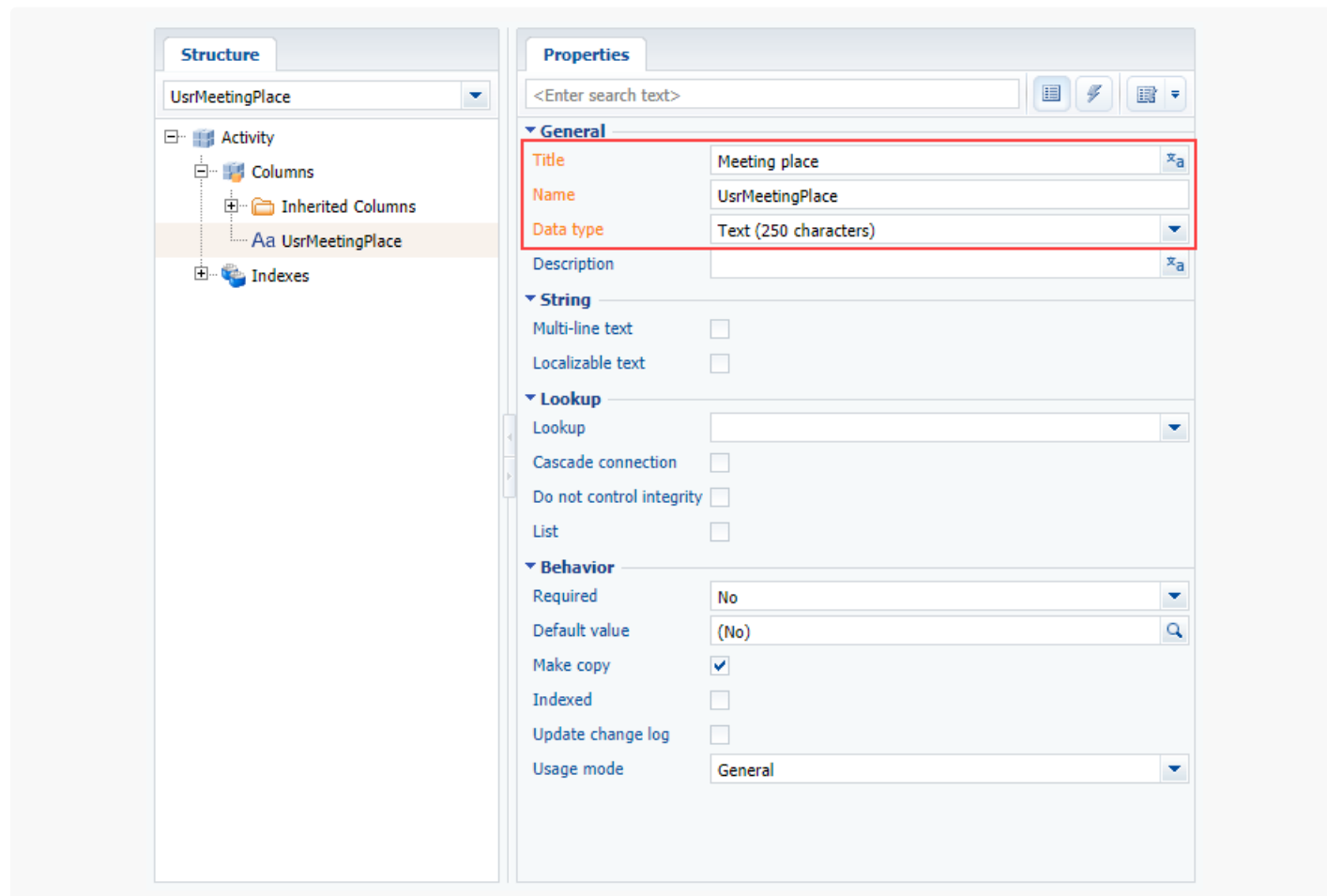
Пакет с реализацией примера можно скачать по [ссылке](#).

Алгоритм реализации примера

1. Создать схему замещающего объекта и добавить в него новую колонку

Создайте схему замещающего объекта [*Активность*] ([*Activity*]) и добавьте в нее новую колонку [*Место встречи*] ([*Meeting place*]) типа "строка" (рис. 1). Процесс создания схемы замещающего объекта описан в статье "[Создать схему объекта](#)".

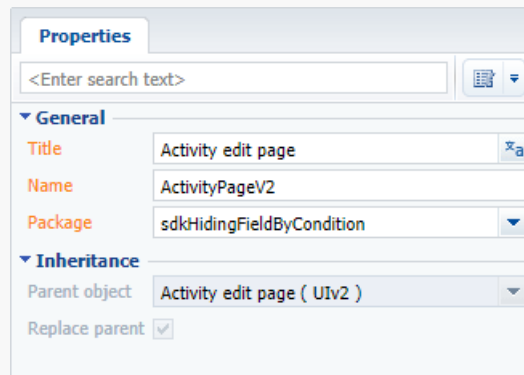
Рис. 1. — Добавление пользовательской колонки в схему замещающего объекта



2. Создать схему замещающей модели представления страницы активности

Создайте схему замещающей модели представления, в которой в качестве родительского объекта укажите схему [*Страница редактирования активности*] ([*Activity edit page*]) (рис. 2). Процесс создания схемы замещающей модели представления описан в статье "[Создать клиентскую схему](#)".

Рис. 2. — Свойства схемы замещающей модели представления страницы записи

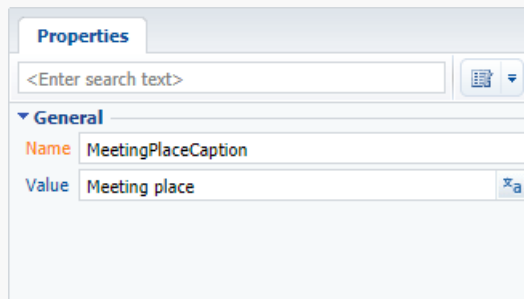


3. Добавить новое поле на страницу активности

Добавьте в массив `diff` конфигурационный объект с настройками расположения поля [*Место встречи*] ([*Meeting place*]) на странице. Добавление поля подробно рассмотрено в статье "[Добавление нового поля](#)".

Для возможности локализации приложения, создайте локализуемую строку (рис. 3) и свяжите ее с заголовком поля.

Рис. 3. — Свойства локализуемой строки



4. В свойство `rules` модели представления страницы добавить правило

Добавьте в свойство `rules` модели представления страницы правило `BINDPARAMETER` для колонки `UsrMeetingPlace`. Свойству `property` правила установите значение `BusinessRuleModule.enums.Property.VISIBLE`. В массив `conditions` добавьте условие выполнения правила — значение колонки `ActivityCategory` модели должно быть равно конфигурационной константе `ConfigurationConstants.Activity.ActivityCategory.Meeting`.

На заметку.

Конфигурационная константа `ConfigurationConstants.Activity.ActivityCategory.Meeting` содержит идентификатор записи "Встреча" справочника [*Категория активности*].

Исходный код схем замещающей модели представления:

```

// В список зависимостей модуля добавить модуль BusinessRuleModule и ConfigurationConstants.
define("ActivityPageV2", ["BusinessRuleModule", "ConfigurationConstants"],
    function(BusinessRuleModule, ConfigurationConstants) {
        return {
            // Название схемы объекта страницы записи.
            entitySchemaName: "Activity",
            // Отображение нового поля на странице записи.
            diff: /**SCHEMA_DIFF*/[
                // Метаданные для добавления поля [Место встречи].
                {
                    // Операция добавления компонента на страницу.
                    "operation": "insert",
                    // Мета-имя родительского контейнера, в который добавляется поле.
                    "parentName": "Header",
                    // Поле добавляется в коллекцию компонентов
                    // родительского элемента.
                    "propertyName": "items",
                    // Имя колонки схемы, к которой привязан компонент.
                    "name": "UsrMeetingPlace",
                    "values": {
                        // Заголовок поля.
                        "caption": {"bindTo": "Resources.Strings.MeetingPlaceCaption"},
                        // Расположение поля.
                        "layout": { "column": 0, "row": 5, "colSpan": 12 }
                    }
                }
            ]/**SCHEMA_DIFF*/,
            // Правила модели представления страницы редактирования.
            rules: {
                // Набор правил для колонки [Место встречи] модели представления.
                "UsrMeetingPlace": {
                    // Зависимость видимости поля [Место встречи] от значения в поле [Категор
                    "BindParametrVisibilePlaceByType": {
                        // Тип правила BINDPARAMETER.
                        "ruleType": BusinessRuleModule.enums.RuleType.BINDPARAMETER,
                        // Правило регулирует свойство VISIBLE.
                        "property": BusinessRuleModule.enums.Property.VISIBLE,
                        // Массив условий, при выполнении которых срабатывает правило.
                        // Определяет равно ли значение в колонке [Категория] значению "Встр
                        "conditions": [{
                            // Выражение левой части условия.
                            "leftExpression": {
                                //Тип выражения – атрибут (колонка) модели представления.
                                "type": BusinessRuleModule.enums.ValueType.ATTRIBUTE,
                                // Название колонки модели представления, значение которой с
                                "attribute": "ActivityCategory"
                            },
                            // Тип операции сравнения.

```

```

"comparisonType": Terrasoft.ComparisonType.EQUAL,
// Выражение правой части условия.
"rightExpression": {
    // Тип выражения — константное значение.
    "type": BusinessRuleModule.enums.ValueType.CONSTANT,
    // Значение, с которым сравнивается выражение левой части.
    "value": ConfigurationConstants.Activity.ActivityCategory.Me
}
}
}
}
}
};
});

```

После сохранения схемы и обновления веб-страницы приложения на странице активности поле [Место встречи] ([Meeting place]) будет отображаться только в том случае, если категория активности — "Встреча" ("Meeting") (рис.4, 5).

Рис. 4. — Результат выполнения примера. Тип активности — "Выполнить" ("To do"), поле [Место встречи] ([Meeting place]) скрыто

Subject * Prepare quotation			
Start *	1/11/2018	6:30 PM	
Due *	1/11/2018	8:00 PM	
Status *	Not started		
Show in calendar	☒		
Owner *	John Best		
Reporter *	John Best		
Priority *	Medium		
Category *	To do		

Рис. 5. — Результат выполнения примера. Тип активности — "Встреча" ("Meeting"), поле [Место встречи] ([Meeting place]) отображено

Subject * Prepare quotation			
Start *	1/11/2018	6:30 PM	
Due *	1/11/2018	8:00 PM	
Status *	Not started		
Show in calendar	☒		
Owner *	John Best		
Reporter *	John Best		
Priority *	Medium		
Category *	Meeting		
Meeting place	office		

Побавити автоматизацию к полю

ДОБАВИТЬ АВТОНУМЕРАЦИЮ К ПОЛЮ



Описание примера

В разделе [*Продукты*] установить автогенерацию для поля [*Код*] страницы записи. Код продукта должен формироваться следующим образом — ART_00001, ART_00002 и т. д.

Важно.

Ниже представлено два алгоритма выполнения примера — на стороне клиента и на стороне сервера. Задача добавления автонумерации реализуется одним из способов.

Исходный код

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

Важно.

Пакет не содержит привязанных системных настроек `ProductCodeMask` и `ProductLastNumber`. Их необходимо добавить вручную.

Алгоритм реализации кейса на стороне клиента

1. Создать две системные настройки

Создайте системную настройку [*Маска кода продукта*] с маской номера "ART_{0:00000}" (рис. 1). Укажите значения для следующих полей:

- [Название] ([Name]) — "Маска кода продукта" ("Product code mask").
- [Код] ([Code]) — "ProductCodeMask".
- [Тип] ([Type]) — строка, длина которой зависит от количества символов в маске. В большинстве случаев достаточно строки длиной 50 символов. В этом примере используется строка неограниченной длины.
- [Значение по умолчанию] ([Default value]) — "ART_{0:00000}".

Рис. 1. — Системная настройка [*Маска кода продукта*]

Product code mask

What can I do for you? >

Creatio

SAVE CANCEL

Name* Product code mask

Type* Unlimited length text

Default value ART_{0:00000}

Code* ProductCodeMask

Cached ☒

Personal ☐

Allow for portal users ☐

Description

Создайте системную настройку [Текущий код продукта] (рис. 2). Укажите:

- [Название] ([Name]) — "Текущий код продукта" ("Product last number").
- [Код] ([Code]) — "ProductLastNumber";
- [Тип] ([Type]) — "Целое число" ("Integer").

Рис. 2. — Системная настройка [Текущий код продукта]

Product last number

What can I do for you? >

Creatio 7.15.1.1293

SAVE CANCEL

Name* Product last number

Type* Integer

Default value

Description

Code* ProductLastNumber

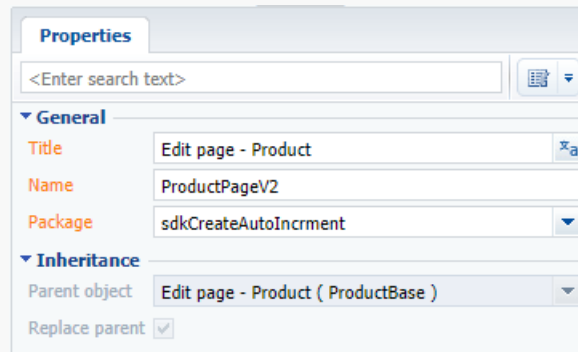
Cached ☐

Save value for current user ☐

2. В пользовательском пакете создать схему замещающей модели представления

Создайте схему замещающей модели представления, в которой в качестве родительского объекта укажите схему `ProductPageV2` (рис. 3). Процесс создания схемы замещающей модели представления описан в статье ["Создать клиентскую схему"](#).

Рис. 3. — Свойства схемы замещающей модели представления страницы продукта



3. Переопределить метод onEntityInitialized()

В коллекции методов модели представления страницы записи переопределите метод `onEntityInitialized()`. В методе `onEntityInitialized()` нужно вызвать метод `getIncrementCode()`, в callback-функции которого следует присвоить сгенерированный номер колонке `[Code]`. Исходный код схемы замещающей модели представления:

```
define("ProductPageV2", [], function() {
    return {
        // Название схемы объекта страницы записи.
        entitySchemaName: "Product",
        // Коллекция методов модели представления страницы.
        methods: {
            // Переопределение базового метода Terrasoft.BasePageV2.onEntityInitialized, который
            // срабатывает после окончания инициализации схемы объекта страницы записи.
            onEntityInitialized: function() {
                // Вызывается родительская реализация метода onEntityInitialized.
                this.callParent(arguments);
                // Код генерируется, только если создается новый элемент или копия существующего
                if (this.isAddMode() || this.isCopyMode()) {
                    // Вызов базового метода Terrasoft.BasePageV2.getIncrementCode, который гене
                    // по заданной ранее маске.
                    this.getIncrementCode(function(response) {
                        // Сгенерированный номер возвращается в колонку [Code].
                        this.set("Code", response);
                    });
                }
            }
        }
    };
});
```

После сохранения схемы и обновления страницы приложения с очисткой кэша, при добавлении нового продукта будет отображен автоматически сгенерированный код (рис. 4).

Рис. 4. — Результат выполнения кейса на стороне клиента

Алгоритм реализации кейса на стороне сервера

1. Создать две системные настройки

Этот шаг полностью идентичен первому шагу алгоритма реализации кейса на стороне клиента.

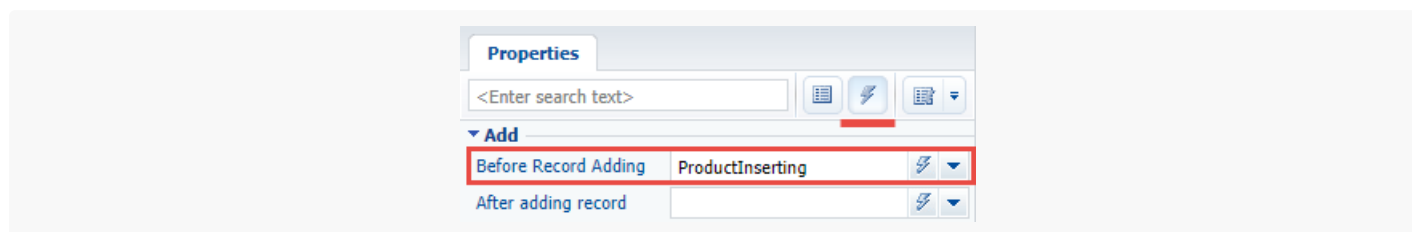
2. Создать схему замещающего объекта [Продукт]

Выберите пользовательский пакет и на вкладке [Схемы] выполните команду меню [Добавить] — [Замещающий объект]. В свойствах нового объекта нужно указать в качестве родительского объект [Продукт] ([Product]) (рис. 5).

Рис. 5. — Свойства схему замещающего объекта продукта

3. Добавить в схему замещающего объекта обработчик события [Перед добавлением записи]

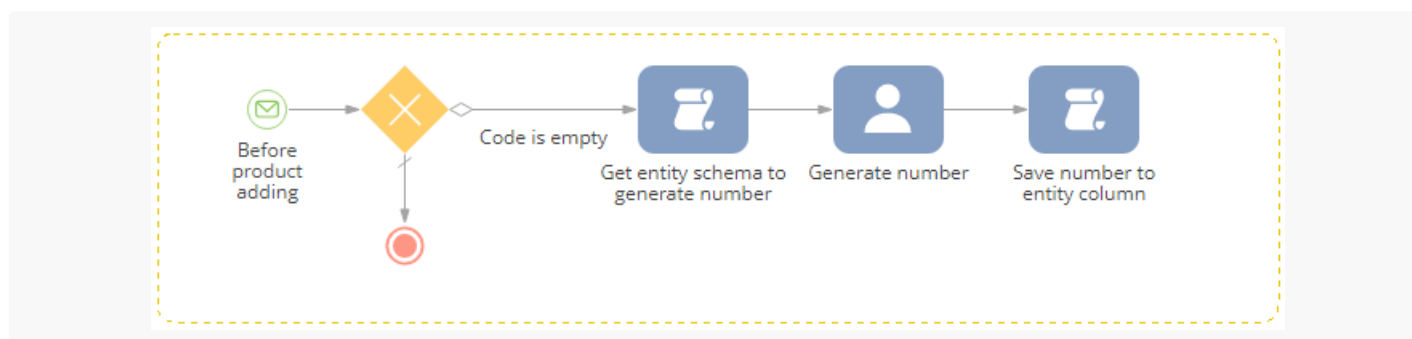
В свойствах объекта, отображающихся в дизайнера объектов, добавьте новый обработчик события. Для этого перейдите на вкладку событий и дважды кликните по полю [Перед добавлением записи] ([Before Record Adding]) либо кликните по пиктограмме события в этом поле (рис. 6).

Рис. 6. — Обработчик события [*Перед добавлением записи*] продукта

После этого откроется [дизайнер процессов](#) объекта.

4. Добавить событийный подпроцесс

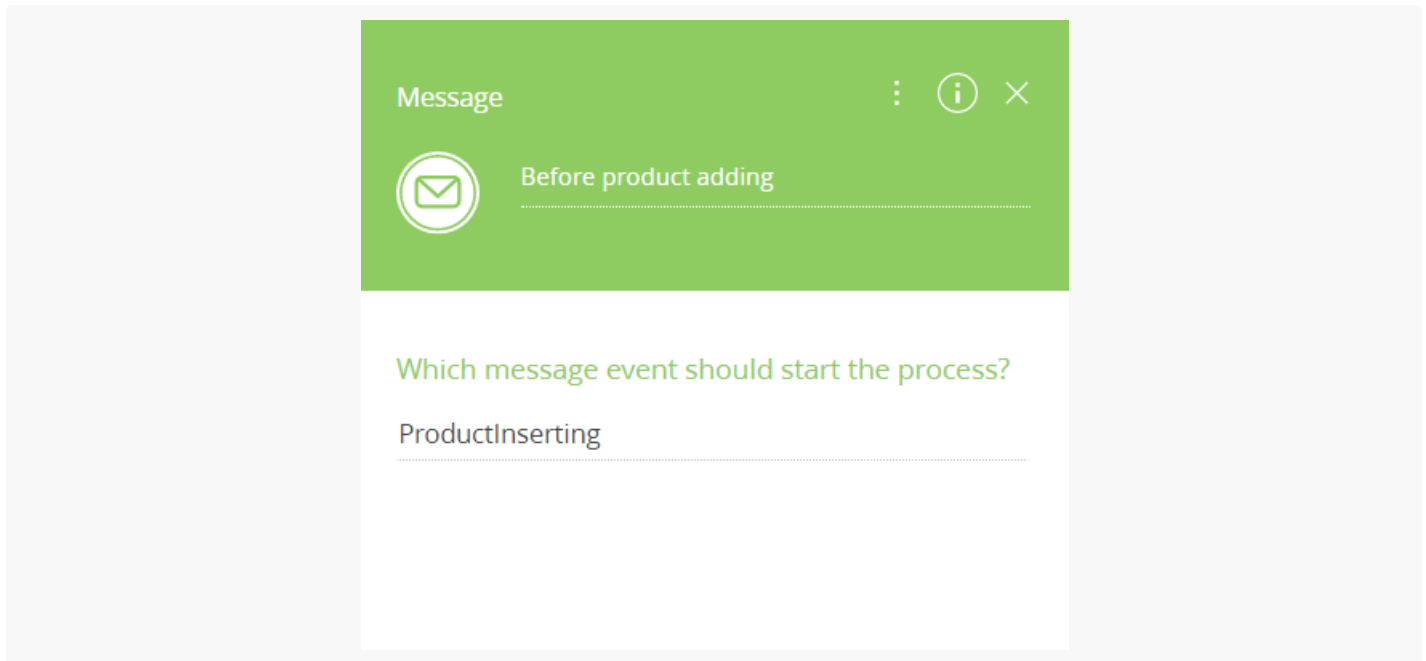
Для реализации обработчика события [*Перед добавлением записи*] в рабочую область дизайнера процессов объекта добавьте [событийный подпроцесс](#). В нем реализуйте бизнес-процесс добавления автогенерируемого номера (рис. 7).

Рис. 7. — Событийный подпроцесс — обработчик события [*Перед добавлением записи*]

Элементы событийного подпроцесса:

1. [Начальное сообщение](#) [*Перед добавлением продукта*] ([*Before product adding*]) (рис. 8) — подпроцесс будет запущен при получении сообщения `ProductInserting`, добавленном на шаге 3.

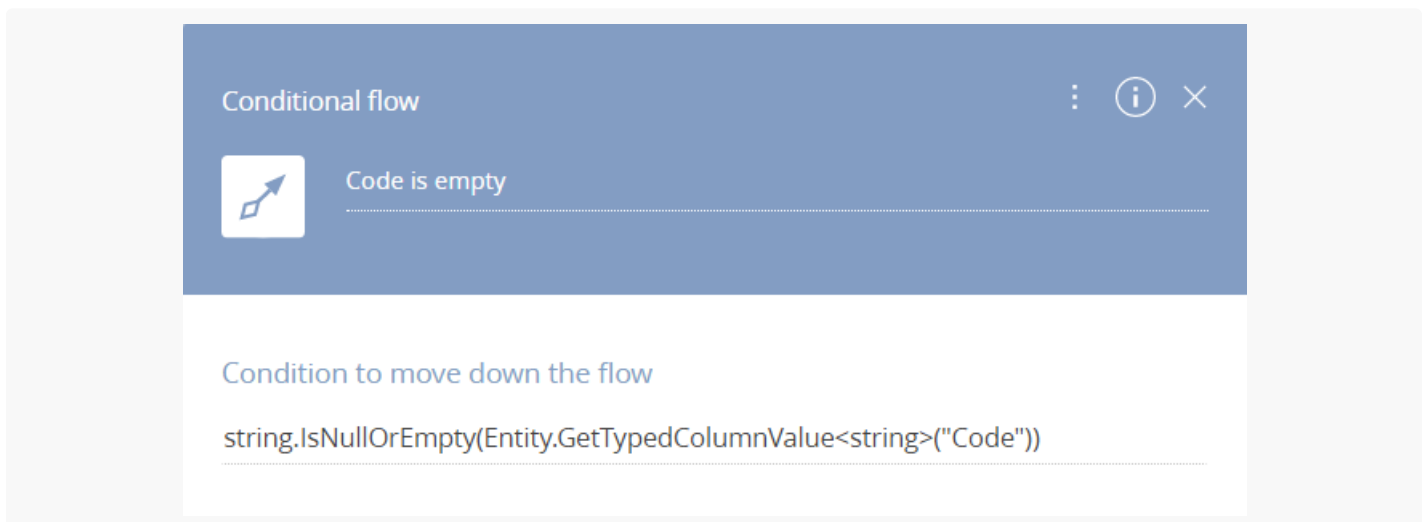
Рис. 8. — Свойства начального сообщения [*Перед добавлением продукта*]



2. [Логический оператор \[Исключающее ИЛИ\]](#), от которого исходят две ветви подпроцесса:

- [Поток по умолчанию](#) — переход по нему произойдет в случае, если не может быть выполнен переход по условному потоку. Эта ветвь завершается [событием \[Останов\]](#).
- [Условный поток](#) [Код не заполнен] ([Code is empty]), в котором проверяется, заполнена ли колонка [Code] (рис. 9). Дальнейшее выполнение подпроцесса возможно только в том случае, если колонка не заполнена.

Рис. 9. — Свойства условного потока [Код не заполнен]

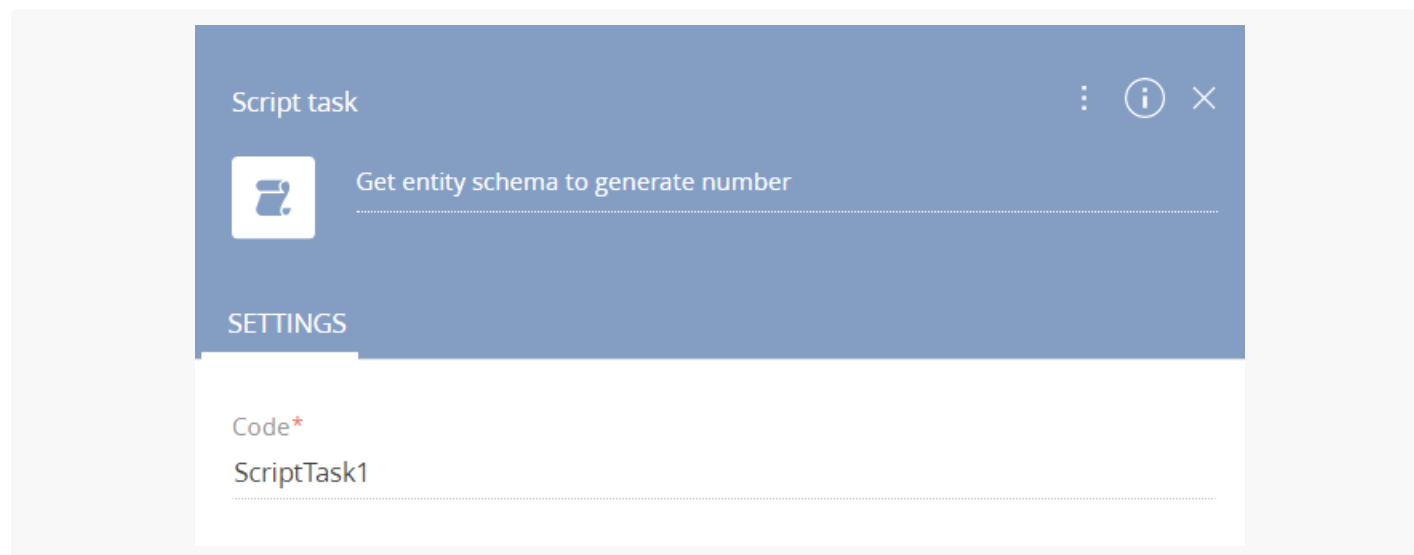


Код, который необходимо добавить в поле Условие условного потока

```
string.IsNullOrEmpty(Entity.GetTypedColumnValue<string>("Code"))
```

3. [Задание-сценарий](#) [*Определить схему объекта для генерации номера*] ([*Get entity schema to generate number*]) (рис. 10).

Рис. 10. — Свойства задания-сценария [*Определить схему объекта для генерации номера*]



В этом элементе выполняется программный код сценария C#. Для его добавления дважд..

```
// Установка схемы для генерации номера.
UserTask1.EntitySchema = Entity.Schema;
return true;
```

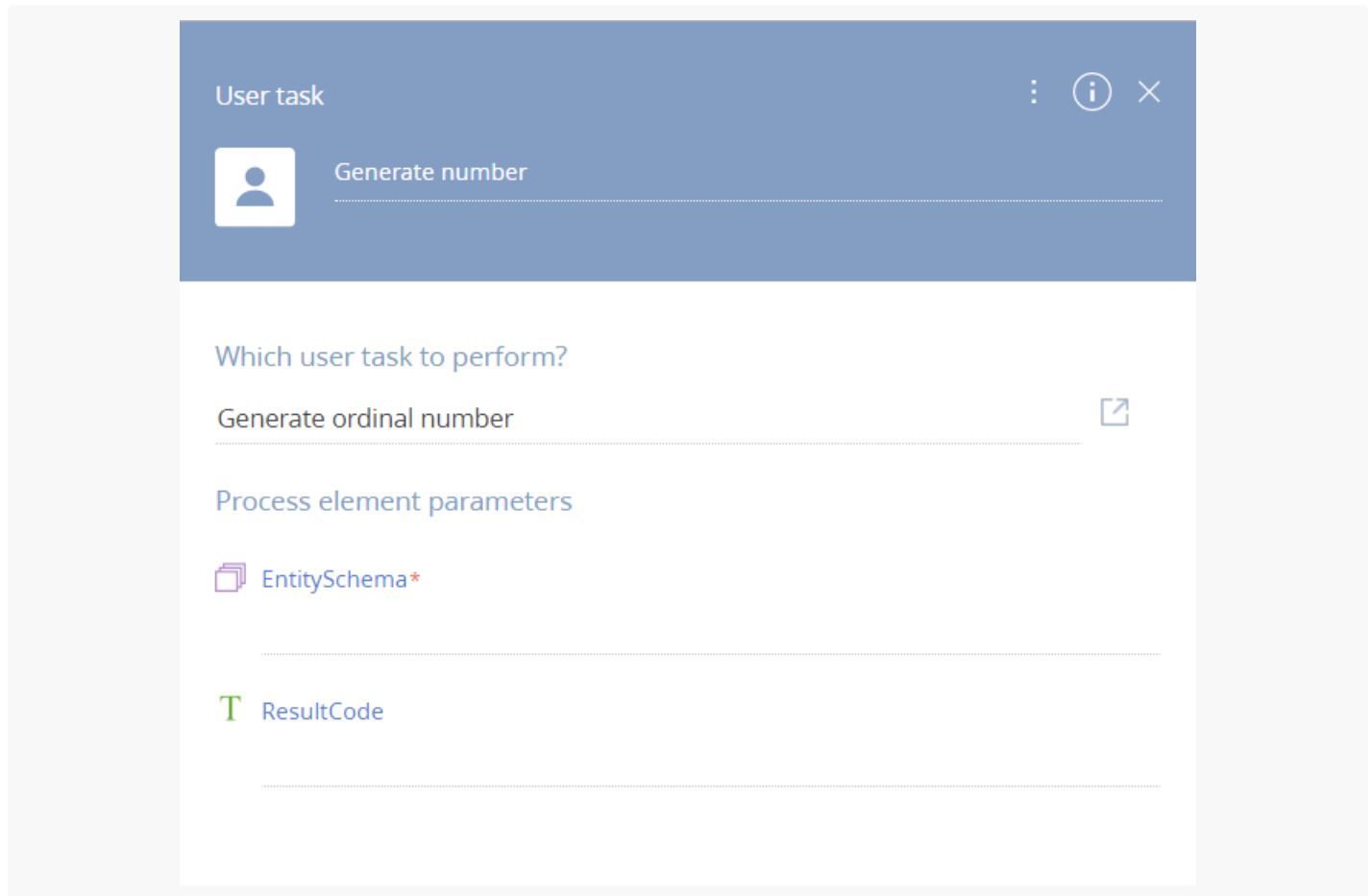
Здесь UserTask1 — это название пользовательского действия [*Выполнить генерацию номера*] ([*Generate number*]).

Важно.

После добавления исходного кода обязательно сохраните сценарий. Для этого на вкладке [*ScriptTask1*] выполните действие меню [*Сохранить*].

4. [Действие процесса](#) [*Выполнить генерацию номера*] ([*Generate number*]) (рис. 11).

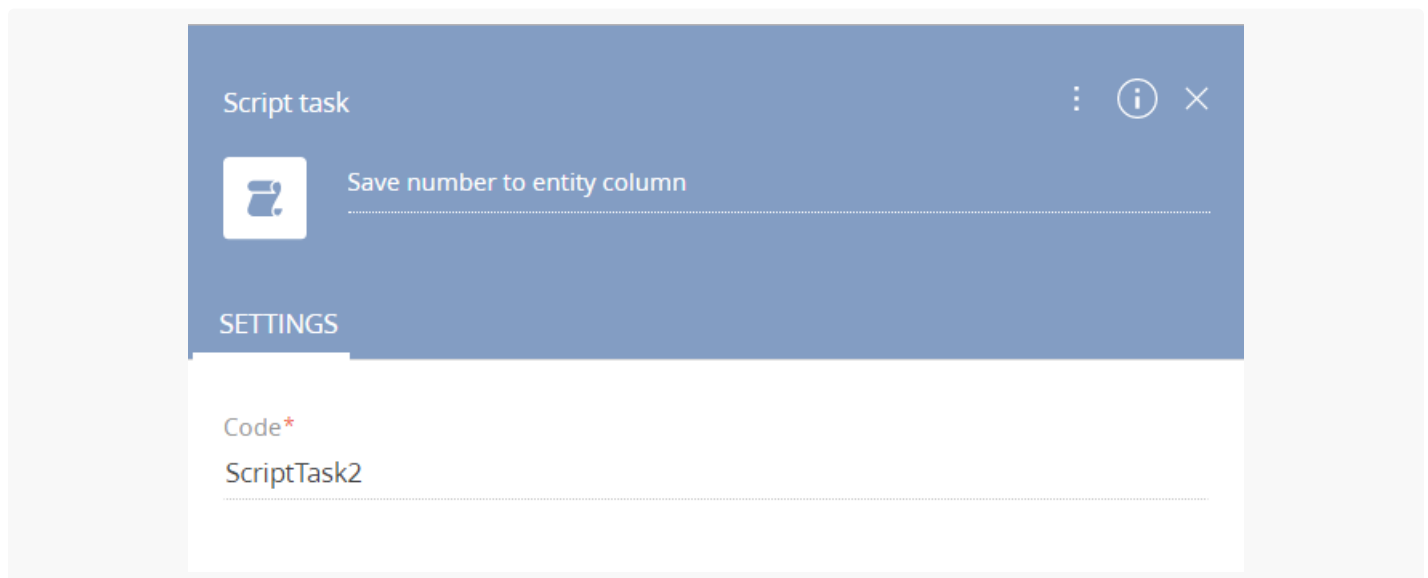
Рис. 11. — Свойства действия процесса [*Выполнить генерацию номера*]



В этом элементе выполняется системное действие [*Сгенерировать номер по порядку*] ([*Generate ordinal number*]). Именно системное действие [*Сгенерировать номер по порядку*] генерирует текущий порядковый номер в соответствии с установленной в системной настройке `ProductCodeMask` маской.

5. [Задание-сценарий](#) [*Записать полученный номер в колонку объекта*] ([*Save number to entity column*]) (рис. 12).

Рис. 12. — Свойства задания-сценария [*Записать полученный номер в колонку объекта*]



В этом элементе будет выполняться программный код сценария C#. Значение, которое ...

```
Entity.SetColumnValue("Code", UserTask1.ResultCode);
return true;
```

Сохраните и закройте дизайнер встроенного процесса, а затем опубликуйте схему объекта [*Продукт*]. В результате после сохранения нового продукта на его странице в поле [*Код*] будет указан автоматически сгенерированный код (рис. 13, рис. 14).

Важно.

Поскольку автогенерация кода и его сохранение в колонку выполняется на стороне сервера при возникновении события [*Перед сохранением записи*], то значение кода невозможно сразу же отобразить на странице добавления продукта. Это связано с тем, что событие [*Перед сохранением записи*] возникает на стороне сервера уже после отправки запроса на добавление записи из клиентской части приложения.

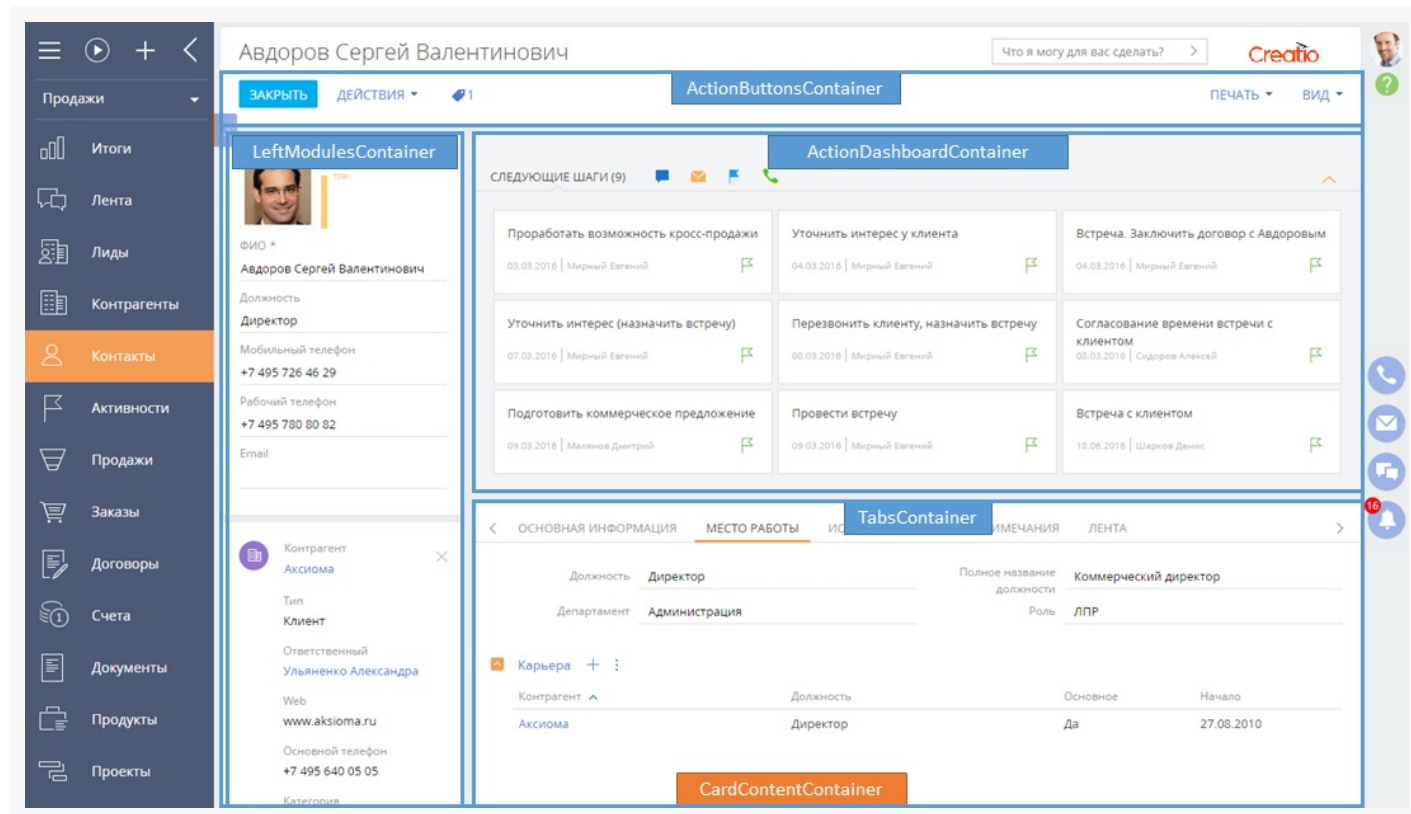
Рис. 13. — При создании продукта код не отображается

Рис. 14. — Код отображается в сохраненном продукте

Страница записи

Оснoвы

Страница записи — элемент интерфейса, предназначенный для работы с записями реестра. На странице хранится вся информация о бизнес-объектах в виде полей, вкладок и деталей, дашбордов. Страница именуется по названию соответствующих объектов системы (например, страница контрагента, страница контакта и т. д.). Она открывается при создании новой или редактировании существующей записи [реестра раздела](#). У каждого раздела есть одна или несколько страниц записей.



Каждой странице записи соответствует своя схема клиентского модуля. Например, конфигурирование страницы контакта выполнено в схеме `ContactPageV2` пакета `UIv2`. Все схемы страниц записи унаследованы от родительской схемы `BasePageV2` пакета `NUI`.

Элементы пользовательского интерфейса приложения, относящиеся к странице записи, размещены в соответствующих контейнерах, которые конфигурируются в базовой или унаследованной схеме страницы записи. К основным контейнерам страницы записи относятся:

- контейнер кнопок действий ([`ActionButtonsContainer`]), в котором размещены кнопки действий страницы записи;
- контейнер левой части страницы записи ([`LeftModulesContainer`]), в котором размещены основные поля ввода и редактирования данных;
- контейнер панели действий ([`ActionDashboardContainer`]), в котором размещены непосредственно панель действий и полоса стадий;
- контейнер вкладок ([`TabsContainer`]), содержащий вкладки с полями ввода и редактирования,

сгруппированными по какому-либо признаку, например, месту работы.

Если на страницу записи необходимо добавить пользовательские поля ввода и редактирования данных, то ее нужно заместить пользовательской страницей записи.

Настройка логики страницы записи

Одним из инструментов настройки логики страниц в Creatio являются бизнес-правила.

Бизнес-правила — это стандартный механизм системы, который позволяет настраивать поведение полей на странице путем конфигурирования колонок модели представления.

Бизнес-правила позволяют решать следующие типовые задачи:

- скрыть или отобразить поле;
- заблокировать или разблокировать поле;
- сделать поле обязательным или необязательным для заполнения;
- отфильтровать значения справочного поля в зависимости от значения в другом поле.

К основным элементам управления страницы относятся:

- поле ввода;
- кнопка;
- поле с изображением;
- кнопка выбора цвета;
- мультивалютное поле.

Creatio предоставляет возможность добавлять на страницу записи стандартные элементы управления, изменять их и создавать пользовательские элементы управления.

Средства конфигурирования страницы позволяют настраивать поведение существующих на странице элементов управления:

- [добавлять валидацию](#);
- [настраивать вычисляемые поля](#);
- [применять фильтрацию к справочным полям](#);
- [устанавливать для полей значения по умолчанию](#).

Добавить действие на страницу записи

В Creatio реализована возможность настройки списка действий из стандартного меню [*Действия*] на странице записи.

Список действий страницы представляет собой экземпляр класса `Terrasoft.BaseViewModelCollection`. Каждый элемент списка действий — это модель представления.

Настройка действия выполняется в конфигурационном объекте, в котором можно как явно задавать свойства модели представления действий, так и использовать базовый механизм привязки.

Наполнение меню [*Действия*] страницы записи реализовано в базовом классе страниц `BasePageV2`.

Защищенный виртуальный метод `getActions()` из схемы `BasePageV2` возвращает список действий раздела.

Добавление отдельного действия в коллекцию выполняется вызовом метода `addItem()`. В качестве параметра ему передается callback-метод `getButtonItem()`. Метод создает экземпляр модели представления действия по конфигурационному объекту, который передан ему в качестве параметра.

Базовая реализация добавления действия

```
/**
 * Возвращает коллекцию действий страницы записи
 * @protected
 * @virtual
 * @return {Terrasoft.BaseViewModelCollection} Возвращает коллекцию действий страницы записи
 */
getActions: function() {
    // Список действий – экземпляр Terrasoft.BaseViewModelCollection
    var actionMenuItems = this.Ext.create("Terrasoft.BaseViewModelCollection");
    // Добавление действия в коллекцию. В качестве callback-метода передается метод,
    // инстанцирующий экземпляр модели действия по переданному конфигурационному
    // объекту.
    actionMenuItems.addItem(this.getButtonItem({
        // Конфигурационный объект настройки действия.
        ...
    }));
    // Возвращает новую коллекцию действий.
    return actionMenuItems;
}
```

Свойства конфигурационного объекта действия страницы, который передается параметром в метод `getButtonItem()` представлены в таблице 1.

Табл. 1. — Свойства конфигурационного объекта

Свойство	Описание
Type	Тип элемента меню [<i>Действия</i>]. С помощью этого свойства можно добавить в меню действия горизонтальную линию для отделения блоков меню. Для этого в качестве значения свойства нужно указать строку <code>Terrasoft.MenuSeparator</code> . Если значение свойства не указано, то по умолчанию будет добавлен пункт меню.
Caption	Заголовок пункта меню [<i>Действия</i>]. Для задания заголовков рекомендуется использовать локализуемые строки.
Tag	В этом свойстве задается имя метода обработчика действия.
Enabled	Логическое свойство, регулирующее доступность пункта меню.
Visible	Логическое свойство, которое регулирует видимость пункта меню.

Последовательность добавления пользовательского действия

1. Создать схему замещающей модели представления страницы или схему модели представления страницы.
2. Переопределить метод `getActions()`.
3. Добавить действие в коллекцию действий методом `addItem()`.
4. В callback-метод `getButtonMenuItem()` передать конфигурационный объект с настройками добавляемого действия.

Важно. При замещении базовых страниц в методе `getActions()` замещающей схемы модели представления сначала нужно вызвать родительскую реализацию этого метода для инициализации действий родительской страницы. Для этого необходимо выполнить метод `this.callParent(arguments)`, который вернет коллекцию действий базовой страницы.

В версии 7.8 появилась возможность работы с часовыми поясами. На странице контакта добавлена информация о часовом поясе. Значение часовых поясов контакта рассчитывается автоматически. Подробнее о расчете часовых поясов контакта можно узнать из статьи "[Как узнать текущее время контакта](#)". Информация отображается в элементе, который формируется при помощи генератора представления. Данный элемент невозможно добавить на страницу при помощи дизайнера страницы. Для добавления элемента отображения часового пояса контакта на страницу пользователя необходимо:

1. Создать схему замещающей модели представления страницы.
2. Добавить элемент отображения часового пояса.
3. Подключить поиск часового пояса.
4. Настроить стили отображения.

Добавить действие на страницу записи

 Средний

Описание примера

На страницу заказа необходимо добавить действие [*Показать дату выполнения*] ([*Show execution date*]), которое отобразит в информационном окне планируемую дату выполнения заказа. Действие будет доступно только для заказов, находящихся на стадии [*Исполнение*] ([*In progress*]).

Важно. Пример актуален для кастомизации линейки продуктов Sales Creatio.

На заметку. Действие страницы записи применяется к конкретному объекту, который открыт на странице. Для доступа к значениям полей объекта страницы записи в методе-обработчике действия необходимо использовать методы модели представления: `get()` — для получения значения и `set()` — для установки значения.

Исходный код

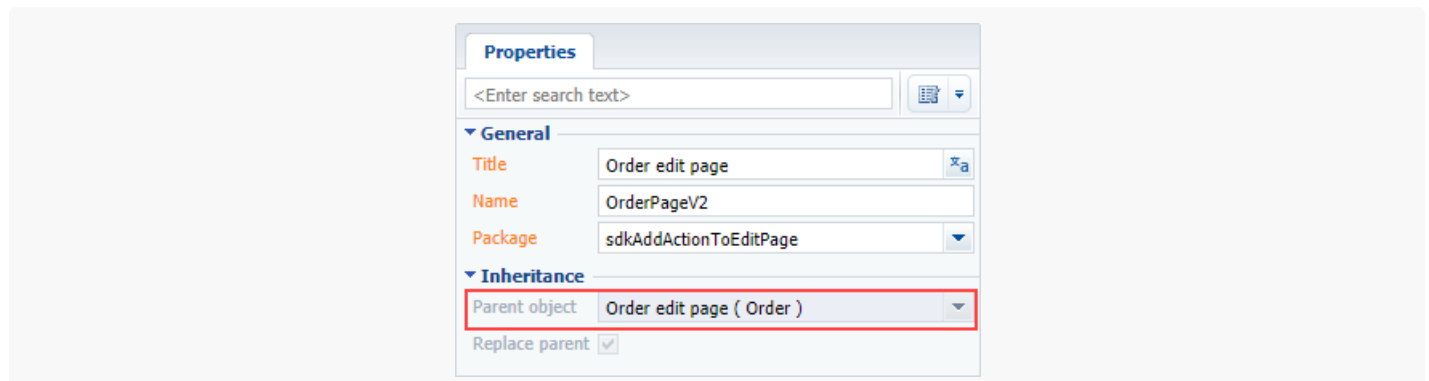
Пакет с реализацией примера можно скачать по [ссылке](#).

Алгоритм реализации примера

1. В пользовательском пакете создать схему замещающей модели представления страницы заказа

Создайте схему замещающей модели представления, в которой в качестве родительского объекта укажите схему [*Страница редактирования заказа*] ([*Order edit page*], `OrderPageV2`) (рис. 1). Процесс создания схему замещающей модели представления описан в статье "[Создать клиентскую схему](#)".

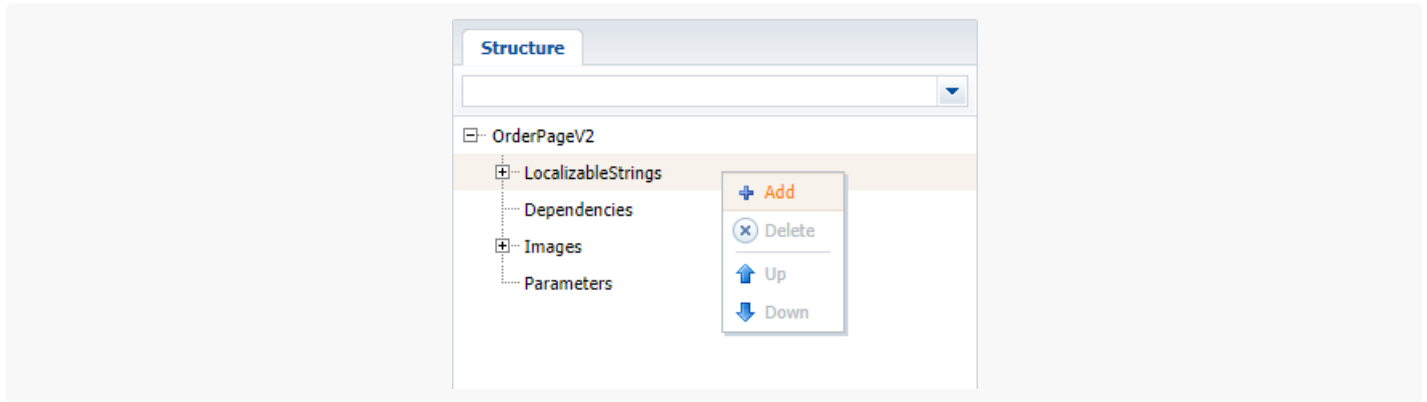
Рис. 1. — Свойства замещающей страницы записи



2. В коллекцию локализуемых строк схемы замещающей модели представления страницы добавить строку с заголовком пункта меню [Действия] ([Actions])

Создайте новую локализуемую строку (рис. 2).

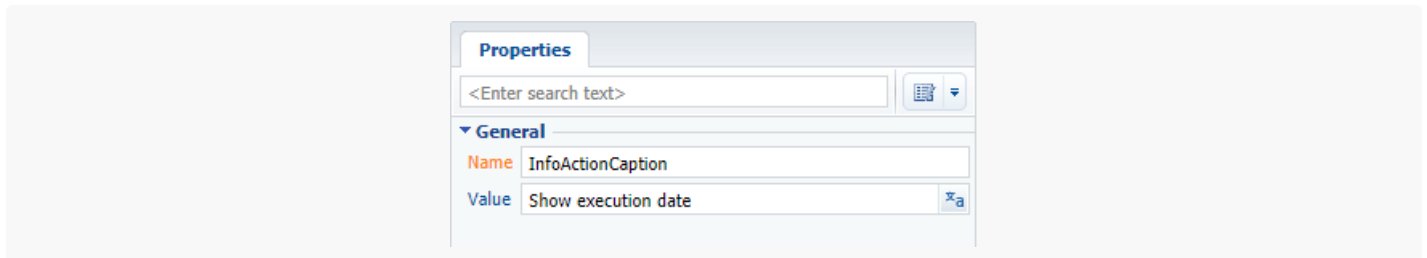
Рис. 2 — Добавление в схему локализуемой строки



Для созданной строки укажите (рис. 3):

- [Название] ([Name]) — "InfoActionCaption";
- [Значение] ([Value]) — "Показать дату выполнения" ("Show execution date").

Рис. 3. — Свойства пользовательской локализуемой строки



3. В коллекцию методов схемы замещающей модели представления страницы добавить реализацию методов

- `isRunning()` — проверяет, находится ли заказ на стадии [Исполнение] ([In progress]), и определяет доступность добавленного пункта меню.
- `showOrderInfo()` — метод-обработчик действия, который отображает в информационном окне планируемую дату завершения заказа;
- `getActions()` — переопределенный метод родительской схемы, возвращающий коллекцию действий страницы переопределенный.

Исходный код схемы замещающей модели представления:

```
define("OrderPageV2", ["OrderConfigurationConstants"], function(OrderConfigurationConstants) {
    return {
```

```

// Название схемы объекта страницы записи.
entitySchemaName: "Order",
// Методы модели представления страницы записи.
methods: {
    // Проверяет, на какой стадии находится заказ, для определения доступности пункта меню
    isRunning: function() {
        // Метод возвращает true, если статус заказа [Исполнение], иначе возвращает false
        if (this.get("Status")) {
            return this.get("Status").value === OrderConfigurationConstants.Order.OrdersStatusRunning;
        }
        return false;
    },
    // Метод-обработчик действия. Отображает в информационном окне дату выполнения заказа
    showOrderInfo: function() {
        // Получение даты выполнения заказа.
        var dueDate = this.get("DueDate");
        // Отображение информационного окна.
        this.showInformationDialog(dueDate);
    },
    // Переопределение базового виртуального метода, возвращающего коллекцию действий страницы
    getActions: function() {
        // Вызывается родительская реализация метода для получения
        // коллекции проинициализированных действий базовой страницы.
        var actionMenuItems = this.callParent(arguments);
        // Добавление линии-разделителя.
        actionMenuItems.addItem(this.getButtonMenuItem({
            Type: "Terrasoft.MenuSeparator",
            Caption: ""
        }));
        // Добавление пункта меню в список действий страницы записи.
        actionMenuItems.addItem(this.getButtonMenuItem({
            // Привязка заголовка пункта меню к локализуемой строке схемы.
            "Caption": {bindTo: "Resources.Strings.InfoActionCaption"},
            // Привязка метода-обработчика действия.
            "Tag": "showOrderInfo",
            // Привязка свойства доступности пункта меню к значению, которое возвращает
            "Enabled": {bindTo: "isRunning"}
        }));
        return actionMenuItems;
    }
}
};
});

```

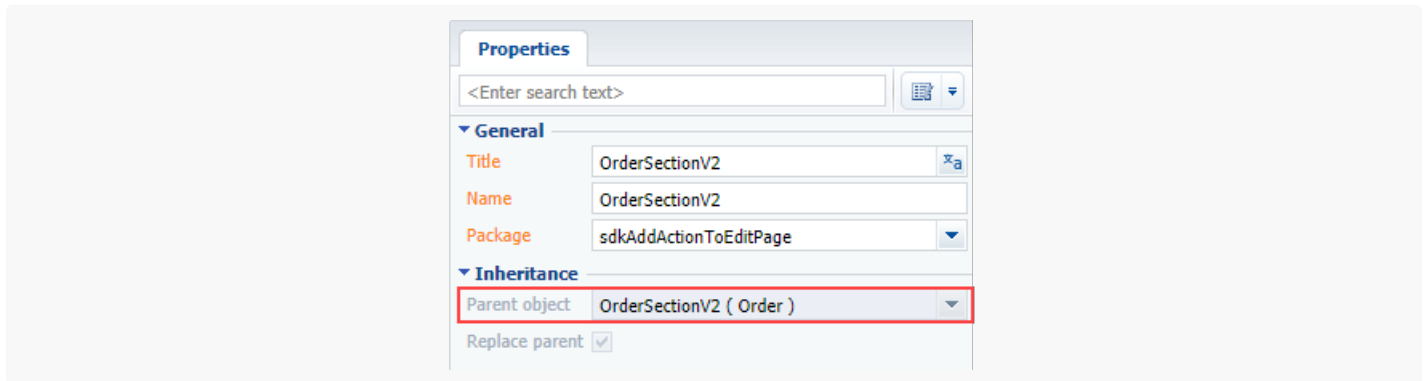
Важно. Предыдущих шагов достаточно для добавления действия на страницу записи. Но в режиме вертикального представления реестра пользовательское действие страницы отображаться не будет.

Для корректного отображения действия страницы добавьте локализуемую строку заголовка и метод, определяющий доступность пункта меню, в схему модели представления раздела.

4. Создать схему замещающей модели представления раздела [Заказы] ([Orders])

Создайте замещающий клиентский модуль, в котором в качестве родительского объекта укажите схему `OrderSectionV2` (рис. 4). Процесс создания замещающей страницы описан в статье ["Создать клиентскую схему"](#).

Рис. 4. — Свойства схему замещающей модели представления страницы раздела [Заказы]



5. Добавить локализуемую строку с заголовком пункта меню [Действия] ([Actions])

Повторите действия из пункта 2.

6. Добавить реализацию метода

В коллекцию методов схемы модели представления раздела добавьте реализацию метода `isRunning()`, который будет проверять, находится ли выбранный заказ на стадии [Исполнение] ([In progress]), и определяет доступность добавленного пункта меню.

Исходный код схемы замещающей модели представления:

```
define("OrderSectionV2", ["OrderConfigurationConstants"], function(OrderConfigurationConstants)
    return {
        // Название схемы раздела.
        entitySchemaName: "Order",
        // Методы модели представления раздела.
        methods: {
            // Проверяет, на какой стадии находится заказ, для определения доступности пункта меню.
            isRunning: function(activeRowId) {
                activeRowId = this.get("ActiveRow");
                // Получение коллекции данных списочного представления реестра раздела.
                var gridData = this.get("GridData");
```

```

// Получение модели выбранного заказа по заданному значению первичной колонки.
var selectedOrder = gridData.get(activeRowId);
// Получение свойства модели – статуса выбранного заказа.
var selectedOrderStatus = selectedOrder.get("Status");
// Метод возвращает true, если статус заказа [Исполнение]. Иначе возвращает false
return selectedOrderStatus.value === OrderConfigurationConstants.Order.OrderStat

    }
}
};
});

```

После сохранения схемы и обновления страницы приложения с очисткой кэша, при выборе заказа, находящегося на стадии [*Исполнение*] ([*In progress*]), на странице заказа появится новое действие (рис. 5, 6).

