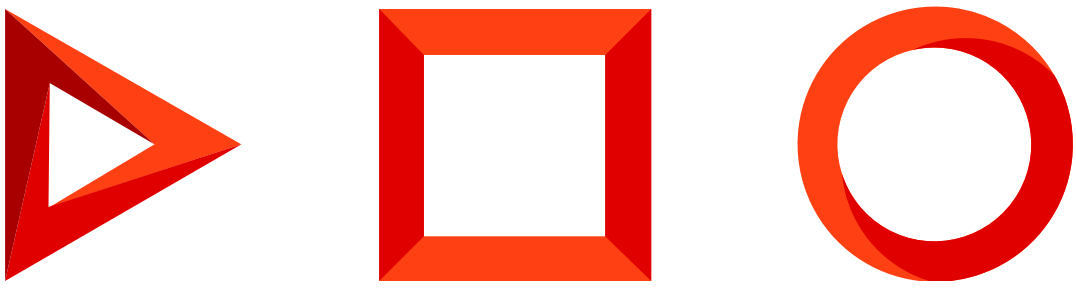
Creatio Academy

Деталь

Версия 8.0



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

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

Содержание

Деталь

Основы

Деталь — элемент интерфейса на странице записи, который отображает записи определенного объекта, связанного 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*] ([*Communication options*]), 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).

Назначение детали — отображение дополнительных данных для основного объекта раздела. Детали раздела отображаются во вкладках страницы записи раздела в контейнере вкладок.

Структура и типы деталей

Составляющие детали:

- **Схема объекта детали** связана с объектом раздела. Например, детали [*Адреса*] ([*Addresses*]) страницы контакта соответствует схема объекта `ContactAddress` пакета `Base`. Связь с объектом раздела выполняется по обязательной колонке [*Contact*] объекта детали.
- **Схема модели представления детали** позволяет настроить структуру, расположение и поведение элементов пользовательского интерфейса детали. Например, деталь [*Адреса*] ([*Addresses*]) страницы контакта настраивается в схеме `ContactAddressDetailV2` модели представления детали, которая наследует схему `BaseAddressDetailV2`) пакета `UIv2`.
- **Схема модели представления страницы записи детали** позволяет настроить страницу детали. Например, свойства страницы детали [*Адреса*] ([*Addresses*]) страницы контакта настраивается в схеме `ContactAddressPageV2` модели представления страницы записи детали, которая наследует схему `BaseAddressPageV2` пакета `UIv2`.

Типы деталей, которые предоставляет Creatio:

- Деталь с редактируемым реестром.
- Деталь со страницей добавления.
- Деталь с выбором из справочника.
- Деталь с полями.
- Деталь типа [*Файлы и ссылки*] ([*Attachments*]).

Тип детали зависит от метода ввода и отображения данных.

Реализовать деталь

Функциональность базовой детали реализована в схеме `BaseDetailV2` пакета `NUI`.

Инструменты, которые позволяют реализовать деталь:

- Мастер деталей.
- Creatio IDE.

Реализовать некоторые типы деталей невозможно исключительно в мастере деталей. В этом случае необходимо использовать комбинацию мастера деталей и Creatio IDE. Особенности использования инструментов при реализации разных типов деталей будут рассмотрены далее.

Общий алгоритм реализации детали с использованием **мастера деталей**:

1. **Создайте пользовательскую деталь**. Для этого воспользуйтесь инструкцией, которая приведена в статье Создать новую деталь.
2. **Добавьте пользовательскую деталь на страницу записи**. Для этого воспользуйтесь инструкцией, которая приведена в статье Настроить существующую деталь на странице записи.
3. **Настройте внешний вид пользовательской детали** (опционально). Для этого воспользуйтесь инструкцией, которая приведена в статье Настроить существующую деталь на странице записи.

Общий алгоритм реализации детали с использованием **Creatio IDE**:

1. **Создайте пользовательскую деталь**.
   a. Создайте схему объекта детали. Для этого воспользуйтесь инструкцией, которая приведена в статье Разработка конфигурационных элементов.
   b. Создайте схему модели представления детали. Для этого воспользуйтесь инструкцией, которая приведена в статье Разработка конфигурационных элементов.
   c. Добавьте пользовательские стили детали (опционально).
   d. Зарегистрируйте деталь в базе данных (опционально).
2. **Добавьте пользовательскую деталь на страницу записи**.
   Создайте схему замещающей модели представления страницы записи, на которой будет размещена деталь. Для этого воспользуйтесь инструкцией, которая приведена в статье Разработка конфигурационных элементов.
3. **Настройте внешний вид пользовательской детали** (опционально). Для этого воспользуйтесь инструкцией, которая приведена в статье Настроить существующую деталь на странице записи.