Рис. 5. — Результат выполнения примера. Заказ на стадии [*Завершен*] ([*Completed*]) — действие неактивно

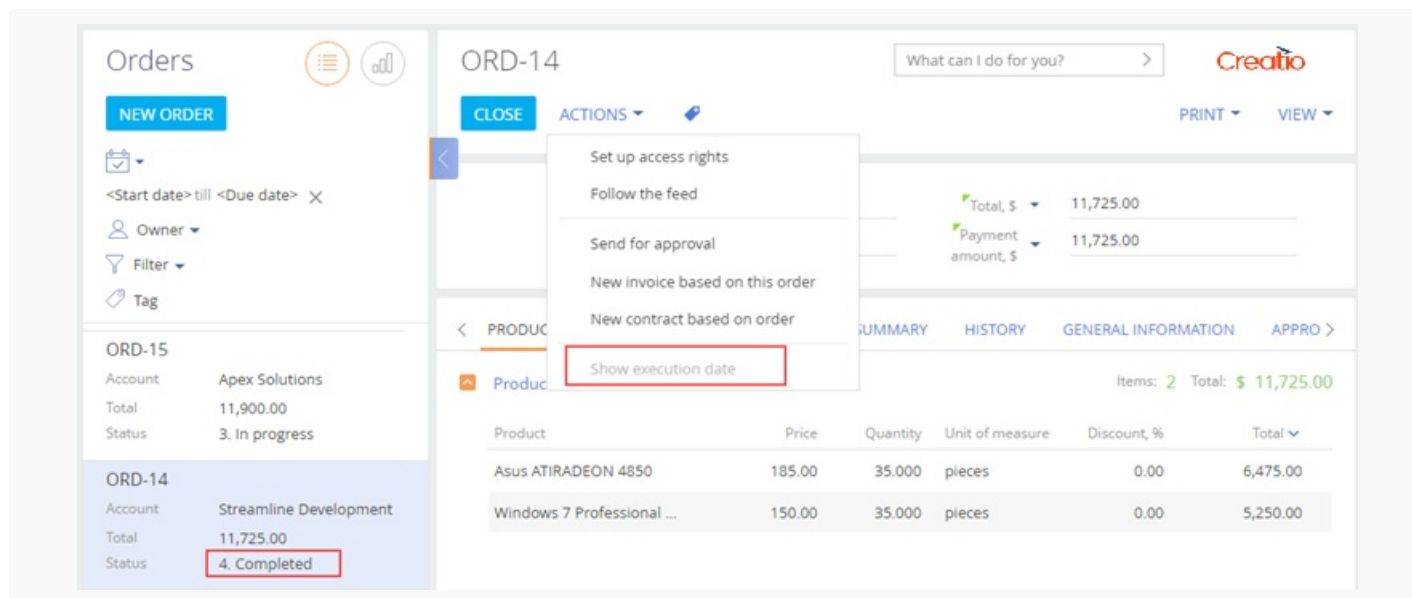
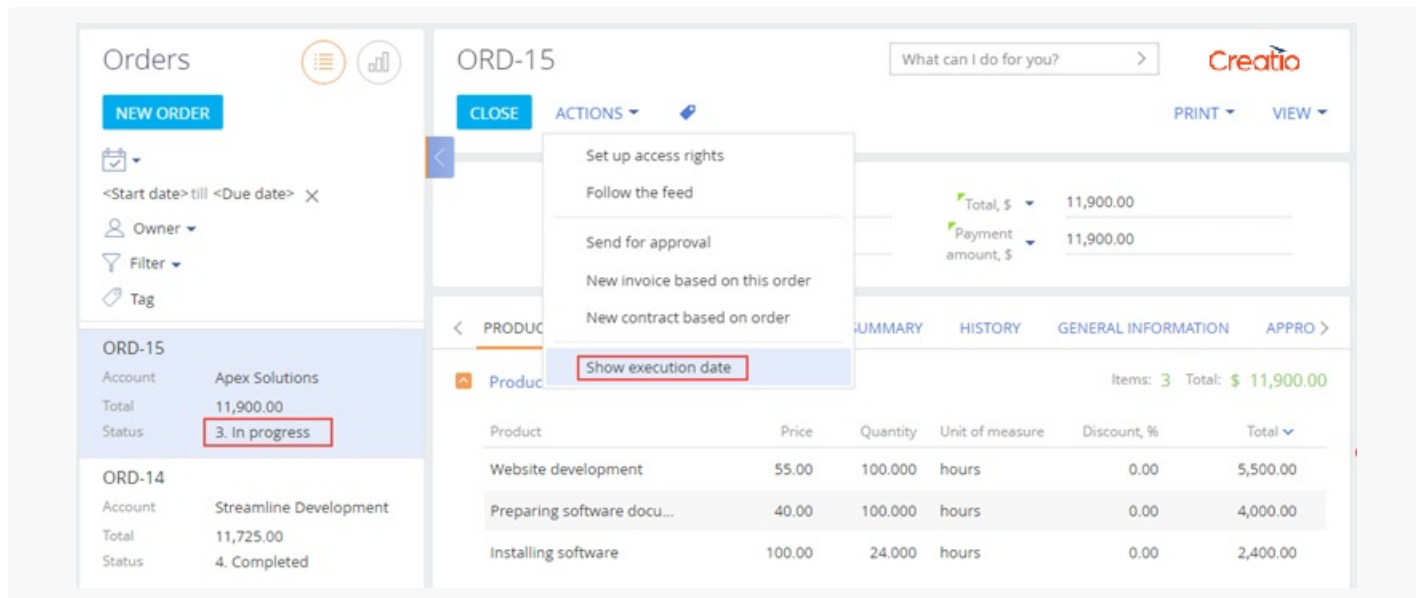


Рис. 6. — Результат выполнения примера. Заказ на стадии [*Исполнение*] ([*In progress*]) — действие активно



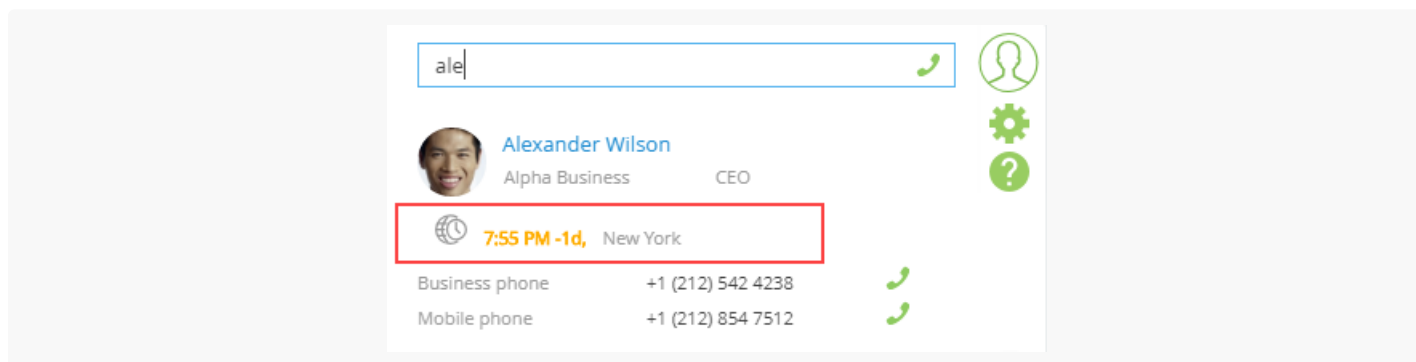
Отобразить часовой пояс контакта

 Сложный

Описание примера

Необходимо добавить элемент отображения часового пояса контакта на вкладку **Звонки** коммуникационной панели. Для удобства совершения звонков абонентам в другом часовом поясе, при поиске номера телефона необходимо отображать текущее время контакта (рис.1).

Рис. 1. — Отображение текущего времени абонента



Исходный код

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

Алгоритм реализации примера

1. Создать схему замещающей модели представления страницы

В пользовательском пакете создайте схему замещающей модели представления (рис. 2), для которого укажите родительский объект [*Схема найденного абонента*] ([*Found subscriber schema*], `SubscriberSearchResultItem`) (рис. 3). Процесс создания схему замещающей модели представления описан в статье "[Создать клиентскую схему](#)".

Рис. 2. — Создание схемы замещающей модели представления

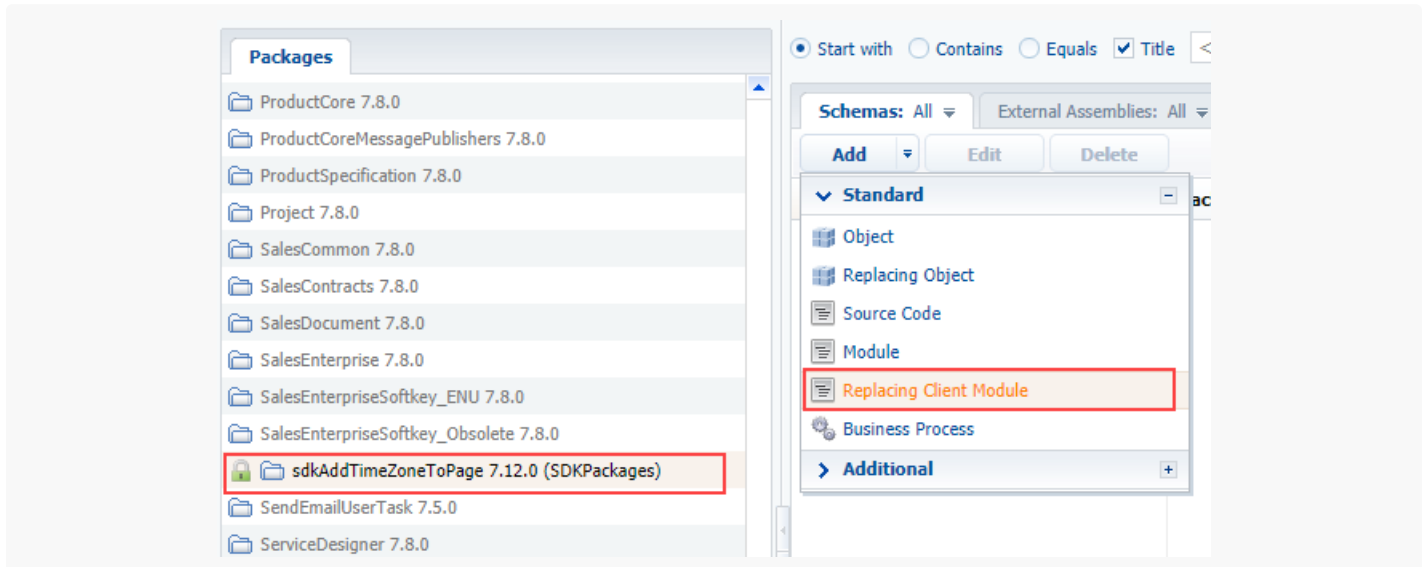
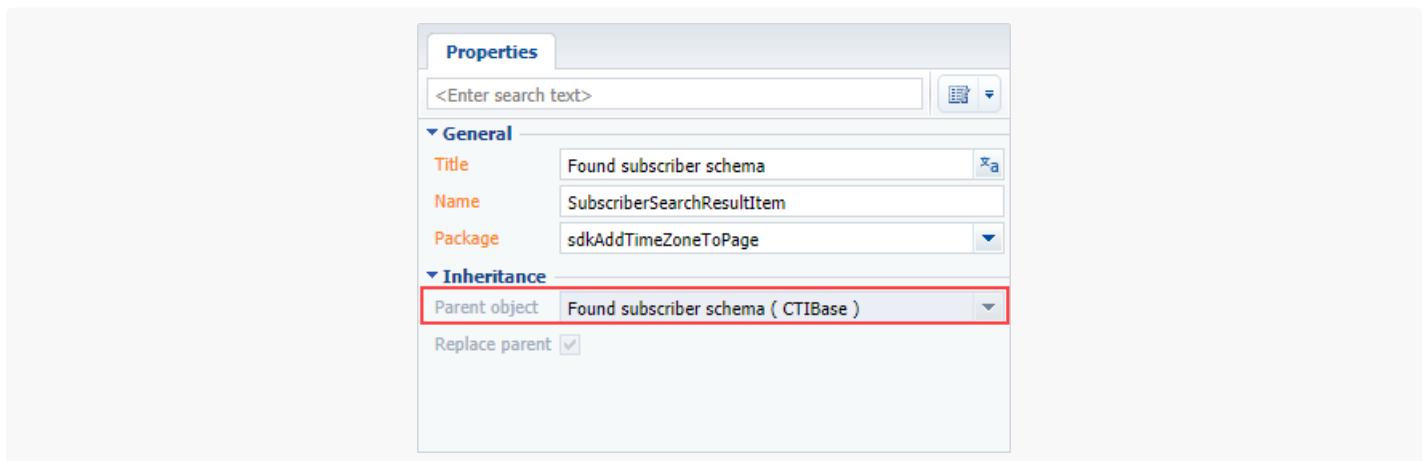


Рис. 3 — Свойства схемы замещающей модели представления



2. Добавить элемент отображения часового пояса

В зависимости схемы добавьте модули:

- `TimezoneGenerator` — модуль [*Генератор компонента "Часовой пояс"*], при помощи которого формируется элемент отображения часового пояса контакта.
- `TimezoneMixin` — миксин, который используется для поиска часового пояса контакта.

В массив модификаций `diff` добавьте конфигурационный объект для отображения элемента часового пояса контакта.

3. Подключить поиск часового пояса

Для запуска поиска часового пояса контакта в метод `init()` миксина `TimezoneMixin` передайте уникальный идентификатор контакта. В результате выполнения будут установлены атрибуты:

- `TimeZoneCaption` — название временной зоны контакта и текущее время.
- `TimeZoneCity` — название города, для которого определена временная зона.

Исходный код схемы:

```
// Объявление схемы.
define("SubscriberSearchResultItem",
  // Зависимость от TimezoneGenerator, TimezoneMixin.
  ["TimezoneGenerator", "TimezoneMixin",
  // Зависимость от модуля со стилями.
  "css!UsrSubscriberSearchResultItemCss"],
  function() {
    return {
      // Блок для создания атрибутов.
      attributes: {
        // Название атрибута, который отвечает за состояние отображения элемента час
        "IsShowTimeZone": {
          // Тип данных атрибута.
          "dataValueType": Terrasoft.DataValueType.BOOLEAN,
          // Тип атрибута в модели.
          "type": Terrasoft.ViewModelColumnType.VIRTUAL_COLUMN,
          // Значение по умолчанию.
          "value": true
        }
      },
      // Блок для подключения миксинов.
      mixins: {
        // Подключение миксина.
        TimezoneMixin: "Terrasoft.TimezoneMixin"
      },
      // Блок для определения методов.
      methods: {
        // Конструктор класса.
        constructor: function() {
          // Вызов базового конструктора.
          this.callParent(arguments);
          // Признак того, что абонент является контактом.
          var isContact = this.isContactType();
          // Если абонент – контакт, то элемент отображается.
          this.set("IsShowTimeZone", isContact);
          // Если абонент – контакт.
          if (isContact) {
            // Идентификатор контакта.
```

```

        var contactId = this.get("Id");
        // Поиск часового пояса контакта.
        this.mixins.TimezoneMixin.init.call(this, contactId);
    }
},
// Возвращает признак того, что абонент — контакт.
isContactType: function() {
    // Тип абонента.
    var type = this.get("Type");
    // Возвращает результат сравнения.
    return type === "Contact";
}
},
// Массив модификаций.
diff: [
    {
        // Добавление нового элемента.
        "operation": "insert",
        // Родительским элементом является SubscriberSearchResultItemContainer.
        "parentName": "SubscriberSearchResultItemContainer",
        // Новый элемент добавляется в коллекцию элементов родителя.
        "propertyName": "items",
        // Название элемента.
        "name": "TimezoneContact",
        // Свойства элемента.
        "values": {
            // Тип элемента.
            "itemType": Terrasoft.ViewItemType.CONTAINER,
            // Для формирования конфигурации представления вызывается метод генератора.
            "generator": "TimezoneGenerator.generateTimeZone",
            // Видимость контейнера привязывается к атрибуту.
            "visible": {"bindTo": "IsShowTimeZone"},
            // Стил элемент.
            "wrapClass": ["subscriber-data", "timezone"],
            // Заголовок привязываем к атрибуту.
            "timeZoneCaption": {"bindTo": "TimeZoneCaption"},
            // Город привязываем к атрибуту.
            "timeZoneCity": {"bindTo": "TimeZoneCity"}
        },
        // Позиция элемента в родительском контейнере.
        "index": 2
    }
]
};
});

```

На этом шаге уже отображается текущее время контакта и его город.

4. Настроить стили отображения

На предыдущих шагах в конфигурационном объекте, помещенном в массив `diff`, уже выполнена предварительная настройка для подключения стилей.

Позиционирование элемента выполняется при помощи свойства `index`. В контейнере `SubscriberSearchResultItemContainer` по умолчанию элементы расположены поочередно:

- на первом месте с индексом 0 отображается фотография абонента,
- на втором с индексом 1 — информация об абоненте,
- на третьем с индексом 2 — телефоны абонента.

После установки значения индекса 2, элемент отобразится между данными абонента и списком телефонных номеров.

Стили текстовых элементов в схеме определяются CSS-классом `subscriber-data`. Генератор элемента предоставляет свойство `wrapClass` для возможности управления стилями.

Для позиционирования элемента и его визуального выделения создайте в пользовательском пакете модуль `UsrSubscriberSearchResultItemCss`.

На вкладке LESS созданного модуля добавьте CSS-селекторы классов, определяющие нужные стили.

```
/* Настройка стилей для отображения добавляемого элемента.*/
.ctiPanelMain .search-result-items-list-container .timezone {
  /* Отступ сверху.*/
  padding-top: 13px;
  /* Смещение снизу.*/
  margin-bottom: -10px;
}
/* Настройка стилей для отображения времени контакта.*/
.ctiPanelMain .search-result-items-list-container .timezone-caption {
  /* Отступ слева.*/
  padding-left: 10px;
  /* Цвет текста.*/
  color: rgb(255, 174, 0);
  /* Шрифт текста — жирный.*/
  font-weight: bold;
}
/* Настройка стилей для отображения города контакта.*/
.ctiPanelMain .search-result-items-list-container .timezone-city {
  /* Отступ слева.*/
  padding-left: 10px;
}
```

Для загрузки модуля со стилями добавьте исходный код в модуль и сохраните схему.

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

```
return {};  
});
```

Добавьте модуль `UsrSubscriberSearchResultItemCss` в зависимости `SubscriberSearchResultItem`.

После сохранения схемы и обновления страницы приложения результаты поиска абонента будут отображаться так, как показано на рисунке 1.

Деталь

Оснoвы

Деталь — элемент интерфейса на странице записи, который отображает записи определенного объекта. Как правило, это записи, связанные с текущей записью. Например, на странице контакта детали используются для хранения информации о связанных с ним активностях, адресах, документах, и т. д. Большинство деталей имеют собственный реестр. Отдельные детали, например, [*Средства связи*], отображаются не в виде реестра. Визуально деталь отличается от группы полей наличием панели инструментов для управления данными (добавления и изменения записей, сортировки, фильтрации, настройки детали и других действий).

Детали предназначены для отображения дополнительных данных для основного объекта раздела. Детали раздела отображаются во вкладках страницы редактирования раздела в контейнере вкладок.

В зависимости от метода ввода и отображения данных, различают следующие типы деталей:

- Деталь с полями редактирования** — данные вводятся и редактируются непосредственно в полях данных, размещенных в детали. При необходимости в деталь можно добавить новое поле (1). Примером такой детали может служить деталь [*Средства связи*].

Деталь с полями — деталь, которая может включать несколько групп полей редактирования. Базовая

деталь с полями реализована в схеме `BaseFieldsDetail` пакета `BaseFinance`, доступного в банковских продуктах Creatio — Financial Services Creatio sales edition, customer journey edition и lending edition. Модель представления записи детали реализована в схеме `BaseFieldRowViewModel`.

Базовая деталь с полями редактирования позволяет выполнять следующие действия:

- Добавлять записи на деталь без сохранения страницы, на которой эта деталь размещена.
- Работать с деталью как со страницей записи.
- Использовать базовую валидацию полей с возможностью добавления пользовательской.
- Добавлять виртуальную запись.
- Расширять логику поведения записей.

На основе базовой детали с полями редактирования может быть создана пользовательская деталь. Для этого при [создании схемы](#) пользовательской детали следует унаследовать ее от схемы базовой детали.

- **Деталь со страницей добавления** — данные вводятся и редактируются на странице детали. К этому типу относится, например, деталь [*Адреса контакта*], данные каждого адреса которой вводятся и редактируются на странице [*Адрес контакта*].

Деталь со страницей добавления является стандартной деталью приложения Creatio и может быть добавлена в раздел при помощи мастера детали и мастера разделов.

Общая последовательность действий для добавления детали со страницей добавления в существующий раздел системы:

- Создать схему объекта детали.
- Создать схему модели представления реестра детали и схему модели представления страницы детали с помощью мастера детали.
- Разместить деталь на странице записи раздела с помощью мастера раздела.

Ниже будет рассмотрен способ добавления детали со страницей добавления без использования мастера деталей.

Для добавления пользовательской детали со страницей добавления в существующий раздел необходимо:

- Создать схему объекта детали.
- Создать схему реестра детали.
- Создать схему страницы записи детали.
- Выполнить настройку детали в схеме замещающей модели представления страницы раздела.
- Зарегистрировать связи между схемами объекта, реестра детали и страницы записи детали при помощи специальных SQL-запросов в системные таблицы.
- Выполнить настройку полей детали.

Важно.

Чтобы связать пользовательские схемы создаваемой детали, необходимо внести изменения в системные таблицы [SysModuleEdit], [SysModulentity] и [SysDetail] базы данных Creatio при помощи SQL-запросов.

Однако при составлении и выполнении SQL-запроса к базе данных следует быть предельно внимательным. Выполнение неправильно составленного SQL-запроса может привести к повреждению существующих данных и нарушению работы системы.

- **Деталь с редактируемым реестром** — данные отображаются в списочном виде, вводятся и редактируются непосредственно в реестре. Примером служит деталь [*Продукт в заказе*].

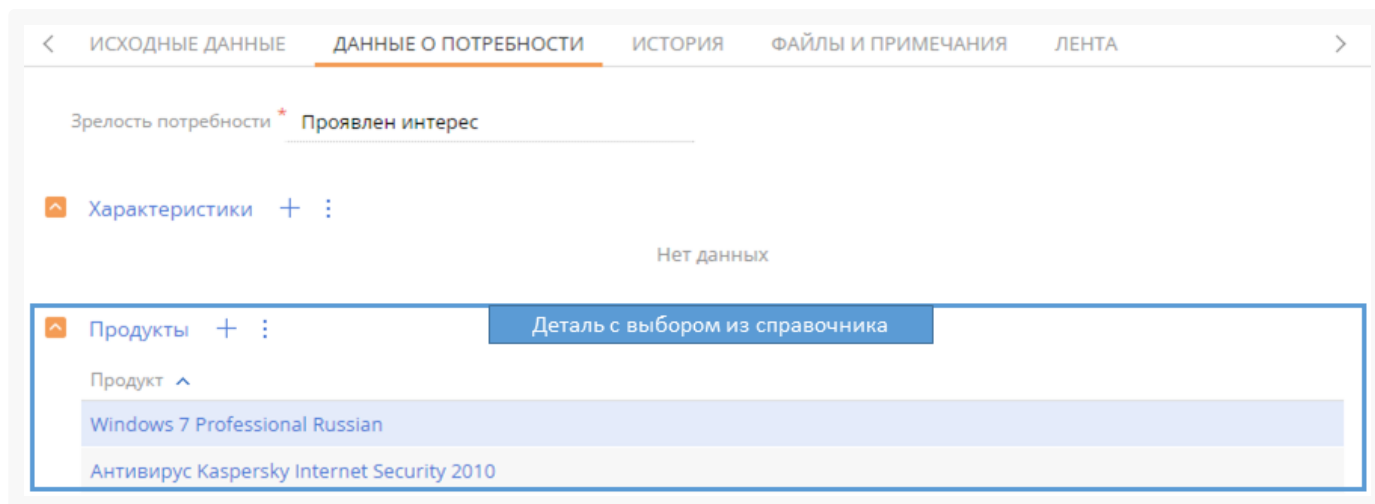
ПРОДУКТЫ ПАСПОРТ ЗАКАЗА ДОСТАВКА ИТОГИ ИСТОРИЯ ОБЩАЯ ИНФОРМАЦИЯ ВИЗЫ ФАЙЛЫ И ПРИМЕЧАНИЯ ЛЕНТА						
Продукты + :		Деталь с редактируемым реестром			Позиций: 3 на сумму: руб. 349 200,00	
Продукт	Цена	Количество	Единица измерения	Скидка, %	Итого	
Intel Core i7-4930K	21 900,00	10,00	штук	0,00	219 000,00	
Intel Core i5-4670K	8 700,00	10,00	штук	0,00	✓ ↺ 🗑	
Документирование ПО	1 200,00	36,00	часов	0,00	43 200,00	

Поскольку деталь с редактируемым реестром не является стандартной деталью приложения Creatio, то она не может быть добавлена в раздел при помощи мастера деталей.

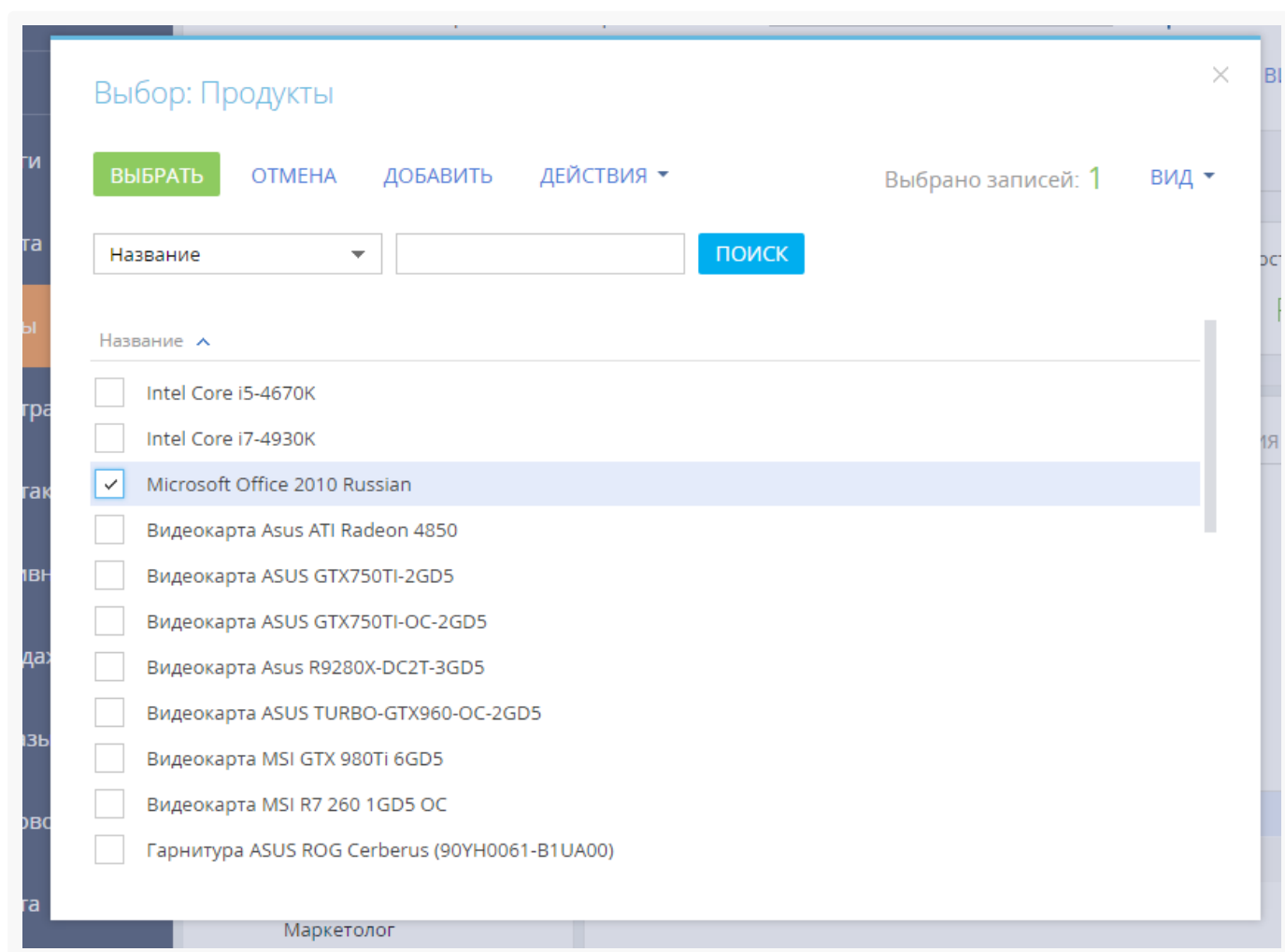
Для добавления пользовательской детали с редактируемым реестром в существующий раздел необходимо:

- Создать схему объекта детали.
- Создать и сконфигурировать схему детали.
- Выполнить настройку детали в схеме замещающей модели представления страницы раздела.
- Выполнить настройку полей детали.
- **Деталь с выбором из справочника** — данные детали выбираются из справочника, который отображается в модальном окне. Например, для детали [*Продукт в лиде*] отображаемые данные выбираются из справочника в модальном диалоговом окне [*Выбор: Продукт*].

Деталь [*Продукт в лиде*] с выбором из данных из справочника



Выбор продуктов из справочника детали [*Продукт в лиде*]



Поскольку деталь с выбором из справочника не является стандартной деталью приложения Creatio, то для ее добавления в раздел недостаточно использования мастера деталей. Ниже будет рассмотрен способ добавления этой детали на страницу записи раздела.

Для добавления пользовательской детали с выбором из справочника в существующий раздел необходимо:

- Создать схему объекта детали.
- Создать деталь с помощью мастера деталей и добавить ее в раздел.
- Сконфигурировать схему детали.
- Выполнить настройку полей детали.

Каждой детали соответствует схема бизнес-объекта, связанная с объектом текущего раздела. Так, детали [*Адреса Контакта*] соответствует схема объекта `Адрес контакта ( ContactAddress )` пакета `Base` . Связь с объектом раздела выполняется по обязательной колонке [*Контакт*] объекта детали.

Состав, расположение и поведение элементов пользовательского интерфейса детали конфигурируется схемой детали. Так, например, деталь [*Адреса Контакта*] конфигурируется схемой `Деталь адресов контакта ( ContactAddressDetailV2 )` , наследующей схему `Базовая схема детали с реестром ( BaseAddressDetailV2 )` пакета `UIv2` . Схемы деталей приложения унаследованы от базовой схемы детали с реестром (`BaseGridDetailV2`) и базовой схемы детали (`BaseDetailV2`) пакета `NUI` .

Страница детали конфигурируется схемой страницы записи. Например, свойства страницы детали [*Адреса Контакта*] задаются схемой `Страница адреса контакта ( ContactAddressPageV2 )` , которая унаследована от `Базовой страницы адреса ( BaseAddressPageV2 )` пакета `UIv2` .

Деталь [*Файлы и ссылки*] ([*Attachments*]) предназначена для хранения файлов и ссылок на web-ресурсы и статьи базы знаний, связанные с записью раздела. Деталь доступна во всех разделах Creatio (см. "[Файлы и примечания](#)"). Основная функциональность детали реализована в схеме `FileDetailV2` пакета `UIv2` .

Важно. Порядок добавления детали [*Файлы и ссылки*] ([*Attachments*]), который описан в статье, актуален для версии приложения 7.12.0 или ниже.

Чтобы добавить деталь на страницу записи раздела, необходимо выполнить следующие шаги:

1. С помощью мастера деталей создать обычную деталь, используя схему объекта раздела `Название объекта разделаFile` (см. статью "[Создать новый раздел](#)").
2. Изменить родительский объект для схемы модуля реестра детали.
3. С помощью мастера разделов добавить деталь на страницу записи.
4. Выполнить дополнительную настройку детали.

Деталь [*Файлы и ссылки*] ([*Attachments*]) предназначена для хранения файлов и ссылок на web-ресурсы и статьи базы знаний, связанные с записью раздела. Деталь доступна во всех разделах Creatio (см. статью "[Файлы и примечания](#)"). Основная функциональность детали реализована в схеме `FileDetailV2` пакета `UIv2` .

По умолчанию деталь [*Файлы и ссылки*] ([*Attachments*]) настроена так, что в ее списочном представлении отображаются только колонки [*Название*] ([*Name*]) и [*Версия*] ([*Version*]). Также на странице добавления новой ссылки доступна колонка [*Описание*] ([*Description*]) (рис. 1). Но в реестре детали она не отображается.

Рис. 1. — Отображение поля [*Описание*] ([*Description*]) на странице добавления

На заметку.

В плиточном представлении детали [*Файлы и ссылки*] ([*Attachments*]) всегда отображается только колонка [*Название*] ([*Name*]) и ссылка или файл.

Настройку реестра детали удобно выполнять с помощью [функциональности настройки колонок реестра](#). Однако для детали [*Файлы и ссылки*] ([*Attachments*]) эта функциональность по умолчанию отключена.

Чтобы добавить функциональность настройки колонок реестра детали, необходимо выполнить следующие действия:

- В пользовательском пакете создать схему замещающей модели представления детали `FileDetailV2`.
- В схеме замещающей модели представления реализовать вызов метода открытия страницы настройки колонок реестра.

Множественное добавление записей на деталь

Обычно деталь позволяет добавлять только одну запись. С помощью миксина `LookupMultiAddMixin` можно осуществлять множественное добавление записей на деталь.

Миксин `LookupMultiAddMixin` предназначен для расширения схем деталей. Он позволяет расширить действие добавления записи на деталь, предоставляя пользователю возможность выбирать несколько записей из справочника одновременно.

Последовательность добавления функциональности множественного выбора записей на деталь:

1. Создать схему замещающей модели представления детали.
2. Использовать методы миксина с замещением базовых методов детали.

Удалить деталь

Чтобы удалить пользовательскую деталь Creatio, необходимо иметь доступ к конфигурации системы и базе данных.

Важно. Прежде чем удалить пользовательскую деталь, необходимо снять блокировку с соответствующих этой детали файлов в хранилище версий SVN.

В первую очередь потребуется удалить записи из базы данных. Для удаления детали воспользуйтесь следующим скриптом:

```
DECLARE @Caption nvarchar(max);
SET @Caption = 'ToDelete';
DECLARE @UID UNIQUEIDENTIFIER;
select @UID = EntitySchemaUID from SysDetail
where Caption = @Caption
delete from SysDetail where EntitySchemaUID = @UID
```

Значение “ToDelete” необходимо заменить на название схемы детали, которое можно посмотреть в разделе [*Управление конфигурацией*] ([*Advanced settings*]) дизайнера системы. После удаления данных из БД нужно удалить схему пользовательской детали при помощи конфигурации системы. Пункт дизайнера системы [*Управление конфигурацией*] ([*Advanced settings*]) также позволяет удалить объект, на котором основывается данная деталь.

Настроить деталь с полями



Важно. При разработке детали с полями для продуктов линейки Financial Services Creatio используется схема `BaseFieldsDetail` пакета `BaseFinance`. Этот пакет присутствует только в продуктах линейки Financial Services Creatio.

Чтобы **использовать деталь с полями в CRM продуктах Creatio**:

1. [Скачайте пакет](#) `sdkFieldsDetailPackage`.
2. Импортируйте пакет в пользовательское приложение. Для этого воспользуйтесь инструкцией, которая приведена в статье [Перенести пакеты](#).
3. Добавьте пакет `sdkFieldsDetailPackage` в зависимости пользовательского пакета.

Пример. Реализовать пользовательскую деталь [*Медицинские документы*] ([*Medical documents*]), которая содержит виртуальные поля [*Номер*] ([*Number*]) и [*Серия*] ([*Series*]). Добавить деталь на вкладку [*История*] ([*History*]) страницы физического лица. Значение, введенное в поле [*Номер*] ([*Number*]), не должно быть отрицательным. Названия полей детали отображать синим цветом.

1. Создать пользовательскую деталь

1. Создайте пользовательский пакет и установите его в качестве текущего. Подробнее читайте в статье [Общие принципы работы с пакетами](#).
2. Перейдите в дизайнер системы по кнопке

3. В блоке [*Настройка системы*] ([*System setup*]) перейдите по ссылке [*Мастер деталей*] ([*Detail wizard*]).
4. Заполните свойства детали:

- [*Заголовок*] ([*Title*]) — "Медицинские документы" ("Medical documents").
- [*По какому объекту создать деталь?*] ([*How to create detail?*]) — выберите "Новому объекту" ("Create new object").

Заполните свойства объекта:

- [*Заголовок*] ([*Title*]) — "Медицинский документ" ("Medical document").
- [*Код*] ([*Code*]) — "UsrMedDocument".

После сохранения в конфигурации будут созданы:

- Схема `UsrMedDocument` объекта детали.
- Схема `UsrSchemac6fd3fd0Detail` модели представления реестра детали [*Медицинские документы*] ([*Medical documents*]).
- Схема `UsrUsrMedDocument4988cee4Page` модели представления страницы записи детали [*Медицинские документы*] ([*Medical documents*]).

5. Перейдите на вкладку [*Страница*] ([*Page*]) для настройки страницы записи детали.

6. Настройте поля детали.

- a. Заполните свойства поля [*Contact*] типа [*Справочник*] ([*Lookup*]).

- a. [*Заголовок*] ([*Title*]) — "Физ. лицо" ("Contact").
- b. [*Код*] ([*Code*]) — "UsrContact".
- c. Установите признак [*Обязательное*] ([*Required*]).
- d. [*Справочник*] ([*Lookup*]) — выберите "Контакт" ("Contact").

New column

SAVE CANCEL

General

Title *
Contact

Code *
UsrContact

☒ Required

☒ Copy this value when copying records

Data source

Lookup *
Contact

Lookup view

☒ Selection window

☐ List

- b. Заполните свойства поля [*Series*] типа [*Строка*] ([*String*]).
- [*Заголовок*] ([*Title*]) — "Серия" ("Series").
 - [*Код*] ([*Code*]) — "UsrSeries".
 - [*Длина строки*] ([*Text length*]) — выберите "Строка (50 символов)" ("Text (50 characters)").
 - Установите признак [*Обязательное*] ([*Required*]).

New column

SAVE **CANCEL**

General

Title *

Series XA

Code *

UsrSeries

Text length

Text (50 characters)

☒ Required

☒ Copy this value when copying records

Editability

☐ Read-only

Appearance

- с. Заполните свойства поля [*Number*] типа [*Целое число*] ([*Integer*]).
 - а. [*Заголовок*] ([*Title*]) — "Номер" ("Number").
 - б. [*Код*] ([*Code*]) — "UsrNumber".
 - с. Установите признак [*Обязательное*] ([*Required*]).

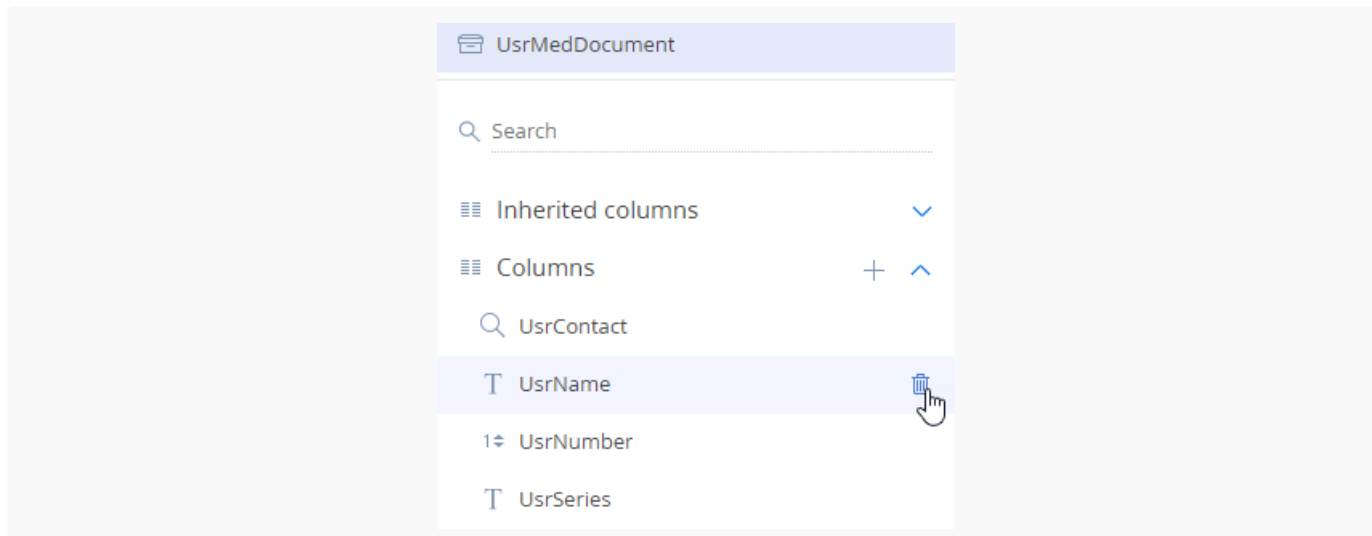
7. При необходимости, измените расположение полей детали.

8. На панели инструментов мастера деталей нажмите [Сохранить] ([Save]).

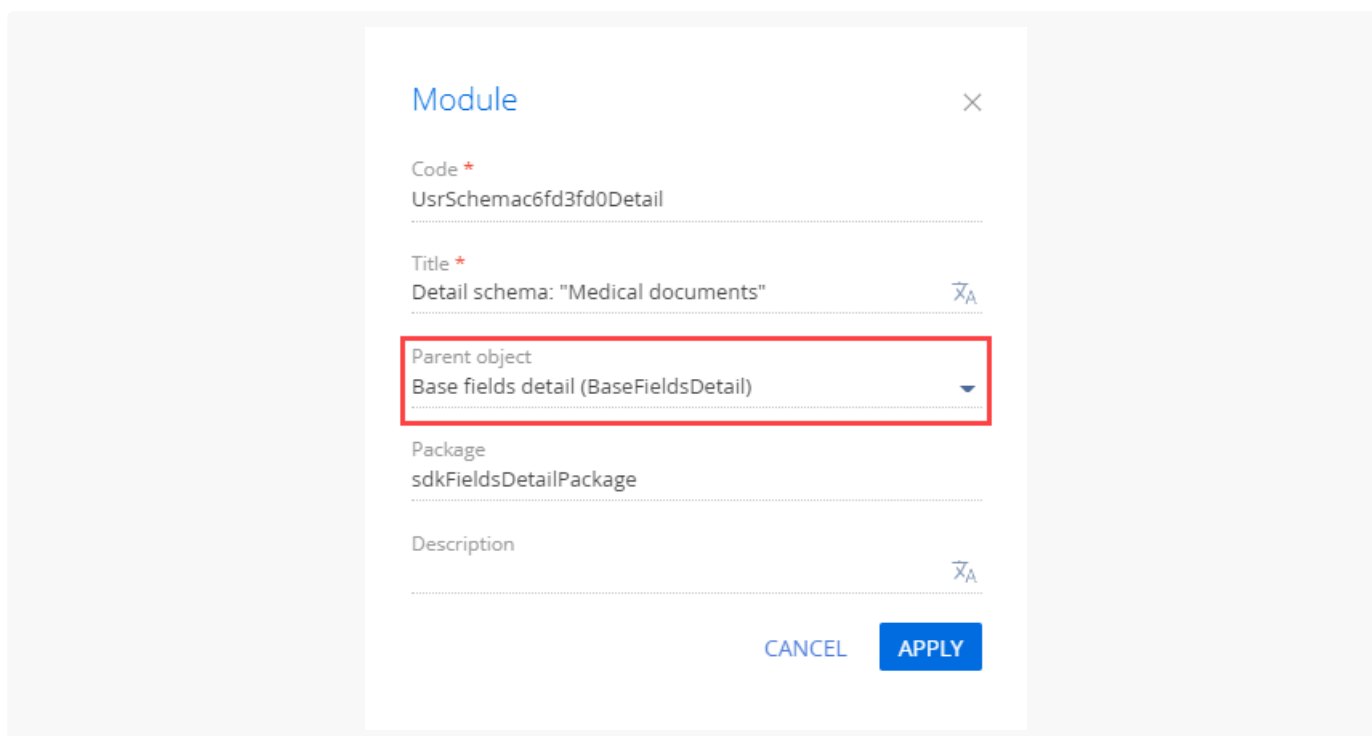
После сохранения в конфигурации будет модифицирована схема `UsrUsrMedDocument4988cee4Page` модели представления страницы записи детали [Медицинские документы] ([Medical documents]).

2. Настроить пользовательскую деталь

1. [Перейдите в раздел \[Конфигурация \]](#) ([Configuration]) и выберите пользовательский [пакет](#), который был установлен в качестве текущего.
2. Откройте схему `UsrMedDocument` объекта детали.
3. В контекстном меню узла [Колонки] ([Columns]) структуры объекта удалите обязательную колонку `[UsrName]`.



4. На панели инструментов дизайнера объектов нажмите [Опубликовать] ([Publish]).
5. Откройте схему `UsrSchemac6fd3fd0Detail` модели представления реестра детали [Медицинские документы] ([Medical documents]).
6. На панели свойств нажмите кнопку  и измените значение поля [Родительский объект] ([Parent object]) на `BaseFieldsDetail`. Схема `BaseFieldsDetail` реализует деталь с полями. По умолчанию в мастере деталей в качестве родительского объекта устанавливается базовая схема детали с реестром.



7. В свойство `methods` схемы модели представления детали добавьте метод `getDisplayColumns`, который возвращает массив с названиями колонок, отображающихся как поля в детали.

Исходный код схемы `UsrSchemac6fd3fd0Detail` представлен ниже.

```
UsrSchemac6fd3fd0Detail
```

```
define("UsrcSchemac6fd3fd0Detail", [], function() {
    return {
        entitySchemaName: "UsrcMedDocument",
        details: /**SCHEMA_DETAILS*/{}/**SCHEMA_DETAILS*/,
        diff: /**SCHEMA_DIFF*/[]/**SCHEMA_DIFF*/,
        methods: {
            getDisplayColumns: function() {
                return ["UsrcSeries", "UsrcNumber"];
            }
        }
    };
});
```

8. На панели инструментов дизайнера модуля нажмите [*Сохранить*] ([*Save*]).

3. Добавить деталь на страницу записи раздела

1. Перейдите в раздел [*Физ. лица*] ([*Contacts*]) и откройте страницу физического лица.
2. На панели инструментов кликните [*Вид*] —> [*Открыть мастер раздела*] ([*View*] —> [*Open section wizard*]).
3. В рабочей области мастера разделов перейдите на вкладку [*История*] ([*History*]) и нажмите кнопку [*Добавить деталь*] ([*New detail*]).
4. Заполните настройки детали.
 - [*Деталь*] ([*Detail*]) — выберите "Медицинские документы" ("Medical documents"). Поля [*Заголовок*] ([*Title*]) и [*Код (на английском)*] ([*Code*]) заполнятся автоматически.
 - [*У которых колонка детали*] ([*Where detail column*]) — выберите "Физ. лицо" ("Contact").

Значения остальных колонок оставьте без изменений.

Detail settings

SAVE **CANCEL**

Detail *
Medical documents

Title *
Medical documents

Code *
UsrSchemac6fd3fd0Detail7b52c652

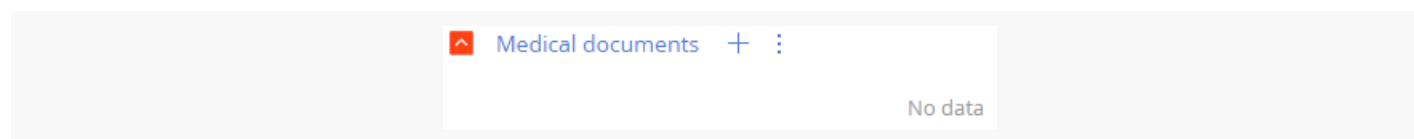
What records to show on the page?

Where detail column *
Contact

Equals to page column *
Id

5. Нажмите [Сохранить] —> [Мастер раздела] —> [Сохранить] ([Save] —> [Section wizard] —> [Save]).

В результате деталь [Медицинские документы] ([Medical documents]) будет добавлена на вкладку [История] ([History]) страницы физического лица.



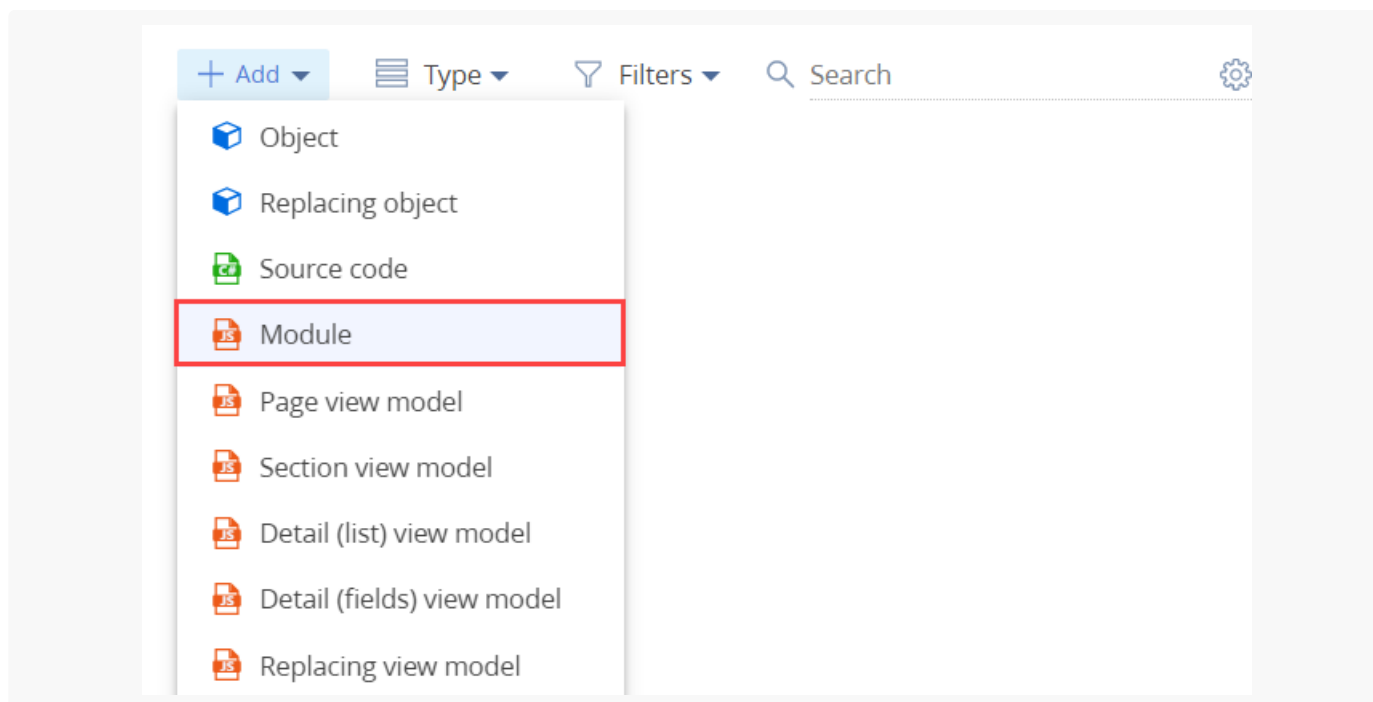
4. Добавить пользовательские стили детали

Поскольку в схеме модели представления страницы детали невозможно задать стили для отображения, необходимо:

1. Создать схему модуля, в которой определить стили.
2. Добавить модуль со стилями в зависимости модуля детали.

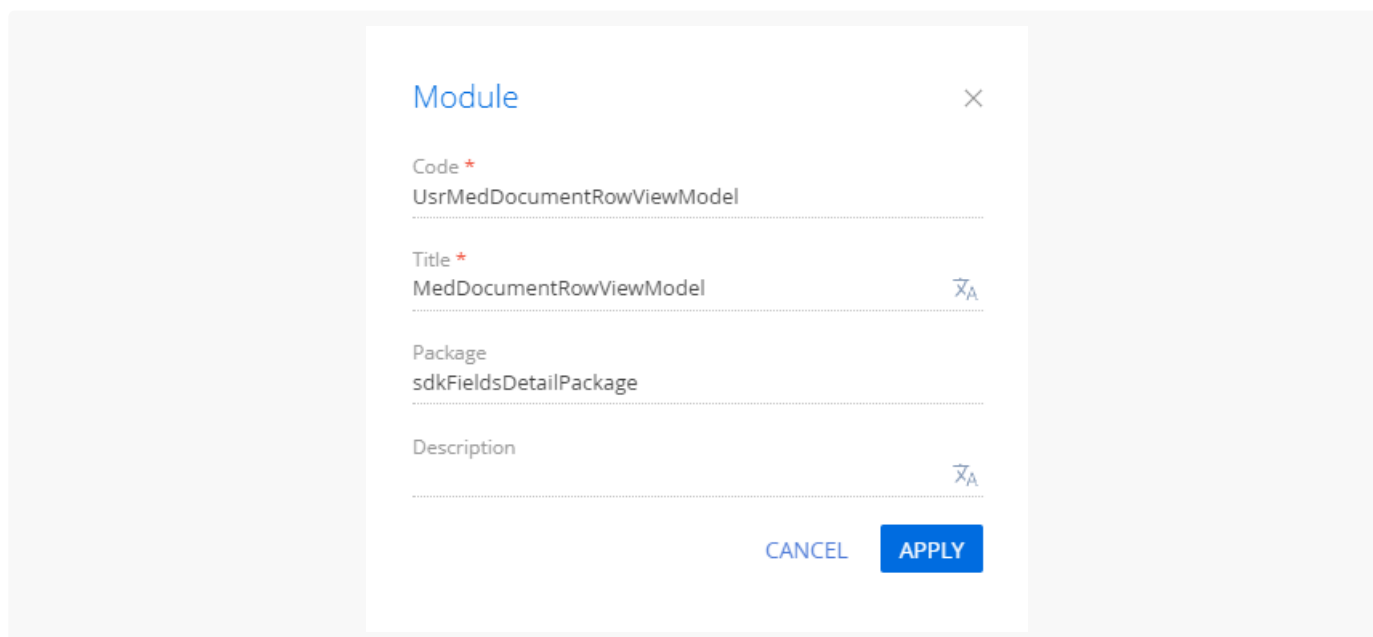
1. Создать схему модуля

1. [Перейдите в раздел \[Конфигурация \]](#) ([Configuration]) и выберите пользовательский [пакет](#), который был установлен в качестве текущего.
2. На панели инструментов реестра раздела нажмите [Добавить] —> [Модуль] ([Add] —> [Module]).



3. Заполните свойства схемы:

- [Код] ([Code]) — "UsrMedDocumentRowViewModel".
- [Заголовок] ([Title]) — "MedDocumentRowViewModel".



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

4. В дизайнера схем добавьте исходный код. В исходном коде схемы создайте описание модуля и определите в нем класс `Terrasoft.configuration.UsrMedDocumentRowViewModel`, унаследованный от класса `Terrasoft.BaseFieldRowViewModel`.

UsrMedDocumentRowViewModel

```
define("UsrMedDocumentRowViewModel", ["BaseFieldRowViewModel"], function() {
    Ext.define("Terrasoft.configuration.UsrMedDocumentRowViewModel", {
        extend: "Terrasoft.BaseFieldRowViewModel",
        alternateClassName: "Terrasoft.UsrMedDocumentRowViewModel"
    });
    return Terrasoft.UsrMedDocumentRowViewModel;
});
```

5. Перейдите в узел [LESS] структуры объекта и задайте необходимые стили отображения детали.

Настройка стилей отображения детали

```
.med-document-left-row-container {
    .t-label {
        color: blue;
    }
}
.field-detail-row {
    width: 100%;
    display: inline-flex;
    margin-bottom: 10px;

    .field-detail-row-left {
        display: flex;
        flex-wrap: wrap;

        .control-width-15 {
            min-width: 300px;
            width: 50%;
            margin-bottom: 5px;
        }
    }
    .field-detail-row-left.singlecolumn {
        width: 50%;
    }
}
```

6. На панели инструментов дизайнера нажмите [Сохранить] ([Save]).

2. Модифицировать схему модели представления детали

Чтобы **использовать созданный модуль и его стили** в схеме детали:

1. Откройте схему `UsrSchemac6fd3fd0Detail` модели представления реестра детали [Медицинские

документы] ([*Medical documents*]).

- В зависимости схемы `UsrSchemac6fd3fd0Detail` добавьте модуль `UsrMedDocumentRowViewModel`.
- В определение модуля схемы детали добавьте методы переопределения базовых CSS-классов стилей:
 - `getRowViewModelClassName()` — метод, который возвращает имя класса модели представления записи на детали.
 - `getLeftRowContainerWrapClass()` — метод, который возвращает массив строк с названиями CSS-классов, используемых для генерации представления контейнеров, содержащих подписи полей записей.

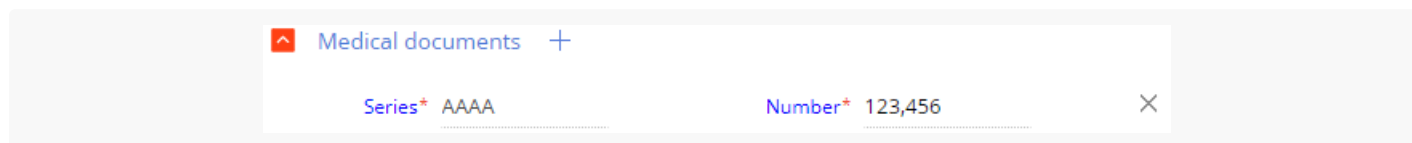
Исходный код модифицированной схемы представлен ниже.

`UsrSchemac6fd3fd0Detail`

```
define("UsrSchemac6fd3fd0Detail", ["UsrMedDocumentRowViewModel", "css!UsrMedDocumentRowViewMo
return {
  entitySchemaName: "UsrMedDocument",
  details: /**SCHEMA_DETAILS*/{}/**SCHEMA_DETAILS*/,
  diff: /**SCHEMA_DIFF*/ [], /**SCHEMA_DIFF*/
  methods: {
    getDisplayColumns: function() {
      return ["UsrSeries", "UsrNumber"];
    },
    getRowViewModelClassName: function() {
      return "Terrasoft.UsrMedDocumentRowViewModel";
    },
    getLeftRowContainerWrapClass: function() {
      return ["med-document-left-row-container", "field-detail-row"];
    }
  }
};
});
```

- На панели инструментов дизайнера нажмите [Сохранить] ([*Save*]).

В результате названия полей детали [*Медицинские документы*] ([*Medical documents*]), которая была добавлена на вкладку [*История*] ([*History*]) страницы физического лица, отображаются синим цветом.



5. Добавить валидацию к полю детали

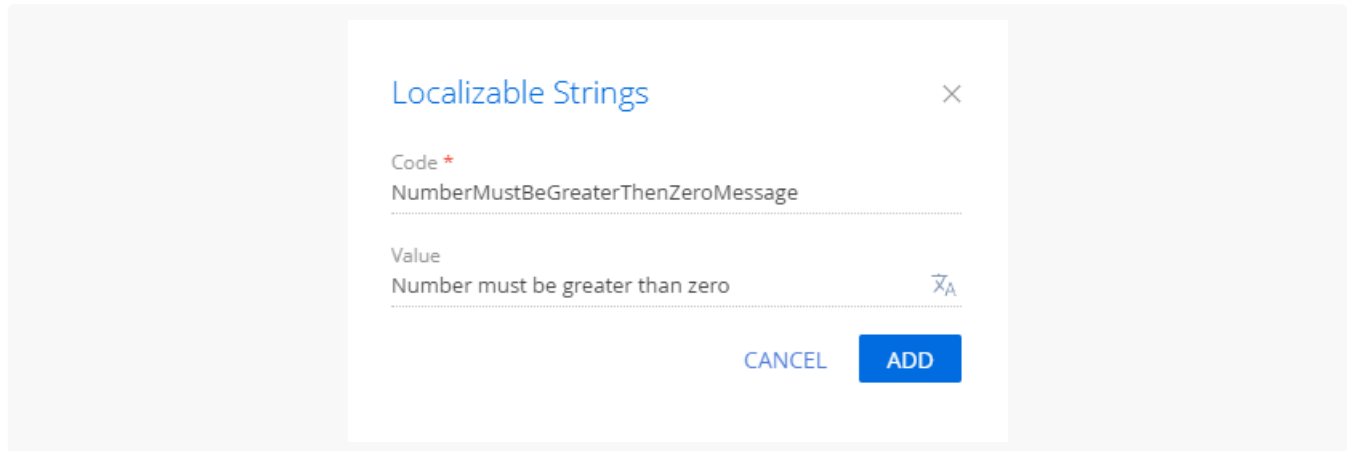
- Откройте схему `UsrMedDocumentRowViewModel` модуля.

2. Добавьте локализуемую строку с сообщением о неверном значении поля [*Номер*] ([*Number*]).

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

b. Заполните свойства локализуемой строки:

- [*Код*] ([*Code*]) — "NumberMustBeGreaterThanZeroMessage".
- [*Значение*] ([*Value*]) — "Number must be greater than zero" ("Введите номер больше нуля").



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

f. В зависимости модуля `UsrMedDocumentRowViewModel` добавьте модуль ресурсов `UsrMedDocumentRowViewModelResources`. Это необходимо, чтобы значение локализуемой строки отобразилось во front-end части приложения.

3. Добавьте логику работы валидации значения поля [*Номер*] ([*Number*]). Для этого реализуйте методы:

- `validateNumberMoreThenZero()` — метод, который содержит логику валидации значения поля.
- `setValidationConfig()` — метод, который связывает колонку [*Number*] и метод-валидатор `validateNumberMoreThenZero()`.
- `init()` — переопределенный базовый метод, в котором выполняется вызов базовой логики и метода `setValidationConfig()`.

Исходный код модифицированной схемы представлен ниже.

`UsrMedDocumentFieldsDetail`

```
define("UsrMedDocumentRowViewModel", ["BaseFieldRowViewModel", "UsrMedDocumentRowViewModelRes
function(resources) {
    Ext.define("Terrasoft.configuration.UsrMedDocumentRowViewModel", {
        extend: "Terrasoft.BaseFieldRowViewModel",
        alternateClassName: "Terrasoft.UsrMedDocumentRowViewModel",
        validateNumberMoreThenZero: function(columnValue) {
            var invalidMessage = "";
            if (columnValue < 0) {
                invalidMessage = resources.localizableStrings.NumberMustBeGreaterThanZeroMess
```

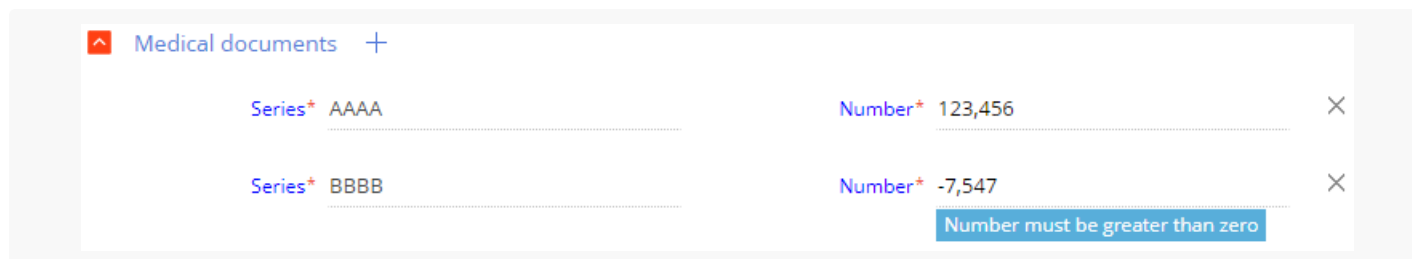
```

    }
    return {
        fullInvalidMessage: invalidMessage,
        invalidMessage: invalidMessage
    };
},
setValidationConfig: function() {
    this.addColumnValidator("UsrNumber", this.validateNumberMoreThenZero);
},
init: function() {
    this.callParent(arguments);
    this.setValidationConfig();
}
});
return Terrasoft.UsrMedDocumentRowViewModel;
});

```

4. На панели инструментов дизайнера нажмите [*Сохранить*] ([*Save*]).

В результате при вводе отрицательного значения в поле [*Номер*] ([*Number*]) отображается соответствующее предупреждение.



6. Сделать виртуальными поля детали

1. Откройте схему `UsrSchemac6fd3fd0Detail` модели представления реестра детали [*Медицинские документы*] ([*Medical documents*]).
2. Добавьте реализацию метода `useVirtualRecord()`.

Исходный код модифицированной схемы представлен ниже.

UsrSchemac6fd3fd0Detail

```

define("UsrSchemac6fd3fd0Detail", ["UsrMedDocumentRowViewModel", "css!UsrMedDocumentRowViewMo
return {
    entitySchemaName: "UsrMedDocument",
    details: /**SCHEMA_DETAILS*/{}/**SCHEMA_DETAILS*/,
    diff: /**SCHEMA_DIFF*/ [], /**SCHEMA_DIFF*/
    methods: {
        getDisplayColumns: function() {

```

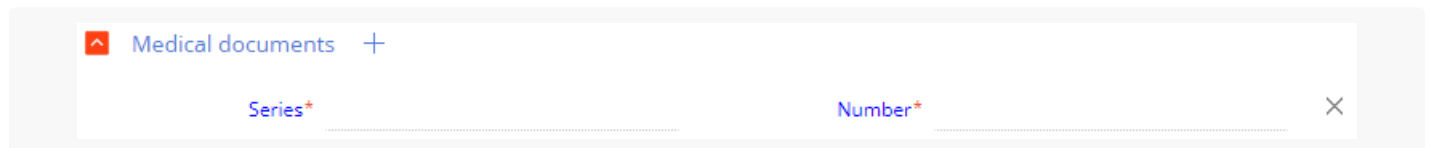
```

        return ["UsrSeries", "UsrNumber"];
    },
    getRowViewModelClassName: function() {
        return "Terrasoft.UsrMedDocumentRowViewModel";
    },
    getLeftRowContainerWrapClass: function() {
        return ["med-document-left-row-container", "field-detail-row"];
    },
    useVirtualRecord: function() {
        return true;
    }
}
};
});

```

3. На панели инструментов дизайнера нажмите [Сохранить] ([Save]).

В результате при открытии вкладки [История] ([History]), которая содержит деталь [Медицинские документы] ([Medical documents]), отображается виртуальная запись.



Скрыть пункты меню детали с реестром

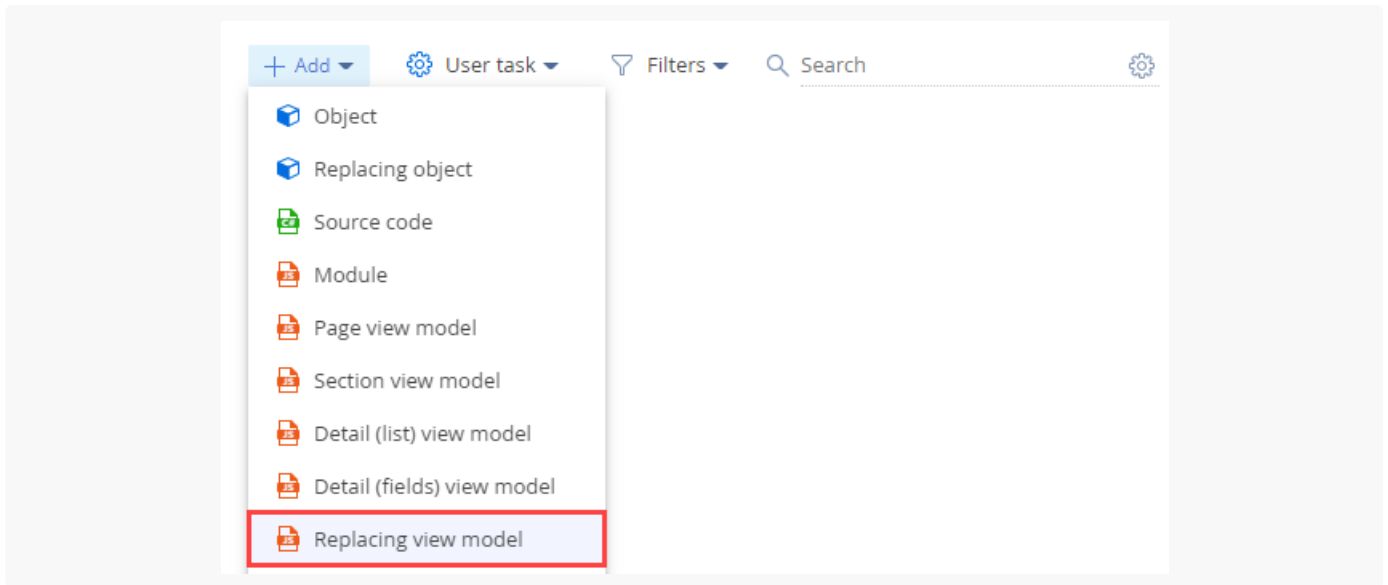
 Средний

Пример. Для детали [Адреса] ([Addresses]), которая находится на вкладке [Основная информация] ([Contact info]) страницы контакта, скрыть пункты [Копировать] ([Copy]), [Изменить] ([Edit]), [Удалить] ([Delete]) меню.

Назначение пунктов [Копировать] ([Copy]), [Изменить] ([Edit]), [Удалить] ([Delete]) меню — управление записями реестра детали.

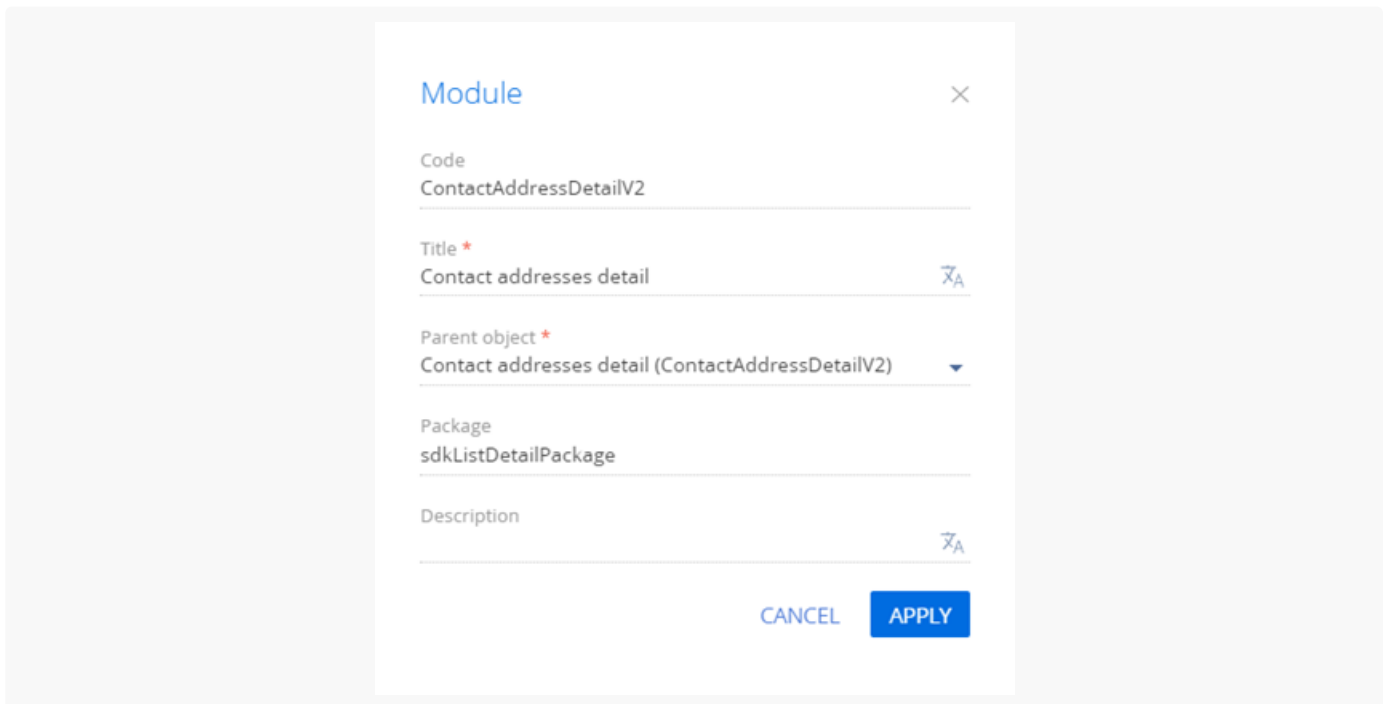
Создать схему замещающей модели представления реестра детали

1. [Перейдите в раздел \[Конфигурация \]](#) ([Configuration]) и выберите пользовательский [пакет](#), в который будет добавлена схема.
2. На панели инструментов реестра раздела нажмите [Добавить] —> [Замещающая модель представления] ([Add] —> [Replacing view model]).



3. Заполните свойства схемы:

- [Код] ([Code]) — "ContactAddressDetailV2".
- [Заголовок] ([Title]) — "Деталь адресов контакта" ("Contact addresses detail").
- [Родительский объект] ([Parent object]) — выберите "ContactAddressDetailV2".



4. В дизайнера модуля добавьте исходный код.

ContactAddressDetailV2

```
define("ContactAddressDetailV2", [], function() {
    return {
        entitySchemaName: "AccountAddress",
```



```

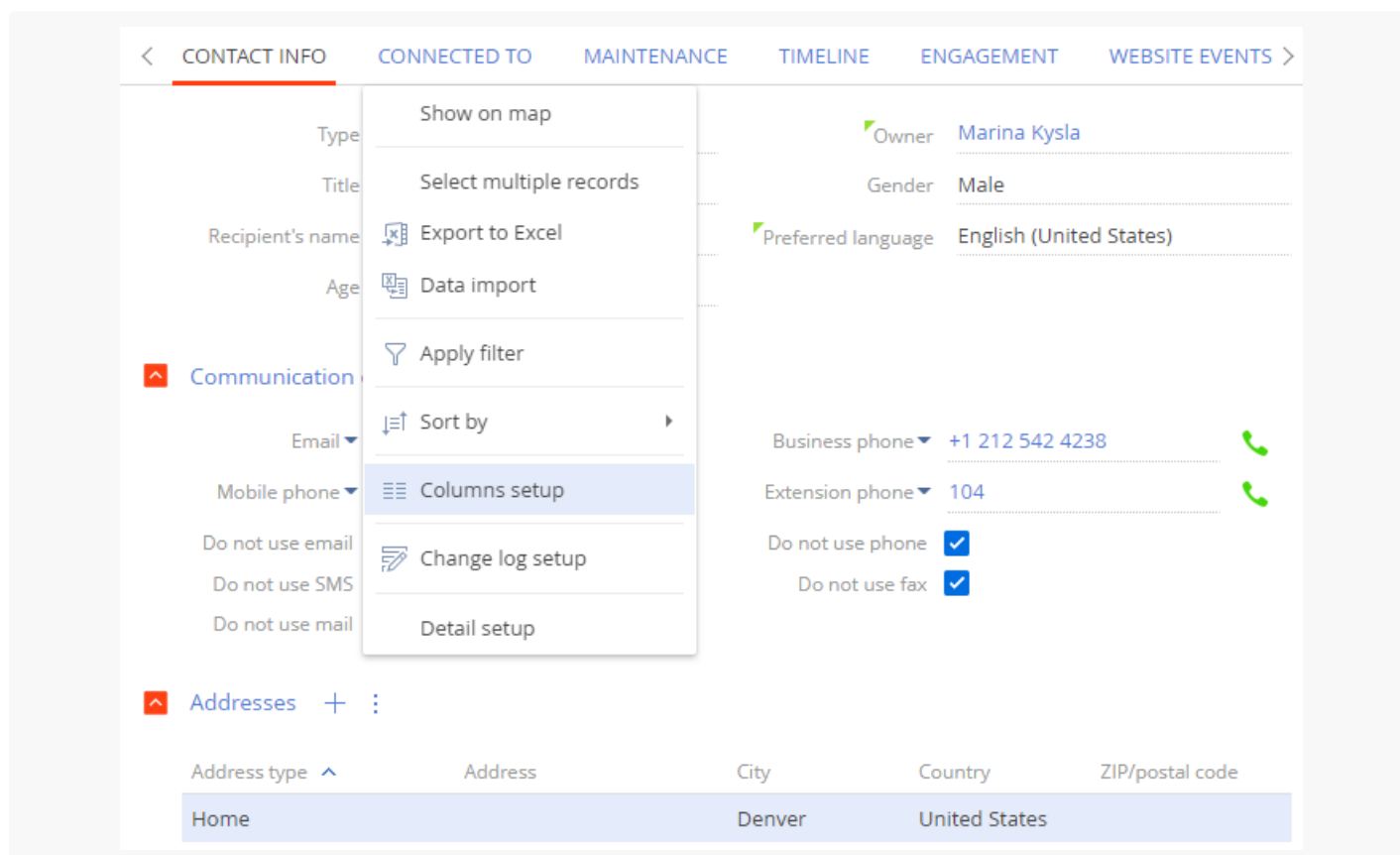
methods: {
  /* Удаление пункта [Копировать] ([Copy]) меню. */
  getCopyRecordMenuItem: Terrasoft.emptyFn,
  /* Удаление пункта [Редактировать] ([Edit]) меню. */
  getEditRecordMenuItem: Terrasoft.emptyFn,
  /* Удаление пункта [Удалить] ([Delete]) меню. */
  getDeleteRecordMenuItem: Terrasoft.emptyFn
},
diff: /**SCHEMA_DIFF*/[ ]/**SCHEMA_DIFF*/
};
});

```

5. На панели инструментов дизайнера нажмите [Сохранить] ([Save]).

Результат выполнения примера

В результате выполнения примера пункты [Копировать] ([Copy]), [Изменить] ([Edit]) скрыты из меню детали [Адреса] ([Addresses]).



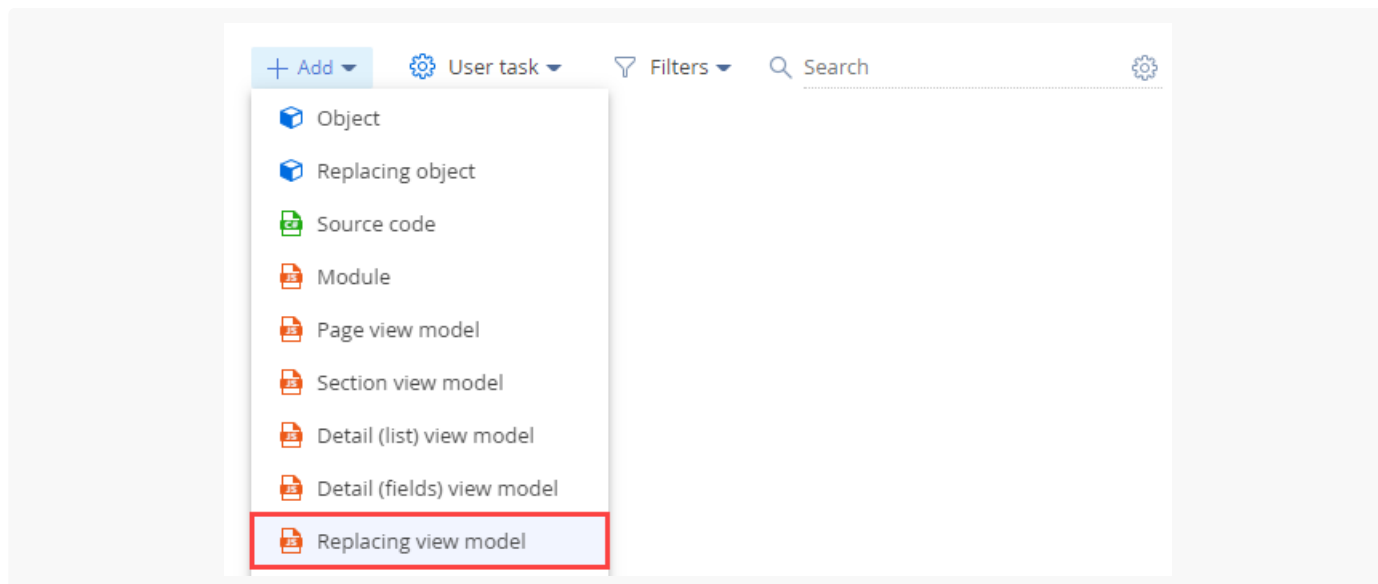
Реализовать множественное добавление записей на деталь



Пример. Реализовать множественное добавление записей на деталь [*Контакты*] ([*Contacts*]) страницы записи раздела [*Продажи*] ([*Opportunities*]).

1. Создать схему замещающей модели представления детали

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



3. В дизайнере модуля заполните свойства схемы:

- [*Код*] ([*Code*]) — "OpportunityContactDetailV2".
- [*Заголовок*] ([*Title*]) — "OpportunityContactDetailV2".
- [*Родительский объект*] ([*Parent object*]) — выберите "OpportunityContactDetailV2".

2. Реализуйте бизнес-логику детали

1. В свойство `mixins` схемы детали добавьте миксин `LookupMultiAddMixin`.
2. В переопределенном методе `init()` схемы детали инициализируйте миксин `LookupMultiAddMixin`. Переопределение метода `init()` описано в статье [Виды модулей](#).
3. Переопределите метод `getAddRecordButtonVisible()`, который отвечает за отображение кнопки добавления.
4. Переопределите метод `onCardSaved()`, который отвечает за сохранение страницы детали. Используйте метод `openLookupWithMultiSelect()`, который вызывает справочное окно для множественного выбора.
5. Реализуйте метод `getMultiSelectLookupConfig()`, который выполняет конфигурирование справочного окна. Связанный с методом `openLookupWithMultiSelect()`. Возвращает объект конфигурации для справочного окна.

Свойства объекта:

- `rootEntitySchemaName` — корневая схема объекта.
 - `rootColumnName` — связующая колонка, которая указывает на запись корневой схемы.
 - `relatedEntitySchemaName` — связанная схема.
 - `relatedColumnName` — колонка, которая указывает на запись связанной схемы.
6. Переопределите метод `addRecord()`, который отвечает за добавление записи на деталь. Как и для метода `onCardSaved()`, используйте метод `openLookupWithMultiSelect()`. Значение `true` указывает на необходимость выполнения проверки является ли запись новой.

В текущем примере справочное окно использует данные из таблицы `[OpportunityContact]`, которая

связана с колонкой [Opportunity] корневой схемы [Opportunity] и колонкой [Contact] связанной схемы [Contact].

Исходный код схемы детали представлен ниже.

OpportunityContactDetailV2

```
define("OpportunityContactDetailV2", ["LookupMultiAddMixin"], function() {
    return {
        mixins: {
            /* Подключение миксина к схеме. */
            LookupMultiAddMixin: "Terrasoft.LookupMultiAddMixin"
        },
        methods: {
            /* Переопределение базового метода инициализации схемы. */
            init: function() {
                this.callParent(arguments);
                /* Инициализация миксина. */
                this.mixins.LookupMultiAddMixin.init.call(this);
            },
            /* Переопределение базового метода отображения кнопки добавления. */
            getAddRecordButtonVisible: function() {
                /* Отображать кнопку добавления если деталь развернута, даже если для детали не
                return this.getToolsVisible();
            },
            /* Переопределение базового метода.
            Обработчик события сохранения страницы записи детали. */
            onCardSaved: function() {
                /* Открывает справочное окно с множественным выбором записей. */
                this.openLookupWithMultiSelect();
            },
            /* Переопределение базового метода добавления записи на деталь. */
            addRecord: function() {
                /* Открывает справочное окно с множественным выбором записей. */
                this.openLookupWithMultiSelect(true);
            },
            /* Метод, который возвращает конфигурационный объект для справочного окна. */
            getMultiSelectLookupConfig: function() {
                return {
                    /* Корневая схема – [Продажа]. */
                    rootEntitySchemaName: "Opportunity",
                    /* Колонка корневой схемы. */
                    rootColumnName: "Opportunity",
                    /* Связанная схема – [Контакт]. */
                    relatedEntitySchemaName: "Contact",
                    /* Колонка связанной схемы. */
                    relatedColumnName: "Contact"
                };
            }
        }
    };
});
```

```

    }
  }
};
});

```

На панели инструментов дизайнера модуля нажмите [Сохранить] ([Save]).

Результат выполнения примера

1. Обновите страницу приложения.
2. На детали [Контакты] ([Contacts]) страницы записи раздела [Продажи] ([Opportunities]) нажмите кнопку .

	Contacts					
Contact	Primary c...	Role	Influence	Decision-making factors	Loyalty	
Andrew Baker	No	Decision maker	High		2 – Supportive	

В результате выполнения примера приложение позволяет выбрать несколько записей из справочника.

Select: Contact

SELECT

CANCEL

NEW

ACTIONS

Records selected: 4 VIEW

Full name

SEARCH

Full name	Email	Account
☒ James Smith	smith@gateway-invest.co.uk	Gateway
☐ Tran Manzo	TranManzo@gmail.com	
☒ Symon Clarke	symon-clarke@yahoo.com	Our company
☐ Youlanda Mcwhorter	YoulandaMcwhorter@gmail.com	
☒ Jason Robinson	jason_r@gmail.com	Our company
☐ Stasia Henrickson	StasiaHenrickson@hotmail.com	
☐ Stephen Washington	StephenWashington@gmail.com	
☐ Stewart Crispin	StewartCrispin@hotmail.com	
☒ Nora Wesley	TyroneRigg@hotmail.com	Gtech
☐ Winter Hodge	winterhodge@gmail.com	Console Solutions
☐ Zachariah Kershner	ZachariahKershner@gmail.com	
☐ Wilbur Brochberg	WilburBrochberg@gmail.com	

После подтверждения выбранные записи добавляются на деталь [Контакты] ([Contacts]) на странице записи раздела [Продажи] ([Opportunities]).

Contacts + ⋮

Contact	Primary contact	Role	Influen...	Decision-making factors	Loyalty
Jason Robinson	No	Contact person	Low		1 – Interested
James Smith	No	Decision maker	High		2 – Supportive
Nora Wesley	No	Decision maker	Medium		1 – Interested
Symon Clarke	No	Influencer	High		3 – Active supporter

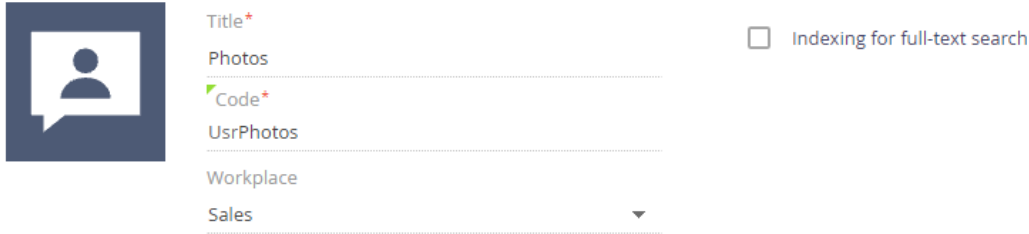
Реализовать деталь типа [Файлы и ссылки]

 Сложный

Пример. На страницу записи пользовательского раздела [*Фотографии*] ([*Photos*]) добавить деталь [*Прикрепленные фотографии*] ([*Photos attachment*]) типа [*Файлы и ссылки*] ([*Attachments*]).

1. Создать пользовательский раздел

1. Создайте пользовательский пакет и установите его в качестве текущего. Подробнее читайте в статье [Общие принципы работы с пакетами](#).
2. Перейдите в дизайнер системы по кнопке .
3. В блоке [*Настройка системы*] ([*System setup*]) перейдите по ссылке [*Мастер разделов*] ([*Section wizard*]).
4. Заполните настройки раздела:
 - [*Заголовок*] ([*Title*]) — "Фотографии" ("Photos").
 - [*Код (на английском)*] ([*Code*]) — "UsrPhotos".
 - [*Рабочее место*] ([*Workplace*]) — выберите "Продажи" ("Sales").



Section settings



Title*
Photos

Code*
UsrPhotos

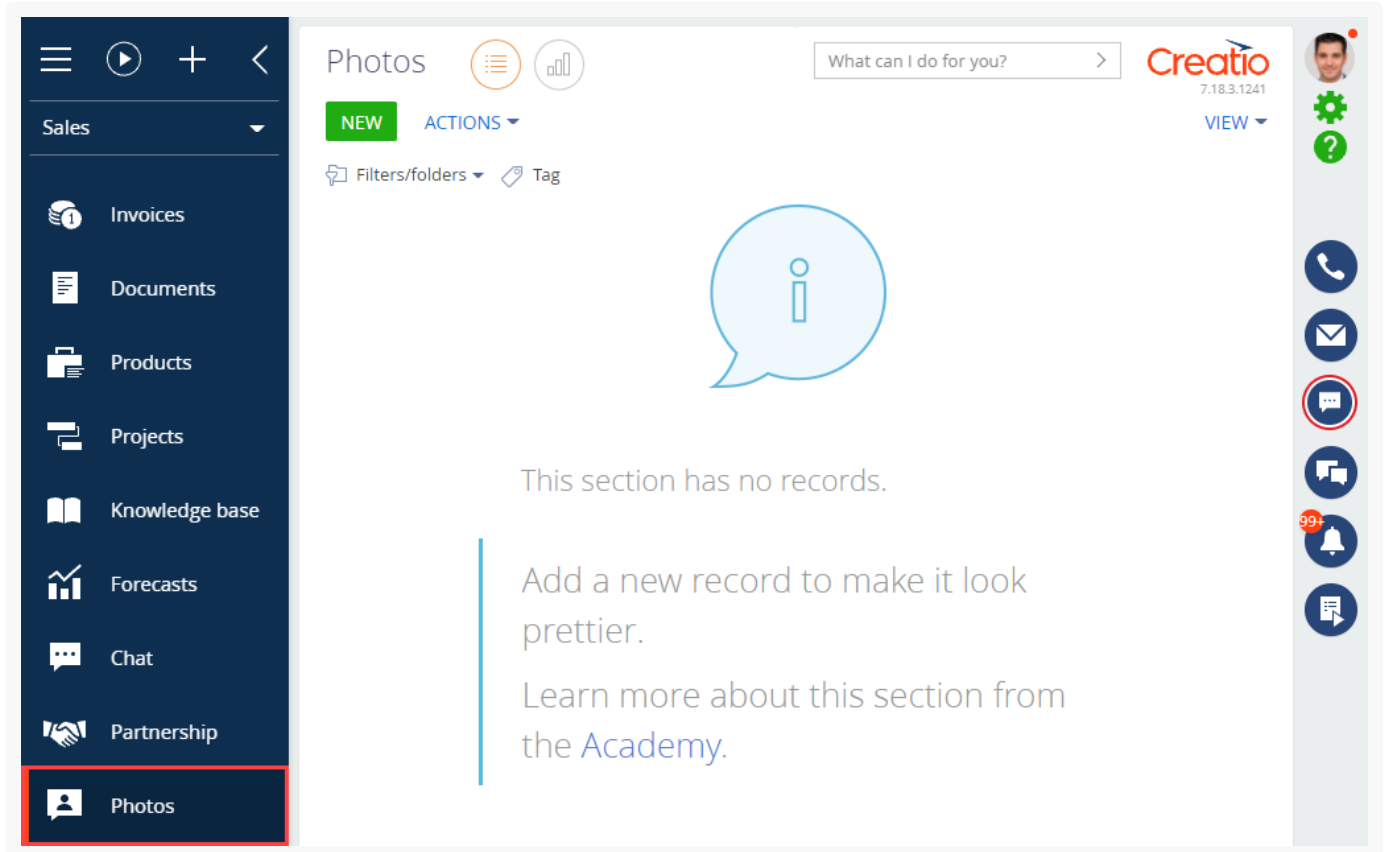
Workplace
Sales ▼

☐ Indexing for full-text search

5. На панели инструментов мастера разделов нажмите [*Сохранить*] ([*Save*]).

В результате:

- Пользовательский раздел [*Фотографии*] ([*Photos*]) отображается в рабочем месте [*Продажи*] ([*Sales*]).



- В конфигурации созданы схемы раздела [Фотографии] ([Photos]).

☐	UsrPhotos *	Photos	Object	10/11/2021, 7:43:02 AM	sdkDetailAttachme ntPackage	⋮
☐	UsrPhotos1Page *	Edit page: "Photos"	Client module	10/11/2021, 7:42:47 AM	sdkDetailAttachme ntPackage	⋮
☐	UsrPhotos60ec3f85Section *	Section schema: "Photos"	Client module	10/11/2021, 7:42:36 AM	sdkDetailAttachme ntPackage	⋮
☐	UsrPhotosFile *	Photos attachment	Object	10/11/2021, 7:43:11 AM	sdkDetailAttachme ntPackage	⋮
☐	UsrPhotosFolder *	Photos folder	Object	10/11/2021, 7:43:27 AM	sdkDetailAttachme ntPackage	⋮
☐	UsrPhotosInFolder *	Photos in Folder	Object	10/11/2021, 7:43:36 AM	sdkDetailAttachme ntPackage	⋮
☐	UsrPhotosInTag *	Photos section record tag	Object	10/11/2021, 7:43:57 AM	sdkDetailAttachme ntPackage	⋮
☐	UsrPhotosTag *	Photos section tag	Object	10/11/2021, 7:43:48 AM	sdkDetailAttachme ntPackage	⋮

2. Создать пользовательскую деталь

- Перейдите в дизайнер системы по кнопке
- В блоке [Настройка системы] ([System setup]) перейдите по ссылке [Мастер деталей] ([Detail wizard]).
- Заполните свойства детали:
 - [Заголовок] ([Title]) — "Прикрепленные фотографии" ("Photos attachment").
 - [По какому объекту создать деталь?] ([How to create detail?]) — выберите "Существующему объекту" ("Based on existing object").

- [Объект] ([Object]) — выберите "Файл и ссылка объекта Фотографии" ("Photos attachment").

4. На панели инструментов мастера деталей нажмите [Сохранить] ([Save]).

После сохранения в конфигурации созданы:

- Схема `UsrSchemae9733d1bDetail` модели представления детали [Прикрепленные фотографии] ([Photos attachment]).
- Схема `UsrUsrPhotosFiled6a229baPage` страницы записи детали [Прикрепленные фотографии] ([Photos attachment]).

3. Настроить пользовательскую деталь

1. [Перейдите в раздел \[Конфигурация \]](#) ([Configuration]) и выберите пользовательский [пакет](#), который был установлен в качестве текущего.
2. Откройте схему `UsrUsrPhotosFiled6a229baPage` страницы записи детали [Прикрепленные фотографии] ([Photos attachment]).
3. На панели свойств нажмите кнопку  и измените значение поля [Родительский объект] ([Parent object]) на `FileDetailV2`. Схема `FileDetailV2` пакета `UIv2` реализует деталь [Файлы и ссылки] ([Attachments]). По умолчанию в мастере деталей в качестве родительского объекта устанавливается базовая схема детали с реестром.

Module [X]

Code *
UsrUsrPhotosFiled6a229baPage

Title *
Card schema: "Photos attachment" [X_A]

Parent object
FileDetailV2 (FileDetailV2) [v]

Package
sdkDetailAttachmentPackage

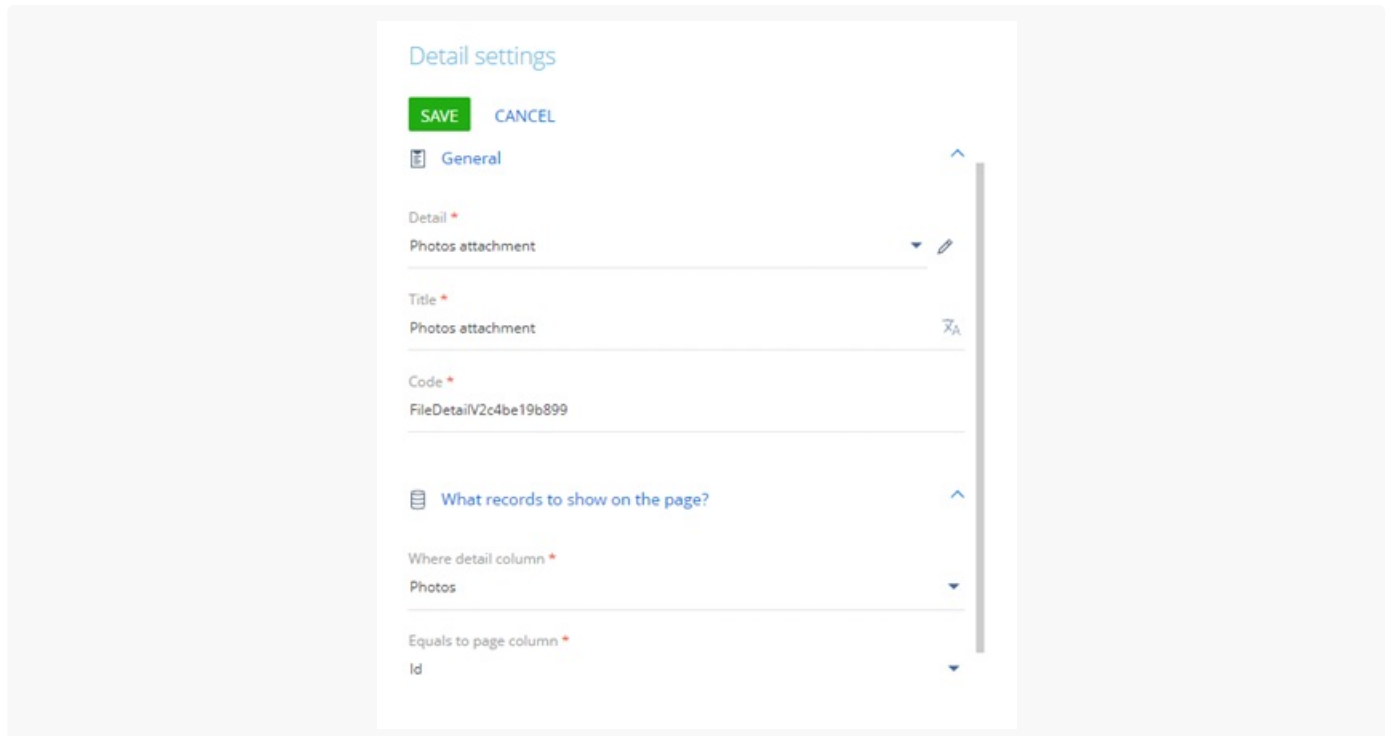
Description [X_A]

CANCEL APPLY

4. Для применения заданных свойств нажмите [Применить] ([Apply]).
5. На панели инструментов дизайнера нажмите [Сохранить] ([Save]).

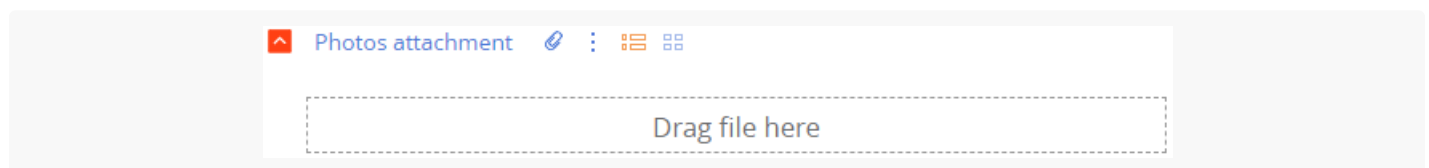
4. Добавить деталь в раздел

1. Перейдите в раздел [Фотографии] ([Photos]).
2. На панели инструментов кликните [Вид] —> [Открыть мастер раздела] ([View] —> [Open section wizard]).
3. В блоке [Страницы раздела] ([Section pages]) нажмите кнопку [Редактировать страницу] ([Edit page]).
4. В рабочей области мастера разделов нажмите кнопку [Добавить деталь] ([New detail]).
5. Заполните настройки детали.
 - [Деталь] ([Detail]) — выберите "Прикрепленные фотографии" ("Photos attachment"). Поля [Заголовок] ([Title]) и [Код (на английском)] ([Code]) заполнятся автоматически.
 - [Заголовок] ([Title]) — измените на "Прикрепленные фотографии" ("Photos attachment").



6. Нажмите [Сохранить] —> [Мастер раздела] —> [Сохранить] ([Save] —> [Section wizard] —> [Save]).

В результате деталь [Прикрепленные фотографии] ([Photos attachment]) будет добавлена на страницу записи раздела [Фотографии] ([Photos]).



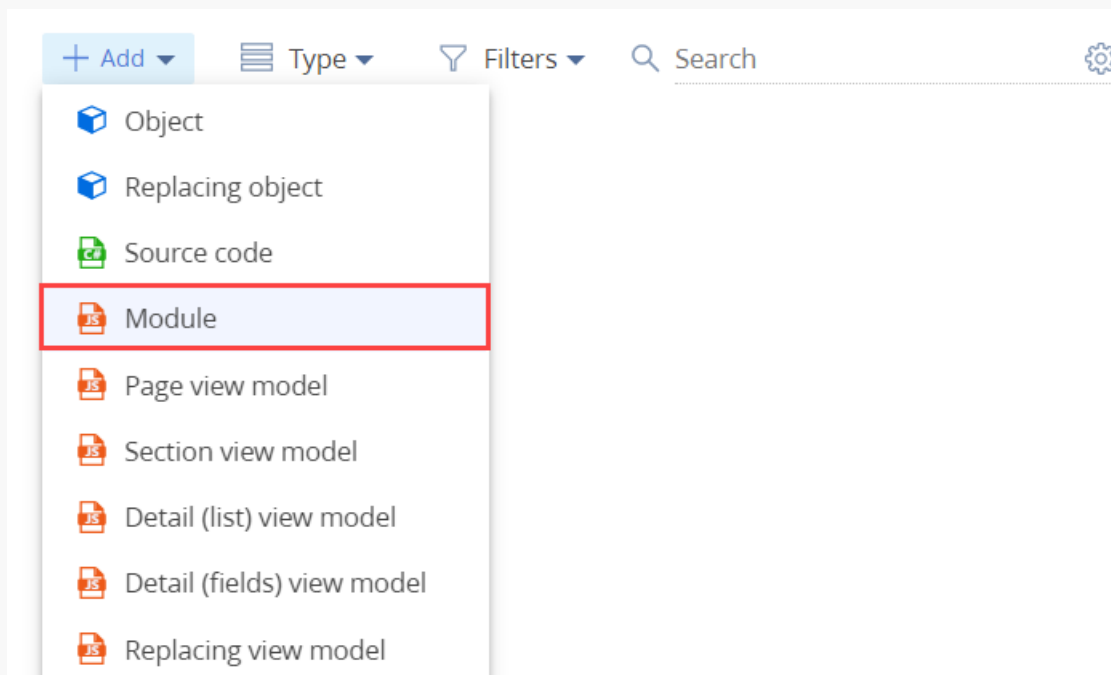
5. Добавить пользовательские стили детали

Поскольку в схеме модели представления страницы детали невозможно задать стили для отображения, необходимо:

1. Создать схему модуля, в которой определить стили.
2. Добавить модуль со стилями в зависимости модуля детали.

1. Создать схему модуля

1. [Перейдите в раздел \[Конфигурация \]](#) ([Configuration]) и выберите пользовательский [пакет](#), который был установлен в качестве текущего.
2. На панели инструментов реестра раздела нажмите [Добавить] —> [Модуль] ([Add] —> [Module]).



3. Заполните свойства схемы:

- [Код] ([Code]) — "UsrSchemaDetailCSS".
- [Заголовок] ([Title]) — "SchemaDetailCSS".

Module
×

Code *
UsrSchemaDetailCSS

Title *
SchemaDetailCSS

Package
sdkDetailAttachmentPackage

Description

CANCEL APPLY

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

4. Перейдите в узел [LESS] структуры объекта и задайте необходимые стили отображения детали.

Настройка стилей отображения детали

```
div[id*="UsrSchemae9733d1bDetail"] {
```

```

.grid-status-message-empty {
    display: none;
}
.grid-empty > .grid-bottom-spinner-space {
    height: 5px;
}
.dropzone {
    height: 35px;
    width: 100%;
    border: 1px dashed #999999;
    text-align: center;
    line-height: 35px;
}
.dropzone-hover {
    border: 1px dashed #4b7fc7;
}
.DragAndDropLabel {
    font-size: 1.8em;
    color: rgb(110, 110, 112);
}
}

div[data-item-marker*="added-detail"] {
    div[data-item-marker*="tiled"], div[data-item-marker*="listed"] {
        .entity-image-class {
            width: 165px;
        }
        .entity-image-container-class {
            float: right;
            width: 128px;
            height: 128px;
            text-align: center;
            line-height: 128px;
        }
        .entity-image-view-class {
            max-width: 128px;
            max-height: 128px;
            vertical-align: middle;
        }
        .images-list-class {
            min-height: 0.5em;
        }
        .images-list-class > .selectable {
            margin-right: 10px;
            display: inline-block;
        }
        .entity-label {
            display: block;
            max-width: 128px;

```

```

        margin-bottom: 10px;
        text-align: center;
    }
    .entity-link-container-class > a {
        font-size: 1.4em;
        line-height: 1.5em;
        display: block;
        max-width: 128px;
        margin-bottom: 10px;
        color: #444;
        text-decoration: none;
        text-overflow: ellipsis;
        overflow: hidden;
        white-space: nowrap;
    }
    .entity-link-container-class > a:hover {
        color: #0e84cf;
    }
    .entity-link-container-class {
        float: right;
        width: 128px;
        text-align: center;
    }
    .select-entity-container-class {
        float: left;
        width: 2em;
    }
    .listed-mode-button {
        border-top-right-radius: 1px;
        border-bottom-right-radius: 1px;
    }
    .tiled-mode-button {
        border-top-left-radius: 1px;
        border-bottom-left-radius: 1px;
    }
    .tiled-mode-button, .listed-mode-button {
        padding-left: 0.308em;
        padding-right: 0.462em;
    }
}
.button-pressed {
    background: #fff;

    .t-btn-image {
        background-position: 0 16px !important;
    }
}
div[data-item-marker*="tiled"] {

```

```

        .tiled-mode-button {
            .button-pressed;
        }
    }
    div[data-item-marker="listed"] {
        .listed-mode-button {
            .button-pressed;
        }
    }
}

```

5. На панели инструментов дизайнера нажмите [Сохранить] ([Save]).

2. Модифицировать схему модели представления детали

Чтобы **использовать созданный модуль и его стили** в схеме детали:

1. Откройте схему `UsrSchemae9733d1bDetail` модели представления детали [Прикрепленные фотографии] ([*Photos attachment*]).
2. В зависимости схемы `UsrSchemae9733d1bDetail` добавьте модуль `UsrSchemaDetailCSS`.

Исходный код модифицированной схемы представлен ниже.

`UsrSchemae9733d1bDetail`

```

define("UsrSchemae9733d1bDetail", ["css!UsrSchemaDetailCSS"], function() {
    return {
        entitySchemaName: "UsrPhotosFile",
        details: /**SCHEMA_DETAILS*/{}/**SCHEMA_DETAILS*/,
        methods: {},
        diff: /**SCHEMA_DIFF*/[]/**SCHEMA_DIFF*/
    };
});

```

3. На панели инструментов дизайнера нажмите [Сохранить] ([Save]).

Результат выполнения примера

В результате на страницу записи пользовательского раздела [Фотографии] ([*Photos*]) добавлена деталь [Прикрепленные фотографии] ([*Photos attachment*]).

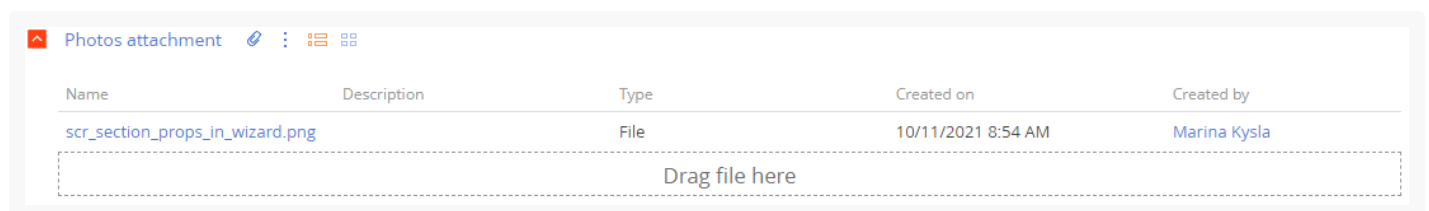


Схема BaseDetailV2 JS

 Сложный

Для того чтобы мастер деталей мог корректно работать с созданной разработчиком деталью, ее необходимо зарегистрировать. Для регистрации детали необходимо в таблицу `SysDetail` добавить запись с указанием заголовка детали, идентификатора схемы детали `DetailSchemaUid` (колонок `Uid` таблицы `SysSchema`) и идентификатора схемы объекта детали `EntitySchemaUid` (колонок `Uid` таблицы `SysSchema`).

Все схемы деталей должны быть наследниками базовой схемы детали `BaseDetailV2`. В схеме реализована базовая логика инициализации данных и общения со страницей записи.

Сообщения

Сообщения предназначены для общения детали со страницей записи. Полный перечень сообщений, их режим вещания и направление приведены в таблице.

Сообщения базовой детали

Название	Режим	Направление	Описание
<code>GetCardState</code>	Адресное	Публикация	Возвращает состояние страницы записи.
<code>SaveRecord</code>	Адресное	Публикация	Сообщает странице записи о необходимости сохранить данные.
<code>DetailChanged</code>	Адресное	Публикация	Сообщает странице записи об изменении данных детали.
<code>UpdateDetail</code>	Адресное	Подписка	Подписка на обновление страницы записи.
<code>OpenCard</code>	Адресное	Публикация	Открывает страницу записи.

Виды режимов сообщений определены в перечислении `Terrasoft.core.enums.MessageMode`, а направление сообщений — в перечислении `Terrasoft.core.enums.MessageDirectionType`. Подробнее о них можно узнать из [библиотеки классов клиентской части ядра платформы](#).

Атрибуты

Свойство `attributes` содержит атрибуты модели представления детали.

`CanAtd` `BOOLEAN`

Признак возможности добавления данных.

CanEdit BOOLEAN

Признак возможности редактирования данных.

CanDelete BOOLEAN

Признак возможности удаления данных.

Collection COLLECTION

Коллекция данных детали.

Filter CUSTOM_OBJECT

Фильтр детали. Используется для фильтрации данных в детали.

DetailColumnName STRING

Имя колонки, по которой выполняется фильтрация.

MasterRecordId GUID

Значение ключа родительской записи.

IsDetailCollapsed BOOLEAN

Признак свернутости детали.

DefaultValues CUSTOM_OBJECT

Значения колонок модели по умолчанию.

Caption STRING

Заголовок детали.

Возможные типы данных атрибутов представлены перечислением `Terrasoft.DataValueType`. Подробнее о типах данных можно узнать из [библиотеки классов клиентской части ядра платформы](#).

Методы

init

Инициализирует страницу детали.

Параметры

<code>{Function} callback</code>	Функция обратного вызова.
<code>{Object} scope</code>	Контекст выполнения метода.

`initProfile`

Инициализирует профиль схемы. По умолчанию содержит значение `Terrasoft.emptyFn`.

`initDefaultCaption`

Устанавливает заголовок детали по умолчанию.

`initDetailOptions`

Инициализирует коллекцию данных представления реестра.

`subscribeSandboxEvents`

Подписывается на сообщения, необходимые для работы детали.

`getUpdateDetailSandboxTags`

Генерирует массив тегов для сообщения `UpdateDetail`.

`updateDetail`

Обновляет деталь согласно переданным параметрам. По умолчанию содержит значение `Terrasoft.emptyFn`.

Параметры

<code>{Object} config</code>	Конфигурационный объект, содержащий свойства детали.

`initData`

Инициализирует коллекцию данных представления реестра.

Параметры

{Function} callback	Функция обратного вызова.
{Object} scope	Контекст выполнения метода.

getEditPageName

Возвращает имя страницы записи в зависимости от типа выбранной записи (при редактировании) или от выбранного типа записи для добавления (при добавлении).

onDetailCollapsedChanged

Обработчик сворачивания или разворачивания детали.

Параметры

{Boolean} isCollapsed	Признак свернутой/развернутой детали.

getToolsVisible

Возвращает значение свернутости детали.

getDetailInfo

Публикует сообщение для получения информации о детали.

Массив модификаций

В массиве модификаций `diff` базовой детали определен только базовый контейнер для представления детали:

```
diff: /**SCHEMA_DIFF*/[
  // Базовый контейнер для представления детали.
  {
    "operation": "insert",
    "name": "Detail",
    "values": {...}
  }
]/**SCHEMA_DIFF*/
```

Схема BaseGridDetailV2



Является наследником `BaseDetailV2`. Все детали с реестром должны быть наследниками `BaseGridDetailV2`. В схеме `BaseGridDetailV2` реализована базовая логика работы с реестром (загрузка, фильтрация), удаление, добавление и редактирование записей на детали.

Процесс создания пользовательской детали с реестром подробно изложен в [статье](#).

Сообщения

Основные сообщения схемы `BaseGridDetailV2` приведены в таблице.

Сообщения базовой детали с реестром

Название	Режим	Направление	Описание
<code>getCardInfo</code>	Адресное	Подписка	Возвращает информацию о странице записи — значения по умолчанию, название колонки типизации, значение колонки типизации.
<code>CardSaved</code>	Широковещательное	Подписка	Обработывает сообщение сохранения страницы записи.
<code>UpdateFilter</code>	Широковещательное	Подписка	Обновляет фильтры в детали.
<code>GetColumnsValues</code>	Адресное	Публикация	Получает значения колонок модели страницы записи.

Виды режимов сообщений определены в перечислении `Terrasoft.core.enums.MessageMode`, а направление сообщений — в перечислении `Terrasoft.core.enums.MessageDirectionType`. Подробнее о них можно узнать из [библиотеки классов клиентской части ядра платформы](#).

Атрибуты

`ActiveRow` GUID

Значение первичной колонки активной записи реестра.

`IsGridEmpty` BOOLEAN

Признак того, что реестр пуст.

`MultiSelect` BOOLEAN

Признак, разрешен ли множественный выбор.

SelectedRows `COLLECTION`

Массив выбранных записей.

RowCount `INTEGER`

Количество строк в реестре.

IsPageable `BOOLEAN`

Признак активности постраничной загрузки.

SortColumnIndex `INTEGER`

Индекс колонки сортировки.

CardState `TEXT`

Режим открытия страницы записи.

EditPageUid `GUID`

Уникальный идентификатор страницы записи.

ToolsButtonMenu `COLLECTION`

Коллекция выпадающего списка функциональной кнопки.

DetailFilters `COLLECTION`

Коллекция фильтров детали.

IsDetailWizardAvailable `BOOLEAN`

Признак, что мастер детали доступен.

Возможные типы данных атрибутов представлены перечислением `Terrasoft.DataValueType`. Подробнее о типах данных детали можно узнать из [библиотеки классов клиентской части ядра платформы](#).

Миксины

`GridUtilities` Terrasoft.GridUtilities

Миксин для работы с реестром.

`WizardUtilities` Terrasoft.WizardUtilities

Миксин для работы с мастером деталей.

Подробнее миксин `GridUtilities` описан ниже.

Методы

`init`

Замещает метод класса `BaseDetailV2`. Вызывает логику метода `init` родителя, регистрирует сообщения, инициализирует фильтры.

Параметры

<code>{Function} callback</code>	Функция обратного вызова.
<code>{Object} scope</code>	Контекст выполнения метода.

`initData`

Замещение метода класса `BaseDetailV2`. Вызывает логику метода `initData` родительского класса, инициализирует коллекцию данных представления реестра.

Параметры

<code>{Function} callback</code>	Функция обратного вызова.
<code>{Object} scope</code>	Контекст выполнения метода.

`loadGridData`

Выполняет загрузку данных реестра.

`initGridData`

Выполняет инициализацию значений по умолчанию для работы со списком.

`getGridData`

Возвращает коллекцию реестра.

`getFilters`

Возвращает коллекцию фильтров детали.

`getActiveRow`

Возвращает идентификатор выбранной записи в реестре.

`atdRecord`

Добавляет новую запись в реестр. Сохраняет страницу записи в случае необходимости.

Параметры

<code>{String} editPageUid</code>	Идентификатор типизированной страницы записи.

`copyRecord`

Копирует запись и открывает страницу записи.

Параметры

<code>{String} editPageUid</code>	Идентификатор типизированной страницы записи.

`editRecord`

Открывает страницу выбранной записи.

Параметры

<code>{Object} record</code>	Модель записи для редактирования.

`subscribeSandboxEvents`

Подписывается на сообщения, необходимые для работы детали.

`updateDetail`

Замещение метода класса `BaseDetailV2`. Вызывает логику метода `updateDetail` родителя, обновляет деталь.

Параметры

<code>{Object} config</code>	Конфигурационный объект, содержащий свойства детали.

`openCard`

Открывает страницу записи.

Параметры

<code>{String} operation</code>	Тип операции (добавление/редактирование).
<code>{String} typeColumnValue</code>	Значение колонки типизации записи.
<code>{String} recordId</code>	Идентификатор записи.

`onCardSaved`

Обработывает событие сохранения страницы записи, в которой находится деталь.

`atdToolsButtonMenuItems`

Добавляет элементы в коллекцию выпадающего списка функциональной кнопки.

Параметры

<code>{Terrasoft.BaseViewModelCollection} toolsButtonMenu</code>	Коллекция выпадающего списка функциональной кнопки.

`initDetailFilterCollection`

Инициализирует фильтр детали.

`setFilter`

Устанавливает значение фильтров детали.

Параметры

<code>{String} key</code>	Тип фильтров.
<code>{Object} value</code>	Значение фильтров.

loadQuickFilter

Загружает быстрый фильтр.

Параметры

{Object} config

Параметры загрузки модуля фильтров.

destroy

Очищает данные, выгружает деталь.

Массив модификаций

В массиве модификаций `diff` базовой детали определен только базовый контейнер для представления детали:

Массив модификаций `diff`

```
diff: /**SCHEMA_DIFF*/ [
  {
    // Элемент для отображения реестра.
    "operation": "insert",
    "name": "DataGrid",
    "parentName": "Detail",
    "propertyName": "items",
    "values": {
      "itemType": Terrasoft.ViewItemType.GRID,
      ...
    }
  },
  {
    // Кнопка дозагрузки реестра.
    "operation": "insert",
    "parentName": "Detail",
    "propertyName": "items",
    "name": "loadMore",
    "values": {
      "itemType": Terrasoft.ViewItemType.BUTTON,
      ...
    }
  },
  {
    // Кнопка добавления записи.
    "operation": "insert",
    "name": "AddRecordButton",
```

```

        "parentName": "Detail",
        "propertyName": "tools",
        "values": {
            "itemType": Terrasoft.ViewItemType.BUTTON,
            ...
        }
    },
    {
        // Кнопка добавления типизированной записи.
        "operation": "insert",
        "name": "AddTypedRecordButton",
        "parentName": "Detail",
        "propertyName": "tools",
        "values": {
            "itemType": Terrasoft.ViewItemType.BUTTON,
            ...
        }
    },
    {
        // Меню детали.
        "operation": "insert",
        "name": "ToolsButton",
        "parentName": "Detail",
        "propertyName": "tools",
        "values": {
            "itemType": Terrasoft.ViewItemType.BUTTON,
            ...
        }
    }
}
] /**SCHEMA_DIFF*/

```

Миксин GridUtilitiesV2 JS



Сложный

`GridUtilitiesV2` — миксин, реализующий логику работы с элементом управления "реестр". В классе `Terrasoft.configuration.mixins.GridUtilities` реализованы:

1. Подписка на сообщения.
2. Загрузка данных.
3. Работа с реестром:
 - выбор записей (поиск активных);
 - добавление, удаление, редактирование записей;
 - установка фильтров;
 - сортировка;

- экспорт в файл;
- проверка прав доступа к записям реестра.

Методы

`init`

Осуществляет подписку на события.

`destroy`

Очищает подписки на события.

`loadGridData`

Выполняет загрузку данных реестра.

`beforeLoadGridData`

Подготавливает модель представления перед загрузкой данных.

`afterLoadGridData`

Подготавливает модель представления после загрузки данных.

`onGridDataLoaded`

Обработчик события загрузки данных. Выполняется, когда сервер возвращает данные.

Параметры

<code>{Object}</code> response	Результат выборки данных из БД.

`addItemToGridData`

Добавляет коллекцию новых элементов в коллекцию реестра.

Параметры

<code>{Object}</code> dataCollection	Коллекция новых элементов.
<code>{Object}</code> options	Параметры добавления.

`reloadGridData`

Выполняет перезагрузку реестра.

`initQueryOptions`

Инициализирует настройки (постраничность, иерархичность) экземпляра запроса.

Параметры

<code>{Terrasoft.EntitySchemaQuery}</code> <code>esq</code>	Запрос, в котором будут инициализированы необходимые настройки.

`initQuerySorting`

Инициализирует колонки сортировки.

Параметры

<code>{Terrasoft.EntitySchemaQuery}</code> <code>esq</code>	Запрос, в котором будут инициализированы необходимые настройки.

`prepareResponseCollection`

Модифицирует коллекцию данных перед загрузкой в реестр.

Параметры

<code>{Object}</code> <code>collection</code>	Коллекция элементов реестра.

`getFilters`

Возвращает примененные в данной схеме фильтры. Переопределяется в наследниках.

`exportToFile`

Экспортирует содержимое реестра в файл.

`sortGrid`

Выполняет сортировку в реестре.

Параметры

<code>{String} tag</code>	Ключ, указывающий, каким образом пересортировать реестр.

`deleteRecords`

Иницирует удаление выбранных записей.

`checkCanDelete`

Проверяет возможность удаления записи.

Параметры

<code>{Array} items</code>	Идентификаторы выбранных записей.
<code>{Function} callback</code>	Функция обратного вызова.
<code>{Object} scope</code>	Контекст выполнения метода.

`onDeleteAccept`

Выполняет удаление после подтверждения пользователем.

`getSelectedItems`

Возвращает выбранные записи в реестре.

`removeGridRecords`

Убирает из реестра удаленные записи.

Параметры

<code>{Array} records</code>	Удаленные записи.

Модуль ConfigurationGrid



Сложный

Модуль `ConfigurationGrid` содержит реализацию элемента управления "Конфигурационный реестр".

Класс `Terrasoft.ConfigurationGrid` является наследником класса `Terrasoft.Grid`.

Методы

`init`

Инициализирует компонент. Осуществляет подписку на события.

`activateRow`

Выделяет строку и добавляет элементы редактирования.

Параметры

<code>{String Number} id</code>	Идентификатор строки реестра.

`deactivateRow`

Снимает выделение строки и удаляет элементы редактирования.

Параметры

<code>{String Number} id</code>	Идентификатор строки реестра.

`formatCellContent`

Форматирует данные ячейки строки.

Параметры

<code>{Object} cell</code>	Ячейка.
<code>{Object} data</code>	Данные.
<code>{Object} column</code>	Конфигурация ячейки.

`onUpdateItem`

Обработчик события обновления записи.

Параметры

<code>{Terrasoft.BaseViewModel} item</code>	Элемент коллекции.

`onDestroy`

Уничтожает реестр и его компоненты.

Модуль ConfigurationGridGenerator

 Сложный

Класс `Terrasoft.ConfigurationGridGenerator` генерирует конфигурацию реестра и является наследником класса `Terrasoft.ViewGenerator`.

Методы

`addLinks`

Переопределенный метод класса `Terrasoft.ViewGenerator`. В редактируемый реестр не будут добавлены ссылки.

`generateGridCellValue`

Переопределенный метод класса `Terrasoft.ViewGenerator`. Генерирует конфигурацию значения в ячейке.

Параметры

`{Object} config`

Конфигурация колонки.

Модуль ConfigurationGridUtilities

 Сложный

Класс `Terrasoft.ConfigurationGridUtilities` содержит методы инициализации модели представления строки реестра, обработки действий активной записи и обработки горячих клавиш.

Свойства

`currentActiveColumnName String`

Имя текущей выделенной колонки.

`columnsConfig Object`

Конфигурация колонок.

`systemColumns` Array

Коллекция названий системных колонок.

Методы

`onActiveRowAction`

Обрабатывает нажатие действия активной записи.