Реализовать деталь с редактируемым реестром

Деталь с редактируемым реестром позволяет вводить и редактировать данные в реестре детали. Данные отображаются в списочном представлении. Деталь с редактируемым реестром является подвидом детали с реестром. Функциональность базовой детали с реестром реализована в схеме `BaseGridDetailV2` пакета `NUI`. Примером детали с редактируемым реестром является деталь [*Продукты*] ([*Products*]) страницы заказа. Данные каждого продукта редактируются на странице заказа.

Использовать мастер деталей для реализации детали с редактируемым реестром

1. **Создайте пользовательскую деталь**.

   - Настройте редактируемый реестр. Для этого установите признак [*Сделать реестр редактируемым*] ([*Make the list editable*]). В другом случае будет создана деталь со страницей добавления.
   - Настройте многострочный текст (опционально). Для отображения данных в несколько строк установите признак [*Многострочный текст*] ([*Multi-line text*]). Многострочный текст доступен к использованию только для колонок типа [*Строка*] ([*String*]).

2. Выполните шаг 2 общего алгоритма реализации детали с использованием мастера деталей.

Использовать Creatio IDE для реализации детали с редактируемым реестром

1. **Создайте пользовательскую деталь**.

   a. Создайте схему объекта детали.

   - В качестве родительского объекта выберите `BaseEntity`.
   - В схему объекта добавьте колонку типа [*Строка*] ([*String*]) и другие необходимые колонки.

   d. Создайте схему модели представления детали с редактируемым реестром.

   - В качестве родительского объекта выберите `BaseGridDetailV2`.
   - На панели инструментов в контекстном меню узла [*Локализуемые строки*] ([*Localizable strings*]) выберите локализуемую строку [*Caption*] и в свойстве [*Значение*] ([*Value*]) задайте название детали.
   - Реализуйте редактируемый реестр.
     - В зависимости добавьте схемы модулей `ConfigurationGrid`, `ConfigurationGridGenerator`, `ConfigurationGridUtilities`.
     - В свойство `mixins` добавьте миксин `ConfigurationGridUtilities`.
     - В свойство `attributes` добавьте атрибут `IsEditable` со значением `true` свойства `value`.
   - Реализуйте многострочный текст (опционально). Для этого в свойство `attributes` добавьте колонку типа [*Строка*] ([*String*]) со значением `Terrasoft.ContentType.LONG_TEXT` свойства

`contentType`.

Пример схемы `ContactPageV2` модели представления детали с редактируемым реестром `UsrCourierServiceDetail`, у которой для колонки [*UsrDescription*] используется многострочный текст, приведен ниже.