Параметры

<code>{String} buttonTag</code>	Тег выбранного действия.
<code>{String} primaryColumnValue</code>	Идентификатор активной записи.

`saveRowChanges`

Сохраняет запись.

Параметры

<code>{Object} row</code>	Строка реестра.
<code>{Function} [callback]</code>	Функция обратного вызова.
<code>{Object} scope</code>	Контекст вызова функции обратного вызова.

`activeRowSaved`

Обрабатывает результат сохранения записи.

Параметры

<code>{Object} row</code>	Строка реестра.
<code>{Function} [callback]</code>	Функция обратного вызова.
<code>{Object} scope</code>	Контекст вызова функции обратного вызова.

`initActiveRowKeyMap`

Инициализирует подписку на события нажатия кнопок в активной строке.

Параметры

{Array} keyMap	Описание событий.

getCellControlsConfig

Возвращает конфигурацию элементов редактирования ячейки реестра.

Параметры

{Terrasoft.EntitySchemaColumn} entitySchemaColumn	Колонка ячейки реестра.

copyRow

Копирует и добавляет запись в реестр.

Параметры

{String} recordId	Идентификатор копируемой записи.

initEditableGridRowViewModel

Инициализирует классы элементов коллекции редактируемого реестра.

Параметры

{Function} [callback]	Функция обратного вызова.
{Object} scope	Контекст вызова функции обратного вызова.

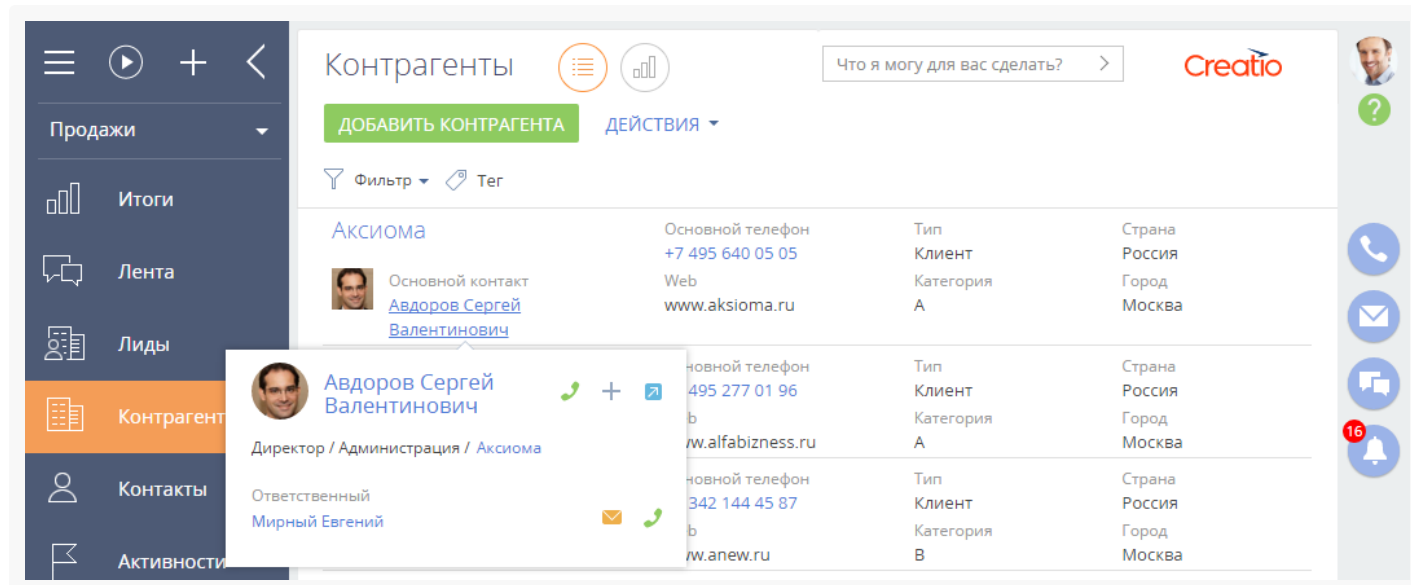
Мини-карточка



Основы

Мини-карточка — сокращенная версия страницы записи с ограниченным количеством полей. Мини-карточки позволяют быстро получить или отредактировать информацию о записи, не открывая отдельную страницу. **Назначение** мини-карточки — ускорение добавления, редактирования и просмотра записей. Набор полей в каждом из типов мини-карточек настраивается отдельно и будет различаться.

Мини-карточка контакта



Мини-карточку можно создать для любого объекта системы.

Подробности о работе с мини-карточками описаны в статье [Мини-карточки](#).

Схема модели представления мини-карточки

В Creatio IDE мини-карточка реализована с помощью [схемы модели представления](#).

Схема модели представления мини-карточки **позволяет настроить**:

- Состав мини-карточки.
- Расположение элементов пользовательского интерфейса мини-карточки.
- Поведение элементов пользовательского интерфейса мини-карточки.

Например, мини-карточка контакта конфигурируется схемой `ContactMiniPage`, а мини-карточка контрагента — схемой `AccountMiniPage`. Родительской схемой для схем-миникарточек является схема `BaseMiniPage` пакета `NUI`.

Структура схемы модели представления мини-карточки не отличается от общей структуры [клиентской схемы модели представления](#).

Обязательные свойства в структуре схемы мини-карточки:

- `entitySchemaName` — имя схемы объекта, к которому будет привязана мини-карточка,
- `diff` — массив модификаций визуальных элементов мини-карточки.

Дополнительные свойства в структуре схемы мини-карточки:

- `attributes` — атрибуты схемы.
- `methods` — методы схемы.
- `mixins` — миксины схемы.
- `messages` — сообщения схемы.

Дополнительные свойства **позволяют**:

- Добавлять пользовательские элементы управления.
- Регистрировать сообщения.
- Формировать бизнес-логику работы мини-карточки.

Существует возможность изменения внешнего вида визуальных элементов мини-карточки с помощью пользовательских стилей.

Важно. В мини-карточках не поддерживается механизм настройки бизнес-логики с помощью бизнес-правил.

Операции с мини-карточками

Добавить мини-карточку в раздел

1. В пользовательский [пакет](#) добавьте [схему модели представления](#) карточки.
2. В качестве родительского объекта выберите схему `BaseMiniPage`.
3. Добавьте необходимую функциональность мини-карточки в исходный код схемы. При этом в элементе `entitySchemaName` обязательно укажите имя схемы объекта, к которому будет привязана мини-карточка, и внесите хотя бы одну модификацию в массив `diff`.
4. Используя SQL-запрос, внесите изменения в системную таблицу `[SysModuleEdit]` базы данных.
5. Добавьте системную настройку `[HasКодРазделаMiniPageAddMode]`. Добавление системной настройки описано в статье [Управление системными настройками](#).

Важно. Выполнение SQL-запроса, который содержит ошибку, может привести к повреждению существующих данных и неработоспособности приложения.

Добавить мини-карточку к произвольному модулю

Для некоторых бизнес-задач возникает необходимость подключения мини-карточки к произвольному модулю Creatio. **Произвольные модули** позволяют создавать ссылки в системе на определенный объект, поэтому подключение отображения мини-карточки при наведении на ссылку позволяет получить информацию об этом объекте, не переходя в раздел этого объекта.

В базовой версии приложения мини-карточка объекта подключена к следующим модулям:

- телефония в коммуникационной панели;
- email в коммуникационной панели;
- центр уведомлений в коммуникационной панели;
- раздел [Лента] в коммуникационной панели;
- графика-списка в разделе итогов.

Чтобы **добавить мини-карточку к произвольному модулю**:

1. Создайте схему модуля.
2. Создайте представление и модель представления модуля. Подключите в свойство `mixins` модели представления утилитный класс `Terrasoft.MinipageUtilities`. Класс позволит использовать методы вызова мини-карточки.
3. Добавьте стили модуля.
4. Создайте контейнер отображения представления модуля.

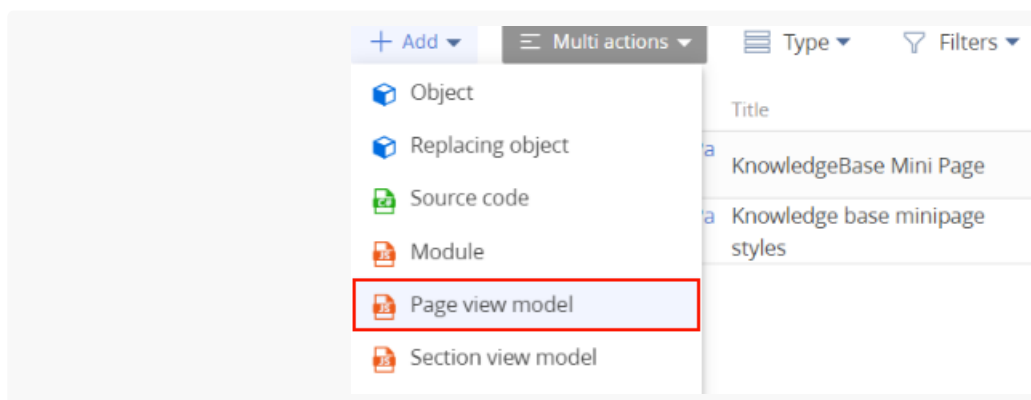
Создать пользовательскую мини-карточку



Пример. Создать пользовательскую мини-карточку для раздела [*База знаний*] ([*Knowledge base*]). Мини-карточка будет служить для просмотра базового набора полей [*Название*] ([*Name*]) и [*Теги*] ([*Tags*]) с возможностью скачивания прикрепленных файлов.

1. Создать схему модели представления мини-карточки

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



3. В дизайнера схем заполните свойства схемы:

- [*Код*] ([*Code*]) — "UsrKnowledgeBaseArticleMiniPage".
- [*Заголовок*] ([*Title*]) — "Миникарточка базы знаний" ("KnowledgeBase Mini Page").
- [*Родительский объект*] ([*Parent object*]) — выберите "BaseMiniPage".

Module [X]

Code *
UsrKnowledgeBaseArticleMiniPage

Title *
KnowledgeBase Mini Page

Parent object
BaseMiniPage (BaseMiniPage)

Package
sdkMiniCardCreating

Description

CANCEL APPLY

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

2. Отобразить поля основного объекта

В дизайнере схем добавьте необходимый исходный код.

1. В качестве схемы объекта укажите схему `KnowledgeBase` .
2. Добавьте необходимые модификации в массив модификаций модели представления `diff` .

Элементы модели представления базовой мини-карточки:

- `MiniPage` — поле карточки.
- `HeaderContainer` — заголовок карточки (по умолчанию размещается в первом ряду поля карточки).

В примере в массив модификаций `diff` добавлены два объекта, которые конфигурируют поля [*Name*] и [*Keywords*].

Исходный код схемы модели представления приведен ниже.

UsrKnowledgeBaseArticleMiniPage.js

```
define("UsrKnowledgeBaseArticleMiniPage", [], function() {
    return {
        entitySchemaName: "KnowledgeBase",
        attributes: {
            "MiniPageModes": {
                "value": [this.Terrasoft.ConfigurationEnums.CardOperation.VIEW]
            }
        }
    },
    {
```

```

diff: /**SCHEMA_DIFF*/[
  {
    "operation": "insert",
    "name": "Name",
    "parentName": "HeaderContainer",
    "propertyName": "items",
    "index": 0,
    "values": {
      "labelConfig": {
        "visible": false
      },
      "isMiniPageModelItem": true
    }
  },
  {
    "operation": "insert",
    "name": "Keywords",
    "parentName": "MiniPage",
    "propertyName": "items",
    "values": {
      "labelConfig": {
        "visible": false
      },
      "isMiniPageModelItem": true,
      "layout": {
        "column": 0,
        "row": 1,
        "colSpan": 24
      }
    }
  }
]/**SCHEMA_DIFF*/
};
});

```

3. Добавить функциональную кнопку в мини-карточку

По условию примера карточка должна обеспечивать скачивание файлов, связанных со статьей базы знаний.

Работа с дополнительными данными обеспечивается с помощью механизма их отображения в виде выпадающего списка преднастроенной кнопки.

В дизайнера схем измените исходный код модели представления.

Для **добавления кнопки** выбора файлов статьи базы знаний:

1. Добавьте в массив `diff` элемент `FilesButton`, который является описанием кнопки
2. Добавьте в свойство `attributes` виртуальную колонку `Article`, которая связывает основную и

дополнительные записи.

3. Добавьте в свойство `attributes` трибут `MiniPageModes` — массив, который содержит коллекцию необходимых операций, выполняемых мини-карточкой.
4. Добавьте в ресурсы схемы изображение кнопки. Например, можно использовать это изображение — . Добавление изображения в ресурсы описано в статье [Добавить поле с изображением](#).
5. Добавьте в свойство `methods` методы работы с выпадающим списком кнопки выбора файла:
 - `init()` — переопределенный базовый метод.
 - `onEntityInitialized()` — переопределенный базовый метод.
 - `setArticleInfo()` — устанавливает значение атрибута `Article`.
 - `getFiles(callback, scope)` — получает информацию о файлах текущей статьи базы знаний.
 - `initFilesMenu(files)` — наполняет коллекцию выпадающего списка кнопки выбора файлов.
 - `fillFilesExtendedMenuData()` — инициирует загрузку файлов и их добавление в выпадающий список кнопки выбора файлов.
 - `downloadFile()` — инициирует скачивание выбранного файла.

Исходный код схемы модели представления, который добавляет функциональную кнопку, приведен ниже.

UsrKnowledgeBaseArticleMiniPage.js

```
define("UsrKnowledgeBaseArticleMiniPage",
  ["terrasoft", "KnowledgeBaseFile", "ConfigurationConstants"],
  function(Terrasoft, KnowledgeBaseFile, ConfigurationConstants) {
    return {
      entitySchemaName: "KnowledgeBase",
      attributes: {
        "MiniPageModes": {
          "value": [this.Terrasoft.ConfigurationEnums.CardOperation.VIEW]
        },
        "Article": {
          "type": Terrasoft.ViewModelColumnType.VIRTUAL_COLUMN,
          "referenceSchemaName": "KnowledgeBase"
        }
      },
      methods: {
        /* Инициализирует коллекцию выпадающего списка кнопки выбора файлов.*/
        init: function() {
          this.callParent(arguments);
          this.initExtendedMenuButtonCollections("File", ["Article"], this.close);
        },
        /* Инициализирует значение атрибута, связывающего основную и дополнительные записи.
        Наполняет коллекцию выпадающего списка кнопки выбора файлов.*/
        onEntityInitialized: function() {
          this.callParent(arguments);
        }
      }
    };
  }
);
```

```

        this.setArticleInfo();
        this.fillFilesExtendedMenuData();
    },
    /* Иницирует загрузку файлов и их добавление в выпадающий список кнопки выбора
    fillFilesExtendedMenuData: function() {
        this.GetFiles(this.initFilesMenu, this);
    },
    /* Устанавливает значение атрибута, связывающего основную и дополнительные записи
    setArticleInfo: function() {
        this.set("Article", {
            value: this.get(this.primaryColumnName),
            displayValue: this.get(this.primaryDisplayColumnName)
        });
    },
    /* Получает информацию о файлах текущей статьи базы знаний.*/
    getFiles: function(callback, scope) {
        var esq = this.Ext.create("Terrasoft.EntitySchemaQuery", {
            rootSchema: KnowledgeBaseFile
        });
        esq.addColumn("Name");
        var articleFilter = >this.Terrasoft.createColumnFilterWithParameter(
            this.Terrasoft.ComparisonType.EQUAL, "KnowledgeBase", this.get(this.primaryColumnName);
        var typeFilter = this.Terrasoft.createColumnFilterWithParameter(
            this.Terrasoft.ComparisonType.EQUAL, "Type", ConfigurationConstants.FileType);
        esq.filters.addItem(articleFilter);
        esq.filters.addItem(typeFilter);
        esq.getEntityCollection(function(response) {
            if (!response.success) {
                return;
            }
            callback.call(scope, response.collection);
        }, this);
    },
    /* Наполняет коллекцию выпадающего списка кнопки выбора файлов.*/
    initFilesMenu: function(files) {
        if (files.isEmpty()) {
            return;
        }
        var data = [];
        files.each(function(file) {
            data.push({
                caption: file.get("Name"),
                tag: file.get("Id")
            });
        }, this);
        var recipientInfo = this.fillExtendedMenuItems("File", ["Article"]);
        this.fillExtendedMenuData(data, recipientInfo, this.downloadFile);
    },
    /* Иницирует скачивание выбранного файла.*/

```



```

downloadFile: function(id) {
    var element = document.createElement("a");
    element.href = "../rest/FileService/GetFile/" + KnowledgeBaseFile.uId + "/"
    document.body.appendChild(element);
    element.click();
    document.body.removeChild(element);
}
},
diff: /**SCHEMA_DIFF*/[
    {
        "operation": "insert",
        "name": "Name",
        "parentName": "HeaderContainer",
        "propertyName": "items",
        "index": 0,
        "values": {
            "labelConfig": {
                "visible": true
            },
            "isMiniPageModelItem": true
        }
    },
    {
        "operation": "insert",
        "name": "Keywords",
        "parentName": "MiniPage",
        "propertyName": "items",
        "values": {
            "labelConfig": {
                "visible": true
            },
            "isMiniPageModelItem": true,
            "layout": {
                "column": 0,
                "row": 1,
                "colSpan": 24
            }
        }
    },
    {
        "operation": "insert",
        "parentName": "HeaderContainer",
        "propertyName": "items",
        "name": "FilesButton",
        "values": {
            "itemType": Terrasoft.ViewItemType.BUTTON,
            /* Настройка изображения кнопки.*/
            "imageConfig": {

```

```

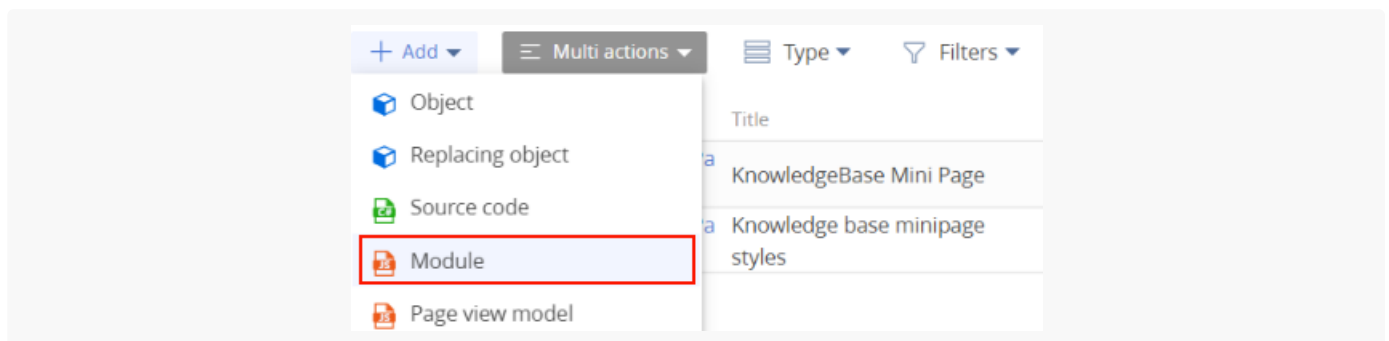
        /* Изображение предварительно необходимо добавить в ресурсы мини-кар
        "bindTo": "Resources.Images.FileImage"
    },
    /* Настройка выпадающего списка.*/
    "extendedMenu": {
        /* Название элемента выпадающего списка.*/
        "Name": "File",
        /* Название атрибута миникарточки, связывающего основную и дополните
        "PropertyName": "Article",
        /* Настройка обработчика нажатия на кнопку.*/
        "Click": {
            "bindTo": "fillFilesExtendedMenuData"
        }
    }
},
    "index": 1
}
]/**SCHEMA_DIFF*/
};
});

```

4. Выполнить стилизацию мини-карточки

Для добавления стилей в модель представления необходимо создать отдельный модуль со стилями и подключить этот модуль к схеме модели представления.

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



3. В дизайнере схем заполните свойства схемы:

- [*Код*] ([*Code*]) — "UsrKnowledgeBaseArticleMiniPageCss".
- [*Заголовок*] ([*Title*]) — "Стили миникарточки базы знаний" ("Knowledge base minipage styles").

Module [X]

Code *
UsrKnowledgeBaseArticleMiniPageCss

Title *
Knowledge base minipage styles

Package
sdkMiniCardCreating

Description

CANCEL APPLY

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

4. В контекстном меню узла LESS и укажите необходимые стили.

UsrKnowledgeBaseArticleMiniPageCss.js

```
div[data-item-marker="UsrKnowledgeBaseArticleMiniPageContainer"] > div {
width: 250px;
}
```

5. В дизайнере схем измените код схемы модели представления: добавьте загрузку этого модуля в исходном коде.

Ниже приведен полный исходный код мини-карточки:

UsrKnowledgeBaseArticleMiniPage.js

```
define("UsrKnowledgeBaseArticleMiniPage",
["terrasoft", "KnowledgeBaseFile", "ConfigurationConstants", "css!UsrKnowledgeBaseArticleMiniPage"],
function(Terrasoft, KnowledgeBaseFile, ConfigurationConstants) {
return {
entitySchemaName: "KnowledgeBase",
attributes: {
"MiniPageModes": {
"value": [this.Terrasoft.ConfigurationEnums.CardOperation.VIEW]
},
"Article": {
"type": Terrasoft.ViewModelColumnType.VIRTUAL_COLUMN,
"referenceSchemaName": "KnowledgeBase"
}
}
},
},
```

```

methods: {
  init: function() {
    this.callParent(arguments);
    this.initExtendedMenuButtonCollections("File", ["Article"], this.close);
  },
  onEntityInitialized: function() {
    this.callParent(arguments);
    this.setArticleInfo();
    this.fillFilesExtendedMenuData();
  },
  fillFilesExtendedMenuData: function() {
    this.GetFiles(this.initFilesMenu, this);
  },
  setArticleInfo: function() {
    this.set("Article", {
      value: this.get(this.primaryColumnName),
      displayValue: this.get(this.primaryDisplayColumnName)
    });
  },
  getFiles: function(callback, scope) {
    var esq = this.Ext.create("Terrasoft.EntitySchemaQuery", {
      rootSchema: KnowledgeBaseFile
    });
    esq.addColumn("Name");
    var articleFilter = this.Terrasoft.createColumnFilterWithParameter(
      this.Terrasoft.ComparisonType.EQUAL, "KnowledgeBase", this.get(this.p
    var typeFilter = this.Terrasoft.createColumnFilterWithParameter(
      this.Terrasoft.ComparisonType.EQUAL, "Type", ConfigurationConstants.F
    esq.filters.addItem(articleFilter);
    esq.filters.addItem(typeFilter);
    esq.getEntityCollection(function(response) {
      if (!response.success) {
        return;
      }
      callback.call(scope, response.collection);
    }, this);
  },
  initFilesMenu: function(files) {
    if (files.isEmpty()) {
      return;
    }
    var data = [];
    files.each(function(file) {
      data.push({
        caption: file.get("Name"),
        tag: file.get("Id")
      });
    }, this);
  },
};

```

```

        var recipientInfo = this.fillExtendedMenuItems("File", ["Article"]);
        this.fillExtendedMenuData(data, recipientInfo, this.downloadFile);
    },
    downloadFile: function(id) {
        var element = document.createElement("a");
        element.href = "../rest/FileService/GetFile/" + KnowledgeBaseFile.uId + "
        document.body.appendChild(element);
        element.click();
        document.body.removeChild(element);
    }
},
diff: /**SCHEMA_DIFF*/[
    {
        "operation": "insert",
        "name": "Name",
        "parentName": "HeaderContainer",
        "propertyName": "items",
        "index": 0,
        "values": {
            "labelConfig": {
                "visible": true
            },
            "isMiniPageModelItem": true
        }
    },
    {
        "operation": "insert",
        "name": "Keywords",
        "parentName": "MiniPage",
        "propertyName": "items",
        "values": {
            "labelConfig": {
                "visible": true
            },
            "isMiniPageModelItem": true,
            "layout": {
                "column": 0,
                "row": 1,
                "colSpan": 24
            }
        }
    },
    {
        "operation": "insert",
        "parentName": "HeaderContainer",
        "propertyName": "items",
        "name": "FilesButton",
        "values": {
            "itemType": Terrasoft.ViewItemType.BUTTON,

```

```

        "imageConfig": {
            "bindTo": "Resources.Images.FilesImage"
        },
        "extendedMenu": {
            "Name": "File",
            "PropertyName": "Article",
            "Click": {
                "bindTo": "fillFilesExtendedMenuData"
            }
        }
    },
    "index": 1
}
]/**SCHEMA_DIFF*/
});
});

```

5. Зарегистрировать мини-карточку в базе данных

Создание мини-карточки предполагает ее обязательную регистрацию в базе данных. Для внесения изменений в базу данных выполните следующий SQL-запрос.

Запрос на создание мини-карточки

```

DECLARE
    -- Название схемы представления создаваемой мини-карточки.
    @ClientUnitSchemaName NVARCHAR(100) = 'UsrKnowledgeBaseArticleMiniPage',
    -- Название схемы объекта, к которому привязывается мини-карточка.
    @EntitySchemaName NVARCHAR(100) = 'KnowledgeBase'

UPDATE SysModuleEdit
SET MiniPageSchemaUid = (
    SELECT TOP 1 Uid
    FROM SysSchema
    WHERE Name = @ClientUnitSchemaName
)
WHERE SysModuleEntityId = (
    SELECT TOP 1 Id
    FROM SysModuleEntity
    WHERE SysEntitySchemaUid = (
        SELECT TOP 1 Uid
        FROM SysSchema
        WHERE Name = @EntitySchemaName
        AND ExtendParent = 0
    )
);

```

В результате выполнения запроса уникальный идентификатор мини-карточки будет добавлен в таблицу [SysModuleEdit] в поле [MiniPageSchemaUId] записи, соответствующей разделу [База знаний] ([Knowledge base]).

	CardSchemaUId	ActionKindCaption	ActionKindName	PageCaption	MiniPageSchemaUId
50	9DBD0611-FA52-4A90-9542-E5FD997B4AFD	New article	KnowledgeBase	Knowledge base article	9C072BC6-93C7-488A-87A6-A41B79CC67...

6. Добавить системную настройку

В разделе [Системные настройки] ([System settings]) дизайнера системы добавьте системную настройку со следующими свойствами:

- [Название] ([Name]) — "HasKnowledgeBaseMiniPageAddMode".
- [Код] ([Code]) — "HasKnowledgeBaseMiniPageAddMode".
- [Тип] ([Type]) — "Логическое" ("Boolean").
- [Значение по умолчанию] ([Default value]) — признак установлен.

HasKnowledgeBaseMiniPageAddMode

What can I do for you?

Creatio

SAVE

CANCEL

Name*

HasKnowledgeBaseMiniPageAddMode

Code*

HasKnowledgeBaseMiniPageAddMode

Type*

Boolean

Cached

☒

Default value

☒

Personal

☐

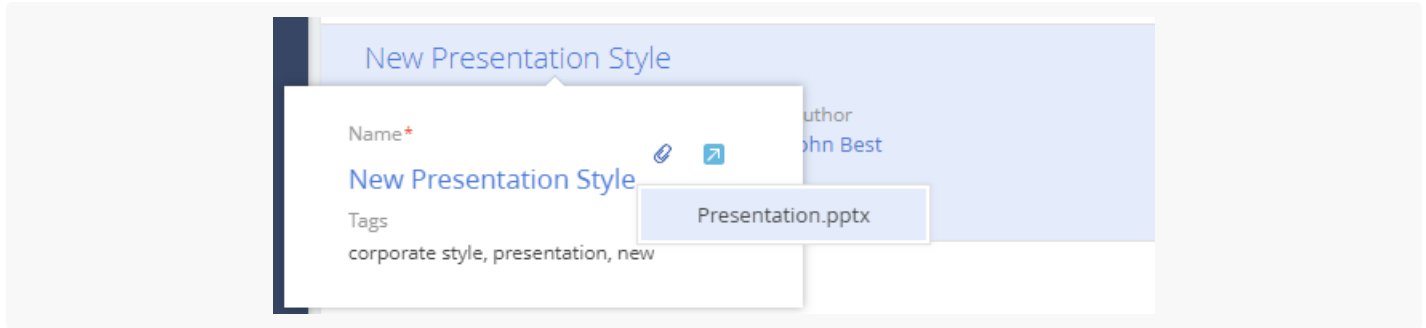
Allow for portal users

☐

Description

Результат выполнения примера

После сохранения схемы и обновления веб-страницы приложения в разделе [База знаний] ([Knowledge base]) при наведении курсора на название будет отображаться пользовательская мини-карточка, в которой будут отображены связанные с записью файлы и реализована возможность скачать их.



Создать мини-карточку добавления

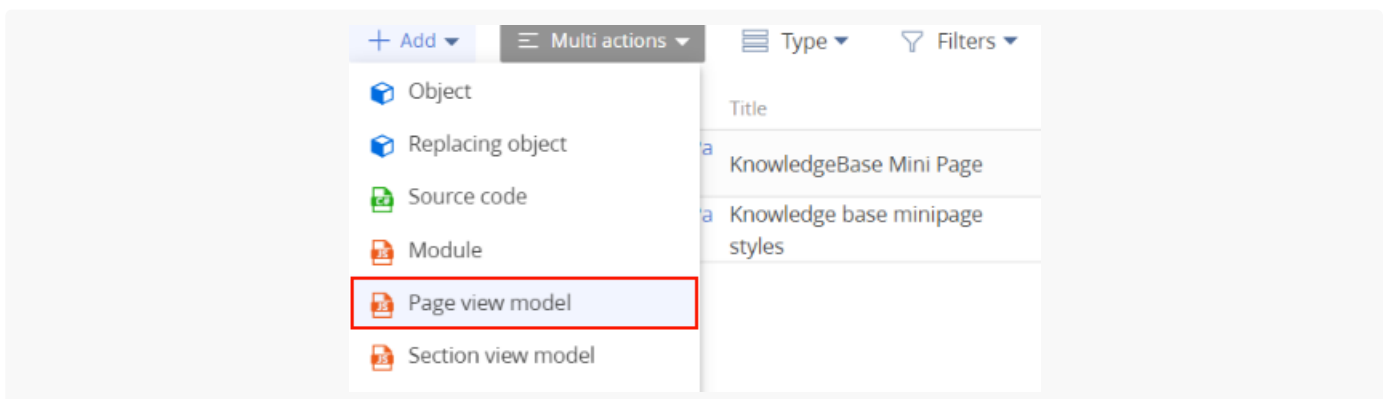


Сложный

Пример. Создать пользовательскую мини-карточку добавления новой записи в раздел [*Продукты*] ([*Products*]). Мини-карточка должна добавлять базовый набор полей [*Название*] ([*Name*]) и [*Код*] ([*Code*]).

1. Создать схему модели представления мини-карточки

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



3. В дизайнере схем заполните свойства схемы:

- [*Код*] ([*Code*]) — "UsrProductMiniPage".
- [*Заголовок*] ([*Title*]) — "Мини-карточка продукта" ("Product Mini Page").
- [*Родительский объект*] ([*Parent object*]) — выберите "BaseMiniPage".

The screenshot shows a 'Module' configuration window with the following fields:

- Code ***: UsrProductMiniPage
- Title ***: Product Mini Page
- Parent object**: BaseMiniPage
- Package**: sdkMiniCardAdding
- Description**: (empty)

At the bottom right, there are two buttons: 'CANCEL' and 'APPLY'.

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

2. Отобразить поля основного объекта

В дизайнере схем добавьте необходимый исходный код.

1. В качестве схемы объекта укажите схему `Product`.
2. Объявите атрибут `MiniPageModes` и присвойте ему массив, содержащий коллекцию необходимых операций, выполняемых мини-карточкой.

На заметку. Если кроме операции добавления новой записи требуется отображение мини-карточки на странице раздела (см. [Создать пользовательскую мини-карточку](#)), то в массив, присваиваемый атрибуту `MiniPageModes`, также необходимо добавить значение `this.Terrasoft.ConfigurationEnums.CardOperation.VIEW`.

3. Добавьте необходимые модификации в массив модификаций `diff` модели представления.

Элементы модели представления базовой мини-карточки:

- `MiniPage` — поле карточки.
- `HeaderContainer` — заголовок карточки (по умолчанию размещается в первом ряду поля карточки).

В примере в массив модификаций `diff` добавлены два объекта, которые конфигурируют поля [`Name`] и [`Code`].

Исходный код схемы модели представления приведен ниже.

UsrProductMiniPage.js — отобразить поля объекта

```

define("UsrProductMiniPage", ["UsrProductMiniPageResources"],
function(resources) {
    return {
        entitySchemaName: "Product",
        details: /**SCHEMA_DETAILS*/{}/**SCHEMA_DETAILS*/,
        attributes: {
            "MiniPageModes": {
                "value": [this.Terrasoft.ConfigurationEnums.CardOperation.ADD]
            }
        },
        diff: /**SCHEMA_DIFF*/[
            {
                "operation": "insert",
                "parentName": "MiniPage",
                "propertyName": "items",
                "name": "Name",
                "values": {
                    "isMiniPageModelItem": true,
                    "layout": {
                        "column": 0,
                        "row": 1,
                        "colSpan": 24
                    },
                    "controlConfig": {
                        "focused": true
                    }
                }
            },
            {
                "operation": "insert",
                "parentName": "MiniPage",
                "propertyName": "items",
                "name": "Code",
                "values": {
                    "isMiniPageModelItem": true,
                    "layout": {
                        "column": 0,
                        "row": 2,
                        "colSpan": 24
                    }
                }
            }
        ]/**SCHEMA_DIFF*/
    };
});

```

3. Зарегистрировать мини-карточку в базе данных

Создание мини-карточки предполагает ее обязательную регистрацию в базе данных. Для внесения изменений в базу данных выполните следующий SQL-запрос.

Запрос на создание мини-карточки

```
DECLARE
    -- Название схемы представления создаваемой мини-карточки.
    @ClientUnitSchemaName NVARCHAR(100) = 'UsrProductMiniPage',
    -- Название схемы объекта, к которому привязывается мини-карточка.
    @EntitySchemaName NVARCHAR(100) = 'Product'

UPDATE SysModuleEdit
SET MiniPageSchemaUid = (
    SELECT TOP 1 Uid
    FROM SysSchema
    WHERE Name = @ClientUnitSchemaName
)
WHERE SysModuleEntityId = (
    SELECT TOP 1 Id
    FROM SysModuleEntity
    WHERE SysEntitySchemaUid = (
        SELECT TOP 1 Uid
        FROM SysSchema
        WHERE Name = @EntitySchemaName
        AND ExtendParent = 0
    )
);
```

В результате выполнения запроса уникальный идентификатор мини-карточки будет добавлен в таблицу [SysModuleEdit] в поле [MiniPageSchemaUid] записи, соответствующей разделу [*Продукты*] ([*Products*]) .

CardSchemaUid	ActionKindCaption	ActionKindName	PageCaption	MiniPageSchemaUid	SearchRowSchemaUid
0DAEC87E-A84D-44BC-9DD0-C90B8D1BAA33	New product	Product	Product	15D85C82-D78F-400F-B10C-44BB13C72282	NULL

4. Добавить системную настройку

В разделе [*Системные настройки*] ([*System settings*]) дизайнера системы добавьте системную настройку со следующими свойствами :

- [*Название*] ([*Name*]) — "HasProductMiniPageAddMode".
- [*Код*] ([*Code*]) — "HasProductMiniPageAddMode".
- [*Тип*] ([*Type*]) — "Логическое" ("Boolean").
- [*Значение по умолчанию*] ([*Default value*]) — признак установлен.

HasProductMiniPageAddM...

SAVE **CANCEL**

Name* HasProductMiniPageAddMode Code* HasProductMiniPageAddMode

Type* Boolean Cached ☒

Default value ☒ Personal ☐

Allow for portal users ☐

Description

Результат выполнения примера

В результате выполнения примера при добавлении нового продукта будет отображаться мини-карточка с двумя полями.

Products

NEW PRODUCT **ACTIONS** ▼

Filter ▼ Tag

Motherboards

Type Motherboards

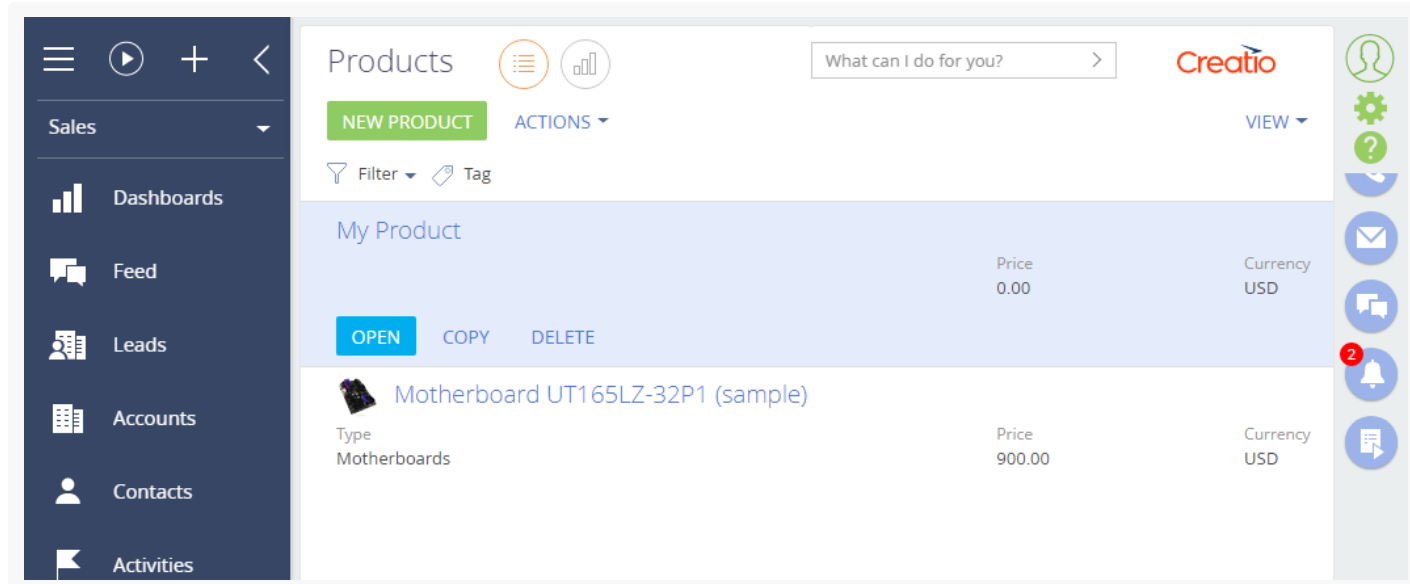
Product

Name* My Product

Code Code12345

SAVE **CANCEL**

После сохранения мини-карточки соответствующая запись появится в реестре раздела.

**Важно.**

Запись в реестре раздела будет отображена только после обновления страницы браузера. Чтобы запись отображалась сразу же после сохранения мини-карточки, необходимо добавить соответствующую функциональность в схему мини-карточки и страницы раздела, используя механизм сообщений (см. статью [Обмен сообщениями между модулями](#)).

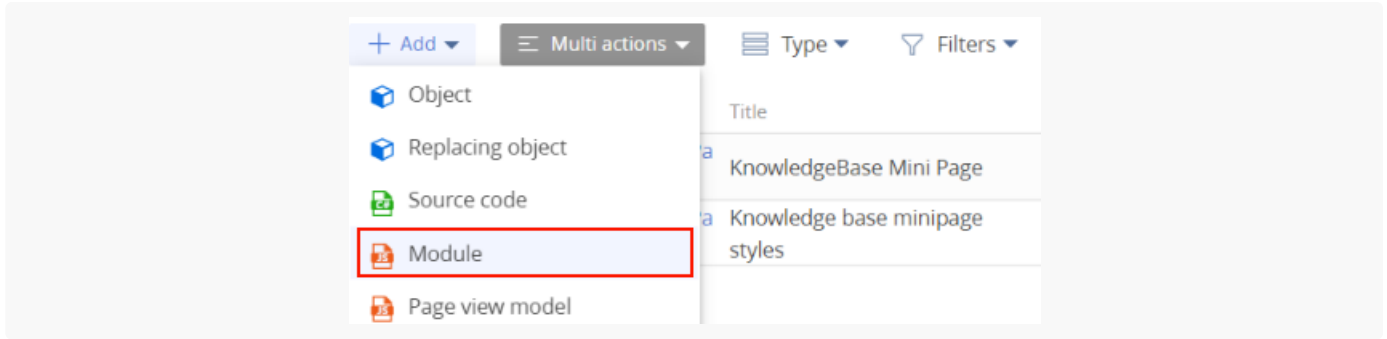
Добавить мини-карточку к произвольному модулю

 Средний

Пример. Отобразить текущего пользователя в правом верхнем углу приложения возле иконки с профилем пользователя. При наведении на ссылку текущего пользователя системы открыть мини-карточку.

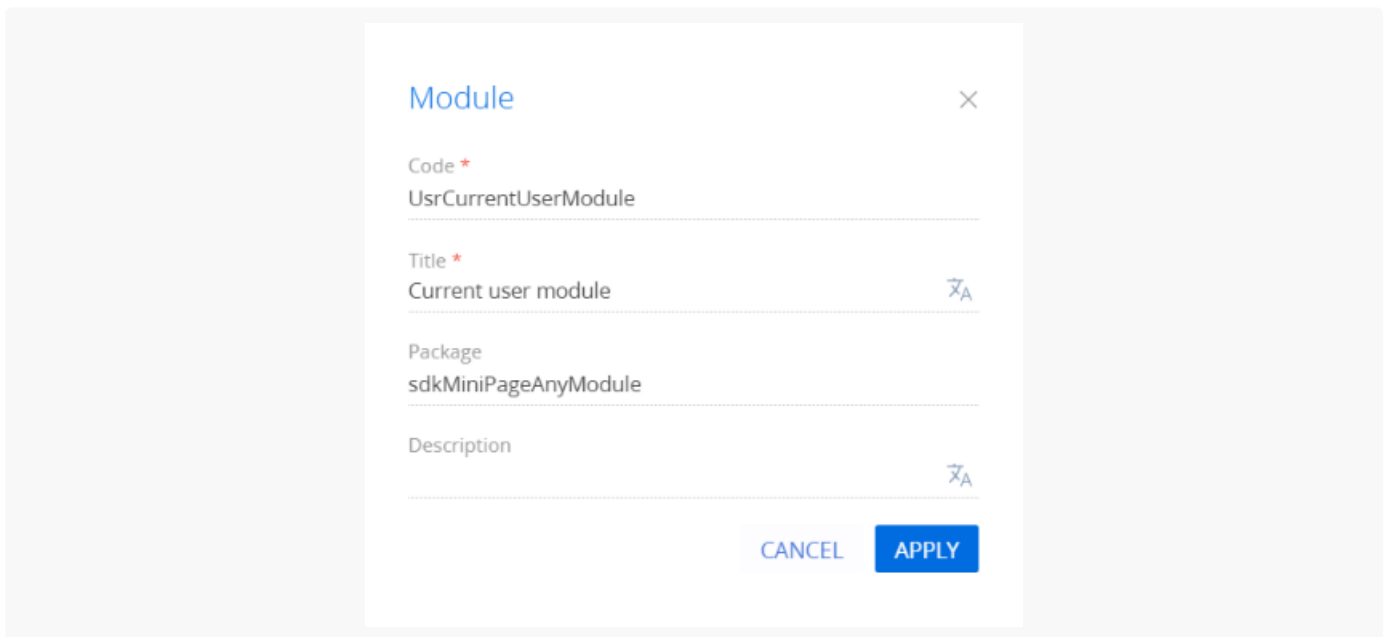
1. Создать схему модуля

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



3. В дизайнере схем заполните свойства схемы:

- [Код] ([Code]) — "UsrCurrentUserModule".
- [Заголовок] ([Title]) — "Модуль "Текущий пользователь" ("Current user module").



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

2. Создать представление и модель представления модуля

В дизайнере схем добавьте в модуль `UsrCurrentUserModule` необходимый исходный код.

1. Для создания модели представления реализуйте класс, унаследованный от `Terrasoft.BaseViewModel`.
2. Подключите утилитный класс `Terrasoft.MinipageUtilities` в свойство `mixins` модели представления. Класс позволит использовать методы вызова мини-карточки.
3. Для создания представления реализуйте класс, унаследованный от `Terrasoft.BaseModule`.
4. В классе переопределите методы базового класса `Terrasoft.BaseModule`:
 - `init()` — инициализирует модель представления модуля.

- `render()` — связывает модель представления с отображением представления в контейнере, передаваемом в параметре `renderTo`.
- `getViewModel()` — используется для создания модели представления.
- `getView()` — используется для получения представления для его дальнейшего отображения. Представление должно отображать ФИО текущего пользователя с гиперссылкой на страницу контакта. При построении гиперссылки определите обработчик события наведения курсора мыши.

5. Определите свойство `viewModel` — используется для хранения ссылки на полученную модель представления.

Исходный код модуля приведен ниже.

UsrCurrentUserModule.js

```
// Определение модуля.
define("UsrCurrentUserModule", ["MiniPageUtilities"], function() {
    // Определение класса CurrentUserViewModel.
    Ext.define("Terrasoft.configuration.CurrentUserViewModel", {
        // Имя родительского класса.
        extend: "Terrasoft.BaseViewModel",
        // Сокращенное название класса.
        alternateClassName: "Terrasoft.CurrentUserViewModel",
        // Используемые миксины.
        mixins: {
            MiniPageUtilitiesMixin: "Terrasoft.MiniPageUtilities"
        }
    });
    // Определение класса UsrCurrentUserModule.
    Ext.define("Terrasoft.configuration.UsrCurrentUserModule", {
        // Сокращенное название класса.
        alternateClassName: "Terrasoft.UsrCurrentUserModule",
        // Имя родительского класса.
        extend: "Terrasoft.BaseModule",
        // Объект Ext.
        Ext: null,
        // Объект sandbox.
        sandbox: null,
        // Объект Terrasoft.
        Terrasoft: null,
        // Модель представления.
        viewModel: null,
        // Создает представления модуля.
        getView: function() {
            // Получение контакта текущего пользователя.
            var currentUser = Terrasoft.SysValue.CURRENT_USER_CONTACT;
            // Представление – экземпляр класса Terrasoft.Hyperlink.
            return Ext.create("Terrasoft.Hyperlink", {
```

```

        // Заполнение заголовка ссылки именем контакта.
        "caption": currentUser.displayName,
        // Обработчик события наведения на ссылку.
        "linkMouseOver": {"bindTo": "linkMouseOver"},
        // Свойство, содержащее дополнительные параметры объекта.
        "tag": {
            // Идентификатор текущего пользователя.
            "recordId": currentUser.value,
            // Название схемы объекта.
            "referenceSchemaName": "Contact"
        }
    });
},
// Создает модель представления модуля.
getViewModel: function() {
    return Ext.create("Terrasoft.CurrentUserViewModel");
},
// Инициализация модуля.
init: function() {
    this.viewModel = this.getViewModel();
},
// Отображает представление модуля.
render: function(renderTo) {
    // Получение объекта представления.
    var view = this.getView();
    // Связывание представления с моделью представления.
    view.bind(this.viewModel);
    // Отображение представления в элементе renderTo.
    view.render(renderTo);
}
});
return Terrasoft.UsrCurrentUserModule;
});

```

3. Добавить стили модуля

Для настройки отображения гиперссылки добавьте стили для созданного модуля.

Чтобы **добавить стили модуля**:

1. В дизайнера схем выберите узел [*LESS*].
2. Добавьте следующий исходный код.

Стили модуля

```

.current-user-class a {
    font-weight: bold;
}

```



```
font-size: 2.0em;
margin: 6px 20px;
}

.current-user-class a:hover {
    text-decoration: none;
}
```

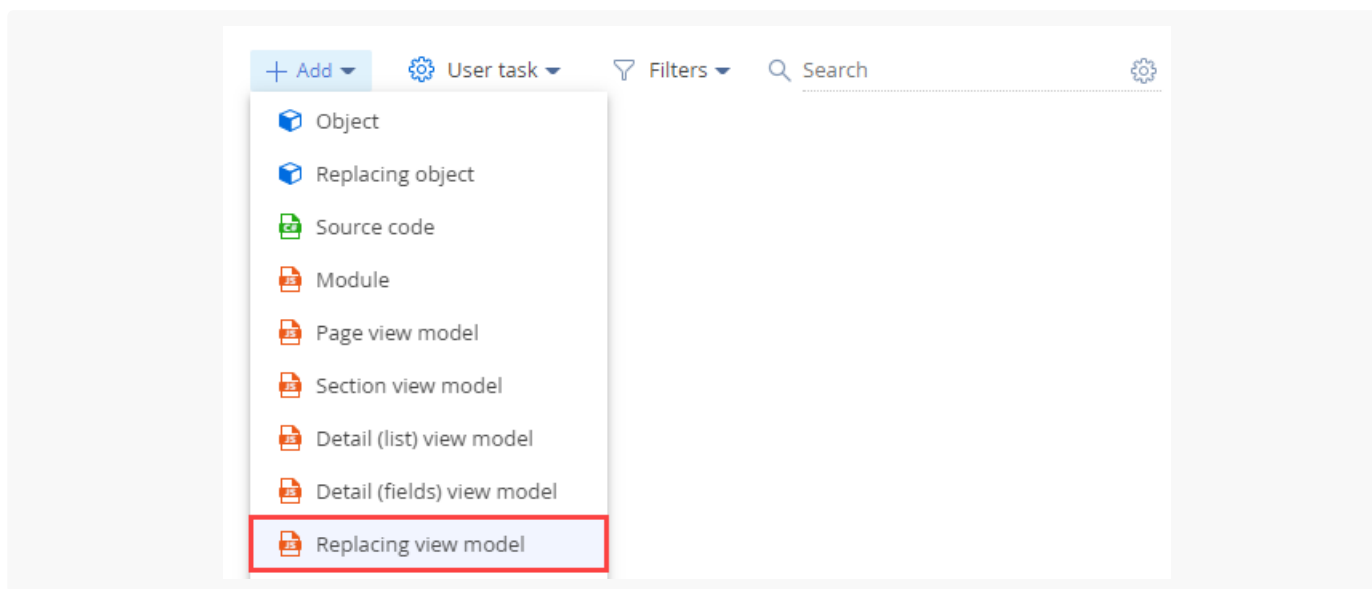
Сохраните созданный модуль.

4. Создать контейнер отображения представления

Для вывода ссылки на профиль пользователя в правом верхнем углу приложения необходимо разместить контейнер и загрузить в него представление созданного модуля.

Для этого создайте схему замещающей модели представления, которая расширит функциональность схемы `MainHeaderSchema`.

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



3. В дизайнера модуля выберите родительский объект `MainHeaderSchema`.

После подтверждения выбранного родительского объекта остальные свойства будут заполнены автоматически.

Module [X]

Code
MainHeaderSchema

Title *
MainHeaderSchema [X]

Parent object
MainHeaderSchema [v]

Package
sdkMiniPageAnyModule

Description [X]

CANCEL APPLY

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

4. В дизайнере модуля добавьте исходный код.

Для отображения представления в исходном коде схемы замещающей модели представления используйте свойство `diff`. Чтобы контейнер отобразился в правом верхнем углу страницы, в качестве родительского элемента создаваемого контейнера установите элемент `RightHeaderContainer`. Далее переопределите метод `onRender()`, в котором выполните загрузку созданного модуля.

Ниже приведен исходный код схемы замещающей модели представления.

MainHeaderSchema.js

```
// Определение модуля.
define("MainHeaderSchema", [], function() {
    return {
        methods: {
            // Выполняет действия после отображения представления.
            onRender: function() {
                // Вызов родительского метода.
                this.callParent(arguments);
                // Загрузка модуля текущего пользователя.
                this.loadCurrentUserModule();
            },
            // Загружает модуль текущего пользователя.
            loadCurrentUserModule: function() {
                // Получение контейнера, в который будет загружен модуль.
                var currentUserContainer = this.Ext.getCmp("current-user-container");
                // Проверка существования контейнера.
                if (currentUserContainer && currentUserContainer.rendered) {
```

```

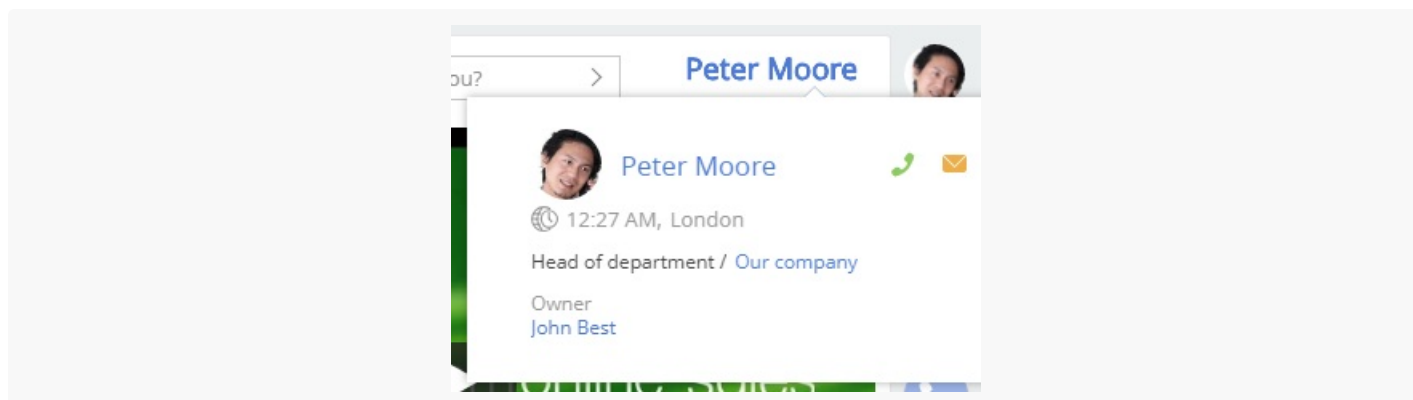
        // Загрузка модуля в контейнер.
        this.sandbox.loadModule("UsrCurrentUserModule", {
            // Название контейнера.
            renderTo: "current-user-container"
        });
    }
}
},
diff: [
    {
        // Операция вставки элемента.
        "operation": "insert",
        // Название элемента.
        "name": "CurrentUserContainer",
        // Название родительского контейнера.
        "parentName": "RightHeaderContainer",
        // Название свойства.
        "propertyName": "items",
        // Значения элемента.
        "values": {
            // Идентификатор контейнера.
            "id": "current-user-container",
            // Тип элемента.
            "itemType": Terrasoft.ViewItemType.CONTAINER,
            // Классы контейнера.
            "wrapClass": ["current-user-class"],
            // Элементы контейнера.
            "items": []
        }
    }
]
};
});

```

5. На панели инструментов дизайнера модуля нажмите [*Сохранить*] ([*Save*]).

Результат выполнения примера

После обновления страницы приложения в правом верхнем углу отобразится ФИО текущего пользователя с гиперссылкой на страницу его контакта. При наведении курсора на гиперссылку появится мини-карточка с данными о текущем пользователе.

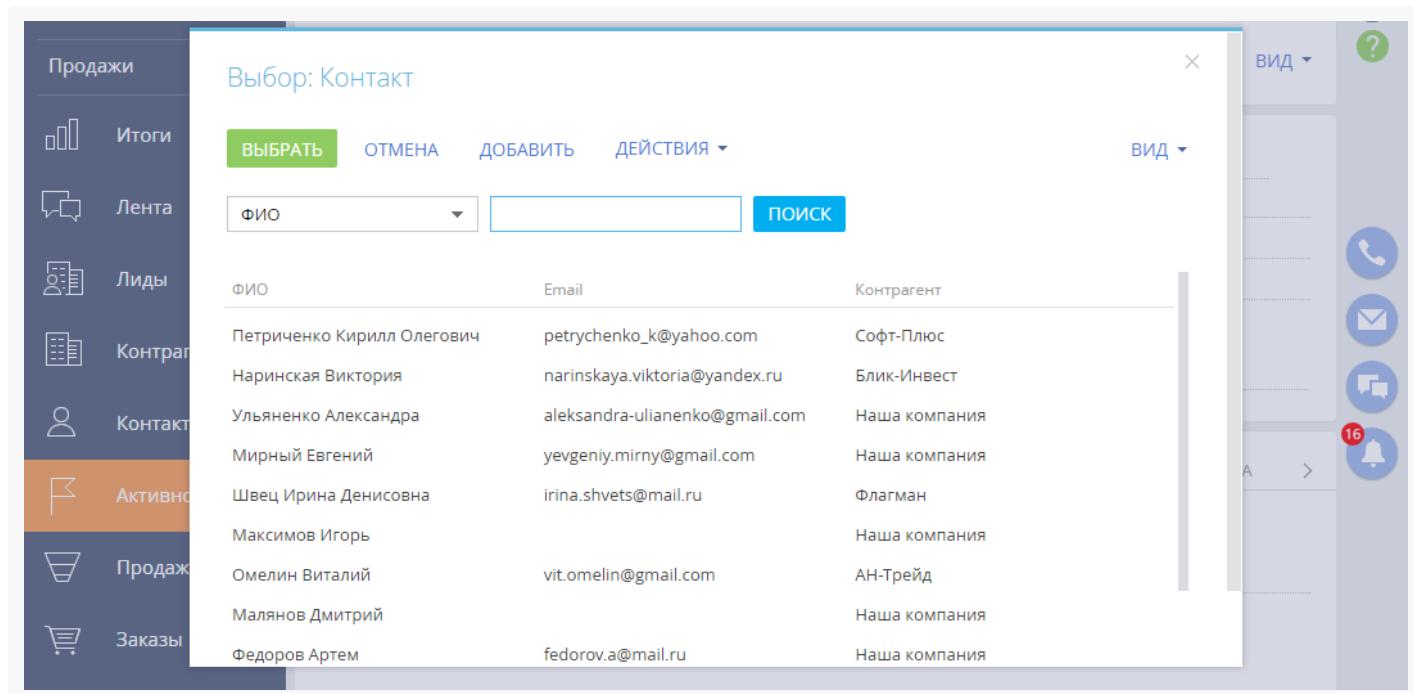


Модальное окно

Основы

Модальное окно предназначено для отображения данных во всплывающем диалоговом окне. При этом переход на новую страницу не происходит, а страница, из которой было вызвано модальное окно, остается открытой. Таким образом страница, которую отображает модальное окно, не попадает в историю переходов по страницам браузера.

Модальные окна предоставляют возможность отображения и выбора данных из разного рода справочников, например, выбор ответственного за активность из справочника контактов.



Основные свойства и поведение модального окна заданы в модулях `ModalBox` и `ModalBoxSchemaModule` пакета `nufl`. А непосредственно вызов модального окна для выбора данных из справочника выполняется в модуле `LookupUtilitiesV2`.

Коммуникационная панель

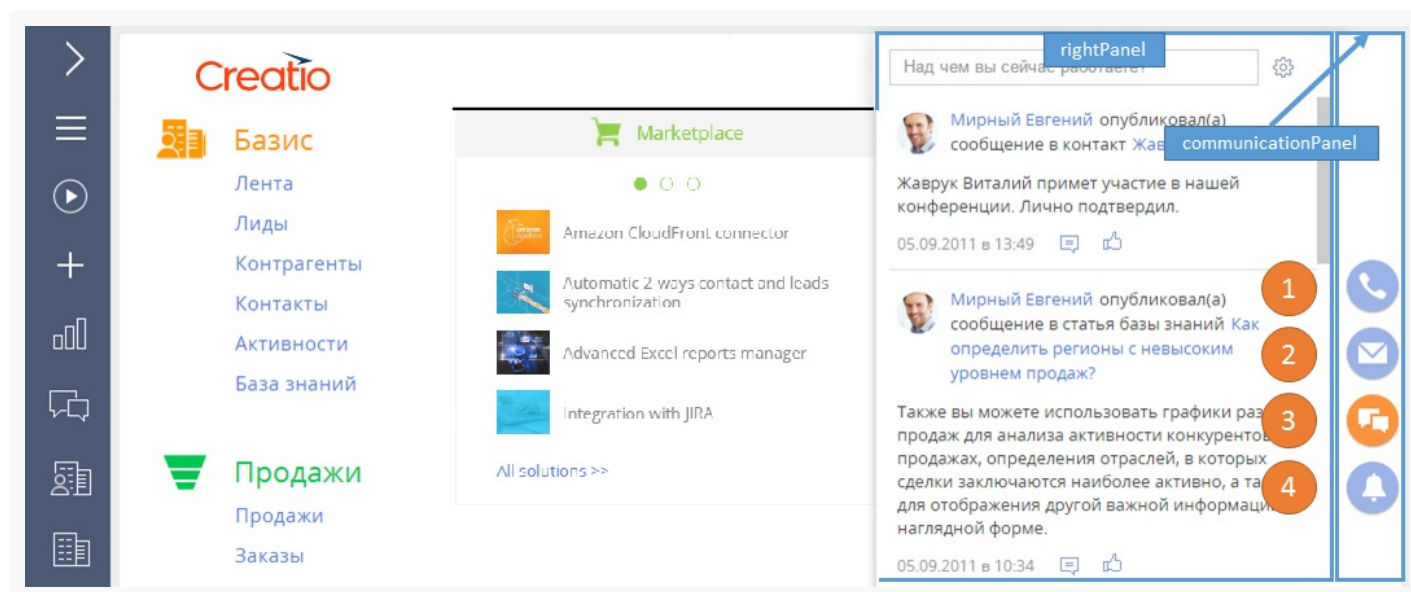
Основы

Коммуникационная панель — инструмент ведения коммуникаций с клиентами и коллегами, а также получения уведомлений в Creatio, расположенный в левой части экрана. С помощью коммуникационной панели можно совершать звонки, обрабатывать почту, общаться в ленте и чатах, выполнять задачи бизнес-процессов, не отрываясь от выполнения текущих задач.

Коммуникационная панель содержит следующие вкладки:

- **Консультационная панель.** Вкладка доступна только в продукте Financial Services Creatio, customer journey edition и предназначена для проведения консультаций клиенту банка менеджером. Вы можете воспользоваться различными критериями для поиска клиента банка, начать проведение консультации, отложить консультацию.
- **СТІ-панель.** Один из инструментов [телефонии в Creatio](#). Вы можете принимать входящие и выполнять исходящие звонки непосредственно в системе.
- **Email.** Вкладка предназначена для [работы с почтой](#). Вы можете отправлять и получать email-сообщения, а также связывать их с другими объектами системы.
- **Лента.** Вкладка отображает сообщения раздела [*Лента*]. Используйте вкладку для просмотра сообщений тех каналов, на которые вы подписаны, а также для добавления новых сообщений и комментариев. Функциональность вкладки аналогична функциональности раздела [*Лента*].
- **Центр уведомлений.** Вкладка [отображает уведомления](#) о различных событиях в системе.
- **Задачи по бизнес-процессам.** Вкладка отображает невыполненные шаги [по запущенным бизнес-процессам](#).

Чтобы открыть нужную вкладку, нажмите соответствующую кнопку на коммуникационной панели. Чтобы закрыть коммуникационную панель, повторно нажмите кнопку вкладки коммуникационной панели, на которой вы находитесь.



Конфигурирование коммуникационной панели осуществляется в схеме коммуникационной панели `CommunicationPanel` пакета `Uiv2`. Вкладка [*Звонки*] конфигурируется в унаследованной схеме

`CommunicationPanel` пакета `CTIBase`. Кнопки перехода во вкладки коммуникационной панели (1...4) расположены в контейнере `communicationPanel`, а непосредственно вкладки — в контейнере `rightPanel`.

Добавить всплывающую кнопку для сообщений из внутренней ленты обращений



В версии 7.12.0 добавлена возможность быстро создать новое обращение из переписки по существующему обращению. Для создания обращения выделите часть текста письма в истории и

нажмите кнопку . В новое обращение будут скопированы значения тех полей исходного обращения, для которых в дизайнера объекта установлен признак [*Выполнять копирование*]. В ответ на исходное письмо будет отправлено автоматическое уведомление о регистрации обращения. Создание обращения по фрагменту письма работает как для email-сообщений, так и для сообщений на портале самообслуживания.

Эта функциональность была реализована с помощью элемента управления `SelectionHandlerMultiLineLabel`, размещенного в пакете `Message`.

Для обработки сообщений вкладки [*Обработка*], созданных во внутренней ленте:

1. Создайте схему замещающей модели представления `SocialMessageHistoryItemPage`.
2. Реализуйте логику обработки выделенного текста с помощью событий `selectedTextChanged` и `selectedTextHandlerButtonClick` элемента управления `SelectionHandlerMultiLineLabel`.
3. Добавьте в массив `diff` конфигурационный объект с настройками элемента `SelectionHandlerMultiLineLabel`.

Описание примера

На вкладке [*Обработка*] страницы раздела [*Обращения*] добавить функциональность всплывающей кнопки при выделении текста в сообщениях, созданных во внутренней ленте обращения. После нажатия на всплывающую кнопку должно быть создано новое обращение с автоматически заполненными полями.

Исходный код

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

Важно.

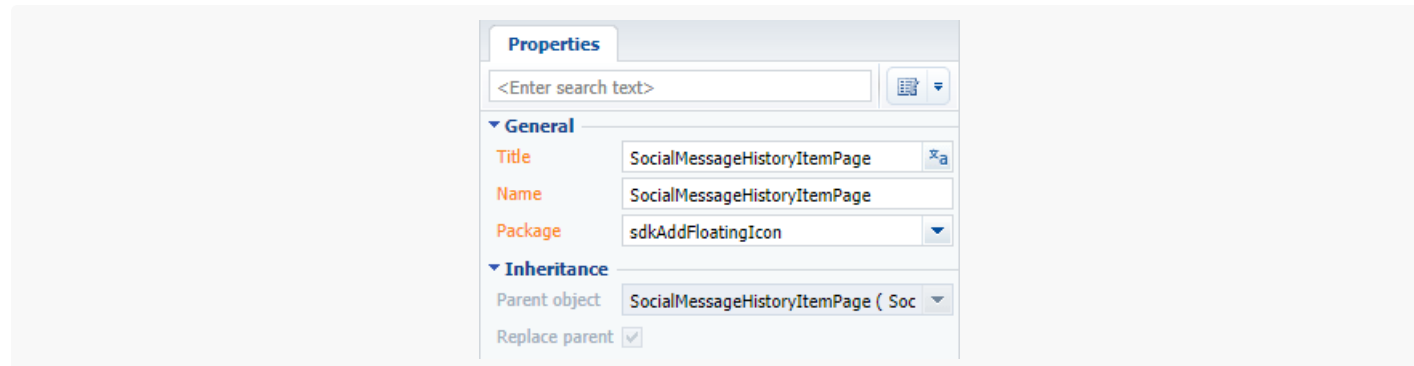
Пакет можно установить только для продуктов линейки `Service` или на линейки продуктов, содержащие пакеты `Message` и `SocialMessage`.

Алгоритм реализации примера

1. Создать схему замещающей модели представления SocialMessageHistoryItemPage

Создайте схему замещающей модели представления, в которой в качестве родительского объекта укажите `SocialMessageHistoryItemPage` (рис. 1). Процесс создания схему замещающей модели представления описан в статье "[Создать клиентскую схему](#)".

Рис. 1. — Свойства схему замещающей модели представления



2. Реализовать логику обработки выделенного текста

В коллекцию методов созданной схемы добавьте следующие методы (исходный код приведен ниже):

- `getMessageFromHistory()` — переопределенный базовый метод. Получает тему для выделенного сообщения.
- `onSelectedTextChanged()` — устанавливает значение выделенного текста в атрибут `HighlightedHistoryMessage`. Срабатывает при выделении текста.
- `onSelectedTextButtonClick()` — создает обращение, тему которого получает из установленного ранее атрибута `HighlightedHistoryMessage`. Логика создания обращения определена в родительской схеме `BaseMessageHistory`. Срабатывает при нажатии на всплывающую кнопку.

3. Настроить элемент SelectionHandlerMultiLineLabel

В массив `diff` созданной схемы добавьте конфигурационный объект с настройками элемента `SelectionHandlerMultiLineLabel`. Исходный код замещающей схемы:

```
define("SocialMessageHistoryItemPage", ["SocialMessageConstants", "css!SocialMessageHistoryItems",
function(socialMessageConstants) {
    return {
        // Название схемы объекта страницы записи.
        entitySchemaName: "BaseMessageHistory",
        details: /**SCHEMA_DETAILS*/{}/**SCHEMA_DETAILS*/,
        // Методы модели представления страницы записи.
        methods: {
```

```

// Переопределенный базовый метод. Получает тему для выделенного сообщения.
getMessageFromHistory: function() {
    var message = this.get("HighlightedHistoryMessage");
    if (this.isHistoryMessageEmpty(message)) {
        message = this.get("[Activity:Id:RecordId].Body");
    }
    return message;
},
// Обработчик события выделения текста.
onSelectedTextChanged: function(text) {
    this.set("HighlightedHistoryMessage", text);
},
// Обработчик нажатия всплывающей кнопки.
onSelectedTextButtonClick: function() {
    // Подготовка данных по обращению из истории.
    this.prepareCaseDataFromHistory();
}
},
diff: /**SCHEMA_DIFF*/[
{
    // Изменить существующий компонент "MessageText".
    "operation": "merge",
    // Название компонента.
    "name": "MessageText",
    // Свойства объекта.
    "values": {
        // Свойства генератора представления.
        "generator": function() {
            return {
                // Значение HTML-тега id.
                "id": "MessageText",
                // Значение маркера.
                "markerValue": "MessageText",
                // Название класса компонента.
                "className": "Terrasoft.SelectionHandlerMultilineLabel",
                // Настройка CSS-стилей.
                "classes": {
                    "multilineLabelClass": ["messageText"]
                },
                // Заголовок.
                "caption": {
                    "bindTo": "Message"
                },
                "showLinks": true,
                // Привязка события изменения выделенного текста к методу-обработчику.
                "selectedTextChanged": {"bindTo": "onSelectedTextChanged"},
                // Привязка события нажатия всплывающей кнопки выделенного текста к методу-обработчику.
                "selectedTextHandlerButtonClick": {"bindTo": "onSelectedTextButt

```



```

// Признак видимости всплывающей кнопки.
"showFloatButton": true
};
}
}
}
]/**SCHEMA_DIFF*/
};
}
);

```

После сохранения схемы и обновления страницы приложения на вкладке [*Обработка*] страницы обращения при выделении текста в сообщениях, созданных во внутренней ленте обращения, будет отображаться всплывающая кнопка (рис. 2). После клика по всплывающей кнопке будет создано новое обращение с автоматически заполненными полями (рис. 3).

Рис. 2. — Всплывающая при выделении текста кнопка

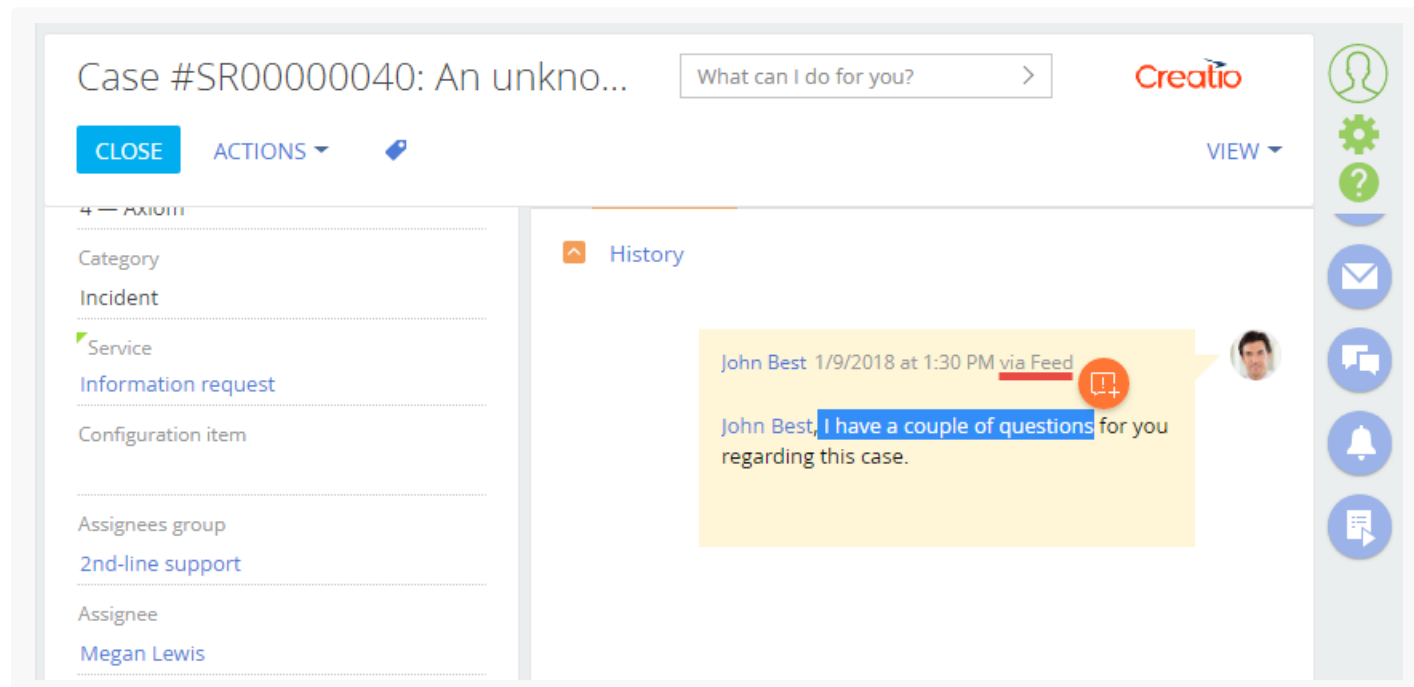
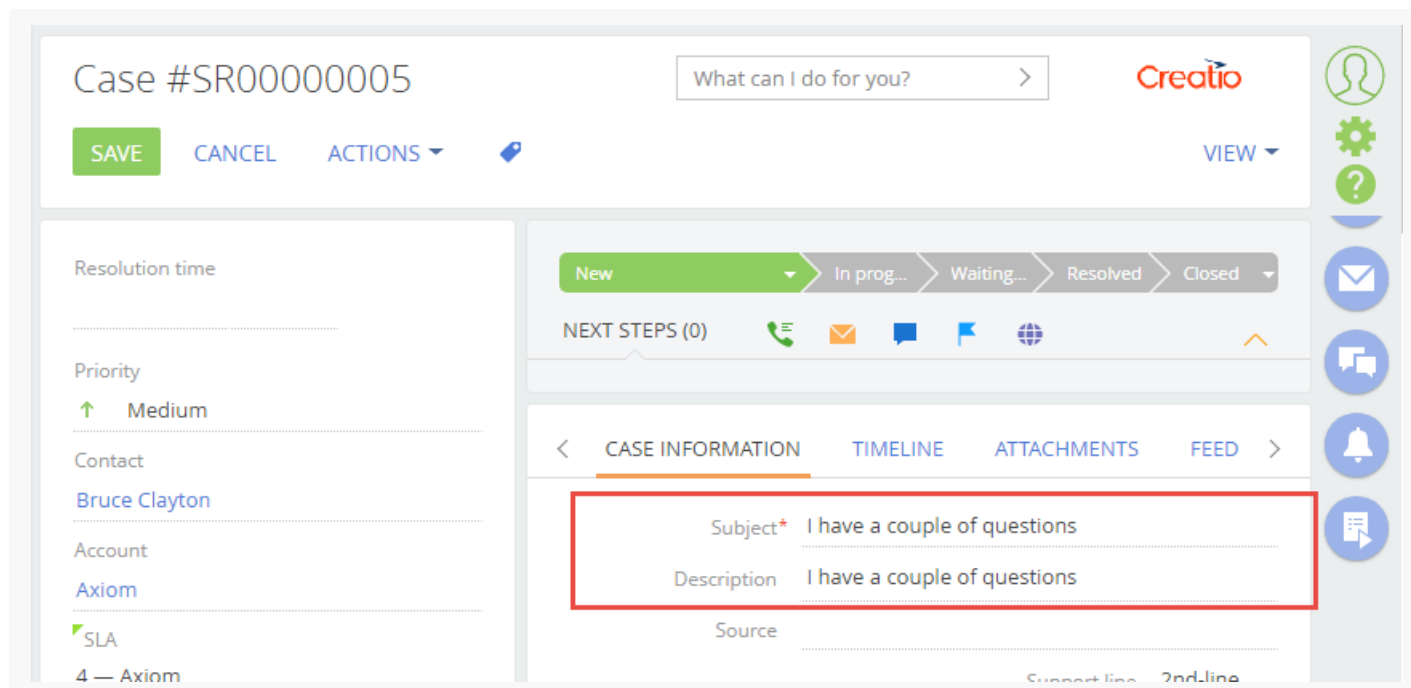


Рис. 3. — Новое обращение

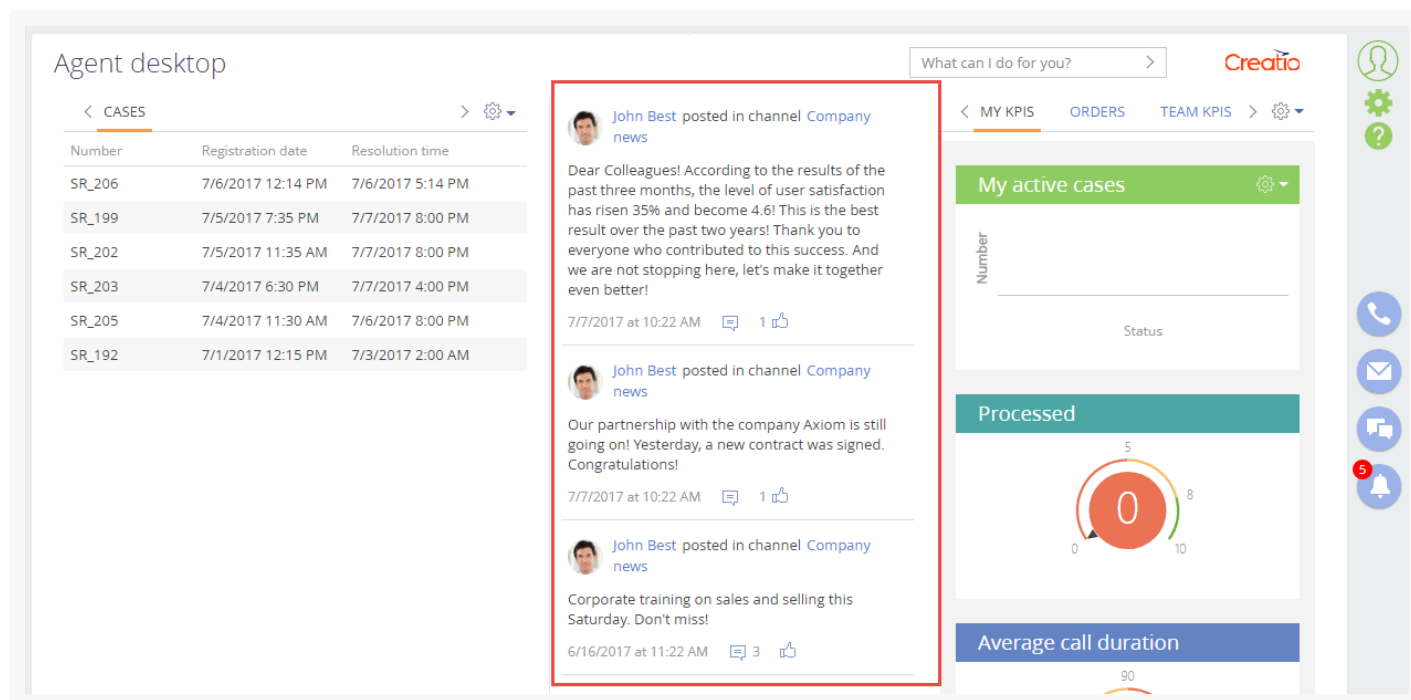


Скрыть область ленты в едином окне

Средний

Единое окно — инструмент контакт-центра, позволяющий принимать и обрабатывать обращения в омниканальном режиме, управлять входящими и исходящими звонками, а также работать с другими преднастроенными элементами очередей. Здесь отображаются все доступные для обработки обращения, которые маршрутизированы на сотрудника по заданной логике настройки очередей (рис. 1).

Рис. 1. — Область ленты единого окна



Чтобы скрыть область ленты необходимо:

1. В пользовательском пакете создать схему замещающей модели представления [*Страница единого окна оператора*] ([*Agent desktop page*]). Как создать схему замещающей модели представления описано в статье "[Создать клиентскую схему](#)".
2. В созданную схему добавить следующий исходный код:

```
define("OperatorSingleWindowPage", [],
function() {
return {
methods: {
// Замещение базового метода для исключения из состава загружаемых модулей модуль
loadContent: function() {
// Т.к. контейнер centerContainer удален, то модуль ESNFeedModule загружать
//this.loadModule("ESNFeedModule", "centerContainer");
this.loadModule("SectionDashboardsModule", "rightContainer");
this.loadModule("OperatorQueuesModule", "leftContainer");
}
},
diff: /**SCHEMA_DIFF*/[
{
"operation": "remove",
"name": "centerContainer"
}
]**SCHEMA_DIFF*/
];
});
```

3. Сохранить изменения.

4. Обновить страницу браузера.

В результате область ленты в едином окне будет скрыта (рис. 2).

Рис. 2. — Раздел [*Единое окно*] ([*Agent desktop*]) без области ленты

The screenshot shows the 'Agent desktop' interface. At the top, there's a search bar with the text 'What can I do for you?' and the 'Creatio' logo. Below the search bar, there's a navigation bar with tabs: 'CASES', 'MY KPIS', 'ORDERS', and 'TEAM KPIS'. The 'CASES' tab is active, displaying a table with columns: 'Number', 'Registration date', and 'Resolution time'.

Number	Registration date	Resolution time
SR_206	7/6/2017 12:14 PM	7/6/2017 5:14 PM
SR_199	7/5/2017 7:35 PM	7/7/2017 8:00 PM
SR_202	7/5/2017 11:35 AM	7/7/2017 8:00 PM
SR_203	7/4/2017 6:30 PM	7/7/2017 4:00 PM
SR_205	7/4/2017 11:30 AM	7/6/2017 8:00 PM
SR_192	7/1/2017 12:15 PM	7/3/2017 2:00 AM

On the right side, there's a 'My active cases' section with a search bar and a 'Status' dropdown. Below it, there's a 'Processed' section with a circular gauge showing a value of 0. At the bottom, there's an 'Average call duration' section showing a value of 90. On the far right, there's a vertical toolbar with icons for phone, mail, chat, and notifications.

Командная строка

Основы

Командная строка предназначена для быстрого доступа к наиболее часто выполняемым операциям, таким как поиск клиента, открытие страницы записи или запуск бизнес-процесса.

Для выполнения команды необходимо ввести ее в командную строку и нажать на [*Выполнить команду*] либо клавишу [*Enter*] клавиатуры. При вводе неполной команды система предложит перечень похожих команд в выпадающем списке.

The screenshot shows the 'Creatio' interface. At the top, there's a search bar with the text 'Найти Лид'. Below the search bar, there's a dropdown menu with the following commands: 'Найти Лид', 'Найти Счет', 'Найти Активность', 'Найти Статья базы знаний', 'Найти Проект', 'Найти Продукт', 'Найти Продажа', 'Найти Договор', 'Найти Контрагент', 'Найти Контакт', and 'Найти Заказ'. The 'Выполнить команду' button is visible next to the search bar. On the left side, there's a sidebar with icons for 'ДОБАВИТЬ КОНТАКТ' and 'ДЕЙСТВИЯ'. The main area displays a list of contacts with their names, positions, and phone numbers. On the right side, there's a vertical toolbar with icons for phone, mail, chat, and notifications.

К возможностям [командной строки](#) относятся:

- Навигация — для перехода в любую доступную группу любого раздела (команда [*Перейти ...*]).
- Поиск записей — для быстрого поиска, например, контакта, контрагента или записей текущего раздела (команда [*Поиск ...*]).
- Создание записей — для создания записей различных разделов (команда [*Создать ...*]).
- Запуск бизнес-процессов — для выполнения настроенных в системе бизнес-процессов (команда [*Запустить процесс ...*]).
- Создание пользовательских команд — для добавления новых команд, распознаваемых командной строкой (команда [*Создать пользовательскую команду*]).

Поле ввода командной строки находится в контейнере `mainHeaderContainer`.

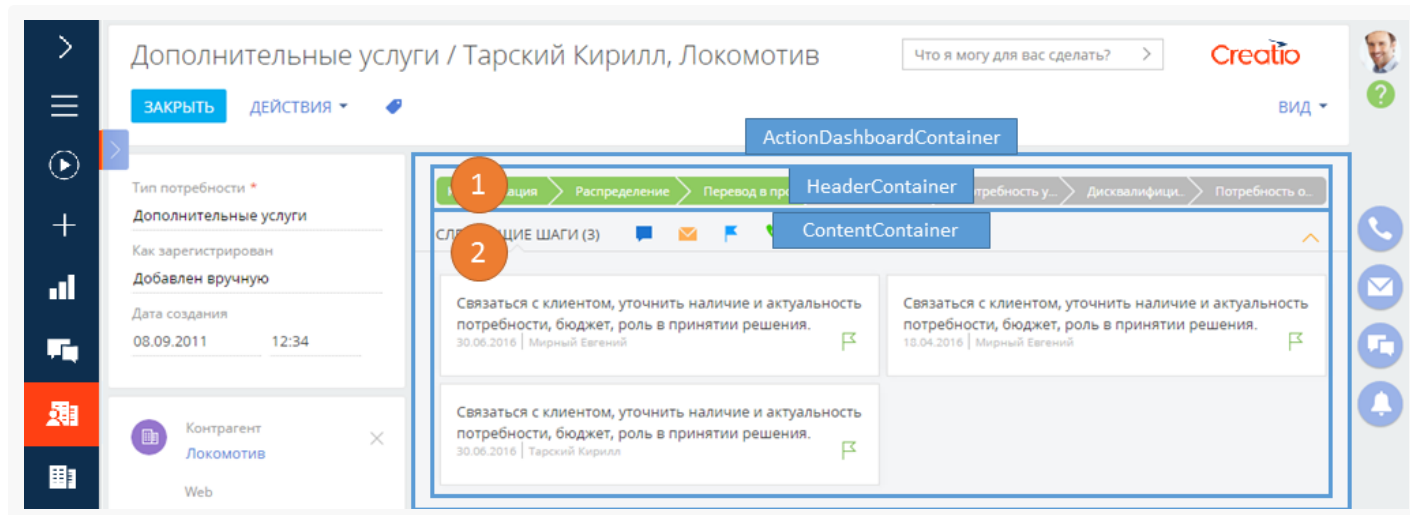
Для отслеживания ввода пользователем команд и их выполнения в системе запущена служба `CommandLineService`. Для хранения команд в системе используется таблица базы данных, структура которой описывается схемой объекта `Command`. Параметры команд описываются при помощи схемы объекта `CommandParams`. Для отображения перечня доступных команд при их частичном вводе, автодополнении и другой функциональности командной строки используются функции модуля `CommandLineModule`.

Инструментальная панель действий

Основы

Инструментальная панель действий предназначена для отображения информации о текущем состоянии работы с записью. Она состоит из двух частей:

- **Полоса стадий** (1) — показывает состояние этапов бизнес-процесса в тех разделах, где работа с записями ведется по бизнес-процессу,
- **Панель действий** (2):
 - Позволяет перейти к выполнению активности, работе с email-сообщениями или с лентой, не покидая раздел.
 - Отображает созданные по бизнес-процессу активности, которые находятся в неконечном состоянии и связаны с объектом раздела по соответствующему полю.
 - Может отображать автогенерируемую страницу, преднастроенную страницу, вопрос и страницу объекта в виде задач.



Инструментальная панель действий расположена в контейнере `ActionDashboardContainer` страницы записи раздела. Полоса стадий расположена во вложенном контейнере `HeaderContainer`, а панель действий — в `ContentContainer`.

Расположение элементов инструментальной панели действий конфигурируется схемой модели представления `BaseActionsDashboard` и унаследованной схемой `SectionActionsDashboard` пакета `ActionsDashboard`.

Добавить инструментальную панель действий на страницу

Чтобы **добавить инструментальную панель действий** на страницу:

1. Создайте схему модели представления, унаследованную от `SectionActionsDashboard`.
2. Создайте схему замещающей модели представления страницы.
3. В свойстве `modules` схемы замещающей модели представления страницы выполните настройку модуля.
4. В свойстве `diff` схемы замещающей модели представления страницы добавьте модуль на страницу.

Добавить новый канал на инструментальную панель действий

Каналы в инструментальной панели действий — это способ коммуникации с контактом. Канал создается для каждого раздела, в котором он подключен, например, для обращения, контакта или лида.

Чтобы **добавить новый канал** на инструментальную панель действий:

1. Создайте класс-наследник базового класса `BaseMessagePublisher`.
2. Создайте схему замещающей модели представления `SectionActionsDashboard`.
3. Установите в свойстве `diff` схемы замещающей модели представления операции вставки вкладки `CallsMessageTab` и контейнера сообщений, а также укажите модуль, который будет отрисовываться в данном канале на одной из вкладок.

4. Переопределите методы:

- `getSectionPublishers()` — добавляет созданный канал в список издателей сообщений.
- `getExtendedConfig()` — определяет параметры вкладки.

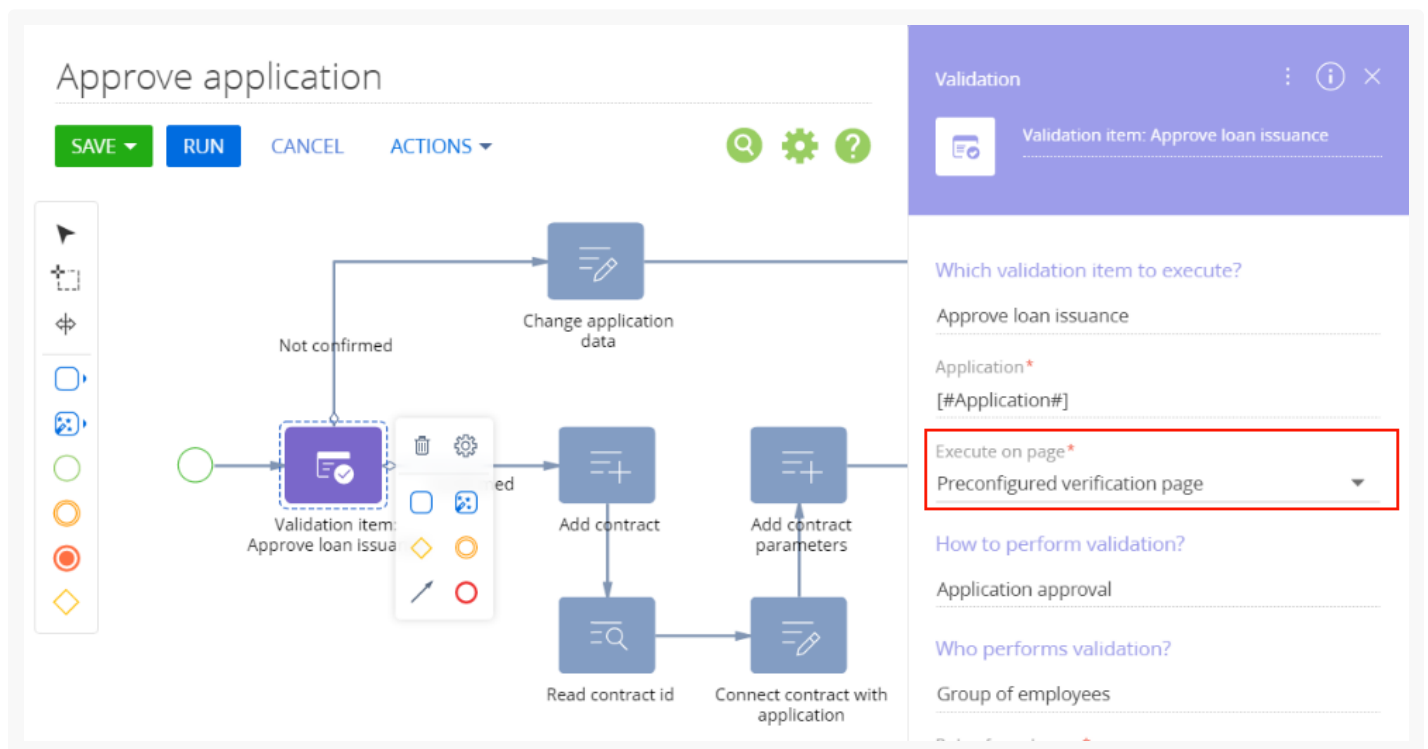
5. Создайте модуль-контейнер для прорисовки в инструментальной панели действий страницы, в которой будет реализована логика добавляемого канала.

6. Создайте схему модели представления, в которой будет реализована логика канала. В качестве родительской схемы установите `BaseMessagePublisherPage`.

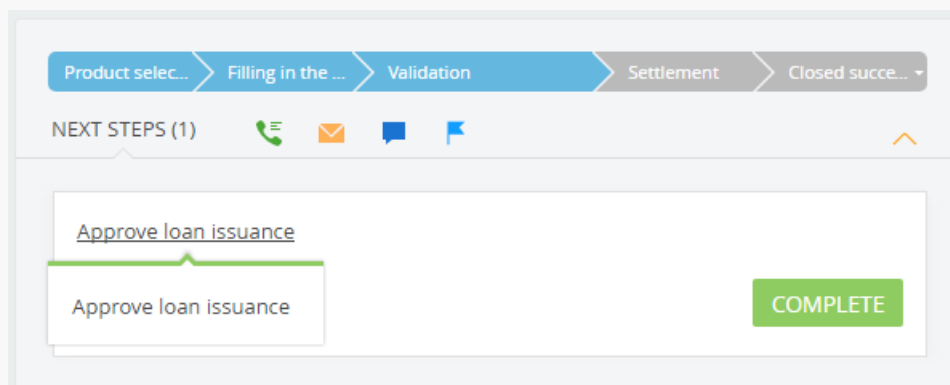
Добавить пользовательское действие верификации

Элемент процесса [*Действие верификации*] используется в ходе верификации заявки сотрудником компании. Элемент используется только в продукте Financial Services Creatio. С его помощью можно создать проверку данных в кредитной заявке — набор необходимых действий верификации, которые должен провести ответственный сотрудник. При помощи этого элемента можно реализовать процесс принятия решения по кредитной заявке. От результата выполнения действия верификации зависит дальнейшее ветвление бизнес-процесса.

При создании настраиваемой страницы действия верификации, например, в бизнес-процессе [*Подтверждение заявки*] ([*Approve application*]), есть возможность выбрать заранее созданную страницу с названием [*Преднастроенная страница верификации*] ([*Preconfigured verification page*]).



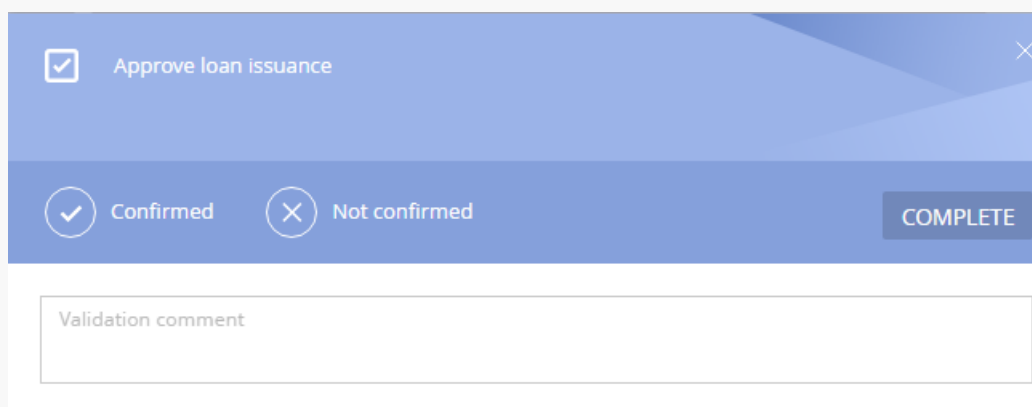
Сама страница отображается, например, после нажатия на кнопку [*Завершить*] ([*Complete*]) активности [*Согласовать выдачу кредита*] ([*Approve loan issuance*]), которая создается при переходе заявки на стадию [*Верификация*] ([*Validation*]).



Преднастроенная страница верификации содержит:

- Кнопки выбора результата выполнения действия верификации.
- Поле [Комментарий] — комментарий к действию верификации.
- Деталь [Сценарий разговора] — содержит скрипт-подсказку для верификатора. Доступна только в режиме чтения.
- Деталь [Файлы и ссылки] — содержит прикрепленные к действию верификации файлы и ссылки. Доступна только в режиме чтения.
- Деталь [Результаты проверок] — содержит контрольные вопросы и ответы на них.

Важно. Если в какой-либо из деталей нет прикрепленных данных, она не будет отображена для экономии пространства страницы.



Creatio предоставляет возможность создавать пользовательские страницы верификации, наследующие преднастроенную.

Чтобы создать **пользовательскую страницу верификации**:

1. Создайте клиентскую схему страницы действия верификации.
2. Используйте созданную схему в бизнес-процессе.

Добавить инструментальную панель

действий

 Средний

Описание примера

Добавить инструментальную панель действий на страницу заказа.

Исходный код

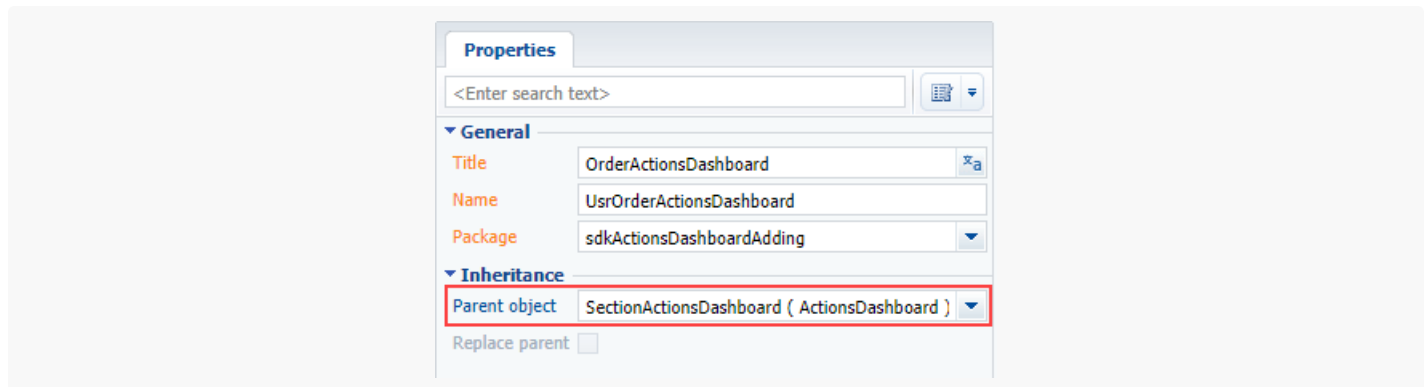
Пакет с реализацией примера можно скачать по [ссылке](#).

Алгоритм реализации примера

1. Создать схему модели представления OrderActionsDashboard

В качестве родительского объекта укажите схему `SectionActionsDashboard` (рис. 1).

Рис. 1. — Свойства схемы модели представления



Исходный код схему модели представления:

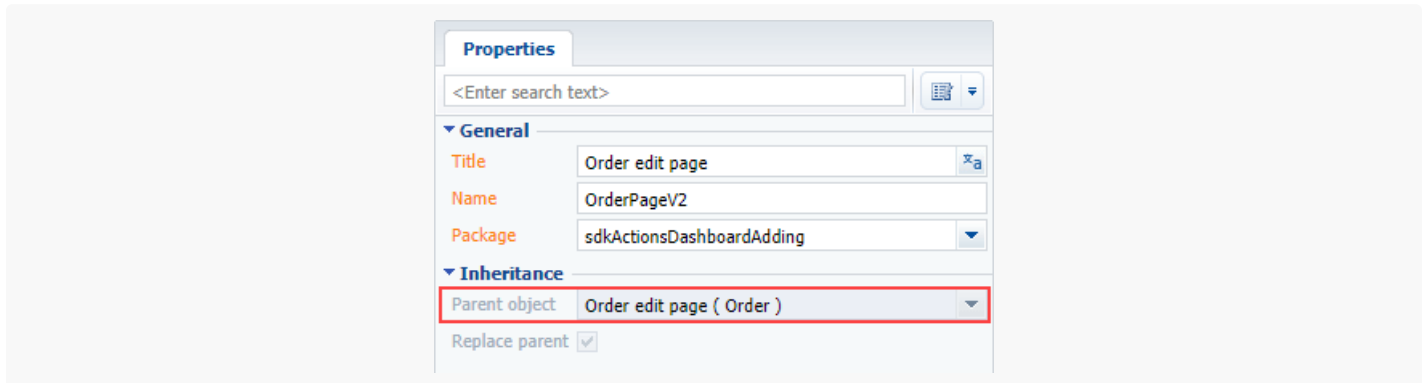
```
define("UsrOrderActionsDashboard", [], function () {
    return {
        details: /**SCHEMA_DETAILS*/{}/**SCHEMA_DETAILS*/,
        methods: {},
        diff: /**SCHEMA_DIFF*/[]/**SCHEMA_DIFF*/
    };
});
```

2. Создать замещающую страницу заказа

Создайте схему замещающей модели представления, в которой в качестве родительского объекта

укажите [*Страница редактирования заказа*] ([*Order edit page*], `OrderPageV2`) (рис. 2). Процесс создания схемы замещающей модели представления описан в статье "[Создать клиентскую схему](#)".

Рис. 2. — Свойства схему замещающей модели представления страницы записи



3. В коллекцию `modules` схемы страницы добавить конфигурационный объект с настройками модуля

На вкладку исходного кода добавьте код замещающего модуля страницы. В нем в коллекцию `modules` модели представления добавьте конфигурационный объект с настройками модуля.

4. В массив `diff` добавить конфигурационный объект с настройками расположения модуля на странице

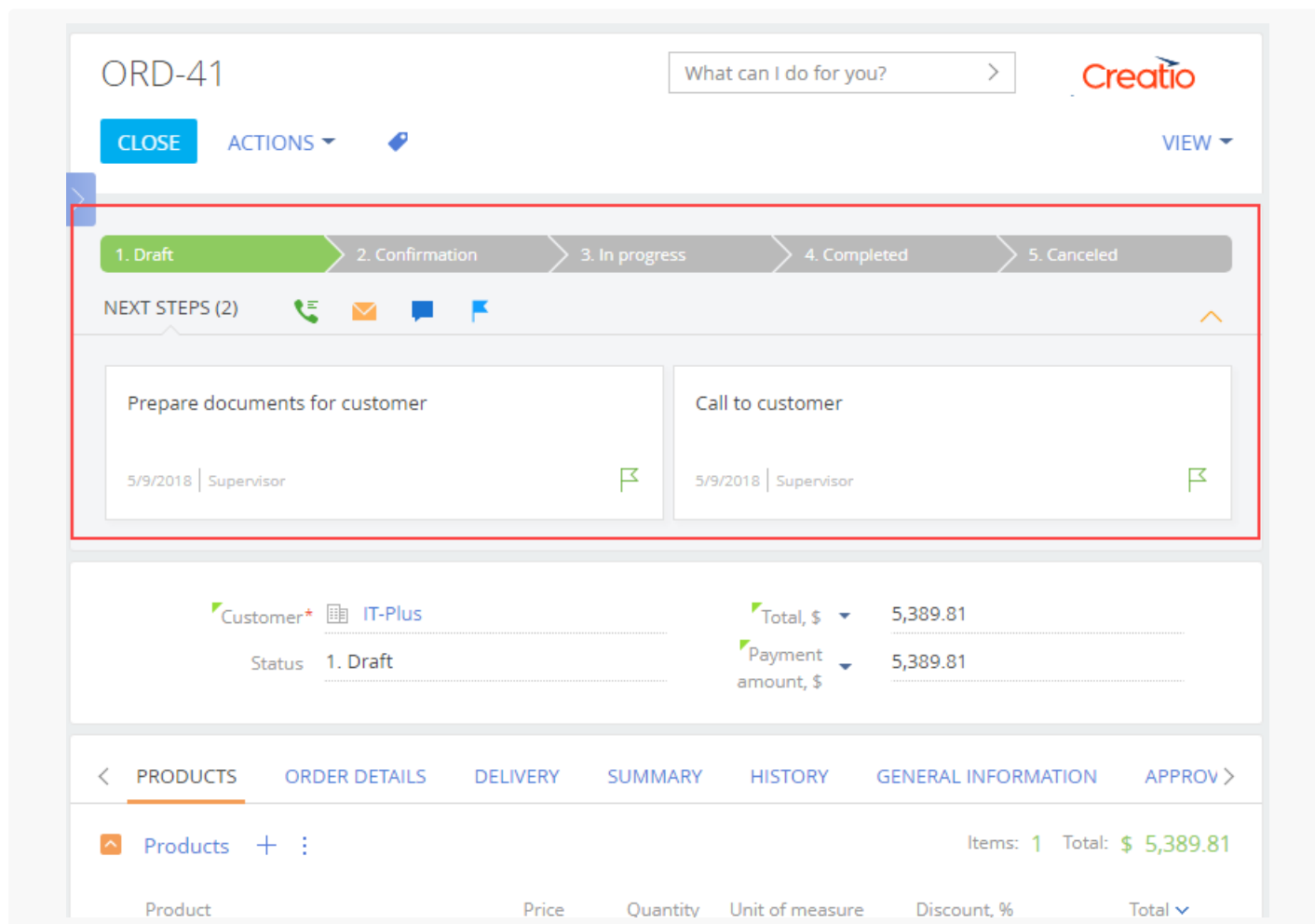
Исходный код схемы замещающей модели представления:

```
define("OrderPageV2", [],
function () {
return {
entitySchemaName: "Order",
attributes: {},
modules: /**SCHEMA_MODULES*/{
"ActionsDashboardModule": {
"config": {
"isSchemaConfigInitialized": true,
// Имя схемы.
"schemaName": "UsrOrderActionsDashboard",
"useHistoryState": false,
"parameters": {
// Конфигурационный объект модели представления.
"viewModelConfig": {
// Имя схемы сущности страницы.
"entitySchemaName": "Order",
// Конфигурационный объект блока Actions.
"actionsConfig": {
// Имя схемы для загрузки элементов в Actions.
```

```
"columnName": "OrderStatus",  
    // Имя колонки в родительской схеме, ссылающейся на схему, c  
    // Если не указана, берет значение равное schemaName.  
    "columnName": "Status",  
    // Имя колонки для сортировки элементов.  
    "orderColumnName": "Position",  
    // Имя колонки для сортировки элементов в меню элемента.  
    "innerOrderColumnName": "Position"  
},  
// Отвечает за отображение модуля панели действий, значение [true]  
"useDashboard": true,  
// Отвечает за отображение блока Content, значение [true] по умолчанию  
"contentVisible": true,  
// Отвечает за отображение блока Header, значение [true] по умолчанию  
"headerVisible": true,  
// Конфигурационный объект элементов панели.  
"dashboardConfig": {  
    // Связь активностей с объектом страницы.  
    "Activity": {  
        // Имя колонки объекта страницы.  
        "masterColumnName": "Id",  
        // Имя колонки в объекте [Activity].  
        "referenceColumnName": "Order"  
    }  
}  
  
} }  
  
} }  
  
} }  
  
} }  
  
}/**SCHEMA_MODULES*/,  
details: /**SCHEMA_DETAILS*/{}//**SCHEMA_DETAILS*/,  
methods: {},  
diff: /**SCHEMA_DIFF*/[  
    {  
        "operation": "insert",  
        "name": "ActionsDashboardModule",  
        "parentName": "ActionDashboardContainer",  
        "propertyName": "items",  
        "values": {  
            "classes": { wrapClassName: ["actions-dashboard-module"] },  
            "itemType": Terrasoft.ViewItemType.MODULE  
        }  
    }  
],  
};
```

После сохранения схемы и обновления страницы приложения на странице заказа появится инструментальная панель действий, которая будет отображать состояние заказа, а также связанные с ним незавершенные активности (рис. 3).

Рис. 3. — Демонстрация результата выполнения примера



Добавить новый канал на инструментальную панель действий

 Сложный

Описание примера

Добавить новый пользовательский канал в инструментальную панель действий страницы контакта. Канал должен полностью повторять функциональность канала фиксации результатов звонка (канал `CallMessagePublisher`).

Исходный код

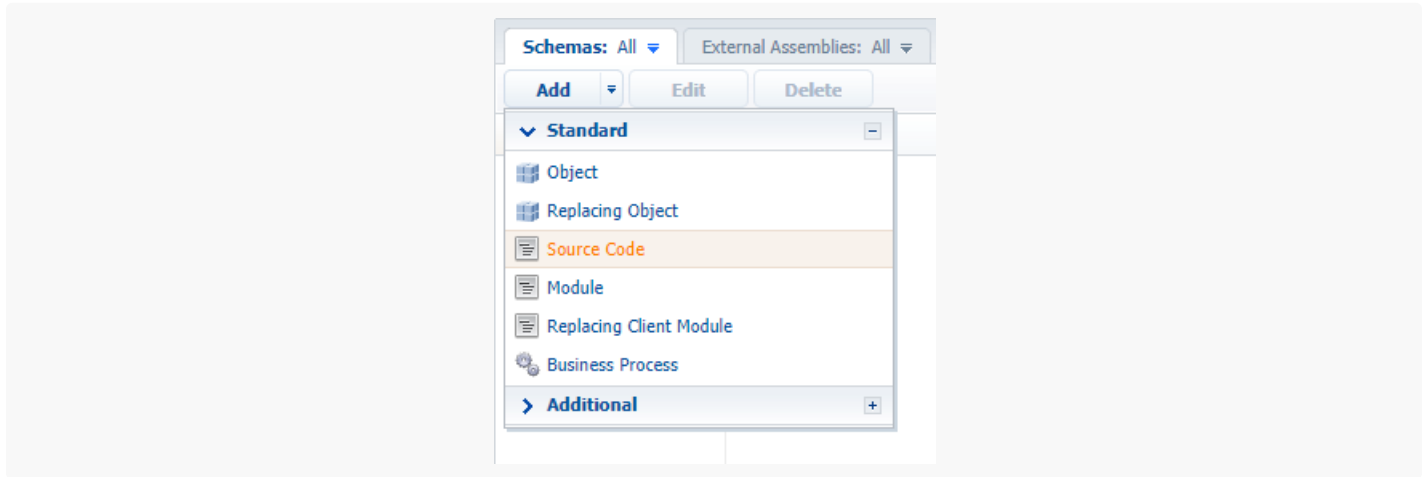
Пакет с реализацией примера можно скачать по [ссылке](#).

Алгоритм выполнения примера

1. Добавить схему исходного кода `UsrCallsMessagePublisher`

Для создания схемы исходного кода в разделе [*Конфигурация*] на вкладке [*Схемы*] выполните пункт меню [*Добавить*] — [*Исходный код*] (рис. 1).

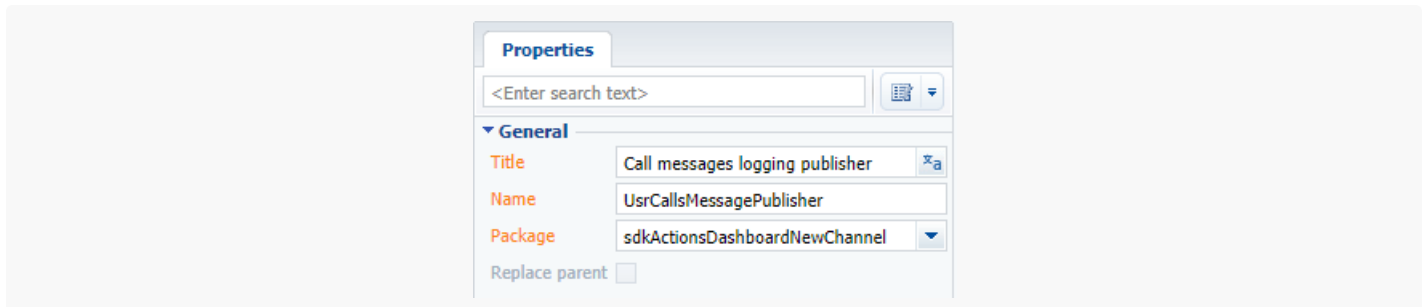
Рис. 1. — Добавление схемы исходного кода



Для созданной схемы укажите (рис. 2):

- [*Заголовок*] ([*Title*]) — "Издатель сообщений логирования звонка" (Call message logging publisher);
- [*Название*] ([*Name*]) — "UsrCallsMessagePublisher".

Рис. 2. — Свойства схемы исходного кода



В созданной схеме в пространстве имен `Terrasoft.Configuration` добавьте новый класс `CallsMessagePublisher`, наследуемый от класса `BaseMessagePublisher`. Класс `BaseMessagePublisher` содержит базовую логику сохранения объекта в базу данных и базовую логику обработчиков событий. Класс-наследник будет содержать логику для конкретного отправителя, например, заполнение колонок объекта `Activity` и последующую отправку сообщения.

Для реализации нового класса `CallsMessagePublisher` в созданную схему добавьте следующий исходный код:

```
using System.Collections.Generic;
```

```

using Terrasoft.Core;

namespace Terrasoft.Configuration
{
    // Класс-наследник BaseMessagePublisher.
    public class CallsMessagePublisher : BaseMessagePublisher
    {
        // Конструктор класса.
        public CallsMessagePublisher(UserConnection userConnection, Dictionary<string, string> entityFieldsData)
            : base(userConnection, entityFieldsData) {
            //Схема, с которой будет работать CallsMessagePublisher.
            EntitySchemaName = "Activity";
        }
    }
}

```

После этого сохраните и опубликуйте схему.

2. Создать схему замещающей модели представления SectionActionsDashboard

Создайте схему замещающей модели представления, в которой в качестве родительского объекта укажите `SectionActionsDashboard` (рис. 3). Процесс создания схемы замещающей модели представления описан в статье ["Создать клиентскую схему"](#).

Рис. 3. — Свойства схемы замещающей модели представления

The screenshot shows a 'Properties' window with a search bar at the top. Below it, the 'General' section contains fields for 'Title' (SectionActionsDashboard), 'Name' (SectionActionsDashboard), and 'Package' (sdkActionsDashboardNewChannel). The 'Inheritance' section is expanded, showing 'Parent object' set to 'SectionActionsDashboard (ActionsDashboard)' and a checked 'Replace parent' checkbox.

На заметку.

Если нужно добавить канал только в одну страницу записи, то необходимо создать новый модуль с названием `имя_разделаSectionActionsDashboard` (например, `BooksSectionActionsDashboard`) и в качестве родительской схемы указать `SectionActionsDashboard` того раздела, в котором будет размещена страница записи.

В свойстве `diff` схемы замещающей модели представления установите операции вставки вкладки

`CallsMessageTab` и контейнера сообщений, а также укажите модуль, который будет отрисовываться в данном канале на одной из вкладок. После этого новый канал будет виден на страницах записей тех разделов, в которых подключен `SectionActionsDashboard`.

В свойстве `methods` переопределите метод `getSectionPublishers()`, который добавит созданный канал в список издателей сообщений, и метод `getExtendedConfig()`, в котором определяются параметры вкладки.

Чтобы метод `getExtendedConfig()` отработал корректно, загрузите изображение иконки канала и укажите ее в параметре `ImageSrc`. Файл изображения иконки, используемый в данном примере, можно скачать [здесь](#).

Переопределите метод `onGetRecordInfoForPublisher()` и добавьте метод `getContactEntityParameterValue()`, определяющие значение контакта из страницы записи раздела, в котором находится панель действий.

Исходный код схемы замещающей модели представления:

```
define("SectionActionsDashboard", ["SectionActionsDashboardResources", "UsrCallsMessagePublisher",
function(resources) {
    return {
        attributes: {},
        messages: {},
        methods: {
            // Метод задает настройки отображения вкладки канала в панели действий.
            getExtendedConfig: function() {
                // Вызов родительского метода.
                var config = this.callParent(arguments);
                var lczImages = resources.localizableImages;
                config.CallsMessageTab = {
                    // Изображение вкладки.
                    "ImageSrc": this.Terrasoft.ImageUrlBuilder.getUrl(lczImages.CallsMessageTabImage),
                    // Значение маркера.
                    "MarkerValue": "calls-message-tab",
                    // Выравнивание.
                    "Align": this.Terrasoft.Align.RIGHT,
                    // Тэг.
                    "Tag": "UsrCalls"
                };
                return config;
            },
            // Переопределяет родительский и добавляет значение контакта из страницы записи
            // раздела, в котором находится панель действий.
            onGetRecordInfoForPublisher: function() {
                var info = this.callParent(arguments);
                info.additionalInfo.contact = this.getContactEntityParameterValue(info.relationSchemaName);
                return info;
            },
            // Определяет значение контакта из страницы записи раздела,
            // в котором находится панель действий.
            getContactEntityParameterValue: function(relationSchemaName) {
                var contact;
```

```

        if (relationSchemaName === "Contact") {
            var id = this.getMasterEntityParameterValue("Id");
            var name = this.getMasterEntityParameterValue("Name");
            if (id && name) {
                contact = {value: id, displayValue: name};
            }
        } else {
            contact = this.getMasterEntityParameterValue("Contact");
        }
        return contact;
    },
    //Добавляет созданный канал в список издателей сообщений.
    getSectionPublishers: function() {
        var publishers = this.callParent(arguments);
        publishers.push("UsrCalls");
        return publishers;
    }
},
// Массив модификаций, с помощью которых строится представление модуля в интерфейсе
diff: /**SCHEMA_DIFF*/[
    // Добавление вкладки CallsMessageTab.
    {
        // Тип операции – вставка.
        "operation": "insert",
        // Название вкладки.
        "name": "CallsMessageTab",
        // Название родительского элемента.
        "parentName": "Tabs",
        // Название свойства.
        "propertyName": "tabs",
        // Конфигурационный объект свойств.
        "values": {
            // Массив дочерних элементов.
            "items": []
        }
    },
    // Добавление контейнера сообщений.
    {
        "operation": "insert",
        "name": "CallsMessageTabContainer",
        "parentName": "CallsMessageTab",
        "propertyName": "items",
        "values": {
            // Тип элемента – контейнер.
            "itemType": this.Terrasoft.ViewItemType.CONTAINER,
            // CSS-класс для контейнера.
            "classes": {
                "wrapClassName": ["calls-message-content"]
            },

```



```

        "items": []
    }
},
// Добавление модуля UsrCallsMessageModule.
{
    "operation": "insert",
    "name": "UsrCallsMessageModule",
    "parentName": "CallsMessageTab",
    "propertyName": "items",
    "values": {
        // CSS-класс для модуля вкладок.
        "classes": {
            "wrapClassName": ["calls-message-module", "message-module"]
        },
        // Тип элемента – модуль.
        "itemType": this.Terrasoft.ViewItemType.MODULE,
        // Название модуля.
        "moduleName": "UsrCallsMessagePublisherModule",
        // Привязка метода, выполняемого после отрисовки элемента.
        "afterrender": {
            "bindTo": "onMessageModuleRendered"
        },
        // Привязка метода, выполняемого после перерисовки элемента.
        "afterrerender": {
            "bindTo": "onMessageModuleRendered"
        }
    }
}
]/**SCHEMA_DIFF*/
};
}
);

```

3. Создать модуль UsrCallsMessagePublisherModule

Модуль `UsrCallsMessagePublisherModule` служит контейнером для прорисовки в `SectionActionsDashboard` страницы `UsrCallsMessagePublisherPage`, в которой будет реализована логика добавляемого канала.

Для модуля установите следующие значения (рис. 4):

- [*Заголовок*] ([*Title*]) — "Модуль издателя сообщений логирования звонка" ("Call messages logging publisher module");
- [*Название*] ([*Name*]) — "UsrCallsMessagePublisherModule".

Рис. 4. — Свойства модуля

Properties

<Enter search text>

▼ **General**

Title: Call messages logging publisher module

Name: UsrCallsMessagePublisherModule

Package: sdkActionsDashboardNewChannel

▼ **Inheritance**

Parent object: [dropdown]

Forbid substitution: ☐

Replace parent: ☐

Исходный код модуля:

```
define("UsrCallsMessagePublisherModule", ["BaseMessagePublisherModule"],
function() {
    // Определение класса.
    Ext.define("Terrasoft.configuration.UsrCallsMessagePublisherModule", {
        // Базовый класс.
        extend: "Terrasoft.BaseMessagePublisherModule",
        // Сокращенное имя класса.
        alternateClassName: "Terrasoft.UsrCallsMessagePublisherModule",
        // Инициализация страницы, которая будет отрисовываться в данном модуле.
        initSchemaName: function() {
            this.schemaName = "UsrCallsMessagePublisherPage";
        }
    });
    // Возвращает объект класса, определенного в модуле.
    return Terrasoft.UsrCallsMessagePublisherModule;
});
```

4. Создать страницу UsrCallsMessagePublisherPage

Для создаваемой страницы установите в качестве родительского объекта схему `BaseMessagePublisherPage` пакета `MessagePublisher`. В качестве названия и заголовка укажите значение `"UsrCallsMessagePublisherPage"`.

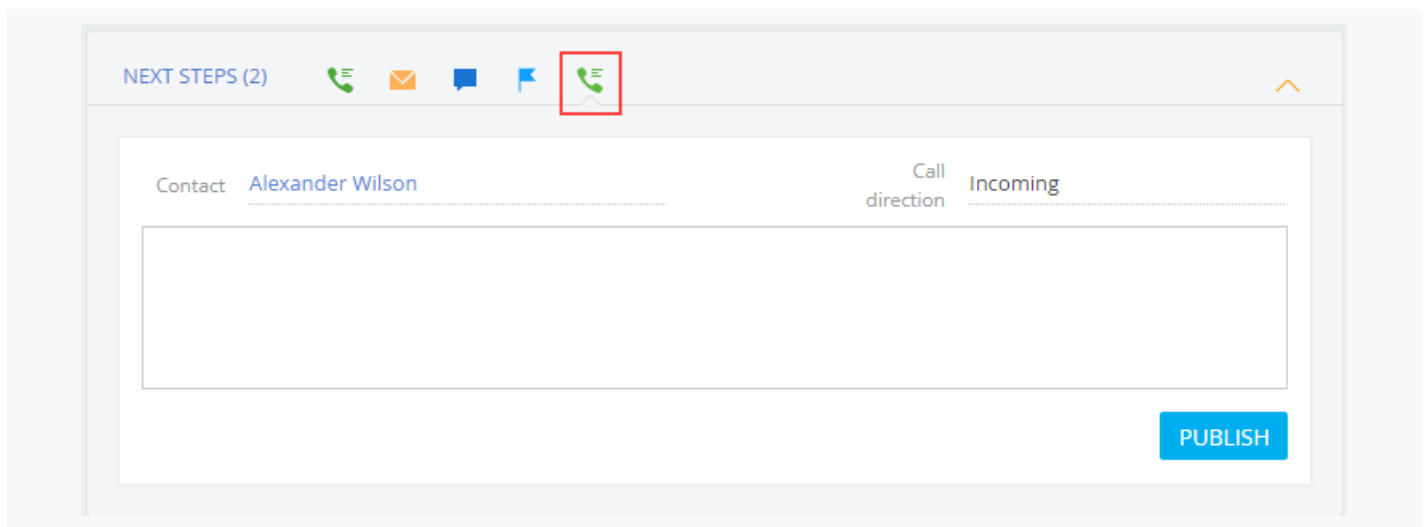
В исходном коде страницы укажите имя схемы объекта, с которым будет работать страница (в данном случае `Activity`), реализуйте логику публикации сообщения и переопределите метод `getServiceConfig`, в котором укажите имя класса из конфигурации.

```
// Задает класс, который будет работать с данной страницей.
```

```
getServiceConfig: function() {
    return {
        className: "Terrasoft.Configuration.CallsMessagePublisher"
    };
}
```

Реализация логики публикации сообщения содержит довольно большое количество методов, атрибутов и свойств. Полностью исходный код схемы `UsrCallsMessagePublisherPage` вы можете скачать по [ссылке](#). В исходном коде показана реализация рабочего канала `CallMessagePublisher`, который используется для логирования входящих и исходящих звонков. Результатом выполнения данного примера будет новый рабочий канал в `SectionActionsDashboard` (рис. 5).

Рис. 5. — Пример пользовательского канала `CallsMessagePublisherPage` в `SectionActionsDashboard` раздела [*Контакты*]



Добавить мультиязычные шаблоны email-сообщений



Сложный

Creatio предоставляет возможность использовать собственную логику подбора шаблона email-сообщения по языку. На инструментальной панели действий (`ActionsDashboard`) записи раздела можно выбирать шаблоны email-сообщений на необходимом языке. Подбор выполняется на основании специализированных правил, которые могут быть определены в зависимости от раздела. Если специфические правила не определены, подбор ведется на основе контакта, связанного с редактируемой записью (колонок [*Контакт*]). Если колонки связи с контактом в объекте раздела нет, используется значение системной настройки `DefaultMessageLanguage`.

Для добавления собственной логики по подбору мультиязычных шаблонов:

1. Создайте класс или классы, унаследованные от `BaseLanguageRule` и определите правила подбора языка (один класс определяет одно правило).
2. Создайте класс, унаследованный от `BaseLanguageIterator`. В конструкторе класса определите свойство

`LanguageRules` как массив экземпляров классов, созданных на предыдущем шаге. Порядок следования соответствует приоритету правил.

3. Создайте класс-наследник от `AppEventListenerBase`, выполняющий привязку класса, определяющего правила подбора языка, к разделу.

4. В справочник [*Шаблоны email сообщений*] ([*Email Templates*]) добавьте нужные мультиязычные шаблоны.

Описание примера

В пользовательский раздел добавить логику выбора языка email-сообщения на основе колонки `UsrContact` основного объекта раздела. Для примера использовать английский и испанский языки.

Исходный код

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

Важно.

Пакет можно установить для продуктов Creatio, содержащих пакет `EmailTemplates`. После установки пакета убедитесь, что выполнены все предварительные настройки, описанные ниже.

Предварительные настройки

Для корректной работы примера:

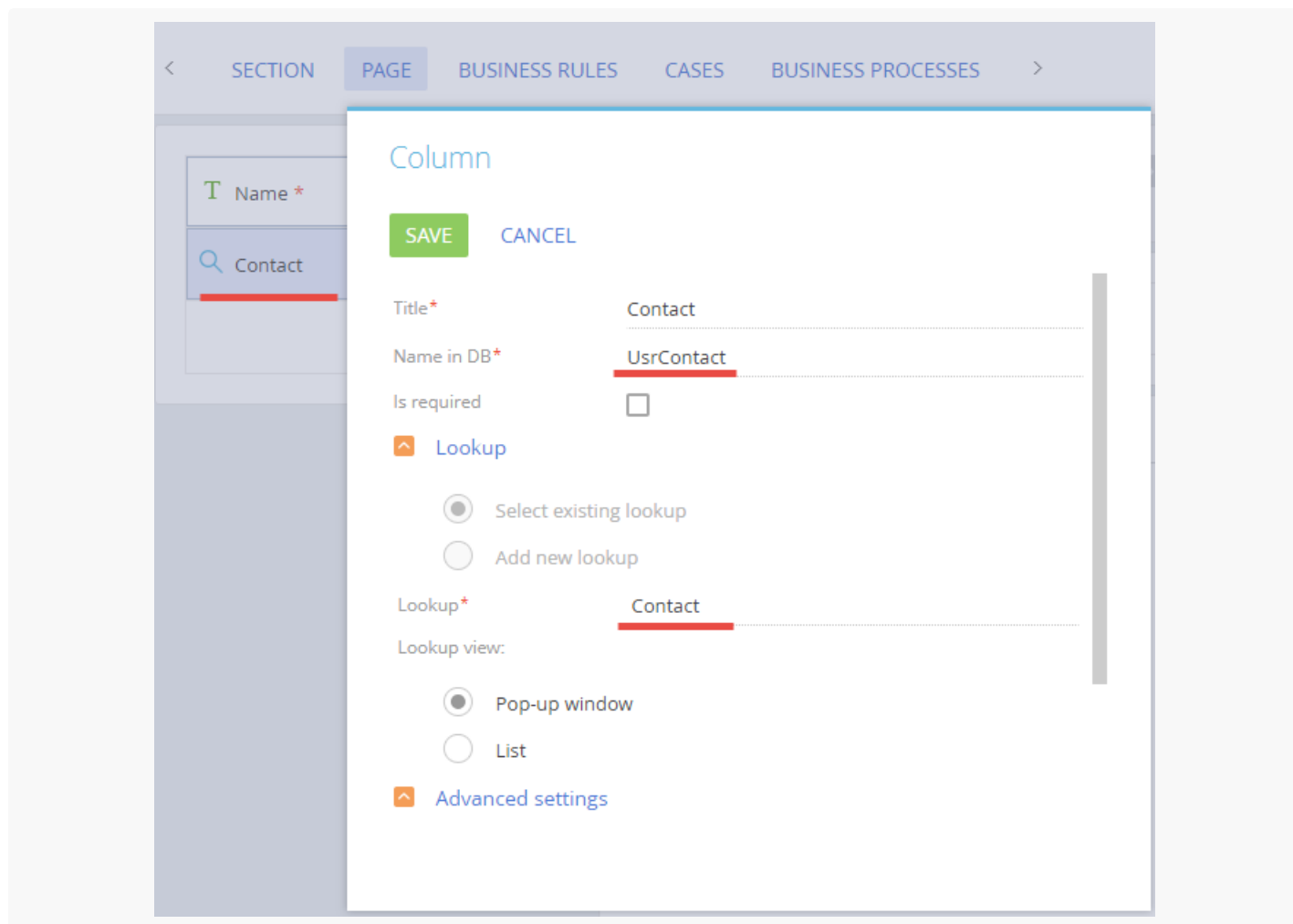
1. Убедитесь, что в справочнике [*Языки общения*] ([*Customer languages*]) используются английский и испанский языки (рис. 1).

Рис. 1. — Справочник [*Языки общения*]

Customer languages			
Filters/folders ▼			
Name	Description	Code	Is used ▼
English (United St...	English (United St...	en-US	Yes
Spanish (Spain)	Español (España, ...	es-ES	Yes

2. В мастере разделов проверьте, что на странице записи пользовательского раздела существует колонка `UsrContact`, связанная со справочником [*Контакт*] ([*Contact*]) (рис. 2).

Рис. 2. — Колонка `UsrContact`



Алгоритм реализации примера

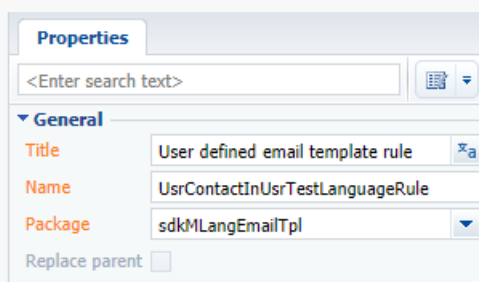
1. Добавить правило подбора языка

В пользовательском пакете создайте схему [*Исходный код*] (см. "[Создать схему \[Исходный код \]](#)").

Для созданной схемы укажите (рис. 3):

- [Название] ([Name]) — "UsrContactInUsrTestLanguageRule";
- [Заголовок] ([Title]) — "Пользовательское правило шаблона email-сообщений" ("User defined email template rule").

Рис. 3. — Свойства схемы [*Исходный код*]



Добавьте в схему следующий исходный код:

```
namespace Terrasoft.Configuration
{
    using System;
    using Terrasoft.Core;
    using Terrasoft.Core.Entities;
    public class ContactInUsrTestLanguageRule : BaseLanguageRule
    {
        public ContactInUsrTestLanguageRule (UserConnection userConnection) : base(userConnection)
        {
        }
        // Определяет идентификатор предпочитаемого языка пользователя.
        // recId – идентификатор текущей записи.
        public override Guid GetLanguageId(Guid recId)
        {
            // Создание экземпляра EntitySchemaQuery для основного объекта пользовательского раздела
            var esq = new EntitySchemaQuery(UserConnection.EntitySchemaManager, "UsrMLangEmailTpl");
            // Определение названия колонки языка контакта.
            var languageColumnName = esq.AddColumn("UsrContact.Language.Id").Name;
            // Получение экземпляра текущей записи.
            Entity usrRecEntity = esq.GetEntity(UserConnection, recId);
            // Получение значения идентификатора предпочитаемого языка пользователя.
            Guid languageId = usrRecEntity.GetTypedColumnValue<Guid>(languageColumnName);
            return languageId;
        }
    }
}
```

Опубликуйте схему.

2. Определить порядок правил подбора языка

В пользовательском пакете создайте схему [Исходный код] (см. ["Создать схему \[Исходный код \]"](#)).

Для созданной схемы укажите (рис. 3):

- [Название] ([Name]) — "UsrTestLanguageIterator";

- [Заголовок] ([Title]) — "Пользовательский итератор правил выбора языка" ("User defined language iterator").

Добавьте в схему исходный код, приведенный ниже:

```
namespace Terrasoft.Configuration
{
    using Terrasoft.Core;
    public class UsrTestLanguageIterator: BaseLanguageIterator
    {
        public UsrTestLanguageIterator(UserConnection userConnection): base(userConnection)
        {
            // Массив правил выбора языка.
            LanguageRules = new ILanguageRule[] {
                // Пользовательское правило.
                new ContactInUsrTestLanguageRule (UserConnection),
                // Правило по умолчанию.
                new DefaultLanguageRule(UserConnection),
            };
        }
    }
}
```

Вторым элементом массива является `DefaultLanguageRule`. Это правило использует для получения языка системную настройку `DefaultLanguage` и используется по умолчанию, если язык не был найден другими, более приоритетными правилами.

Выполните публикацию схемы.

3. Привязать итератор правил выбора языка к разделу

В пользовательском пакете создайте схему [*Исходный код*] (см. ["Создать схему \[Исходный код \]"](#)).

Для созданной схемы укажите (рис. 3):

- [Название] ([Name]) — "UsrTestMLangBinder";
- [Заголовок] ([Title]) — "UsrTestMLangBinder".

Добавьте в схему следующий исходный код:

```
namespace Terrasoft.Configuration
{
    using Terrasoft.Core.Factories;
    using Terrasoft.Web.Common;
    public class UsrTestMLangBinder: AppEventListenerBase
    {
        public override void OnAppStart(AppEventContext context)
        {
```

```

        // Вызов базовой логики.
        base.OnAppStart(context);
        // Привязка итератора к пользовательскому разделу.
        // UsrMLangEmailTpl – название основного объекта раздела.
        ClassFactory.Bind<ILanguageIterator, UsrTestLanguageIterator>("UsrMLangEmailTpl");
    }
}
}

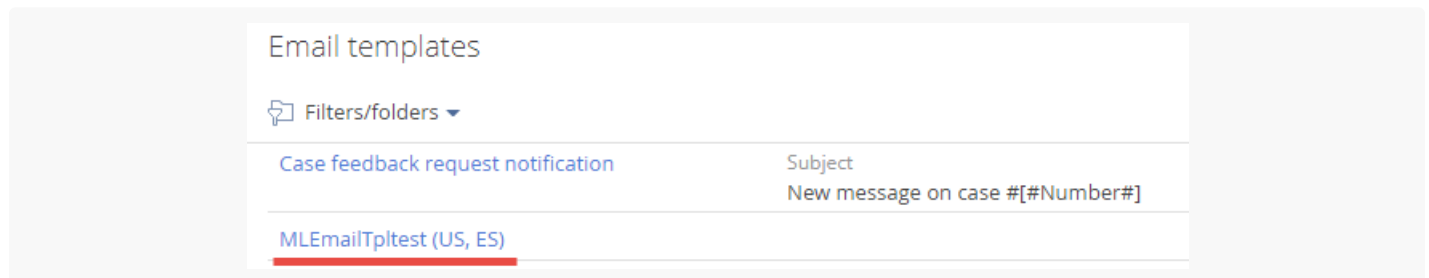
```

Опубликуйте схему.

4. Добавить необходимые мультиязычные шаблоны

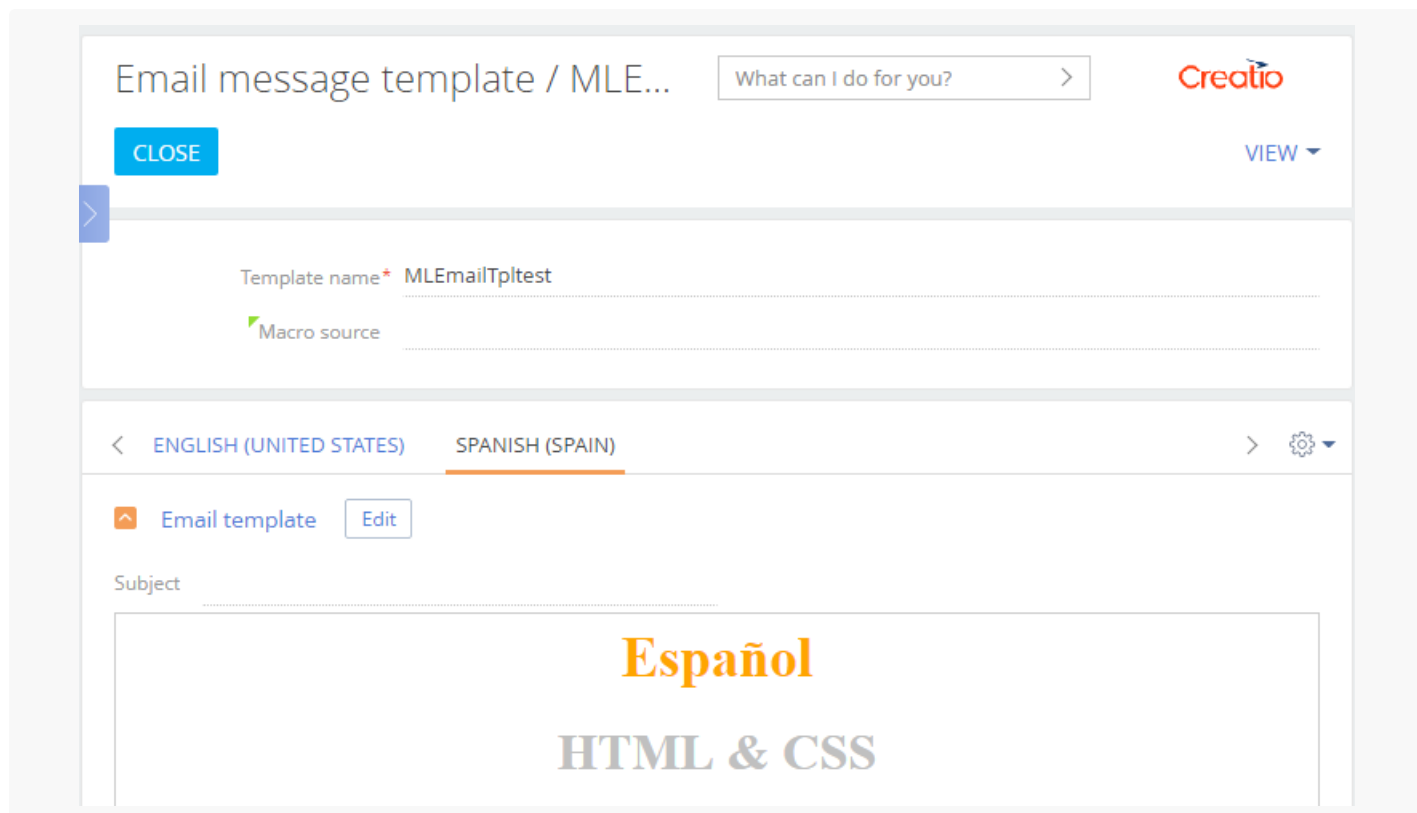
В справочник [*Шаблоны email сообщений*] ([*Email Templates*]) добавьте новую запись (рис. 4), в которой определите шаблоны email-сообщений на требуемых языках (рис. 5).

Рис. 4. — Новая запись в справочнике [*Шаблоны email сообщений*]



Email templates	
Filters/folders ▼	
Case feedback request notification	Subject New message on case #[Number#]
MLEmailTpltest (US, ES)	

Рис. 5. — Добавление шаблонов на требуемых языках



В результате выполнения примера на странице записи пользовательского раздела (рис. 6) в канале инструментальной панели действий (рис. 6, 1) шаблоны email сообщений (рис. 6, 2) будут выбираться автоматически на том языке контакта (рис. 6, 3), который установлен как предпочтительный (рис. 7).

Рис. 6. — Результат выполнения примера

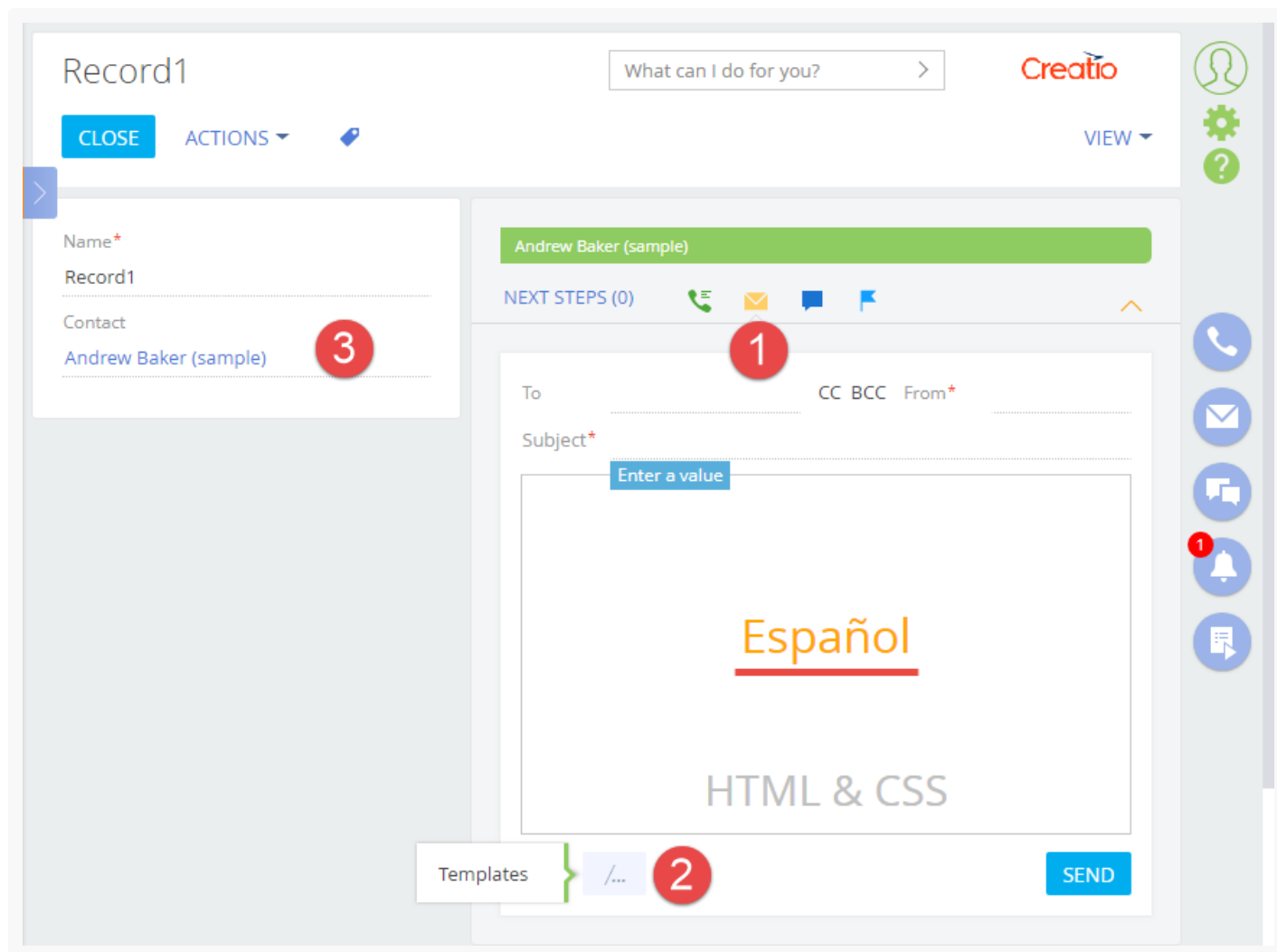
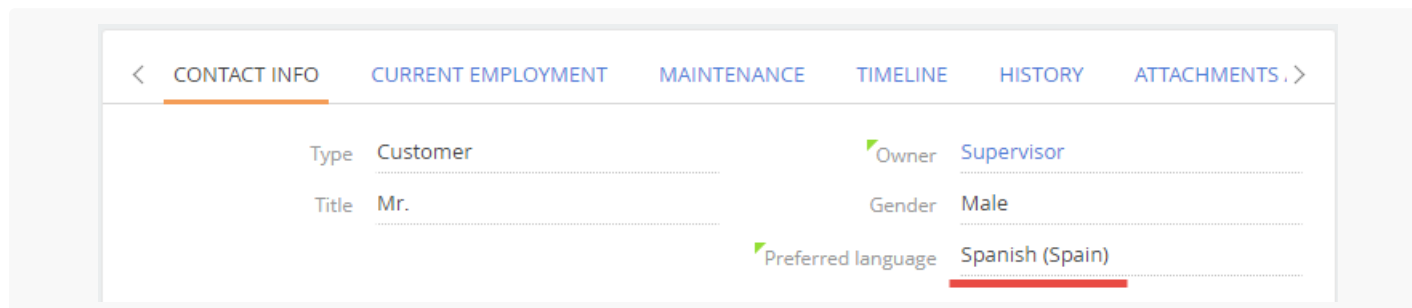


Рис. 7. — Предпочтительный язык контакта



Дашборды

 Сложный

Дашборд (блок итогов) — визуальное представление разных типов аналитических данных, например, в виде графика или виджета. Для работы с аналитикой раздела необходимо перейти в представление аналитики необходимого раздела. Если же необходима работа с данными всех разделов приложения, то нужно перейти в раздел [*Итоги*].

Структура данных для хранения информации по итогам

Раздел итогов представляет собой зависящий от прав пользователя набор элементов-вкладок. Механизм работы с итогами реализован с помощью клиентского менеджера итогов `DashboardManager` и элементов `DashboardManagerItem`, которые и представляют вкладки. За итоги в системе отвечает объект `SysDashboard`. Свойства объекта `SysDashboard` описаны в таблице.

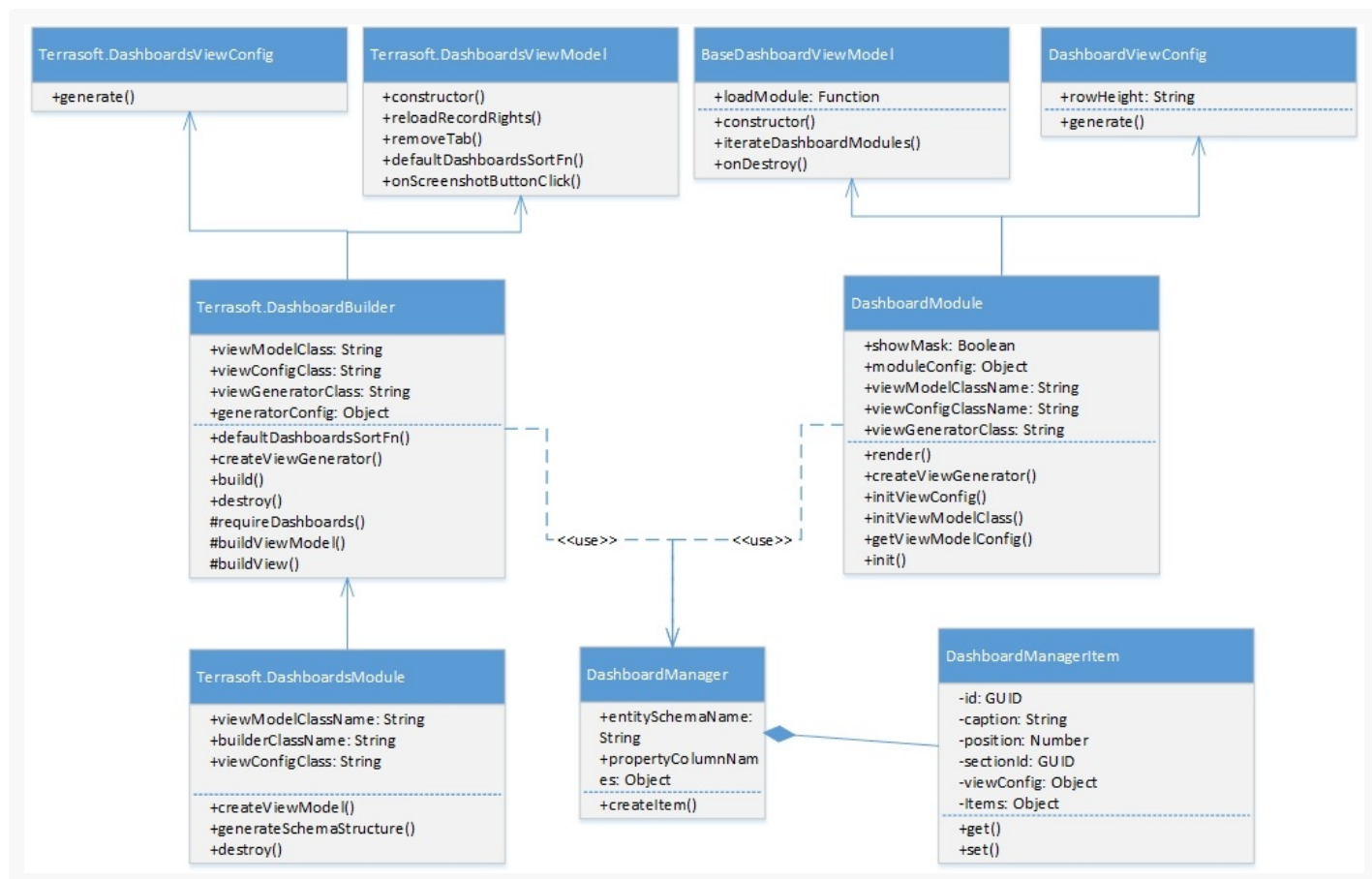
Описание свойств объекта `SysDashboard`

Название	Заголовок	Тип	Описание
<code>Caption</code>	Заголовок	String	Данная информация отображается в заголовке вкладки
<code>Position</code>	Позиция	Number	Если позиция не задана, элементы отображаются в алфавитном порядке
<code>Section</code>	Раздел	Lookup	Раздел системы.
<code>ViewConfig</code>	Конфигурация отображения элементов (дашбордов)	Array	<pre>[{ //Тип элемента (Terrasoft.ViewItemType). itemType: "4", // Название элемента. name: "SomeInvoiceChart", // Конфигурация отображения. layout: { columns: 4, rows: 4, colspan: 4, rowspan: 4 } }, {...}]</pre>
<code>Items</code>	Конфигурация модулей элементов (дашбордов)	JSON Object	<pre>{ // Название элемента, для которого определяю "SomeInvoiceChart": { // Название модуля для отображения элемен "widgetType": "Chart", // Параметры, необходимые для отображения "parameters": { "caption": "some caption", ... }, }, ... }, {...}]</pre>

Название	Заголовок	Тип	Описание

Реализация функциональности в режиме просмотра ИТОГОВ

Иерархия классов, реализующих функциональность в режиме просмотра итогов:



Модуль `SectionDashboardModule` :

- `SectionDashboardBuilder` — класс, инкапсулирующий в себе логику генерации представления и класса модели представления для модуля раздела итогов.
- `SectionDashboardsViewModel` — класс модели представления раздела итогов.
- `SectionDashboardsModule` — класс модуля раздела итогов.

Модуль `DashboardModule` :

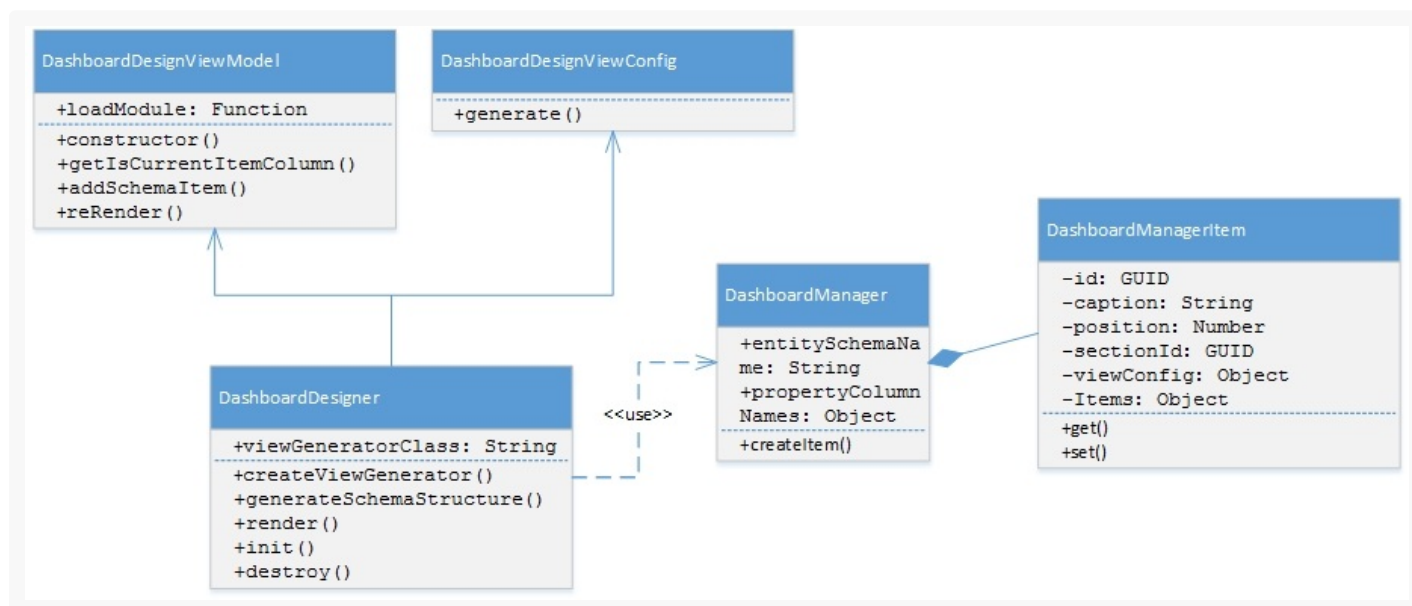
- `DashboardViewConfig` — класс, генерирующий конфигурацию представления для модуля страницы ИТОГОВ.
- `BaseDashboardViewModel` — базовый класс модели представления страницы итогов.
- `DashboardModule` — класс, содержащий функциональность по работе с модулями итогов.

Модуль `DashboardBuilder` :

- `DashboardsViewConfig` — класс, генерирующий конфигурацию представления для модуля итогов.
- `BaseDashboardsViewModel` — базовый класс модели представления итогов.
- `DashboardBuilder` — класс построения модуля итогов.

Реализация функциональности в режиме настройки ИТОГОВ

Иерархия классов, реализующих функциональность в режиме настройки итогов:



Модуль `DashboardDesigner` :

- `DashboardDesignerViewConfig` — класс, генерирующий конфигурацию представления для модуля дизайнера итогов.
- `DashboardDesignerViewModel` — класс модели представления дизайнера итогов.
- `DashboardDesigner` — класс визуального модуля итогов.

Базовые классы, реализующие функциональность дашборда

`BaseWidgetViewModelClass` — базовый класс модели представления дашбордов. Для использования класса необходимо зарегистрировать в модуле такие сообщения:

- `GetHistoryState (publish; ptp) ;`
- `ReplaceHistoryState (publish; broadcast) ;`
- `HistoryStateChanged (subscribe; broadcast) ;`

- `GetWidgetParameters (subscribe; ptp) ;`
- `PushWidgetParameters (subscribe; ptp)` — если используется получение параметров от модулей (`useCustomParameterMethods = true`).

`BaseWidgetDesigner` — базовая схема представления настройки дашбордов. Основные методы:

- `getWidgetConfig()` — возвращает объект актуальных настроек дашборда.
- `getWidgetConfigMessage()` — возвращает название сообщения получения настроек модуля дашборда.
- `getWidgetModuleName()` — возвращает название модуля дашборда.
- `getWidgetRefreshMessage()` — возвращает название сообщения обновления дашборда.
- `getWidgetModulePropertiesTranslator()` — возвращает объект соотношения свойств модуля дашборда и модуля настройки дашборда.

`BaseAggregationWidgetDesigner` — содержит методы для работы с агрегирующими колонками и типами агрегации.

`DashboardEnums` — содержит перечисление свойств, используемых в дашбордах.

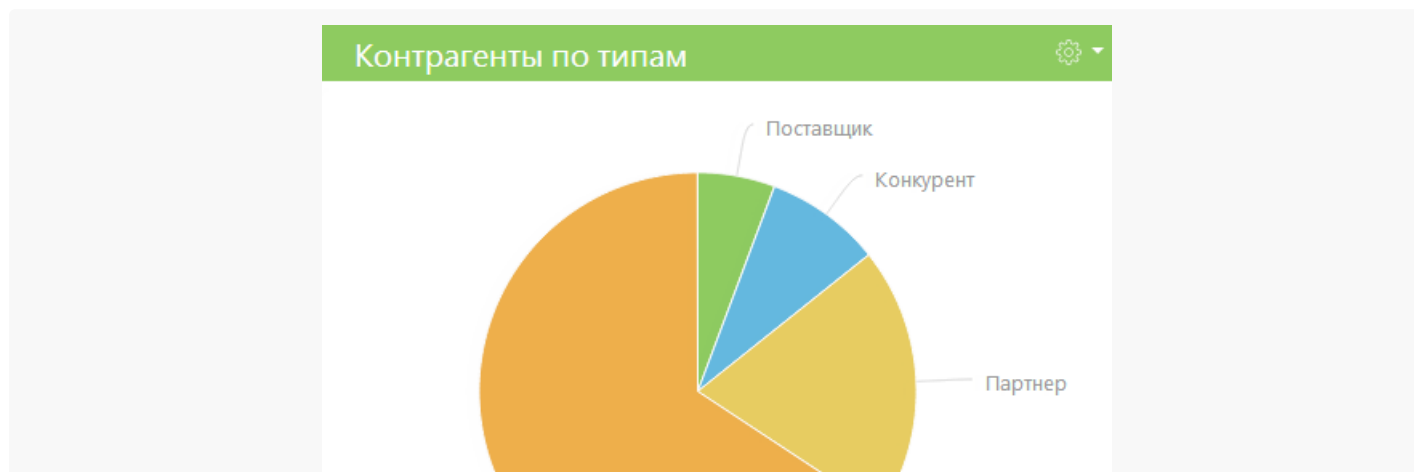
`Terrasoft.DashboardEnums.WidgetType` — содержит конфигурацию дашбордов для режима просмотра (view) и режима настройки (design) итогов. Конфигурация определяется следующими свойствами:

- `moduleName` — название модуля дашборда.
- `configurationMessage` — название сообщения получения настроек модуля.
- `resultMessage` — название сообщения для отдачи параметров настройки модуля дизайнера дашборда.
- `stateConfig (stateObj)` — название схемы дизайнера дашборда.

Виды дашбордов

График

[График](#) в наглядной форме отображает множественные данные из системы. С помощью графика можно отобразить, например, распределение контрагентов по типам. График может отображать информацию в виде диаграмм разных типов, либо в виде реестра данных.





Классы, реализующие функциональность графиков

`ChartViewModel` — модель представления графиков.

`ChartViewConfig` — генерирует конфигурацию представления модуля графика.

`ChartModule` — модуль, предназначенный для работы с графиками.

`ChartDesigner` — схема представления страницы графиков.

`ChartModuleHelper` — используется для формирования запроса с помощью объекта `Terrasoft.EntitySchemaQuery`.

`ChartDrillDownProvider` — содержит методы для работы с функциональностью углубления в элементы (для работы с сериями в графиках).

Параметры настройки графика

Для настройки графика необходимо в конфигурацию модулей дашбордов добавить конфигурационный JSON-объект с настройками свойств графика. Конфигурация модуля дашборда определяется свойством `Items` объекта `SysDashboard`. Подробнее об объекте `SysDashboard` и его свойствах можно узнать из [статьи](#).

В конфигурационном JSON-объекте с настройками дашборда необходимо установить значение "Chart" свойству `widgetType`. Кроме того, свойству `parameters` нужно присвоить объект с необходимыми параметрами. Возможные параметры графика приведены в таблице.

Параметры настройки графика

Название	Тип	Описание
<code>seriesConfig</code>	<code>object</code>	Настройки вложенного графика из серии.
<code>orderBy</code>	<code>string</code>	Поле сортировки.
<code>orderDirection</code>	<code>string</code>	Направление сортировки.
<code>caption</code>	<code>string</code>	Заголовок графика.
<code>sectionId</code>	<code>string</code>	Идентификатор раздела.
<code>xAxisDefaultCaption</code>	<code>string</code>	Заголовок оси X по умолчанию.
<code>yAxisDefaultCaption</code>	<code>string</code>	Заголовок оси Y по умолчанию.
<code>primaryColumnName</code>	<code>string</code>	Название первичной колонки. По умолчанию первичной является колонка Id.

yAxisConfig	object	Массив настроек подписи оси Y.
schemaName	string	Объект, по которому строится график.
sectionBindingColumn	string	Колонка связи с разделом.
func	string	Агрегирующая функция.
type	string	Тип графика.
XAxisCaption	string	Заголовок оси X.
YAxisCaption	string	Заголовок оси Y.
xAxisColumn	string	Колонка группировки оси X.
yAxisColumn	string	Колонка группировки оси Y.
styleColor	string	Цвет графика.
filterData	object	Настройка фильтрации.

Показатель

[Дашборд “Показатель”](#) отображает расчетное числовое значение или дату по определенным данным системы, например, общее количество сотрудников отдела..



Классы, реализующие функциональность показателей

`IndicatorViewModel` — модель представления показателя.

`IndicatorViewConfig` — генерирует конфигурацию представления модуля показателя.

`IndicatorModule` — модуль, предназначенный для работы с показателями.

`IndicatorDesigner` — схема представления страницы показателя.

Параметры настройки показателя

Для настройки показателя необходимо в конфигурацию модулей дашбордов добавить конфигурационный JSON-объект с настройками свойств показателя. Конфигурация модуля дашборда определяется свойством `Items` объекта `SysDashboard`. Подробнее об объекте `SysDashboard` и его свойствах можно узнать из [статьи](#).

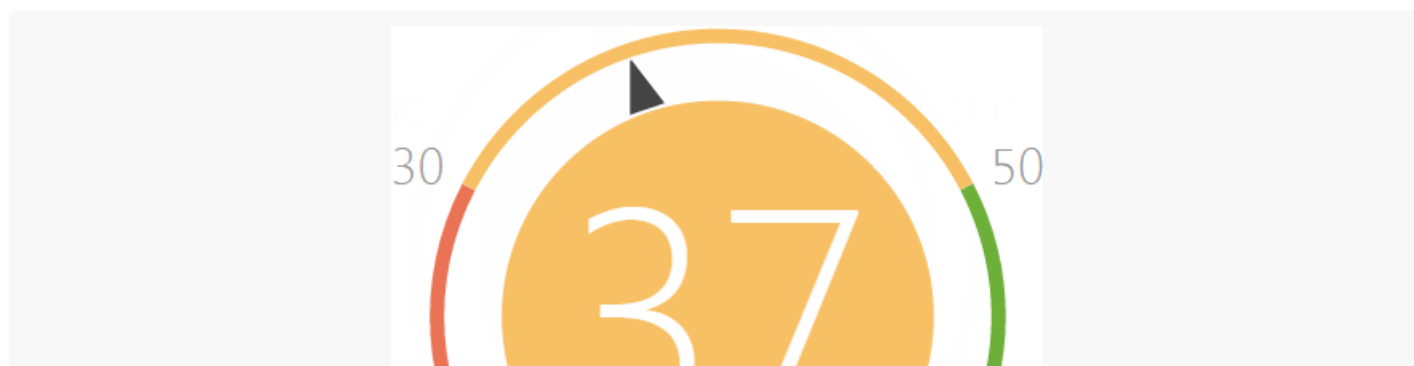
В конфигурационном JSON-объекте с настройками дашборда необходимо установить значение "Indicator" свойству `widgetType`. Кроме того, свойству `parameters` нужно присвоить объект с необходимыми параметрами. Возможные параметры показателя приведены в таблице.

Параметры настройки показателя

Название	Тип	Описание
<code>caption</code>	<code>string</code>	Заголовок показателя.
<code>sectionId</code>	<code>string</code>	Идентификатор раздела.
<code>entitySchemaName</code>	<code>string</code>	Объект, по которому строится показатель.
<code>sectionBindingColumn</code>	<code>string</code>	Колонка связи с разделом.
<code>columnName</code>	<code>string</code>	Название агрегирующей колонки.
<code>format</code>	<code>object</code>	Формат показателя.
<code>filterData</code>	<code>object</code>	Настройка фильтрации.
<code>aggregationType</code>	<code>number</code>	Тип агрегирующей функции.
<code>style</code>	<code>string</code>	Цвет показателя.

Шкала

[Шкала](#) отображает число, полученное в результате запроса к данным системы, относительно нормативных значений. С помощью шкалы можно отобразить, например, реальное количество выполненных активностей относительно запланированного количества.





Классы, реализующие функциональность дашборда "Шкала"

`GaugeViewModel` — модель представления шкалы.

`GaugeViewConfig` — генерирует конфигурацию представления модуля шкалы.

`GaugeModule` — модуль, предназначенный для работы со шкалой.

`GaugeChart` — реализует компонент графика типа шкала.

`GaugeDesigner` — схема представления страницы шкалы.

Параметры настройки шкалы

Для настройки шкалы необходимо в конфигурацию модулей дашбордов добавить конфигурационный JSON-объект с настройками свойств шкалы. Конфигурация модуля дашборда определяется свойством `Items` объекта `SysDashboard`. Подробнее об объекте `SysDashboard` и его свойствах можно узнать из [статьи](#).

В конфигурационном JSON-объекте с настройками дашборда необходимо установить значение "Gauge" свойству `widgetType`. Кроме того, свойству `parameters` нужно присвоить объект с необходимыми параметрами. Возможные параметры шкалы приведены в таблице.

Параметры настройки шкалы

Название	Тип	Описание
<code>caption</code>	<code>string</code>	Заголовок шкалы.
<code>sectionId</code>	<code>string</code>	Идентификатор раздела.
<code>entitySchemaName</code>	<code>string</code>	Объект, по которому строится шкала.
<code>sectionBindingColumn</code>	<code>string</code>	Колонка связи с разделом.

Список

[Список](#) отображает информацию из системы в виде списка с заданным количеством позиций. С помощью списка можно отобразить, например, десятку самых продуктивных менеджеров по количеству закрытых сделок.

Топ-3 самых продуктивных менеджеров	
Мирный Евгений	70
Малянов Дмитрий	68

Классы, реализующие функциональность списков

`DashboardGridViewModel` — модель представления списка.

`DashboardGridViewConfig` — генерирует конфигурацию представления модуля списка.

`DashboardGridModule` — модуль, предназначенный для работы со списками.

`DashboardGridDesigner` — схема представления страницы списка.

Параметры настройки списка

Для настройки списка необходимо в конфигурацию модулей дашбордов добавить конфигурационный JSON-объект с настройками свойств списка. Конфигурация модуля дашборда определяется свойством `Items` объекта `SysDashboard`. Подробнее об объекте `SysDashboard` и его свойствах можно узнать из [статьи](#).

В конфигурационном JSON-объекте с настройками дашборда необходимо установить значение "DashboardGrid" свойству `widgetType`. А также свойству `parameters` нужно присвоить объект с необходимыми параметрами. Возможные параметры списка приведены в таблице.

Параметры настройки списка

Название	Тип	Описание
<code>caption</code>	<code>string</code>	Заголовок списка.
<code>sectionBindingColumn</code>	<code>string</code>	Колонка связи с разделом.
<code>filterData</code>	<code>object</code>	Настройка фильтрации.
<code>sectionId</code>	<code>string</code>	Идентификатор раздела.
<code>entitySchemaName</code>	<code>string</code>	Объект, по которому строится список.
<code>style</code>	<code>string</code>	Цвет списка.
<code>orderDirection</code>	<code>number</code>	Направление сортировки (1 — по возрастанию, 2 — по убыванию).
<code>orderColumn</code>	<code>string</code>	Колонка, по которой сортируется список.
<code>rowCount</code>	<code>number</code>	Количество рядов для отображения.
<code>gridConfig</code>	<code>object</code>	Конфигурация списка.

Web-страница

[Дашборд "Web-страница"](#) предназначен для отображения интернет-страниц на панели итогов. Например, это может быть онлайн-калькулятор валют или ваш корпоративный сайт.

Классы, реализующие функциональность Web-страниц

`WebPageViewModel` — модель представления Web-страницы.

`WebPageViewConfig` — генерирует конфигурацию представления модуля Web-страницы.

`WebPageModule` — модуль, предназначенный для работы с Web-страницами.

`WebPageDesigner` — схема представления страницы дашборда Web-страницы.

Параметры настройки Web-страницы

Для настройки Web-страницы необходимо в конфигурацию модулей дашбордов добавить конфигурационный JSON-объект с настройками свойств Web-страницы. Конфигурация модуля дашборда определяется свойством `Items` объекта `SysDashboard`. Подробнее об объекте `SysDashboard` и его свойствах можно узнать из [статьи](#).

В конфигурационном JSON-объекте с настройками дашборда необходимо установить значение "WebPage" свойству `widgetType`. Кроме того, свойству `parameters` нужно присвоить объект с необходимыми параметрами. Возможные параметры Web-страницы приведены в таблице.

Параметры настройки Web-страницы

Название	Тип	Описание
<code>caption</code>	<code>string</code>	Заголовок дашборда Web-страницы.
<code>sectionId</code>	<code>string</code>	Идентификатор раздела.
<code>url</code>	<code>string</code>	Ссылка на Web-страницу.
<code>style</code>	<code>string</code>	CSS стили дашборда Web-страницы.

Воронка продаж

[Дашборд "Воронка продаж"](#) используется для анализа динамики продвижения продаж по стадиям.

Классы, реализующие функциональность воронки продаж

`OpportunityFunnelChart` — класс, унаследованный от `Chart`.

Параметры настройки воронки продаж

Для настройки воронки продаж необходимо в конфигурацию модулей дашбордов добавить конфигурационный JSON-объект с настройками свойств воронки. Конфигурация модуля дашборда определяется свойством `Items` объекта `SysDashboard`. Подробнее об объекте `SysDashboard` и его

свойствах можно узнать из [статьи](#).

В конфигурационном JSON-объекте с настройками дашборда необходимо установить значение "OpportunityFunnel" свойству `widgetType` . Кроме того, свойству `parameters` нужно присвоить объект с необходимыми параметрами. Возможные параметры воронки продаж приведены в таблице.

Параметры настройки воронки продаж

Название	Тип	Описание
<code>caption</code>	<code>string</code>	Заголовок дашборда воронки.
<code>sectionId</code>	<code>string</code>	Идентификатор раздела.
<code>defPeriod</code>	<code>string</code>	Период воронки (по умолчанию последняя неделя).
<code>sectionBindingColumn</code>	<code>string</code>	Колонка связи с разделом.
<code>type</code>	<code>string</code>	Тип графика ("funnel").
<code>filterData</code>	<code>object</code>	Настройка фильтрации.

Изменить расчеты в воронке продаж



Существует возможность изменять расчеты в воронке, которая подключена к объекту раздела [*Продажи*], для отображения по нему аналитики. Для этого нужно создать новый модуль для расчетов и заменить клиентскую схему отображения воронки.

Последовательность действий для изменения расчетов в воронке:

1. Создать новый класс, который наследуется от `FunnelBaseDataProvider` и задать логику расчетов.
2. Создать схему замещающей модели представления `FunnelChartSchema` и использовать в ней новый класс расчетов.

Пример изменения отображения расчетов в воронке продаж в срезе “по количеству”

Описание кейса

Необходимо изменить расчеты воронки, поменяв отображение количества продаж на количество продуктов, добавленных в продажи.

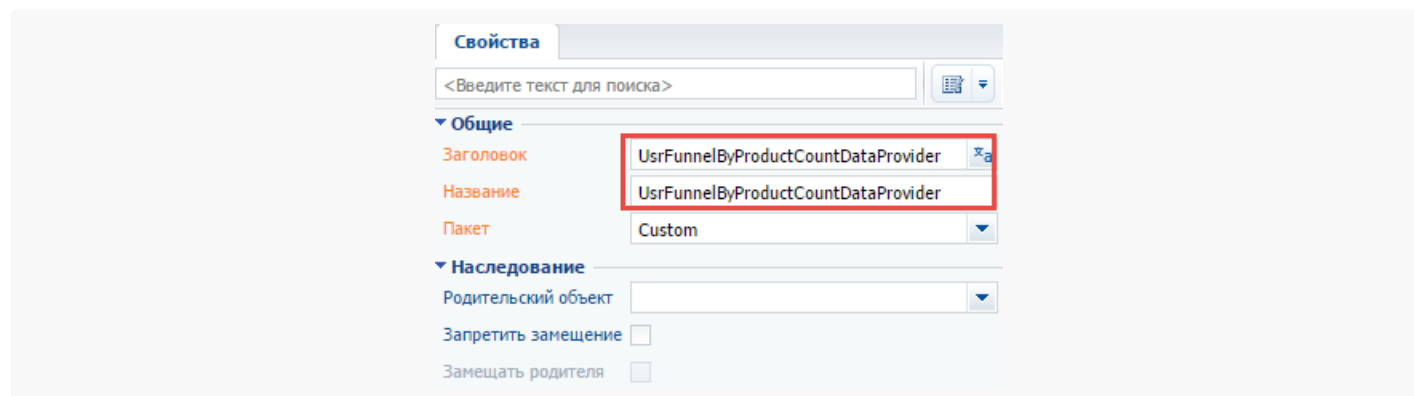
Алгоритм реализации кейса

1. В пользовательском пакете создать новый модуль

В пользовательском пакете необходимо создать новый клиентский модуль провайдера расчетов. Провайдер расчетов — это класс, который отвечает за выборку, фильтрацию и обработку данных для графика воронки.

В качестве имени и заголовка для создаваемого модуля необходимо указать, например, `UsrFunnelByProductCountDataProvider` (рис. 1).

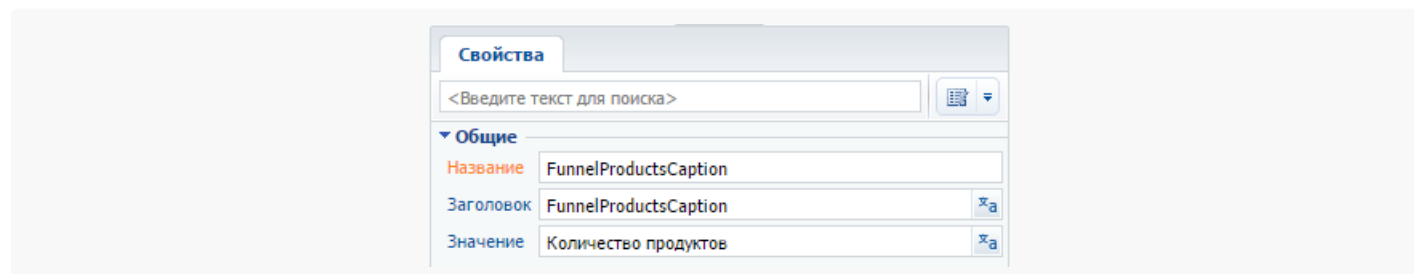
Рис. 1. — Свойства модуля провайдера расчетов



2. Добавить локализуемые строки

В коллекцию локализуемых строк созданного клиентского модуля необходимо добавить строку со значением *Количество продуктов*. Для этого, щелкнув правой клавишей мыши по узлу структуры `[ LocalizableStrings ]`, нужно из всплывающего меню выбрать команду `[ Добавить ]`. Для созданной строки нужно установить свойства так, как показано на рисунке 2.

Рис. 2. — Свойства локализуемой строки



Также аналогичным образом необходимо добавить локализуемую строку `CntOpportunity` со значением *Количество продаж*.

3. Добавить реализацию в модуль провайдера

Для изменения расчетов воронки нужно переопределить:

- метод формирования колонок `addQueryColumns` для выборки данных;

- методы обработки данных выборки.

Для обработки одной записи выборки нужно определить метод `getSeriesDataConfigByItem`. Для обработки всей коллекции нужно определить метод `prepareFunnelResponseCollection`. Для желаемой фильтрации записей нужно переопределить метод `applyFunnelPeriodFilters`.

Ниже приведен исходный код нового модуля провайдера расчетов воронки продаж.

```
define("UsrFunnelByProductCountDataProvider", ["ext-base", "terrasoft", "UsrFunnelByProductCount
    "FunnelBaseDataProvider"],
    function(Ext, Terrasoft, resources) {
        // Определение нового провайдера расчетов.
        Ext.define("Terrasoft.configuration.UsrFunnelByProductCountDataProvider", {
            // Наследование от базового провайдера.
            extend: "Terrasoft.FunnelBaseDataProvider",
            // Сокращенное имя нового провайдера
            alternateClassName: "Terrasoft.UsrFunnelByProductCountDataProvider",
            // Метод для обработки всей коллекции.
            prepareFunnelResponseCollection: function(collection) {
                this.callParent(arguments);
            },
            // Расширение метода базового модуля FunnelBaseDataProvider.
            // Устанавливает колонку количества продуктов для выборки данных.
            addQueryColumns: function(entitySchemaQuery) {
                // Вызов родительского метода.
                this.callParent(arguments);
                // Добавляет в выборку колонку количества продуктов
                entitySchemaQuery.addAggregationSchemaColumn("[OpportunityProductInterest:Opport
                    Terrasoft.AggregationType.SUM, "ProductsAmount");
            },
            // Расширение метода базового класса FunnelBaseDataProvider.
            // Устанавливает фильтрацию для выборки
            applyFunnelPeriodFilters: function(filterGroup) {
                // Вызов родительского метода.
                this.callParent(arguments);
                // Создает группу фильтров.
                var endStageFilterGroup = Terrasoft.createFilterGroup();
                // Устанавливает тип оператора для группы.
                endStageFilterGroup.logicalOperation = Terrasoft.LogicalOperatorType.OR;
                // Добавляет фильтр, который указывает, что стадия на продаже еще не окончена.
                endStageFilterGroup.addItem(
                    Terrasoft.createColumnIsNullFilter(this.getDetailColumnPath("DueDate")));
                // Добавляет фильтр, который указывает, что стадия на продаже является завершаю
                endStageFilterGroup.addItem(
                    Terrasoft.createColumnFilterWithParameter(Terrasoft.ComparisonType.EQUAL,
                        this.getDetailColumnPath("Stage.End"), true, Terrasoft.DataValueType.BOC
                filterGroup.addItem(endStageFilterGroup);
            },
            // Расширение метода базового модуля FunnelBaseDataProvider.
```

```

// Обрабатывает данные для стадий в воронке.
getSeriesDataConfigByItem: function(responseItem) {
    // Объект, хранящий локализуемые строки.
    var lcz = resources.localizableStrings;
    // Получает объект данных стадии из родительского метода.
    var config = this.callParent(arguments);
    // Получает данные о количестве продуктов в продаже из результата выборки.
    var products = responseItem.get("ProductsAmount");
    products = Ext.isNumber(products) ? products : 0;
    // Форматирует строки.
    var name = Ext.String.format("{0}<br/>{1}: {2}<br/>{3}: {4}",
        config.menuHeaderValue, lcz.CntOpportunity, config.y, lcz.FunnelProductsCapt
    var displayValue = Ext.String.format("<br/>{0}: {1}", lcz.FunnelProductsCaption,
    // Устанавливает новые данные в объект данных и возвращает его.
    return Ext.apply(config, {
        name: name,
        displayValue: displayValue
    });
}
});
});

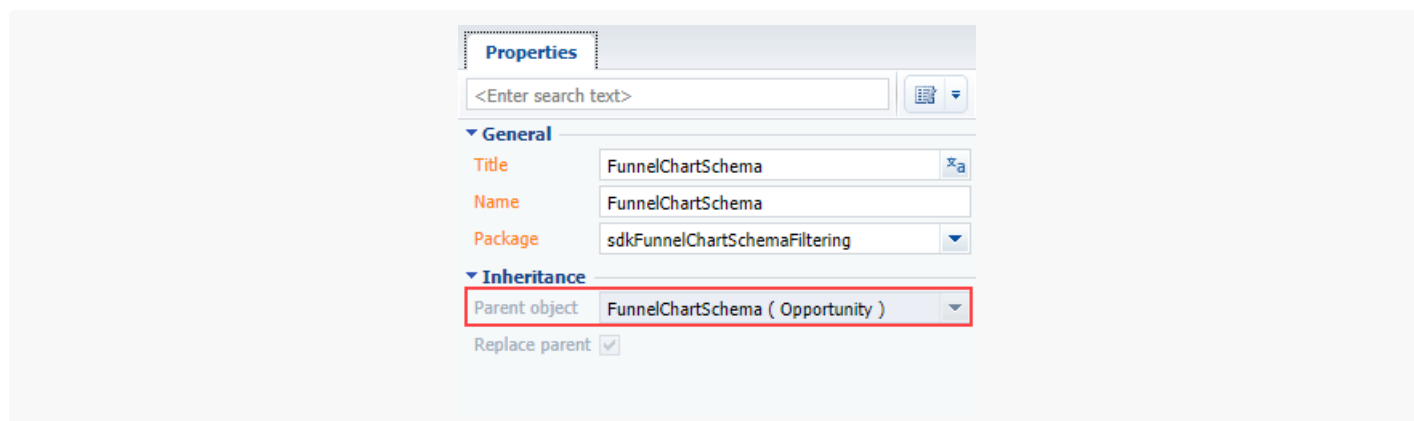
```

4. Создать схему замещающей модели представления для графика воронки

Для того, чтобы в расчетах использовался новый модуль провайдера, нужно переопределить метод формирования провайдеров расчетов воронки продаж.

Для этого необходимо создать схему замещающей модели представления и указать ей схему `FunnelChartSchema` в качестве родительского объекта (рис. 3).

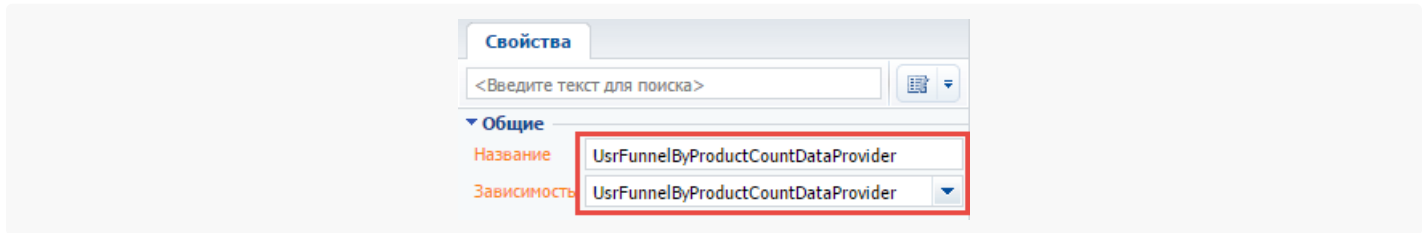
Рис. 3. — Свойства схемы замещающей модели представления



Также необходимо добавить в зависимости новый модуль для расчетов (секция `Dependencies`), указав его

имя в поля [*Зависимость*] и [*Название*] значения `UsrFunnelByProductCountDataProvider` (рис. 4).

Рис. 4. — Свойства зависимости схемы воронки



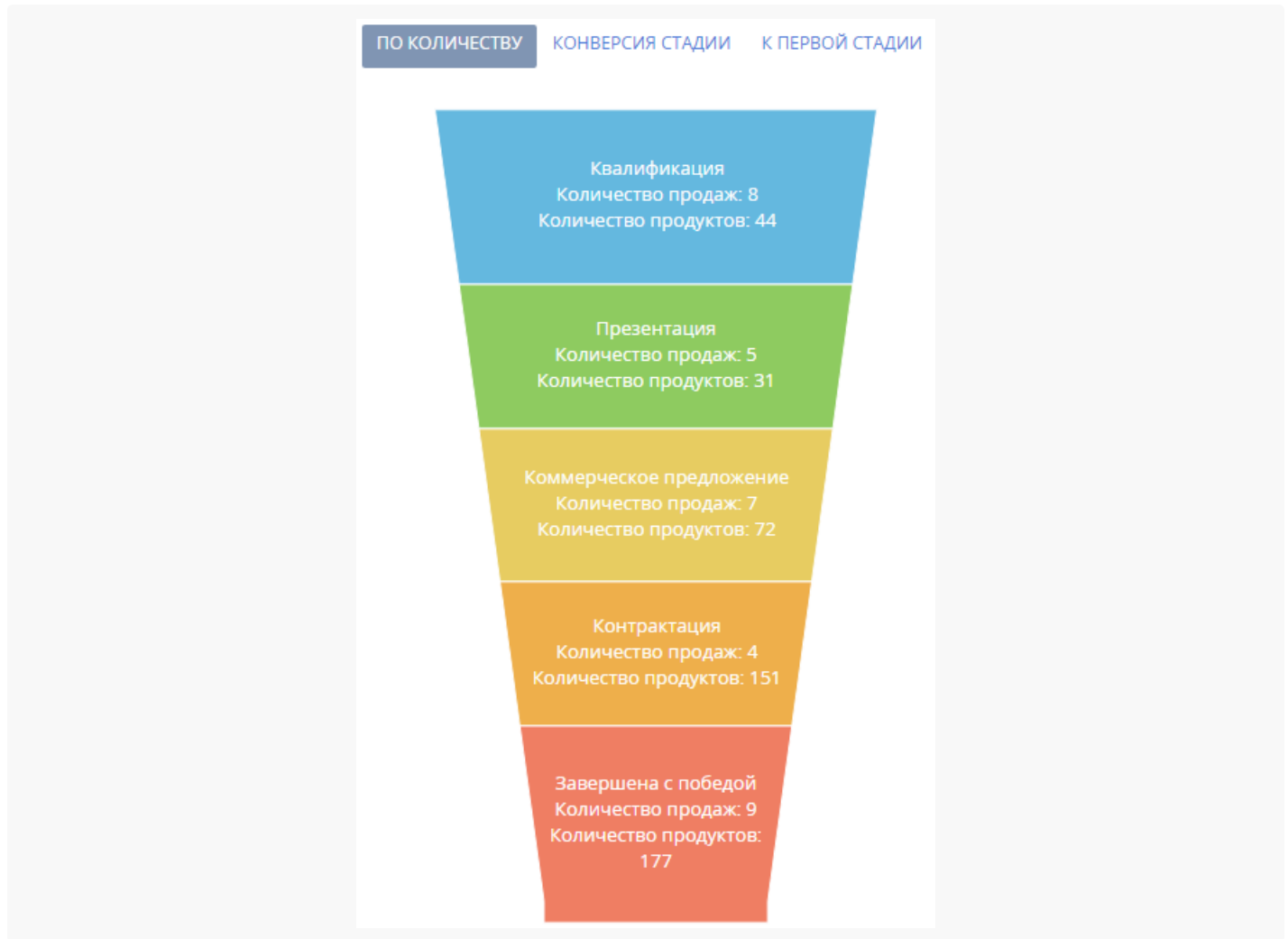
5. Указать новый провайдер расчетов в замещенной схеме воронки

Для этого в замещенной схеме необходимо переопределить метод `getProvidersCollectionConfig`, возвращающий конфигурационный объект с коллекцией провайдеров.

```
define("FunnelChartSchema", ["UsrFunnelByProductCountDataProvider"],
function() {
return {
entitySchemaName: "Opportunity",
methods: {
getProvidersCollectionConfig: function() {
// Вызывает родительский метод.
// Возвращает массив провайдеров.
var config = this.callParent();
// Ищет провайдер данных в срезе по количеству продаж.
var byCount = Terrasoft.findItem(config, {tag: "byNumberConversion"});
// Заменяет на новый класс.
byCount.item.className = "Terrasoft.UsrFunnelByProductCountDataProvider";
return config;
}
}
};
});
```

После сохранения схемы в воронке продаж будет использоваться новый модуль расчетов воронки, а воронка будет отображать общее количество продуктов по стадиям (рис. 5).

Рис. 5. — Воронка продаж с отображением количества продуктов, добавленных в продажи



Подключить дополнительную фильтрацию к воронке



Сложный

Существует возможность подключить дополнительную фильтрацию для расчетов в воронке.

Для этого необходимо реализовать следующую последовательность действий:

1. Создать унаследованный от провайдера расчетов новый класс, в котором нужно реализовать необходимую логику фильтрации.
2. Создать схему замещающей модели представления `FunnelChartSchema` и использовать в ней новый класс расчетов.

Описание примера

Необходимо добавить фильтрацию к расчетам воронки продаж в срезе "по количеству" для выбора только тех продаж, у которых в поле [*Клиент*] указан контрагент.

Исходный код

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

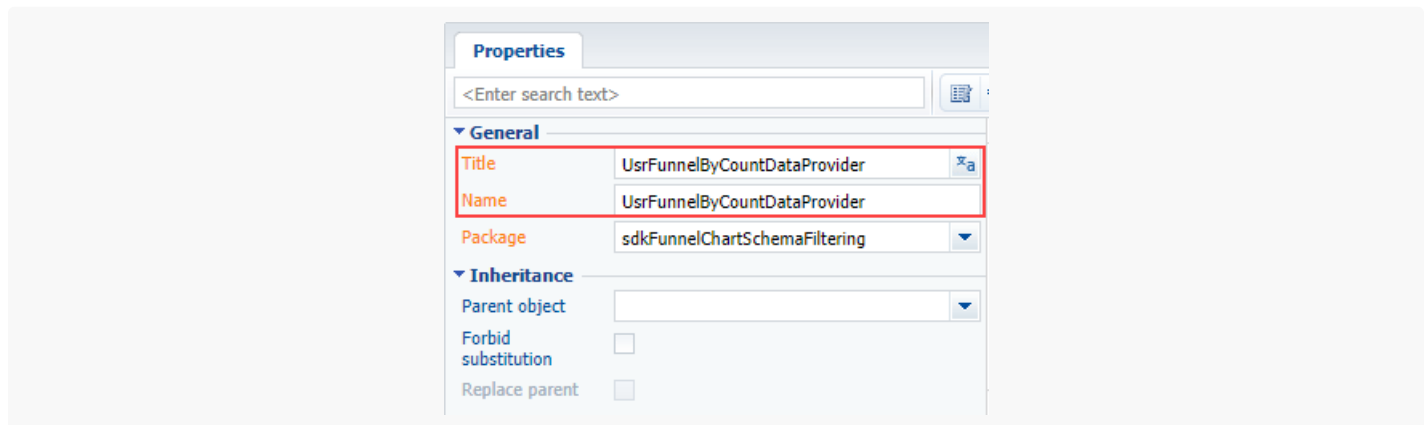
Алгоритм реализации примера

1. В пользовательском пакете создать новый модуль

В пользовательском пакете создайте новый клиентский модуль провайдера расчетов. Провайдер расчетов — это класс, который отвечает за выборку, фильтрацию и обработку данных для графика воронки.

В качестве имени и заголовка для создаваемого модуля укажите, например, `UsrFunnelByCountDataProvider` (рис. 1).

Рис. 1. — Свойства модуля провайдера расчетов



2. Определить новый класс провайдера и задать логику фильтрации

Для добавления фильтрации к расчетам в срезе “по количеству” нужно наследовать созданный класс от класса `FunnelByCountDataProvider` и переопределить метод `getFunnelFixedFilters`.

Исходный код модуля:

```
define("UsrFunnelByCountDataProvider", ["ext-base",
    "terrasoft", "UsrFunnelByCountDataProviderResources",
    "FunnelByCountDataProvider"],
function(Ext, Terrasoft, resources) {
    // Определение нового провайдера расчетов.
    Ext.define("Terrasoft.configuration.UsrFunnelByCountDataProvider", {
        // Наследование от провайдера 'по количеству'.
        extend: "Terrasoft.FunnelByCountDataProvider",
        // Сокращенное имя нового провайдера.
        alternateClassName: "Terrasoft.UsrFunnelByCountDataProvider",
        // Расширение метода базового модуля FunnelByCountDataProvider.
        // Возвращает фильтры для выборки.
```

```

    getFunnelFixedFilters: function() {
        // Вызов родительского метода.
        var esqFiltersGroup = this.callParent(arguments);
        // Добавляет фильтр, который указывает, что в продаже клиентом указан контрагент
        esqFiltersGroup.addItem(
            Terrasoft.createColumnIsNotNullFilter("Account"));
        return esqFiltersGroup;
    }
});
});

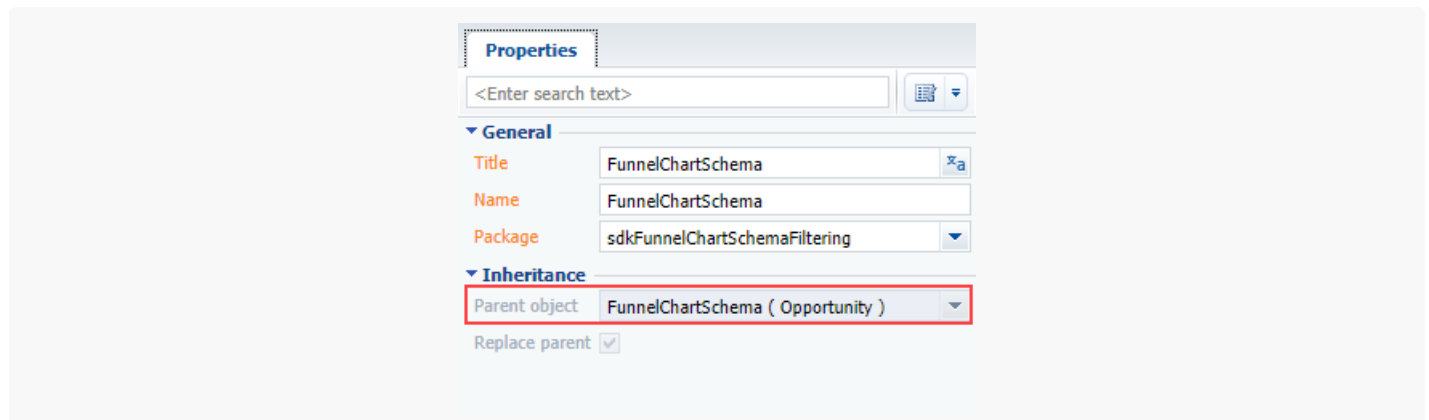
```

После внесения изменений сохраните модуль.

3. В пользовательском пакете реализовать модуль для графика воронки

Чтобы в расчетах использовался новый модуль провайдера, создайте схему замещающей модели представления, в которой в качестве родительской схемы укажите `FunnelChartSchema` из пакета `Opportunity` (рис. 2).

Рис. 2. — Свойства замещающего модуля



4. Указать новый провайдер расчетов в замещенной схеме воронки

Для этого в замещенной схеме переопределите метод формирования провайдеров расчетов воронки продаж и укажите новый класс провайдера расчетов.

Исходный код схемы замещающей модели представления:

```

define("FunnelChartSchema", ["UsrFunnelByCountDataProvider"], function() {
    return {
        entitySchemaName: "Opportunity",
        methods: {
            getProvidersCollectionConfig: function() {
                // Вызывает родительский метод, который возвращает массив провайдеров.
                var config = this.callParent();
            }
        }
    };
});

```

```

        // Ищет провайдер данных для среза по количеству.
        var byCount = Terrasoft.findItem(config, {tag: "byNumberConversion"});
        // Заменяет на новый класс.
        byCount.item.className = "Terrasoft.UsrFunnelByCountDataProvider";
        return config;
    }
}
};
});

```

После сохранения схемы в воронке продаж будет использоваться новый модуль расчетов: будут отображаться только те продажи, у которых в поле [*Клиент*] указан контрагент.

Добавить пользовательский дашборд



Дашборд (блок итогов) — визуальное представление разных типов аналитических данных, например, в виде графика или виджета. Для работы с аналитикой раздела необходимо перейти в представление аналитики необходимого раздела. Если же необходима работа с данными всех разделов приложения, то нужно перейти в раздел [*Итоги*].

Подробнее о дашбордах, реализованных в Creatio, можно узнать в блоке статей [Аналитика](#).

В Creatio предусмотрена возможность создавать пользовательские дашборды.

Для того чтобы создать пользовательский дашборд, необходимо:

1. Создать новый или выбрать существующий модуль.
2. В созданный модуль необходимо добавить исходный код, реализующий необходимую функциональность. В исходном коде модуля класс модуля дашборда должен расширять класс `BaseNestedModule` или же его наследников — `ChartModule`, `IndicatorModule`, `GaugeModule` и др.
3. Проверить наличие необходимых сообщений в соответствующих блоках. Если они отсутствуют, то добавить.
4. Указать параметры дашборда в поле [Параметры модуля] во время добавления дашборда на панель итогов.

Описание примера

Создать пользовательский дашборд, отображающий текущий курс валют.

Исходный код

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

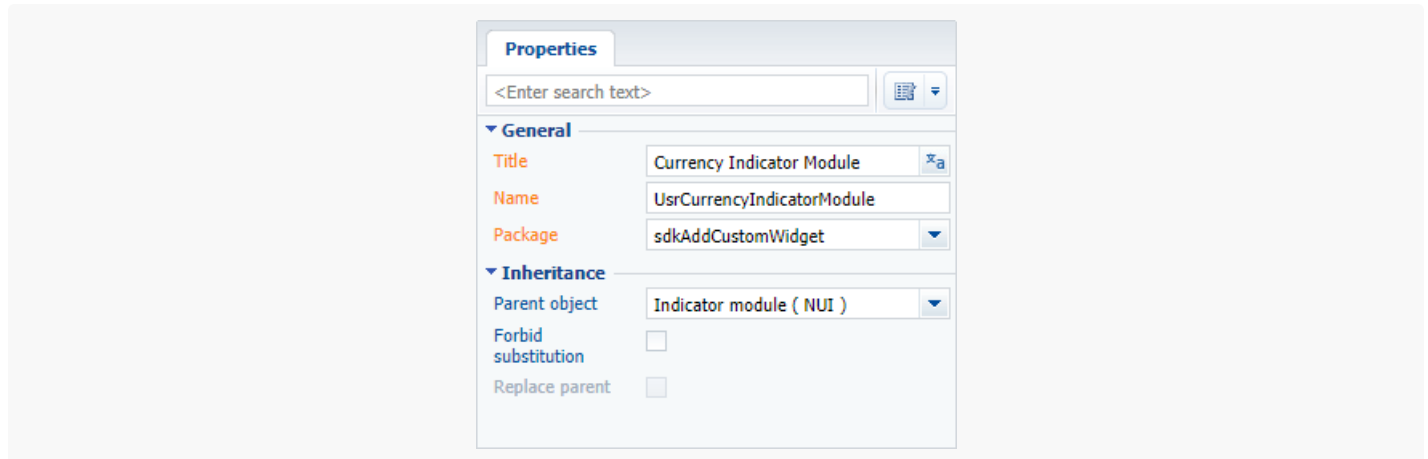
Алгоритм реализации примера

1. Создать модуль показателя валюты.

В дизайнера системы перейдите в раздел [Конфигурация] и на вкладке [Схемы] выполните команду меню [Добавить] — [Стандартные] — [Модуль]. Для создаваемого модуля укажите (рис. 1):

- [Название] ([Name]) — "UsrCurrencyIndicatorModule",
- [Заголовок] ([Title]) — "Модуль показателя валюты" ("Currency Indicator Module").

Рис. 1. — Свойства модуля показателя валюты



2. Добавить исходный код

Исходный код модуля:

```
define("UsrCurrencyIndicatorModule", ["UsrCurrencyIndicatorModuleResources", "IndicatorModule"],

// Класс, генерирующий конфигурацию представления модуля показателя валюты.
Ext.define("Terrasoft.configuration.CurrencyIndicatorViewConfig", {
    extend: "Terrasoft.BaseModel",
    alternateClassName: "Terrasoft.CurrencyIndicatorViewConfig",
    // Генерирует конфигурацию представления модуля показателя валюты.
    generate: function(config) {
        var style = config.style || "";
        var fontStyle = config.fontStyle || "";
        var wrapClassName = Ext.String.format("{0}", style);
        var id = Terrasoft.Component.generateId();
        // Возвращаемый конфигурационный объект представления.
        var result = {
            "name": id,
            "itemType": Terrasoft.ViewItemType.CONTAINER,
            "classes": {wrapClassName: [wrapClassName, "indicator-module-wrapper"]},
            "styles": {
                "display": "table",
                "width": "100%",
                "height": "100%"
            }
        },
```

```

        "items": [
            {
                "name": id + "-wrap",
                "itemType": Terrasoft.ViewItemType.CONTAINER,
                "styles": {
                    "display": "table-cell",
                    "vertical-align": "middle"
                },
                "classes": {wrapClassName: ["indicator-wrap"]},
                "items": [
                    // Отображение названия валюты.
                    {
                        "name": "indicator-caption" + id,
                        "itemType": Terrasoft.ViewItemType.LABEL,
                        "caption": {"bindTo": "CurrencyName"},
                        "classes": {"labelClass": ["indicator-caption"]}
                    },
                    // Отображение курса валюты.
                    {
                        "name": "indicator-value" + id,
                        "itemType": Terrasoft.ViewItemType.LABEL,
                        "caption": {
                            "bindTo": "CurrencyValue"
                        },
                        "classes": {"labelClass": ["indicator-value " + fontStyle]}
                    }
                ]
            }
        ]
    };
    return result;
}
});

// Класс модели представления модуля показателя валюты.
Ext.define("Terrasoft.configuration.CurrencyIndicatorViewModel", {
    extend: "Terrasoft.BaseModel",
    alternateClassName: "Terrasoft.CurrencyIndicatorViewModel",
    Ext: null,
    Terrasoft: null,
    sandbox: null,
    columns: {
        // Название валюты.
        CurrencyName: {
            type: Terrasoft.core.enums.ViewModelSchemaItem.ATTRIBUTE,
            dataValueType: Terrasoft.DataValueType.TEXT,
            value: null
        },
    },

```

```

    // Значение валюты.
    CurrencyValue: {
        type: Terrasoft.core.enums.ViewModelSchemaItem.ATTRIBUTE,
        dataValueType: Terrasoft.DataValueType.FLOAT,
        value: null
    },
    onRender: Ext.emptyFn,
    // Возвращает значение валюты в зависимости от названия. Этот метод приведен в качестве
    // Для каждой конкретной задачи следует выбрать индивидуальный способ получения данных,
    // например REST API, запрос к базе данных и т.п.
    getCurrencyValue: function(currencyName, callback, scope) {
        var result = 0;
        if (currencyName === "USD") {
            result = 26;
        }
        if (currencyName === "EUR") {
            result = 32.3;
        }
        if (currencyName === "RUB") {
            result = 0.45;
        }
        callback.call(scope || this, result);
    },
    // Получает и отображает данные на дашборде.
    prepareIndicator: function(callback, scope) {
        this.getCurrencyValue(this.get("CurrencyName"), function(currencyValue) {
            this.set("CurrencyValue", currencyValue);
            callback.call(scope);
        }, this);
    },
    // Инициализирует дашборд.
    init: function(callback, scope) {
        this.prepareIndicator(callback, scope);
    }
});

// Класс модуля дашборда.
Ext.define("Terrasoft.configuration.CurrencyIndicatorModule", {
    extend: "Terrasoft.IndicatorModule",
    alternateClassName: "Terrasoft.CurrencyIndicatorModule",
    // Название класса модели представления дашборда
    viewModelClassName: "Terrasoft.CurrencyIndicatorViewModel",
    // Название класса-генератора конфигурации представления.
    viewConfigClassName: "Terrasoft.CurrencyIndicatorViewConfig",
    // Подписка на сообщения сторонних модулей.
    subscribeMessages: function() {
        this.sandbox.subscribe("GenerateIndicator", this.onGenerateIndicator, this, [this.s
    }

```



```
});

return Terrasoft.CurrencyIndicatorModule;
});
```

3. Добавить стиль на вкладку LESS

Для того, чтобы текст дашборда отображался по центру на вкладку LESS модуля добавьте стиль:

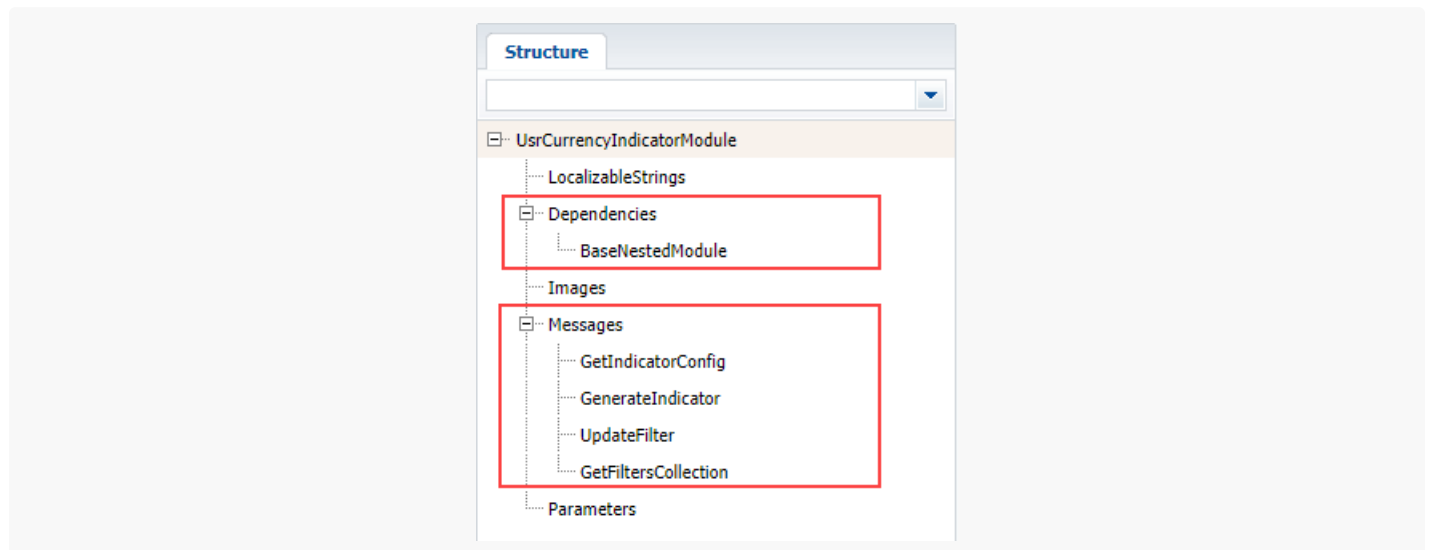
```
.indicator-module-wrapper {
    text-align: center;
}
```

4. Проверить наличие зависимостей и сообщений

В созданный модуль добавьте сообщения родительского модуля (рис. 2):

- адресное сообщение `GetIndicatorConfig`, для которого установите направление "Публикация";
- адресное сообщение `GenerateIndicator`, для которого установите направление "Подписка".

Рис. 2. — Зависимости и сообщения созданного модуля

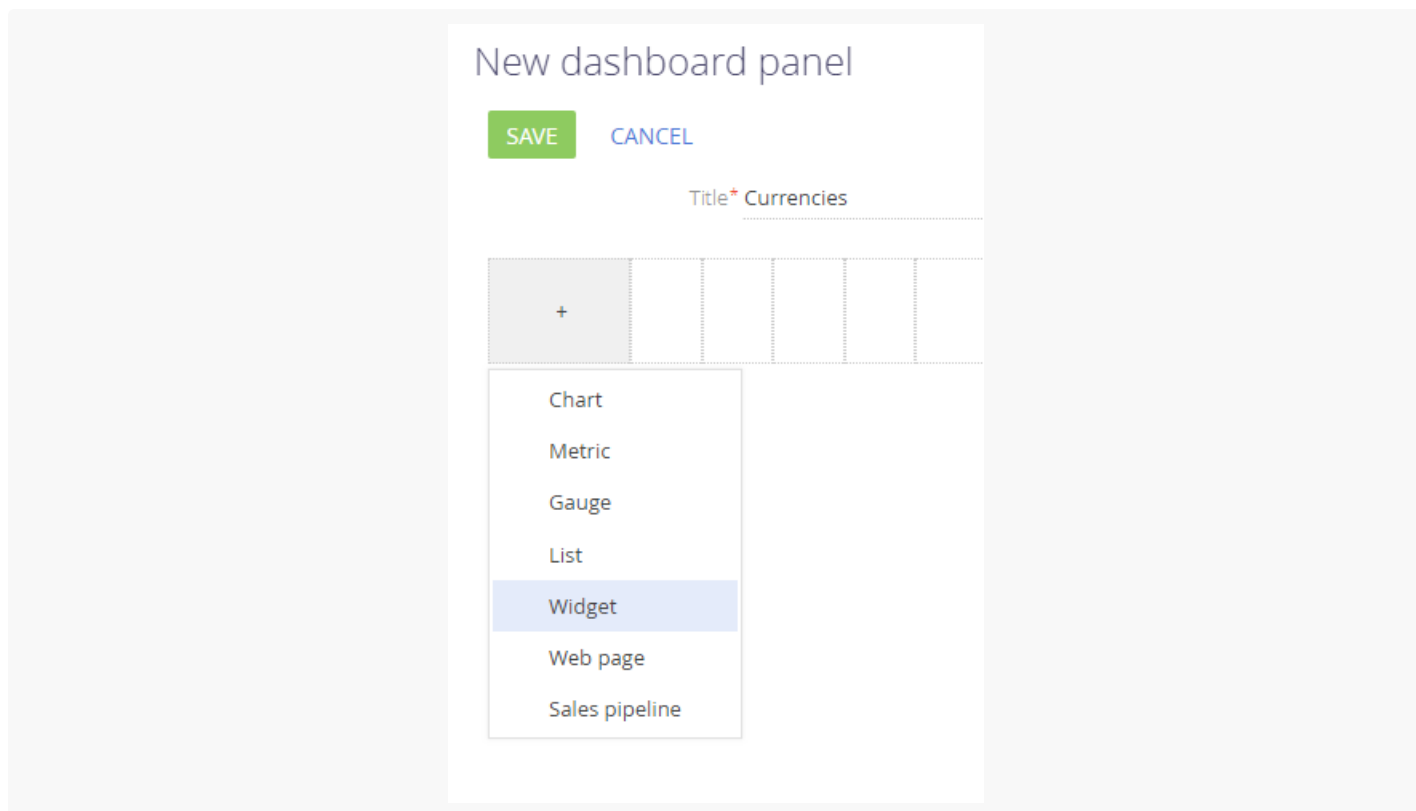


Созданный модуль сохраните.

5. Добавить дашборд на панель итогов и указать его параметры

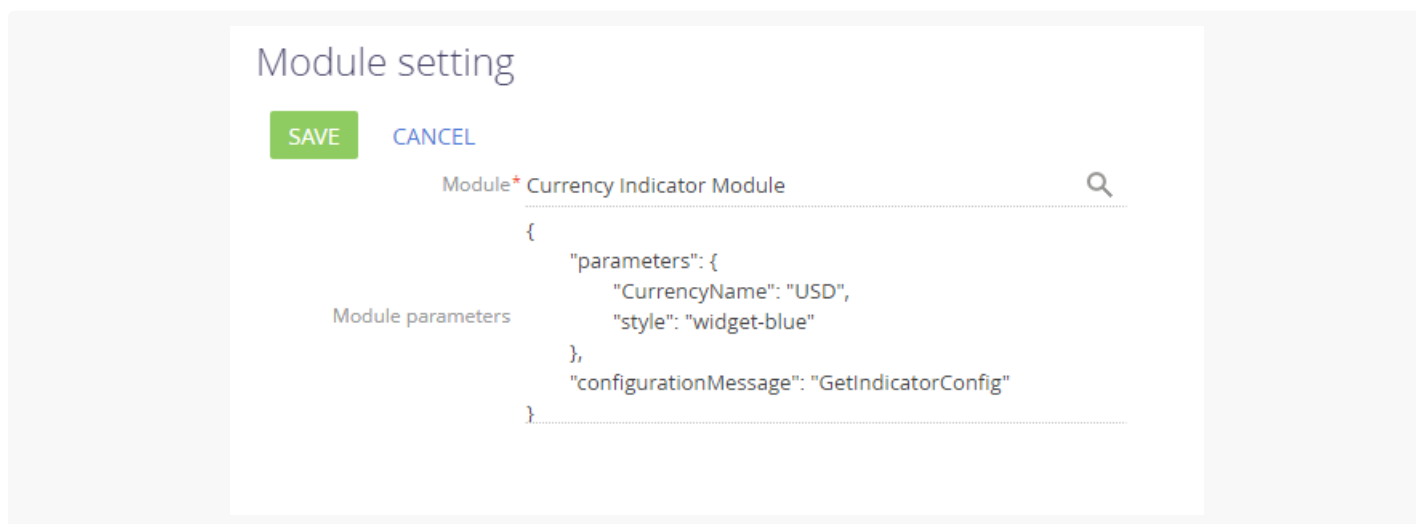
Для отображения дашборда добавьте его на панель итогов (рис. 3).

Рис. 3. — Добавление дашборда на панель итогов



Затем настройте параметры модуля, подключаемого к дашборду (рис. 4).

Рис. 4. — Настройка модуля добавляемого дашборда



Для подключения модуля к создаваемому дашборду необходимо в поле [*Модуль*] добавить значение "Модуль показателя валюты", а в поле [*Параметры модуля*] добавить конфигурационный JSON-объект с необходимыми параметрами.

```
{
  "parameters": {
    "CurrencyName": "USD",
    "style": "widget-blue"
  }
}
```

```

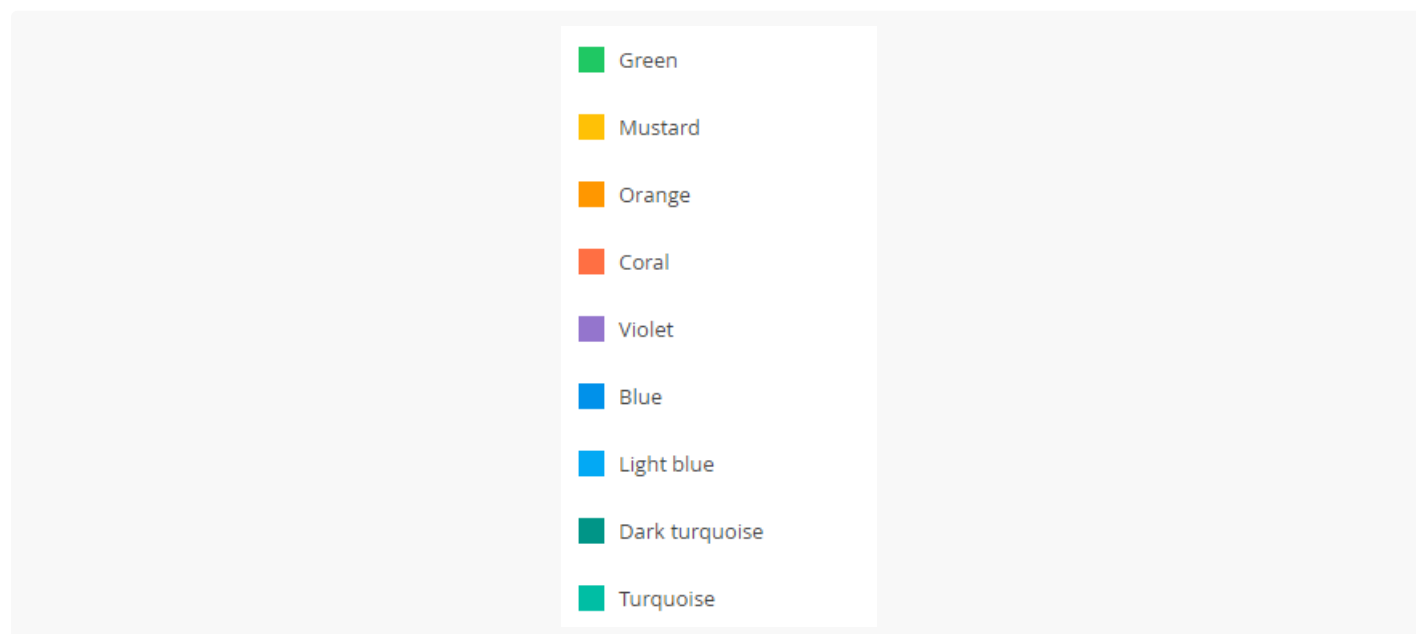
    },
    "configurationMessage": "GetIndicatorConfig"
  }

```

Параметр `CurrencyName` устанавливает значение валюты, для которой необходимо отобразить курс, параметр `style` — стиль дашборда, а параметр `configurationMessage` — название сообщения, при помощи которого будет передан конфигурационный объект.

В параметре `style` можно указать любой цвет дашборда из существующих в Creatio (рис. 5).

Рис. 5. — Варианты стилей дашборда



После сохранения создаваемого дашборда и обновления страницы на панели итогов будет отображен пользовательский дашборд (рис. 6).

Рис. 6. — Дашборд курса валют

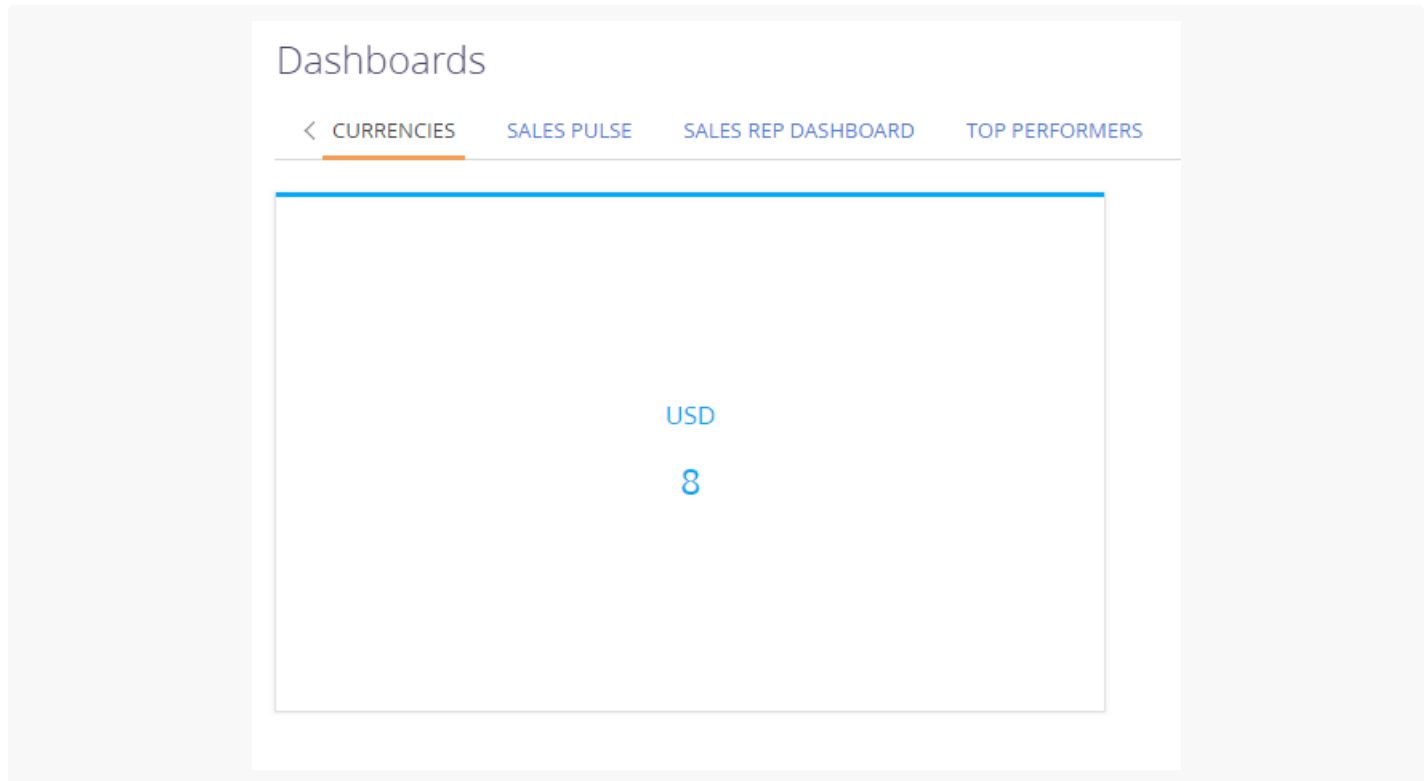


Схема BaseWidgetDesigner JS

 Сложный

`BaseWidgetDesigner` — базовая схема представления настройки дашбордов.

Методы

`getWidgetConfig()`

Возвращает объект актуальных настроек дашборда.

`getWidgetConfigMessage()`

Возвращает название сообщения получения настроек модуля дашборда.

`getWidgetModuleName()`

Возвращает название модуля дашборда.

`getWidgetRefreshMessage()`

Возвращает название сообщения обновления дашборда.

`getWidgetModulePropertiesTranslator()`

Возвращает объект соотношения свойств модуля дашборда и модуля настройки дашборда.

Перечисление DashboardEnums JS



`DashboardEnums` — содержит перечисление свойств, используемых в дашбордах.

`Terrasoft.DashboardEnums.WidgetType` — содержит конфигурацию дашбордов для режима просмотра (view) и режима настройки (design) итогов.

Свойства

`moduleName`

Название модуля дашборда.

`configurationMessage`

Название сообщения получения настроек модуля.

`resultMessage`

Название сообщения для отдачи параметров настройки модуля дизайнера дашборда.

`stateConfig (stateObj)`

Название схемы дизайнера дашборда.

Хронология



Начиная с версии 7.12.0 для быстрого анализа истории работы с клиентами, продажами, обращениями и т.п. используется вкладка [*Хронология*]. Эта вкладка по умолчанию доступна в разделах [*Контакты*], [*Контрагенты*], [*Лиды*], [*Продажи*], [*Обращения*].

Таблицы базы данных

Для настройки хронологии в базе данных предусмотрены следующие [таблицы](#):

- `TimelinePageSetting` — для настройки разделов и их плиток.

- `TimelineTileSetting` — для настройки всех преднастроенных и пользовательских плиток хронологии.
- `SysTimelineTileSettingLcz` — для локализации имен плиток.

Основные колонки таблицы `TimelinePageSetting`

Колонка	Описание
Id	Идентификатор записи.
Key	Ключ — название схемы страницы раздела. Например, <code>AccountPageV2</code> , <code>ContactPageV2</code> и т. д.
Data	Настройки хронологии для раздела в формате JSON.

Основные колонки таблицы `TimelineTileSetting`

Колонка	Описание
Id	Идентификатор записи.
Name	Заголовок плитки, который будет отображаться в меню фильтра. Должен быть во множественном числе, например "Задачи" ("Tasks"). Локализация осуществляется с помощью таблицы <code>SysTimelineTileSettingLcz</code> . Если данное поле не будет указано, тогда заголовок плитки будет взят из названия схемы сущности или типа.
Data	Настройки хронологии для раздела в формате JSON.
<code>Image</code>	Иконка плитки, которая будет отображаться в меню фильтра и слева от плитки во вкладке [<i>Хронология</i>].

Параметры конфигурации плитки хронологии в формате JSON

Колонка	Описание	Обязательность	Пример
<code>entityConfigKey</code>	Ключ плитки. Должен совпадать с Id в таблице <code>TimelineTileSetting</code> соответствующей преднастроенной плитки, которую следует отображать для данной сущности.	Да	706f803d-6a30-4bcd·
<code>entitySchemaName</code>	Название схемы объекта сущности.	Да	Activity

<code>referenceColumnName</code>	Название колонки объекта, по которой будет происходить отбор записей.	Да	Account
<code>masterRecordColumnName</code>	Название колонки родительской записи, по которой будет происходить отбор записей.	Да	Id
<code>typeColumnName</code>	Название колонки типа.	Нет	Type
<code>typeColumnValue</code>	Значение колонки типа.	Указывается только при указании <code>typeColumnName</code>	<code>fbe0acdc-cfc0-df11</code>
<code>viewModelClassName</code>	Название класса модели представления преднастроенной плитки.	Нет. Если значение отсутствует, то будет применен базовый класс <code>BaseTimelineItemViewModel</code>	<code>Terrasoft.Activity</code>
<code>viewClassName</code>	Название класса представления преднастроенной плитки.	Нет. Если значение отсутствует, то будет применен базовый класс <code>BaseTimelineItemView</code>	<code>Terrasoft.Activity</code>
<code>orderColumn</code>	Колонка для сортировки.	Да	StartDate
<code>authorColumnName</code>	Колонка для автора.	Да	Owner
<code>captionColumnName</code>	Колонка для заголовка.	Да, если не указана колонка <code>messageColumnName</code>	Title
<code>messageColumnName</code>	Колонка для информационного сообщения.	Да, если не указана колонка <code>captionColumnName</code>	DetailedResult
<code>caption</code>	Заголовок плитки, который будет отображаться в меню фильтра.	Нет	My Activity

	Должен быть во множественном числе, например "Задачи" ("Tasks"). Используется для задания заголовка плитки, отличного от указанного в поле Name настройки соответствующей плитки в <code>TimelinePageSetting</code> .		
<code>columns</code>	Массив настроек дополнительных колонок для плитки.	Нет	
<code>columnName</code>	Путь к колонке в объекте сущности.	Да	Result
<code>columnAlias</code>	Псевдоним колонки в представлении модели плитки.	Да	ResultMessage
<code>isSearchEnabled</code>	Указывает на возможность текстового поиска по значению в колонке (только для текстовых колонок).	Нет	true

Добавление вкладки [*Хронология*] в раздел

Для [добавления вкладки \[*Хронология* \]](#) на страницу раздела и отображения в нем записей определенных плиток, необходимо:

1. Создать новую запись в таблице `TimelinePageSetting`.
2. Заполнить соответствующие колонки. В колонке `key` необходимо указать название схемы страницы раздела. Например, если необходимо добавить вкладку в раздел [*Контрагенты*], то значением колонки `key` будет "AccountPageV2". Колонка `Data` содержит конфигурацию плиток хронологии, отображаемых на вкладке в указанном разделе, в формате JSON.

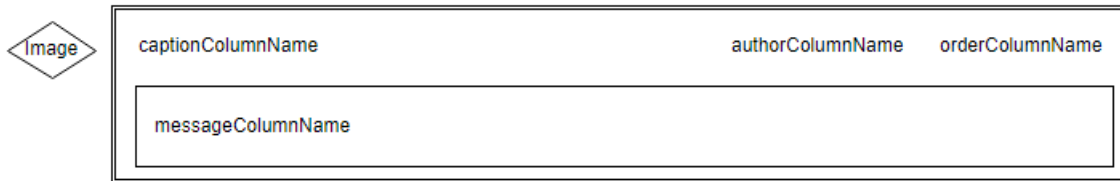
Важно. Вкладка [*Хронология*] не будет отображаться на странице записи раздела, если отсутствует конфигурация плиток в колонке `Data` или если же есть ошибки (например, синтаксические ошибка) в конфигурации.

Пример использования базовой плитки



Чтобы начать использовать хронологию в разделе, достаточно выполнить конфигурацию базовой плитки. Составные элементы базовой плитки:

- иконка;
- заголовок;
- автор;
- дата (сортировка);
- сообщение.



Пример. Добавить плитку [*Договор*] ([*Contract*]) на страницу раздела [*Заказы*] ([*Orders*]). Сортировка должна осуществляться по колонке `StartDate`, значения для заголовка, автора и сообщения плитки брать из колонок `Number`, `Owner` и `Notes` соответственно.

Алгоритм реализации примера

1. Создайте новую запись (или обновите существующую) в таблице [*TimelinePageSetting*].
2. Для колонки `key` установите значение "OrderPageV2", а в колонку `Data` запишите JSON-объект.

JSON-объект

```
[
  {
    "entityConfigKey": "0ef5bd15-f3d3-4673-8af7-f2e61bc44cf0",
    "entitySchemaName": "Contract",
    "referenceColumnName": "Order",
    "orderColumnName": "StartDate",
```

```

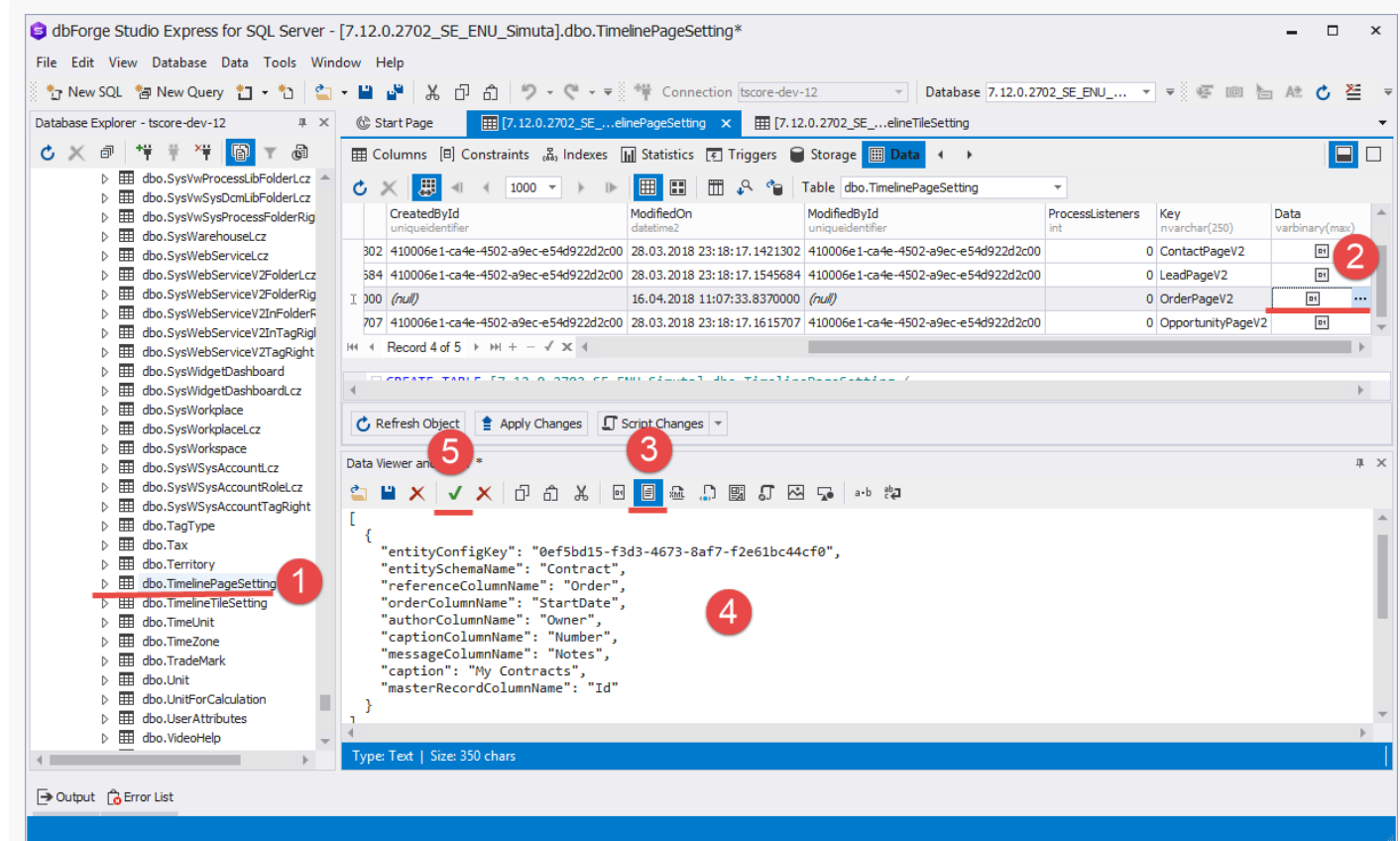
    "authorColumnName": "Owner",
    "captionColumnName": "Number",
    "messageColumnName": "Notes",
    "caption": "My Contracts",
    "masterRecordColumnName": "Id"
  }
]

```

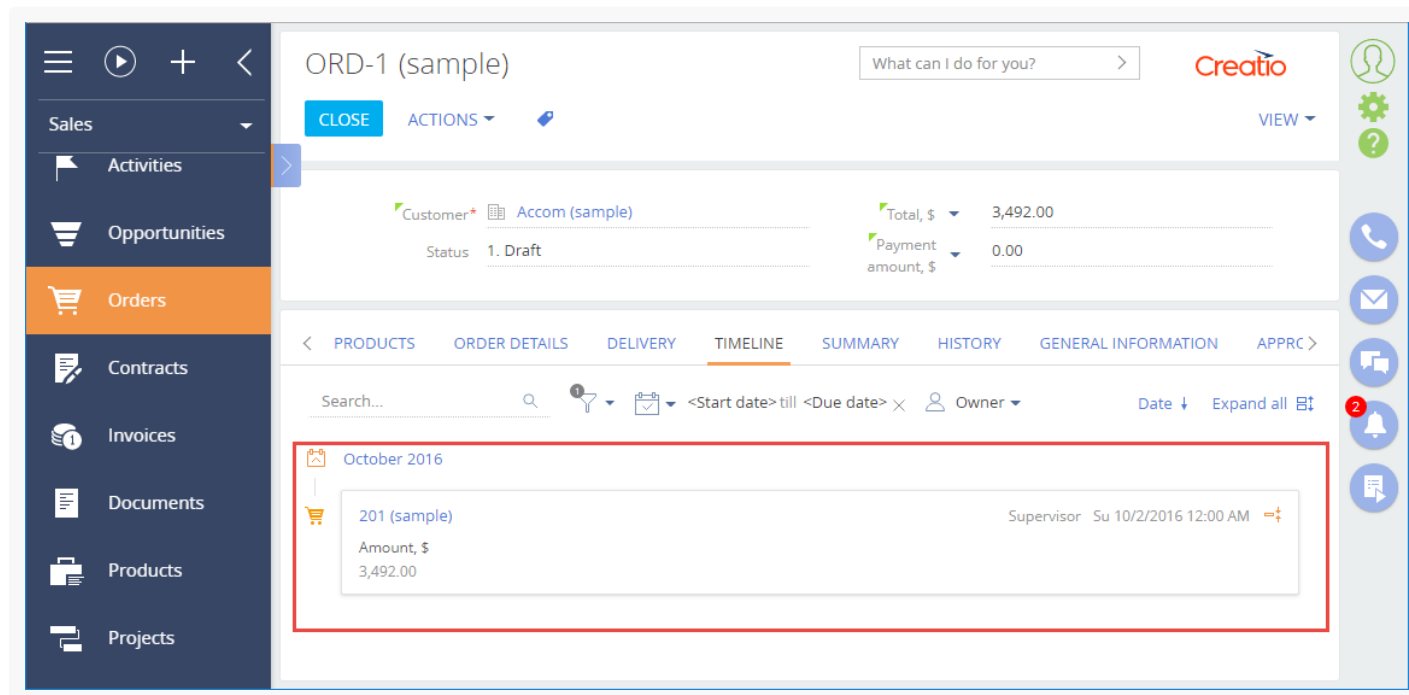
В этом примере используется базовая плитка [*Orders*]. Для нее в таблице [*TimelineTileSettings*] уже существует запись с идентификатором "0ef5bd15-f3d3-4673-8af7-f2e61bc44cf0".

Важно. Поскольку данные в колонке [*Data*] хранятся в формате `varbinary(max)`, то редактировать их удобнее всего с помощью специализированных редакторов, например, dbForge Studio Express for SQL Server. Для этого:

1. Выберите нужную таблицу.
2. Выберите требуемую колонку записи и кликните по кнопке редактирования.
3. В редакторе данных перейдите в режим текстового отображения данных.
4. Добавьте нужные данные.
5. В редакторе данных нажмите на кнопку применения изменений.



Результат использования базовой плитки на вкладке [Хронология] ([Timeline]) в разделе [Заказы]:



Создать плитки вкладки [Хронология], связанные с пользовательским разделом

 Сложный

Для быстрого анализа истории работы с клиентами, продажами, обращениями и другими сущностями. используется вкладка [Хронология]. Эта вкладка по умолчанию доступна в разделах [Контакты], [Контрагенты], [Лиды], [Продажи], [Обращения].

Настройка плитки

Плитку можно настроить используя настройки в таблице [TimelinePageSetting], как показано в [примере с базовой плиткой](#). В таком случае для плитки будут использованы:

- иконка по умолчанию;
- базовые модули представления и модели представления — `BaseTimelineItemView` и `BaseTimelineItemViewModel`;
- поле автора;
- поле заголовка плитки;
- поле сообщения.

При необходимости можно использовать одну и ту же плитку для разных разделов. Однако лучше

воспользоваться таблицей [TimelineTileSetting] и выполнить настройку необходимых плиток для разных разделов.

Таблица [TimelineTileSetting] содержит конфигурации всех преднастроенных плиток в системе. Однако в разделе будут отображаться только те плитки, которые указаны для него в таблице [TimelinePageSetting].

Например, в таблице [TimelineTileSetting] сконфигурированы три преднастроенные плитки: Tasks, Leads и Calls. В таблице [TimelinePageSetting] для раздела [Контрагенты] настроено использование плиток Tasks и Calls, а для раздела [Контакты] — только Calls. Плитка Leads в таком случае нигде отображаться не будет.

На заметку. Хорошей практикой считается разделение настроек плитки. Непосредственную настройку плитки рекомендуется выполнять только в таблице [TimelineTileSetting]. А в таблице [TimelinePageSetting] рекомендуется настраивать добавление плитки в хронологию раздела.

Важно. Если необходимо добавить в раздел преднастроенную плитку Files, то в таблице [TimelinePageSetting] в конфигурации для соответствующего раздела свойство "entitySchemaName" должно содержать имя схемы сущности для файлов (например, AccountFile, ContactFile и т. п.). В то же время название схемы объекта (свойство "entitySchemaName") в таблице [TimelineTileSetting] должно быть всегда "##ReferenceSchemaName##File".

Алгоритм создания преднастроенной плитки

1. Создайте новый раздел (при необходимости).
2. В пользовательском пакете создайте схему модуля, в которой определите класс представления плиток, связанных с новым разделом. Класс должен быть наследником BaseTimelineItemView.
3. В пользовательском пакете создайте схему модуля, в которой определите класс модели представления плиток, связанных с новым разделом. Класс должен быть наследником BaseTimelineItemViewModel.
4. В таблице [TimelineTileSetting] базы данных создайте запись с настройками отображения плитки, связанной с новым разделом.
5. В таблице [TimelinePageSetting] создайте или измените запись, включающую отображение плиток на вкладке [Хронология] в требуемом разделе.

Пример. На вкладке [Хронология] записей раздела [Контрагенты] отображать плитки, связанные с пользовательским разделом [Книги] ([Books]). Плитки должны содержать:

- иконку;
- название;
- автора;
- дату добавления записи о книге;

- СТОИМОСТЬ;
- ISBN номер.

Также при разворачивании плитки должно отображаться краткое описание книги.

Важно. Для реализации примера необходимо внести изменения в таблицы базы данных (см. шаги 4 и 5).

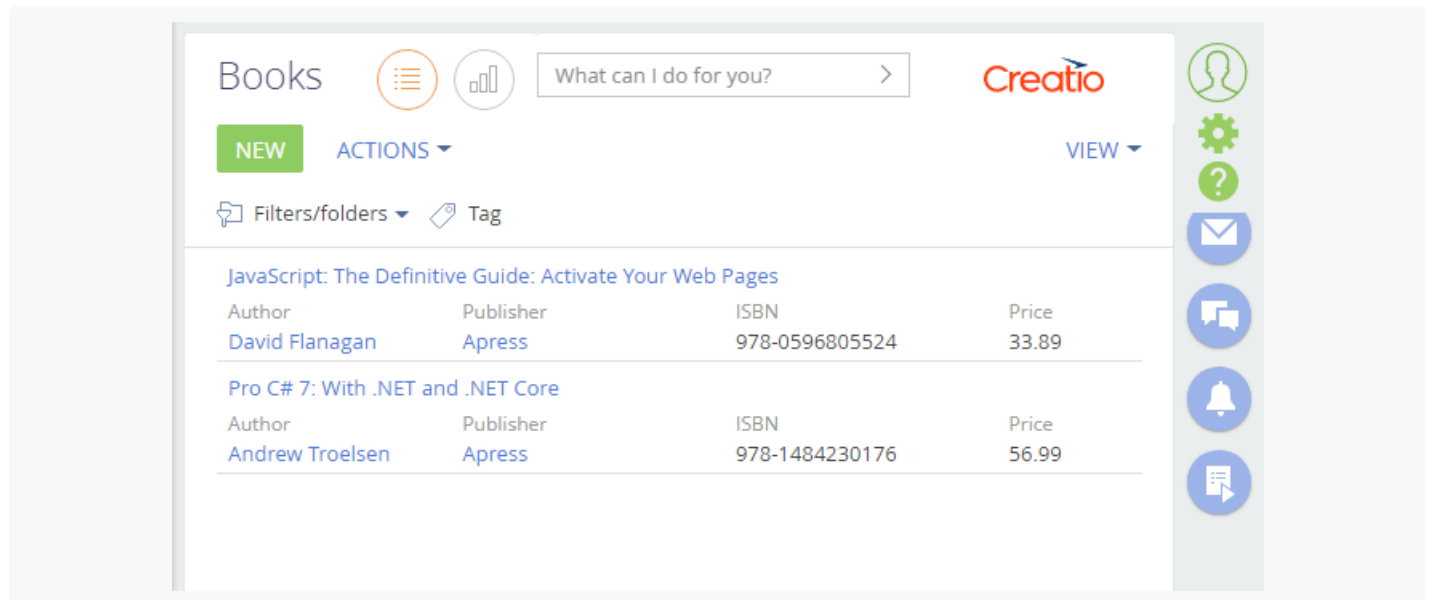
Алгоритм реализации примера

1. Создать новый раздел [Книги]

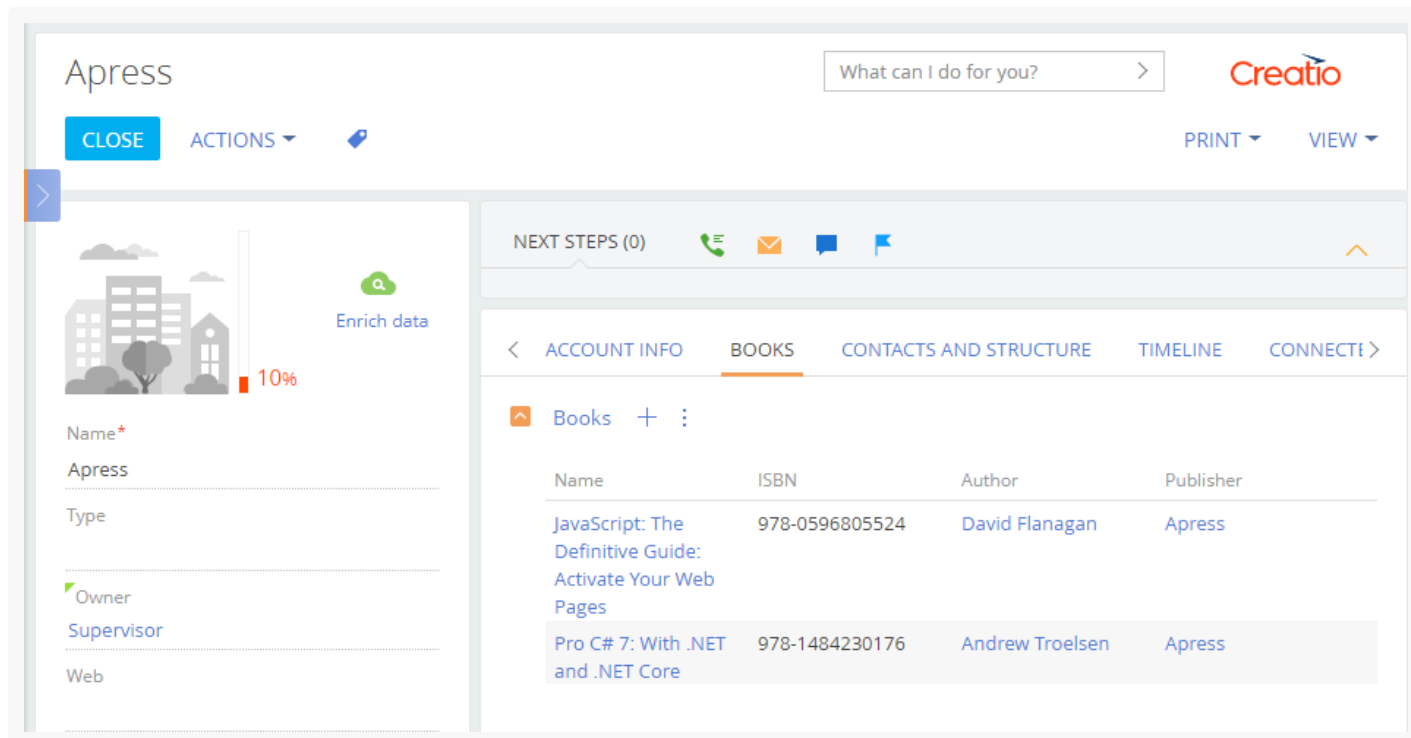
Для создания нового раздела [Книги] воспользуйтесь архивом, содержащим пакет со всей необходимой функциональностью. Установите пакет с помощью функциональности [установки приложений marketplace из *.zip-архива](#).

На заметку. Вы можете создать раздел самостоятельно, используя мастер разделов.

После установки пакета в рабочем месте [Общее] ([General]) будет доступен раздел [Книги] ([Books]).



Также на странице записи раздела [Контрагенты] на вкладке [Books] появится деталь, отображающие связанные записи из раздела [Книги].

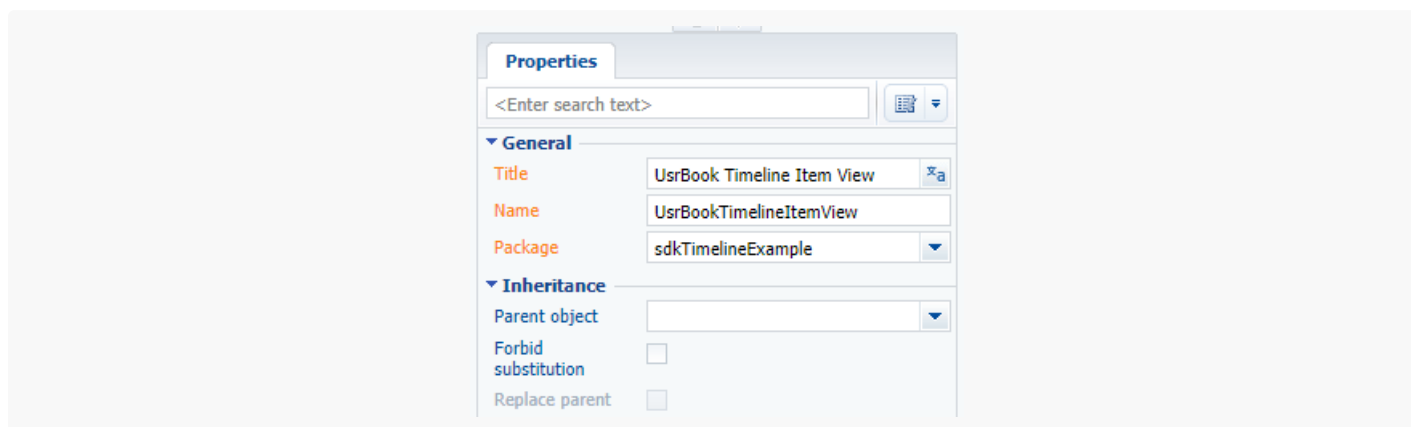


2. Создать модуль представления плитки

В пользовательском пакете, имеющем зависимость от пакета `Timeline`, создайте [схему клиентского модуля](#).

Для созданной схемы модуля укажите:

- [*Название*] ([*Name*]) — "UsrBookTimelineItemView";
- [*Заголовок*] ([*Title*]) — "Представление элемента хронологии UsrBook" ("UsrBook Timeline Item View").



На вкладку [*Исходный код*] ([*Source Code*]) схемы добавьте исходный код модуля.

UsrBookTimelineItemView

```
// Определение модуля и его зависимостей.
define("UsrBookTimelineItemView", ["UsrBookTimelineItemViewResources", "BaseTimelineItemView"],
```

```

// Определение класса представления плитки.
Ext.define("Terrasoft.configuration.UsrBookTimelineItemView", {
    extend: "Terrasoft.BaseTimelineItemView",
    alternateClassName: "Terrasoft.UsrBookTimelineItemView",
    // Метод, возвращающий конфигурацию дополнительного поля [UsrISBN] плитки.
    getUsrISBNViewConfig: function() {
        return {
            // Название поля.
            "name": "UsrISBN",
            // Тип поля – метка.
            "itemType": Terrasoft.ViewItemType.LABEL,
            // Заголовок.
            "caption": {
                "bindTo": "UsrISBN"
            },
            // Видимость.
            "visible": {
                // Привязка к колонке связанной с плиткой сущности.
                "bindTo": "UsrISBN",
                // Настройка видимости.
                "bindConfig": {
                    // Поле видимо, если значение в колонке не пустое.
                    "converter": "checkIsNotEmpty"
                }
            },
            // CSS-стили поля.
            "classes": {
                "labelClass": ["timeline-text-light"]
            }
        };
    },
    // Метод, возвращающий конфигурацию дополнительного поля [UsrPrice] плитки.
    getUsrPriceViewConfig: function() {
        return {
            "name": "UsrPrice",
            "itemType": Terrasoft.ViewItemType.LABEL,
            "caption": {
                "bindTo": "UsrPrice"
            },
            "visible": {
                "bindTo": "UsrPrice",
                "bindConfig": {
                    "converter": "checkIsNotEmpty"
                }
            },
            "classes": {
                "labelClass": ["timeline-item-subject-label"]
            }
        }
    }
});

```

```

    };
  },
  // Переопределенный метод, возвращающий конфигурацию поля [Message] плитки.
  getMessageViewConfig: function() {
    // Получение стандартных настроек.
    var config = this.callParent(arguments);
    // Настройка видимости. Видимо, если плитка развернута.
    config.visible = {
      "bindTo": "IsExpanded"
    };
    return config;
  },
  // Переопределенный метод, возвращающий общую конфигурацию плитки.
  getBodyViewConfig: function() {
    // Получение стандартных настроек.
    var bodyConfig = this.callParent(arguments);
    // Добавление конфигураций дополнительных полей.
    bodyConfig.items.unshift(this.getUsrISBNViewConfig());
    bodyConfig.items.unshift(this.getUsrPriceViewConfig());
    return bodyConfig;
  }
});
});

```

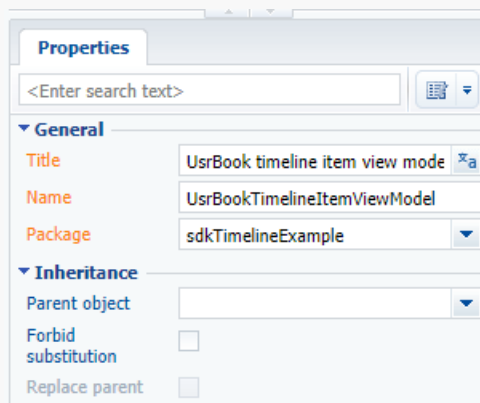
Здесь определяется конфигурация дополнительно отображаемых на плитке полей [*UsrISBN*] и [*UsrPrice*]. Стандартная конфигурация определена в модуле `BaseTimelineItemView`.

3. Создать модуль модели представления плитки

В пользовательском пакете, имеющем зависимость от пакета `Timeline`, создайте [схему клиентского модуля](#).

Для созданной схемы модуля укажите:

- [*Название*] ([*Name*]) — "UsrBookTimelineItemViewModel";
- [*Заголовок*] ([*Title*]) — "Модель представления элемента хронологии UsrBook" ("UsrBook timeline item view model").



На вкладку [*Исходный код*] ([*Source Code*]) схемы добавьте исходный код модуля.

UsrBookTimelineItemViewModel

```
define("UsrBookTimelineItemViewModel", ["UsrBookTimelineItemViewModelResources", "BaseTimelineItemViewModel"], function() {
    Ext.define("Terrasoft.configuration.UsrBookTimelineItemViewModel", {
        alternateClassName: "Terrasoft.UsrBookTimelineItemViewModel",
        extend: "Terrasoft.BaseTimelineItemViewModel"
    });
});
```

Здесь определяется класс `Terrasoft.configuration.UsrBookTimelineItemViewModel`. Поскольку этот класс определен как наследник `Terrasoft.BaseTimelineItemViewModel`, то это позволяет пользоваться всей функциональностью базового класса.

4. В таблице [TimelineTileSetting] создать запись с настройками отображения плитки

Таблица [*TimelineTileSetting*] предназначена для настройки [свойств плиток хронологии](#).

Создайте новую запись в таблице [TimelineTileSetting]. Запись можно создать с помощью SQL-запроса.

SQL-запрос

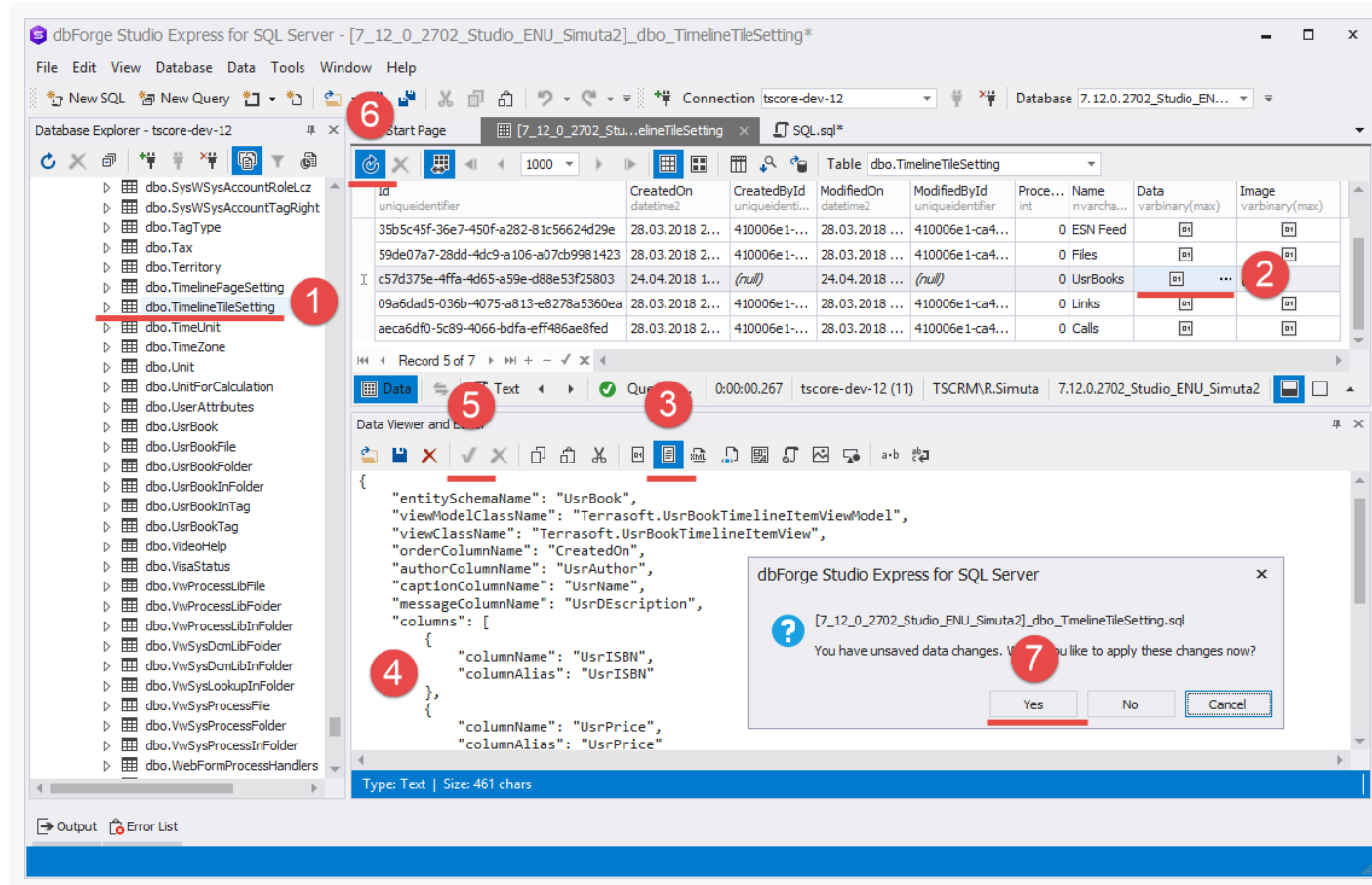
```
INSERT INTO TimelineTileSetting (CreatedOn, CreatedById, ModifiedOn, ModifiedById, Name, Data, Image)
VALUES (GETUTCDATE(), NULL, GETUTCDATE(), NULL, 'UsrBooks', NULL, NULL);
```

Поскольку данные в колонках [*Data*] и [*Image*] хранятся в формате `varbinary(max)`, то редактировать их удобнее всего с помощью специализированных редакторов, например, dbForge Studio Express for SQL Server. Для этого:

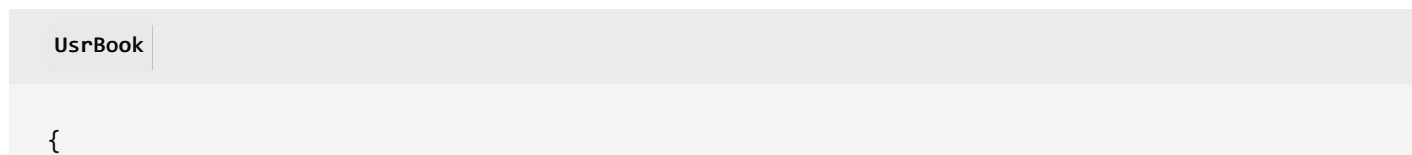
1. Выберите нужную таблицу.

2. Выберите требуемую колонку записи и кликните по кнопке редактирования.
3. В редакторе данных перейдите в режим текстового отображения данных.
4. Добавьте нужные данные.
5. В редакторе данных нажмите на кнопку применения изменений.
6. Нажмите на кнопку обновления данных.
7. В появившемся диалоговом окне согласитесь с применением изменений.

Важно. Это способ подходит только для сред разработки, развернутых on-site. Поскольку изменения вносятся непосредственно в базу данных, то они не привязаны ни к одному пакету. Поэтому при установке пакета со схемами представления и модели представления плитки в другое приложение изменения в базу данных внесены не будут. Для корректного переноса разработанной функциональности следует привязать SQL-скрипты, вносящие соответствующие изменения в базу данных при установке пакета.



Используя приведенную выше последовательность, добавьте в колонку [Data] конфигурационный объект.



```

"entitySchemaName": "UsrBook",
"viewModelClassName": "Terrasoft.UsrBookTimelineItemViewModel",
"viewClassName": "Terrasoft.UsrBookTimelineItemView",
"orderColumnName": "CreatedOn",
"authorColumnName": "UsrAuthor",
"captionColumnName": "UsrName",
"messageColumnName": "UsrDescription",
"columns": [
  {
    "columnName": "UsrISBN",
    "columnAlias": "UsrISBN"
  },
  {
    "columnName": "UsrPrice",
    "columnAlias": "UsrPrice"
  }
]
}

```

Здесь, кроме основных полей, унаследованных от базовой плитки, указывается также массив дополнительных полей, отображение которых сконфигурировано в модуле представления

`UsrBookTimelineItemView` (см. шаг 2).

Для отображения иконки, соответствующей иконке раздела, добавьте в колонку [*Image*] данные в SVG-формате.

Данные колонки [*Image*]

```

<svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 52 52" enable-background="new 0 0 52 52">
<path d="M46.072,31.384c-0.011-0.026-0.025-0.048-0.039-0.073c-0.036-0.064-0.077-0.125-0.123-0.18
c-0.018-0.022-0.034-0.044-0.053-0.064c-0.034-0.036-0.068-0.07-0.105-0.104c-0.062-0.055-0.431
c-1.958-1.307-7.465-4.978-9.424-6.284c-0.388-0.258-0.703-0.845-0.703-1.312V3.938c0-0.401-0.1
c-0.322-0.239-0.739-0.311-1.122-0.193L15.015,8.254c-0.446,0.136-1.154,0.097-1.583-0.086l-1.0
c-0.428-0.184-0.414-0.442,0.031-0.578l15.213-4.646c0.668-0.204,1.045-0.911,0.841-1.58s-0.912
C7.454,5.982,7.429,5.994,7.403,6.005C7.338,6.031,7.276,6.062,7.217,6.097C7.205,6.104,7.191,6
c-0.015,0.01-0.026,0.025-0.041,0.035C7.081,6.191,7.03,6.236,6.982,6.284c-0.02,0.021-0.041,0.
C6.864,6.412,6.813,6.485,6.77,6.562C6.716,6.659,6.683,6.748,6.658,6.838C6.651,6.864,6.648,6.
C6.628,6.985,6.619,7.054,6.616,7.125C6.615,7.142,6.61,7.156,6.61,7.173V29.85c0,0.466-0.036,0
c-0.109,0.058-0.18,0.101-0.246,0.15c-0.025,0.018-0.046,0.037-0.069,0.058c-0.056,0.049-0.107,
c-0.015,0.019-0.032,0.035-0.046,0.056c-0.057,0.079-0.105,0.164-0.142,0.257c-0.006,0.015-0.00
c-0.029,0.077-0.049,0.158-0.062,0.241c-0.002,0.015-0.009,0.027-0.01,0.042c-0.002,0.018,0.002
c-0.003,0.031-0.009,0.062-0.009,0.094v7.312c0,0.393,0.182,0.762,0.493,1.002l14.766,11.391c0.
c0.212,0,0.424-0.053,0.616-0.16l23.203-12.938c0.401-0.224,0.649-0.646,0.649-1.105v-5.766c0-0
C46.145,31.555,46.113,31.468,46.072,31.384z M15.4,11.625c0-0.466,0.361-0.953,0.807-1.089l15.
c0.446-0.136,0.807,0.132,0.807,0.598v14.63c0,0.467-0.314,0.635-0.702,0.376l-1.127-0.752c-0.3
l-13.059,5.805c-0.426,0.189-0.771-0.034-0.771-0.501C15.4,25.943,15.4,11.625,15.4,11.625z M28
c0.425-0.189,1.085-0.134,1.473,0.125l11.43,7.62c0.388,0.259,0.368,0.644-0.045,0.861-18.404,9
c-0.412,0.216-1.047,0.163-1.418-0.121l-11.789-9.001c-0.371-0.283-0.326-0.665,0.1-0.854L28.85

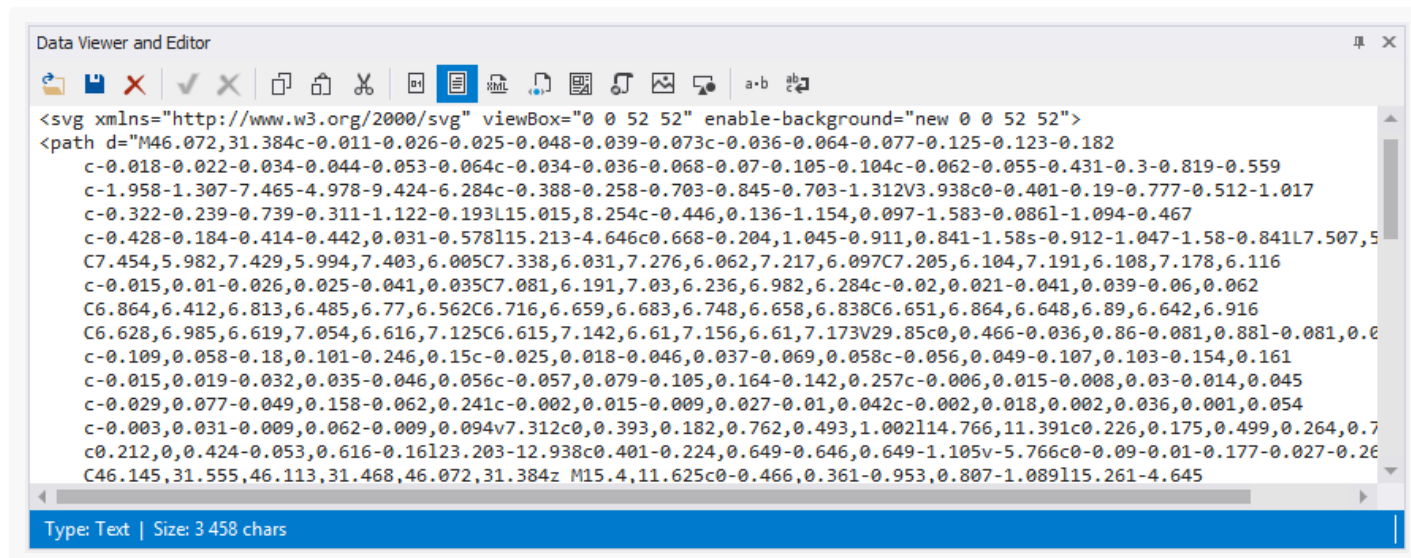
```

```

c0-0.466,0.348-0.695,0.776-0.512l2.174,0.929c0.429,0.183,0.776,0.708,0.776,1.175v2.158c-1.57
L9.142,9.932L9.142,9.932z M9.142,13.152c0.931,0.671,2.22,1.323,3.727,1.372v7.633c-1.57-0.066
C9.142,20.548,9.142,13.152,9.142,13.152z M9.142,21.627c0.931,0.671,2.22,1.323,3.727,1.372v3.
l-2.163,0.876c-0.432,0.175-0.782-0.061-0.782-0.527V21.627z M43.666,36.101c0,0.467-0.33,1.027
c-0.407,0.228-1.036,0.18-1.405-0.104L8.897,39.127c-0.369-0.284-0.668-0.893-0.668-1.358v-2.44
l12.764,9.748c0.225,0.171,0.496,0.26,0.768,0.26c0.201,0,0.403-0.048,0.588-0.146l19.899-10.44
c0.413-0.217,0.747-0.015,0.747,0.452V36.101z" style="fill:#6c91de;"/>
<path d="M33.81,34.064c0.072,0.049,0.155,0.073,0.239,0.073c0.072,0,0.145-0.018,0.209-0.055l4.505
c0.126-0.072,0.207-0.204,0.212-0.349c0.006-0.146-0.063-0.283-0.183-0.365l-9.011-6.192c-0.118
l-5.157,2.123c-0.143,0.059-0.243,0.191-0.259,0.346c-0.017,0.154,0.053,0.304,0.181,0.392L33.8
l8.269,5.682l-3.692,2.11l-8.803-6.052L29.492,25.426z" style="fill:#6c91de;"/>
</svg>

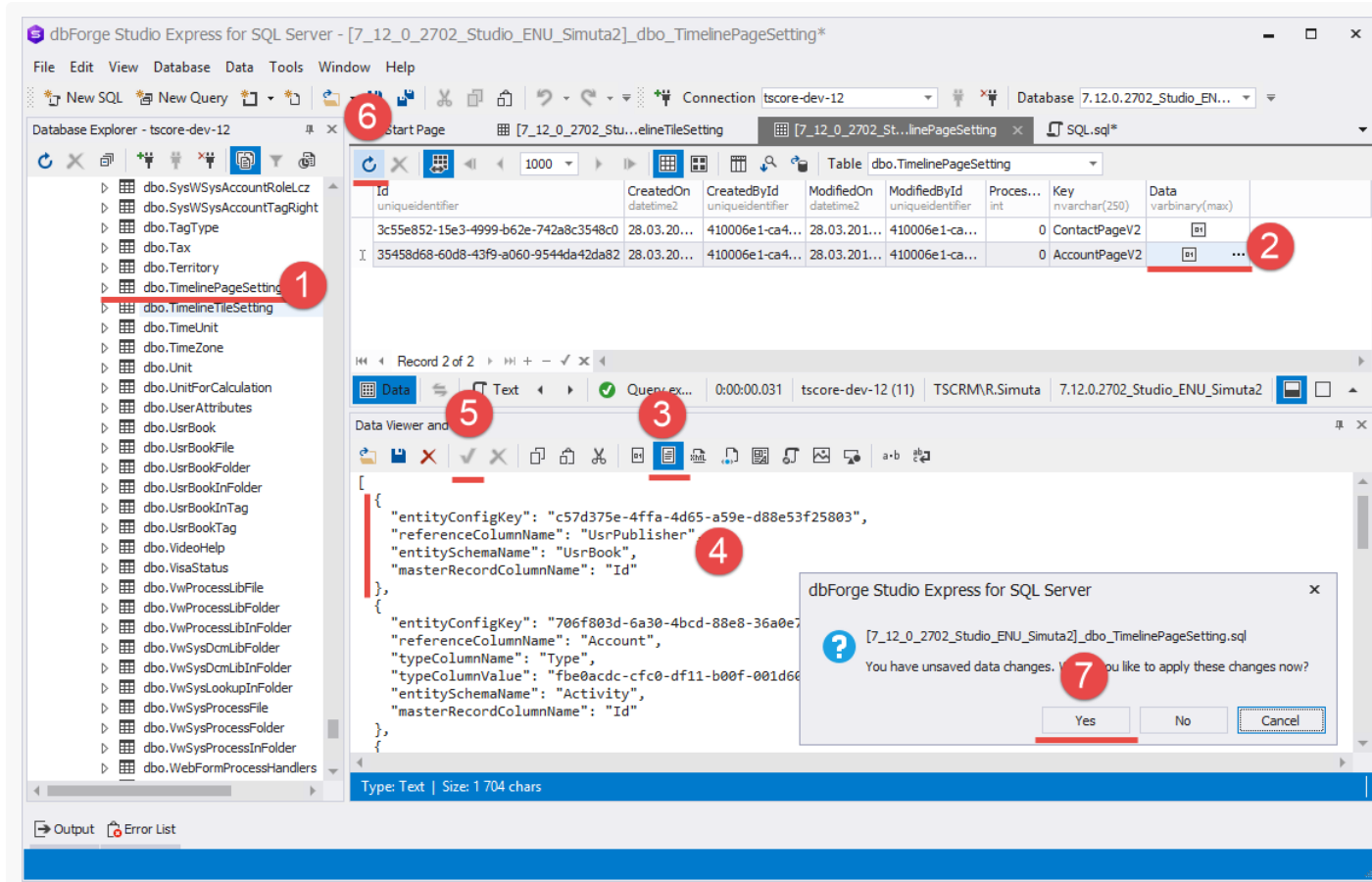
```

Редактирование данных колонки Image с помощью dbForge Studio Express for SQL Server



5. В таблице [TimelinePageSettings] отредактировать запись, подключающую отображение плиток на вкладке [Хронология] страницы контрагента

Для раздела [Контрагенты] в таблице [TimelineTileSettings] уже существует запись с настройкой плиток, связанных с другими разделами. Это запись, содержащая значение "AccountPageV2" в колонке [Key].



Важно. Поскольку в хронологии страницы раздела [*Контрагенты*] используются несколько плиток, то в колонке [*Data*] хранится массив конфигурационных объектов, подключающих соответствующую плитку.

Используя приведенную в п. 4 последовательность, измените массив конфигурационных объектов, добавив в него новую запись.

```
[
  {
    "entityConfigKey": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",
    "referenceColumnName": "UsrPublisher",
    "entitySchemaName": "UsrBook",
    "masterRecordColumnName": "Id"
  },
  ...
]
```

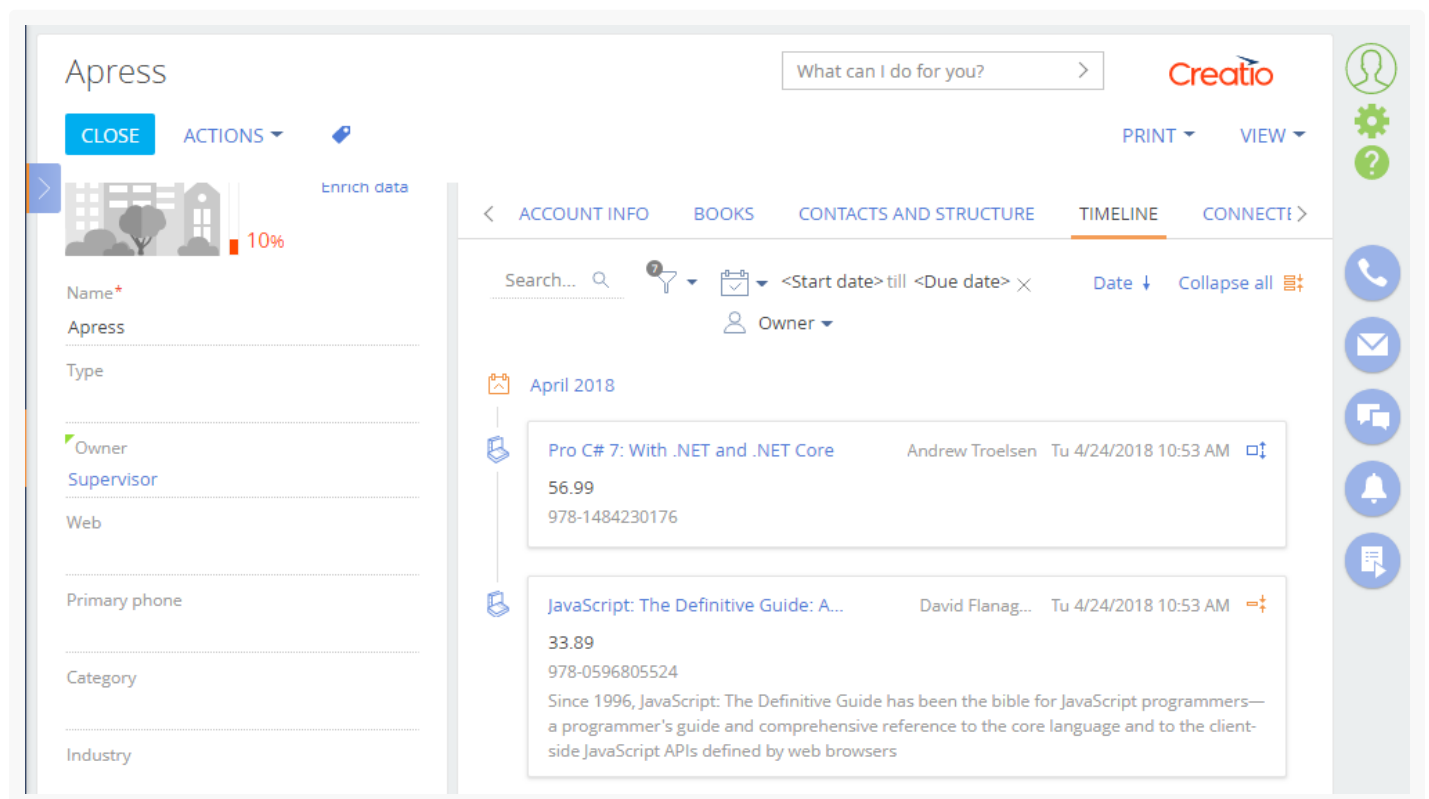
Здесь свойство "entityConfigKey": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx" должно содержать идентификатор записи таблицы [*TimelineTileSettings*], созданной на шаге 4. В этом примере это значение "c57d375e-4ffa-4d65-a59e-d88e53f25803".

Важно. Значение "entityConfigKey" должно быть указано обязательно. При этом оно обязательно должно совпадать со значением колонки [Id] записи с настройкой нужной плитки в таблице [TimelineTileSettings].

Поскольку идентификаторы создаваемых записей генерируются случайным образом, то при повторении вами шага 4 сгенерированный в вашей базе данных идентификатор будет отличаться от приведенного в этом примере.

Важно. Будьте предельно осторожны при изменении значения в колонке [Data]. Внесение некорректных изменений может нарушить работу всех существующих плиток хронологии в разделе.

В результате выполнения примера на вкладке [Хронология] ([Timeline]) страницы раздела [Контрагенты]([Accounts]) отобразятся плитки, связанные с пользовательским разделом [Книги] ([Books]). Эти плитки содержат все поля, приведенные в условиях примера. При этом краткое описание книги отображается только при разворачивании плитки.



Профиль связанной сущности

 Сложный

Элемент управления [Профиль связанной сущности] (класс Profile) — это конфигурационный модуль, который по умолчанию представляет собой информационный блок, наполняемый при загрузке страницы

записи информацией о связанной Entity-сущности. В системе этот элемент используется как профиль связанной записи на странице записи раздела. Например, при открытии страницы записи раздела [*Контакт*] в элементе [*Профиль*] отобразится профиль контакта и информация о связанном контрагенте (связь по колонке [*Контрагент*] объекта [*Контакт*]). Подробнее о профилях записи и связанной записи можно узнать из [статьи](#).

Классы, реализующие функциональность

Родительским классом для класса `Profile` является `BaseProfileSchema` — базовая схема для создания любых профилей связанных записей в системе.

Профиль объекта [*Контакт*]

70%

13:06,
Москва

ФИО*

Авдоров Сергей Валентинович

Полное название должности

Коммерческий директор

Мобильный телефон

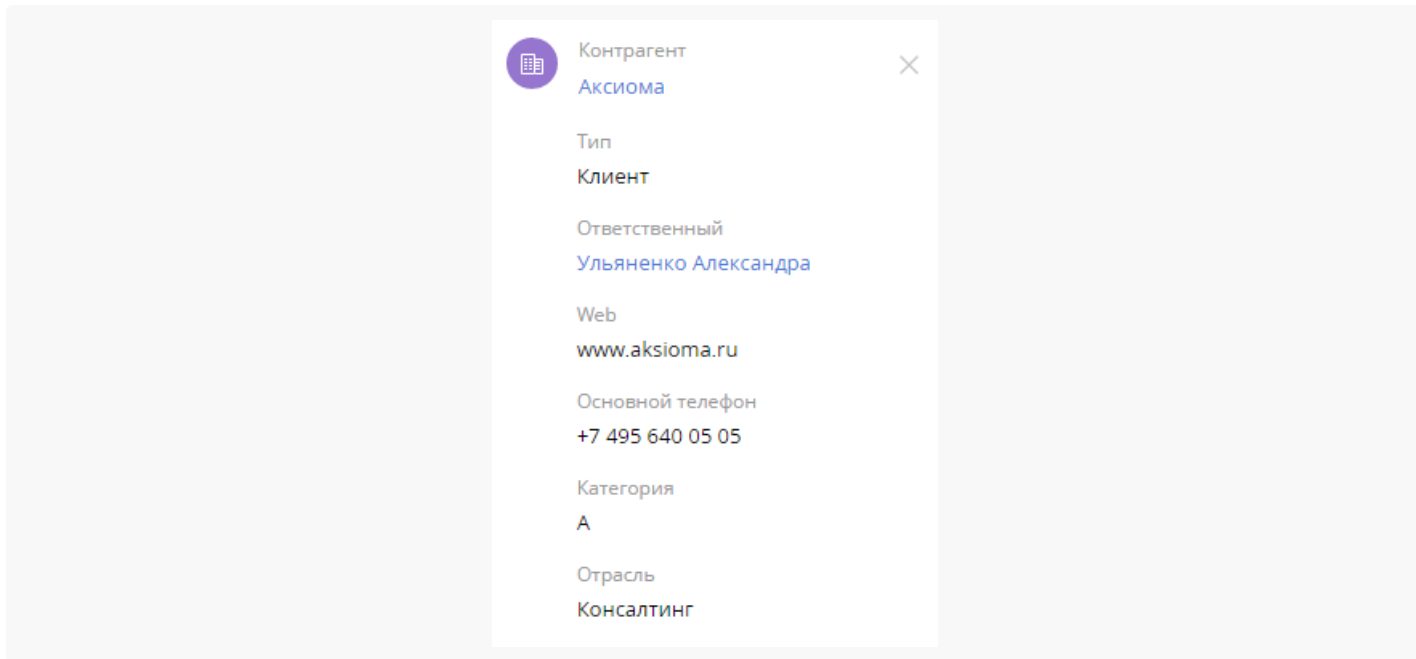
+7 495 726 46 29

Рабочий телефон

+7 495 780 80 82

Email

Связанный с контактом профиль связанной сущности [*Контрагент*]



Контрагент
Аксиома

Тип
Клиент

Ответственный
Ульяненко Александра

Web
www.aksioma.ru

Основной телефон
+7 495 640 05 05

Категория
А

Отрасль
Консалтинг

Родительским классом для класса `Profile` является `BaseProfileSchema` — базовая схема для создания любых профилей связанных записей в системе.

Важно. Все профили должны наследоваться от `BaseProfileSchema`.

Схема `BaseProfileSchema` реализует возможность отобразить любой набор полей по связанной сущности, а также любое количество различных модулей.

Способ описания представления ничем не отличается от страницы записи, т.е. описывается свойством `diff`. При встраивании модуля в страницу записи в атрибутах модуля необходимо указать параметр `masterColumnName`, в котором хранится название колонки, по которой происходит связь профиля с основной схемой страницы записи. Опираясь на значение этой колонки, `Profile` будет загружать данные.

На этапе инициализации объекта профиля, для того чтобы получить дополнительную информацию о колонке, по которой он связан (фильтры, заголовок и т.п.), он отправляет сообщение `GetColumnInfo`. Затем он запрашивает значение колонки, по которой он связан, и, если она заполнена, инициализируются данные по этой записи. При очистке поля или смене значения происходит переинициализация данных в объекте профиля.

Если профиль пуст, т.е. не выбрана запись в поле связи, то в нем отображается название поля, по которому происходит связь, и два действия:

- [*Добавить контрагента*] — создать новую запись в справочнике поля связи.
- [*Выбрать*] — выбрать существующую запись из списка доступных.

**КОНТРАГЕНТ**[Добавить контрагента](#)[Выбрать](#)

На заметку. При выборе существующей связанной сущности на справочник накладывается вся бизнес-логика, которая была определена в странице записи и была связана с атрибутом, ссылающимся на эту сущность. Т.е. сохраняется фильтрация, настройки колонок запроса и прочее.

Если атрибут удалить из схемы страницы записи, вся логика потеряется вместе с ним.

Устройство BaseProfileSchema

Интерфейс взаимодействия реализован стандартным механизмом сообщений. Используются следующие сообщения:

- `OpenCard` — инициирует открытие страницы добавления записи, используя стандартный механизм.
- `UpdateCardProperty` — обновляет значение атрибута страницы записи.
- `GetColumnInfo` — возвращает информацию о поле связи.
- `GetColumnsValues` — возвращает значения запрошенных колонок страницы записи.
- `GetEntityColumnChanges` — подписка на изменение данных на странице записи.
- `CardModuleResponse` — результат добавления новой записи через профиль.

Визуальное содержание профиля (кнопки, ссылки, модули) определяется в массиве модификаций `diff`.

Схема профиля BaseMultipleProfileSchema

Кроме схемы базового профиля `BaseProfileSchema`, имеются также дополнительные схемы, реализующие специфическую функциональность для пользовательских профилей.

Профиль, описываемый схемой `BaseMultipleProfileSchema`, может содержать в себе любое количество профилей и свободно между ними переключаться, пользуясь логикой выбора значений из нескольких справочников. Основное отличие от базового профиля — возможность встраивать другие профили в текущий профиль. При этом встроенные профили могут взаимодействовать друг с другом посредством сообщений. В остальном он работает со страницей записи так же, как и простой профиль.

Важно. Профили `BaseMultipleProfileSchema` должны наследоваться от `BaseRelatedProfileSchema` — базовой схемы для профиля, который может быть зависимым или встроенным в другие профили.

Пример конфигурации схемы профиля



Пример. Сконфигурировать схему профиля.

Реализация примера

Конфигурация схемы профиля представлена ниже.

ContactProfileSchema

```
define("ContactProfileSchema", ["ProfileSchemaMixin"],
function () {
    return {
        // Название схемы объекта.
        entitySchemaName: "Contact",
        // Миксины.
        mixins: {
            // Миксин, содержащий функции для получения иконок и картинок профиля.
            ProfileSchemaMixin: "Terrasoft.ProfileSchemaMixin"
        },
        // Массив модификаций diff.
        diff: /**SCHEMA_DIFF*/[
            {
                // Операция вставки.
                "operation": "insert",
                // Имя сущности.
                "name": "Account",
                // Имя родительского элемента, в который производится вставка.
                "parentName": "ProfileContentContainer",
                // Свойство элемента родителя, с которым производится операция.
                "propertyName": "items",
                // Значения вставляемого элемента.
                "values": {
                    // Привязка к значению свойства Account объекта Contact.
                    "bindTo": "Account",
                    // Конфигурация разметки. Позиционирование элемента.
                    "layout": {
                        "column": 5,
                        "row": 1,
                        "colSpan": 19
                    }
                }
            }
        ]
        // ...Другие конфигурационные объекты массива модификаций.
    }
}/**SCHEMA_DIFF*/
```

```
};
});
```

Пример встраивания профиля в страницу записи



Пример. Выполнить встраивание профиля на страницу записи.

Реализация примера

Встраивание профиля на страницу записи представлено ниже.

ContactPageV2

```
//Определение схемы страницы записи и ее зависимостей.
define("ContactPageV2", ["BaseFiltersGenerateModule", "BusinessRuleModule", "ContactPageV2Re
    "ConfigurationConstants", "ContactCareer", "DuplicatesSearchUtilitiesV2"],
function (BaseFiltersGenerateModule, BusinessRuleModule, resources, ConfigurationConstants,
    return {
        entitySchemaName: "Contact",
        // Используемые модули.
        modules: /**SCHEMA_MODULES*/{
            // Модуль профиля контрагента.
            "AccountProfile": {
                // Конфигурация профиля.
                "config": {
                    // Название схемы.
                    "schemaName": "AccountProfileSchema",
                    // Признак, сообщающий о том, что конфигурация схемы проинициализирована
                    "isSchemaConfigInitialized": true,
                    // Признак, сообщающий о том, что не используется HistoryState.
                    "useHistoryState": false,
                    // Параметры профиля.
                    "parameters": {
                        // Конфигурация модели представления.
                        "viewModelConfig": {
                            // Название колонки связанной сущности.
                            masterColumnName: "Account"
                        }
                    }
                }
            }
        }
    }
}
```

```

    /**SCHEMA_MODULES*/,
    // Массив модификаций.
    diff: /**SCHEMA_DIFF*/[
        {
            "operation": "insert",
            "parentName": "LeftModulesContainer",
            "propertyName": "items",
            // Название профиля.
            "name": "AccountProfile",
            // Значения.
            "values": {
                // Тип элемента — модуль.
                "itemType": Terrasoft.ViewItemType.MODULE
            }
        }
    ]/**SCHEMA_DIFF*/
};
});

```

Схема профиля BaseMultipleProfileSchema

Кроме схемы базового профиля `BaseProfileSchema`, имеются также дополнительные схемы, реализующие специфическую функциональность для пользовательских профилей.

Профиль, описываемый схемой `BaseMultipleProfileSchema`, может содержать в себе любое количество профилей и свободно между ними переключаться, пользуясь логикой выбора значений из нескольких справочников. Основное отличие от базового профиля — возможность встраивать другие профили в текущий профиль. При этом встроенные профили могут взаимодействовать друг с другом посредством сообщений. В остальном он работает со страницей записи так же, как и простой профиль.

Важно. Профили `BaseMultipleProfileSchema` должны наследоваться от `BaseRelatedProfileSchema` — базовой схемы для профиля, который может быть зависимым или встроенным в другие профили.

Пример модуля клиентского профиля BaseMultipleProfileSchema

```

// Определение профиля.
define("ClientProfileSchema", ["ProfileSchemaMixin"],
function () {
    return {
        // Миксины.
        mixins: {
            ProfileSchemaMixin: "Terrasoft.ProfileSchemaMixin"
        },
        // Атрибуты.
    };
});

```

```

attributes: {
    // Видимость профиля контакта.
    "IsVisibleContactProfile": {
        dataValueType: this.Terrasoft.DataValueType.BOOLEAN,
        value: true
    }
},
// Методы.
methods: {
    // Дата-маркер. Нужен для автотестов.
    getProfileModuleContainerDataMarker: function () {
        return "client-profile-module-container";
    },
    // Возвращает заголовок.
    getBlankSlateHeaderCaption: function () {
        return this.get("Resources.Strings.Client");
    },
    // Возвращает иконку-предупреждение.
    getWarningIcon: function () {
        return this.getImageUrlByResourceKey("Resources.Images.WarningIcon");
    },
    // Проверка для отображения предупреждения.
    getIsVisibleWarning: function () {
        var masterColumnNames = this.get("MasterColumnNames");
        if (!masterColumnNames) {
            return false;
        }
        var masterColumnValues = masterColumnNames.filter(function (columnName) {
            var value = this.get(columnName);
            return !this.Ext.isEmpty(value);
        }, this);
        return masterColumnValues.length > 1;
    },
    // Обработчик события изменения колонки профиля.
    onProfileColumnChanged: function () {
        this.set("IsVisibleContactProfile", !this.getIsVisibleWarning());
        return this.callParent(arguments);
    },
    // Обработчик события изменения колонки.
    onColumnChanged: function () {
        this.callParent(arguments);
        this.set("IsVisibleContactProfile", !this.getIsVisibleWarning());
    },
    // Подсказка для кнопки [Очистить].
    getClearButtonHint: function () {
        var clearActionCaption = this.get("Resources.Strings.ClearButtonCaption");
        var masterColumnCaption = this.get("Resources.Strings.Client");
        return this.Ext.String.format("{0} {1}", clearActionCaption, masterColumnCap
    }

```

```

},
// Модули схемы.
modules: /**SCHEMA_MODULES*/{
    // Встраиваемый клиентский профиль контрагента.
    "AccountClientProfile": {
        // Конфигурация профиля.
        "config": {
            "schemaName": "ClientAccountProfileSchema",
            "isSchemaConfigInitialized": true,
            "useHistoryState": false,
            "parameters": {
                "viewModelConfig": {
                    masterColumnName: "Account"
                }
            }
        }
    },
    // Встраиваемый клиентский профиль контакта.
    "ContactClientProfile": {
        "config": {
            "schemaName": "ClientContactProfileSchema",
            "isSchemaConfigInitialized": true,
            "useHistoryState": false,
            "parameters": {
                "viewModelConfig": {
                    masterColumnName: "Contact"
                }
            }
        }
    }
}
/**SCHEMA_MODULES*/,
// Модификации отображения профиля.
diff: /**SCHEMA_DIFF*/[
    {
        "operation": "remove",
        "name": "ProfileIcon"
    },
    {
        "operation": "remove",
        "name": "ProfileHeaderContainer"
    },
    {
        "operation": "insert",
        "name": "ClientProfilesContainer",
        "parentName": "ProfileContentContainer",
        "propertyName": "items",
        "values": {
            "itemType": this.Terrasoft.ViewItemType.CONTAINER,

```

```

        "items": [],
        "layout": {
            "column": 0,
            "row": 0,
            "colSpan": 24,
            "rowSpan": 24
        }
    },
    {
        "operation": "insert",
        "name": "WarningIcon",
        "parentName": "ClientProfilesContainer",
        "propertyName": "items",
        index: 0,
        "values": {
            "getSrcMethod": "getWarningIcon",
            "readonly": true,
            "generator": "ImageCustomGeneratorV2.generateSimpleCustomImage",
            "visible": { "bindTo": "getIsVisibleWarning" },
            "classes": {
                "wrapClass": ["warning-icon"]
            },
            "hint": { "bindTo": "Resources.Strings.WarningMessage" }
        }
    },
    {
        "operation": "insert",
        "parentName": "ClientProfilesContainer",
        "propertyName": "items",
        "name": "AccountClientProfile",
        "values": {
            "itemType": this.Terrasoft.ViewItemType.MODULE
        }
    },
    {
        "operation": "insert",
        "parentName": "ClientProfilesContainer",
        "propertyName": "items",
        "name": "ContactClientProfile",
        "values": {
            "itemType": this.Terrasoft.ViewItemType.MODULE,
            "visible": { "bindTo": "IsVisibleContactProfile" }
        }
    }
]/**SCHEMA_DIFF*/
};
}
);

```

Пример встраиваемого клиентского профиля BaseRelatedProfileSchema

```

define("ClientContactProfileSchema", ["ProfileSchemaMixin"],
function () {
    return {
        // Название схемы объекта.
        entitySchemaName: "Contact",
        // Миксины.
        mixins: {
            ProfileSchemaMixin: "Terrasoft.ProfileSchemaMixin"
        },
        // Методы.
        methods: {
            getProfileHeaderCaption: function () {
                return this.get("Resources.Strings.ProfileHeaderCaption");
            }
        },
        // Массив модификаций.
        diff: /**SCHEMA_DIFF*/[
            {
                "operation": "insert",
                "name": "Account",
                "parentName": "ProfileContentContainer",
                "propertyName": "items",
                "values": {
                    "bindTo": "Account",
                    "enabled": false,
                    "layout": {
                        "column": 5,
                        "row": 1,
                        "colSpan": 19
                    }
                }
            },
            {
                "operation": "insert",
                "name": "Job",
                "parentName": "ProfileContentContainer",
                "propertyName": "items",
                "values": {
                    "bindTo": "Job",
                    "enabled": false,
                    "layout": {
                        "column": 5,

```



```

        "row": 2,
        "colSpan": 19
    }
}
},
{
    "operation": "insert",
    "name": "Type",
    "parentName": "ProfileContentContainer",
    "propertyName": "items",
    "values": {
        "bindTo": "Type",
        "enabled": false,
        "layout": {
            "column": 5,
            "row": 3,
            "colSpan": 19
        }
    }
},
{
    "operation": "insert",
    "name": "MobilePhone",
    "parentName": "ProfileContentContainer",
    "propertyName": "items",
    "values": {
        "bindTo": "MobilePhone",
        "enabled": false,
        "layout": {
            "column": 5,
            "row": 4,
            "colSpan": 19
        }
    }
},
{
    "operation": "insert",
    "name": "Phone",
    "parentName": "ProfileContentContainer",
    "propertyName": "items",
    "values": {
        "bindTo": "Phone",
        "enabled": false,
        "layout": {
            "column": 5,
            "row": 5,
            "colSpan": 19
        }
    }
}
}

```

```

    },
    {
      "operation": "insert",
      "name": "Email",
      "parentName": "ProfileContentContainer",
      "propertyName": "items",
      "values": {
        "bindTo": "Email",
        "enabled": false,
        "layout": {
          "column": 5,
          "row": 6,
          "colSpan": 19
        }
      }
    }
  ]/**SCHEMA_DIFF*/
};
});

```

Пример встраивания модуля клиентского профиля на страницу записи

Встраивание модуля клиентского профиля на страницу записи

```

// Модули страницы записи.
modules: /**SCHEMA_MODULES*/{
  // Название модуля.
  "ClientProfile": {
    // Конфигурация.
    "config": {
      // Признак,сообщающий о том, что конфигурация схемы проинициализирована.
      "isSchemaConfigInitialized": true,
      // Признак, сообщающий о том, что не используется HistoryState.
      "useHistoryState": false,
      // Имя схемы.
      "schemaName": "ClientProfileSchema",
      // Параметры.
      "parameters": {
        // Конфигурация модели представления.
        "viewModelConfig": {
          // Название колонки связанной сущности.
          "masterColumnName": "Client"
        }
      }
    }
  }
}

```

```

    }
  }
}
}/**SCHEMA_MODULES*/,
// Массив модификаций diff.
diff: /**SCHEMA_DIFF*/[
  {
    "operation": "insert",
    // Название профиля.
    "name": "ClientProfile",
    "parentName": "LeftModulesContainer",
    "propertyName": "items",
    // Значения.
    "values": {
      // Тип элемента – модуль.
      "itemType": Terrasoft.ViewItemType.MODULE
    }
  }
]/**SCHEMA_DIFF*/

```

Пример модуля клиентского профиля BaseMultipleProfileSchema



Пример. Выполнить реализацию модуля клиентского профиля `BaseMultipleProfileSchema`.

Реализация примера

Реализация модуля клиентского профиля `BaseMultipleProfileSchema` представлена ниже.

BaseMultipleProfileSchema

```

// Определение профиля.
define("ClientProfileSchema", ["ProfileSchemaMixin"],
function () {
  return {
    // Миксины.
    mixins: {
      ProfileSchemaMixin: "Terrasoft.ProfileSchemaMixin"
    },
    // Атрибуты.

```

```

attributes: {
    // Видимость профиля контакта.
    "IsVisibleContactProfile": {
        dataValueType: this.Terrasoft.DataValueType.BOOLEAN,
        value: true
    }
},
// Методы.
methods: {
    // Дата-маркер. Нужен для автотестов.
    getProfileModuleContainerDataMarker: function () {
        return "client-profile-module-container";
    },
    // Возвращает заголовок.
    getBlankSlateHeaderCaption: function () {
        return this.get("Resources.Strings.Client");
    },
    // Возвращает иконку-предупреждение.
    getWarningIcon: function () {
        return this.getImageUrlByResourceKey("Resources.Images.WarningIcon");
    },
    // Проверка для отображения предупреждения.
    getIsVisibleWarning: function () {
        var masterColumnNames = this.get("MasterColumnNames");
        if (!masterColumnNames) {
            return false;
        }
        var masterColumnValues = masterColumnNames.filter(function (columnName) {
            var value = this.get(columnName);
            return !this.Ext.isEmpty(value);
        }, this);
        return masterColumnValues.length > 1;
    },
    // Обработчик события изменения колонки профиля.
    onProfileColumnChanged: function () {
        this.set("IsVisibleContactProfile", !this.getIsVisibleWarning());
        return this.callParent(arguments);
    },
    // Обработчик события изменения колонки.
    onColumnChanged: function () {
        this.callParent(arguments);
        this.set("IsVisibleContactProfile", !this.getIsVisibleWarning());
    },
    // Подсказка для кнопки [Очистить].
    getClearButtonHint: function () {
        var clearActionCaption = this.get("Resources.Strings.ClearButtonCaption");
        var masterColumnCaption = this.get("Resources.Strings.Client");
        return this.Ext.String.format("{0} {1}", clearActionCaption, masterColumnCaption);
    }
}

```

```

},
// Модули схемы.
modules: /**SCHEMA_MODULES*/{
    // Встраиваемый клиентский профиль контрагента.
    "AccountClientProfile": {
        // Конфигурация профиля.
        "config": {
            "schemaName": "ClientAccountProfileSchema",
            "isSchemaConfigInitialized": true,
            "useHistoryState": false,
            "parameters": {
                "viewModelConfig": {
                    masterColumnName: "Account"
                }
            }
        }
    },
    // Встраиваемый клиентский профиль контакта.
    "ContactClientProfile": {
        "config": {
            "schemaName": "ClientContactProfileSchema",
            "isSchemaConfigInitialized": true,
            "useHistoryState": false,
            "parameters": {
                "viewModelConfig": {
                    masterColumnName: "Contact"
                }
            }
        }
    }
}/**SCHEMA_MODULES*/,
// Модификации отображения профиля.
diff: /**SCHEMA_DIFF*/[
    {
        "operation": "remove",
        "name": "ProfileIcon"
    },
    {
        "operation": "remove",
        "name": "ProfileHeaderContainer"
    },
    {
        "operation": "insert",
        "name": "ClientProfilesContainer",
        "parentName": "ProfileContentContainer",
        "propertyName": "items",
        "values": {
            "itemType": this.Terrasoft.ViewItemType.CONTAINER,

```

```

        "items": [],
        "layout": {
            "column": 0,
            "row": 0,
            "colSpan": 24,
            "rowSpan": 24
        }
    },
    {
        "operation": "insert",
        "name": "WarningIcon",
        "parentName": "ClientProfilesContainer",
        "propertyName": "items",
        index: 0,
        "values": {
            "getSrcMethod": "getWarningIcon",
            "readonly": true,
            "generator": "ImageCustomGeneratorV2.generateSimpleCustomImage",
            "visible": { "bindTo": "getIsVisibleWarning" },
            "classes": {
                "wrapClass": ["warning-icon"]
            },
            "hint": { "bindTo": "Resources.Strings.WarningMessage" }
        }
    },
    {
        "operation": "insert",
        "parentName": "ClientProfilesContainer",
        "propertyName": "items",
        "name": "AccountClientProfile",
        "values": {
            "itemType": this.Terrasoft.ViewItemType.MODULE
        }
    },
    {
        "operation": "insert",
        "parentName": "ClientProfilesContainer",
        "propertyName": "items",
        "name": "ContactClientProfile",
        "values": {
            "itemType": this.Terrasoft.ViewItemType.MODULE,
            "visible": { "bindTo": "IsVisibleContactProfile" }
        }
    }
]/**SCHEMA_DIFF*/
};
}
);

```

Пример встраиваемого клиентского профиля BaseRelatedProfileSchema



Пример. Выполнить встраивание клиентского профиля `BaseRelatedProfileSchema`.

Реализация примера

Встраивание клиентского профиля `BaseRelatedProfileSchema` представлена ниже.

BaseMultipleProfileSchema

```
define("ClientContactProfileSchema", ["ProfileSchemaMixin"],
function () {
    return {
        // Название схемы объекта.
        entitySchemaName: "Contact",
        // Миксины.
        mixins: {
            ProfileSchemaMixin: "Terrasoft.ProfileSchemaMixin"
        },
        // Методы.
        methods: {
            getProfileHeaderCaption: function () {
                return this.get("Resources.Strings.ProfileHeaderCaption");
            }
        },
        // Массив модификаций.
        diff: /**SCHEMA_DIFF*/[
            {
                "operation": "insert",
                "name": "Account",
                "parentName": "ProfileContentContainer",
                "propertyName": "items",
                "values": {
                    "bindTo": "Account",
                    "enabled": false,
                    "layout": {
                        "column": 5,
                        "row": 1,
                        "colSpan": 19
                    }
                }
            }
        ]
    };
});
```

```

    }
  }
},
{
  "operation": "insert",
  "name": "Job",
  "parentName": "ProfileContentContainer",
  "propertyName": "items",
  "values": {
    "bindTo": "Job",
    "enabled": false,
    "layout": {
      "column": 5,
      "row": 2,
      "colSpan": 19
    }
  }
},
{
  "operation": "insert",
  "name": "Type",
  "parentName": "ProfileContentContainer",
  "propertyName": "items",
  "values": {
    "bindTo": "Type",
    "enabled": false,
    "layout": {
      "column": 5,
      "row": 3,
      "colSpan": 19
    }
  }
},
{
  "operation": "insert",
  "name": "MobilePhone",
  "parentName": "ProfileContentContainer",
  "propertyName": "items",
  "values": {
    "bindTo": "MobilePhone",
    "enabled": false,
    "layout": {
      "column": 5,
      "row": 4,
      "colSpan": 19
    }
  }
},
{

```



```

        "operation": "insert",
        "name": "Phone",
        "parentName": "ProfileContentContainer",
        "propertyName": "items",
        "values": {
            "bindTo": "Phone",
            "enabled": false,
            "layout": {
                "column": 5,
                "row": 5,
                "colSpan": 19
            }
        }
    },
    {
        "operation": "insert",
        "name": "Email",
        "parentName": "ProfileContentContainer",
        "propertyName": "items",
        "values": {
            "bindTo": "Email",
            "enabled": false,
            "layout": {
                "column": 5,
                "row": 6,
                "colSpan": 19
            }
        }
    }
}
]/**SCHEMA_DIFF*/
};
});

```

Пример встраивания модуля клиентского профиля на страницу записи



Сложный

Пример. Выполнить встраивание модуля клиентского профиля на страницу записи.

Реализация примера

Встраивание модуля клиентского профиля на страницу записи представлено ниже.

Встраивание модуля клиентского профиля на страницу записи

```
// Модули страницы записи.
modules: /**SCHEMA_MODULES*/{
    // Название модуля.
    "ClientProfile": {
        // Конфигурация.
        "config": {
            // Признак,сообщающий о том, что конфигурация схемы проинициализирована.
            "isSchemaConfigInitialized": true,
            // Признак, сообщающий о том, что не используется HistoryState.
            "useHistoryState": false,
            // Имя схемы.
            "schemaName": "ClientProfileSchema",
            // Параметры.
            "parameters": {
                // Конфигурация модели представления.
                "viewModelConfig": {
                    // Название колонки связанной сущности.
                    "masterColumnName": "Client"
                }
            }
        }
    }
}
/**SCHEMA_MODULES*/,
// Массив модификаций diff.
diff: /**SCHEMA_DIFF*/[
    {
        "operation": "insert",
        // Название профиля.
        "name": "ClientProfile",
        "parentName": "LeftModulesContainer",
        "propertyName": "items",
        // Значения.
        "values": {
            // Тип элемента – модуль.
            "itemType": Terrasoft.ViewItemType.MODULE
        }
    }
]/**SCHEMA_DIFF*/
```