Пример схемы замещающей модели представления

```
/* Определение схемы и установка ее зависимостей от других модулей. */
define("UsrCourierServiceDetail", ["ConfigurationGrid", "ConfigurationGridGenerator",
    "ConfigurationGridUtilities"], function() {
    return {
        /* Название схемы объекта детали. */
        entitySchemaName: "UsrCourierService",
        /* Перечень атрибутов схемы. */
        attributes: {
            /* Признак возможности редактирования. */
            "IsEditable": {
                /* Тип данных – логический. */
                dataValueType: Terrasoft.DataValueType.BOOLEAN,
                /* Тип атрибута – виртуальная колонка модели представления. */
                type: Terrasoft.ViewModelColumnType.VIRTUAL_COLUMN,
                /* Устанавливаемое значение. */
                value: true
            }
        },
        /* Используемые миксины. */
        mixins: {
            ConfigurationGridUtilities: "Terrasoft.ConfigurationGridUtilities"
        },
        /* Массив модификаций модели представления. */
        diff: /**SCHEMA_DIFF*/[
            {
                /* Тип операции – слияние. */
                "operation": "merge",
                /* Название элемента схемы, над которым производится действие. */
                "name": "DataGrid",
                /* Объект, свойства которого будут объединены со свойствами элемента схемы
                "values": {
                    /* Имя класса. */
                    "className": "Terrasoft.ConfigurationGrid",
                    /* Генератор представления должен генерировать только часть представле
                    "generator": "ConfigurationGridGenerator.generatePartial",
                    /* Привязка события получения конфигурации элементов редактирования ак
                    "generateControlsConfig": {"bindTo": "generateActiveRowControlsConfig"
                    /* Привязка события смены активной записи к методу-обработчику. */
                    "changeRow": {"bindTo": "changeRow"},
                    /* Привязка события отмены выбора записи к методу-обработчику. */
```

```
                    "unSelectRow": {"bindTo": "unSelectRow"},
                    /* Привязка  события клика на реестре к методу-обработчику. */
                    "onGridClick": {"bindTo": "onGridClick"},
                    /* Действия, производимые с активной записью. */
                    "activeRowActions": [
                        /* Настройка действия Сохранить. */
                        {
                            /* Имя класса элемента управления, с которым связано действие.
                            "className": "Terrasoft.Button",
                            /* Стиль отображения – прозрачная кнопка. */
                            "style": this.Terrasoft.controls.ButtonEnums.style.TRANSPARENT
                            /* Тег. */
                            "tag": "save",
                            /* Значение маркера. */
                            "markerValue": "save",
                            /* Привязка к изображению кнопки. */
                            "imageConfig": {"bindTo": "Resources.Images.SaveIcon"}
                        },
                        /* Настройка действия Отменить. */
                        {
                            "className": "Terrasoft.Button",
                            "style": this.Terrasoft.controls.ButtonEnums.style.TRANSPARENT
                            "tag": "cancel",
                            "markerValue": "cancel",
                            "imageConfig": {"bindTo": "Resources.Images.CancelIcon"}
                        },
                        /* Настройка действия Удалить. */
                        {
                            "className": "Terrasoft.Button",
                            "style": this.Terrasoft.controls.ButtonEnums.style.TRANSPARENT
                            "tag": "remove",
                            "markerValue": "remove",
                            "imageConfig": {"bindTo": "Resources.Images.RemoveIcon"}
                        }
                    ],
                    /* Привязка к методу, который инициализирует подписку на события нажат
                    "initActiveRowKeyMap": {"bindTo": "initActiveRowKeyMap"},
                    /* Привязка события выполнения действия активной записи к методу-обраб
                    "activeRowAction": {"bindTo": "onActiveRowAction"},
                    /* Признак возможности выбора нескольких записей. */
                    "multiSelect": {"bindTo": "MultiSelect"},
                    /* Колонка описания. */
                    "UsrDescription": {
                        /* Тип отображения - длинный текст. */
                        "contentType": Terrasoft.ContentType.LONG_TEXT
                    }
                }
            }
        ]/**SCHEMA_DIFF*/
```

```
    };
});
```

i. Зарегистрируйте деталь в базе данных. Для этого выполните SQL-запрос к таблице `[SysDetails]` базы данных.

SQL-запрос

```
DECLARE
-- Название схемы детали.
@ClientUnitSchemaName NVARCHAR(100) = 'UsrИмяСхемыДетали',
-- Название схемы объекта детали.
@EntitySchemaName NVARCHAR(100) = 'UsrИмяСхемыОбъектаДетали',
-- Название детали.
@DetailCaption NVARCHAR(100) = 'ИмяДетали'

INSERT INTO SysDetail(
    Caption,
    DetailSchemaUId,
    EntitySchemaUId
)
VALUES(
    @DetailCaption,
    (SELECT TOP 1 UId
    from SysSchema
    WHERE Name = @ClientUnitSchemaName),
    (SELECT TOP 1 UId
    from SysSchema
    WHERE Name = @EntitySchemaName)
)
```

Регистрация детали выполняется, чтобы деталь стала видимой для мастера разделов и мастера деталей.

2. **Добавьте пользовательскую деталь на страницу записи**.

Создайте схему замещающей модели представления страницы записи, на которой будет размещена деталь с редактируемым реестром.

- В качестве родительского объекта выберите схему модели представления, которую необходимо заместить.
- В свойство `details` добавьте деталь.
- В массив модификаций `diff` добавьте конфигурационный объект модели представления детали.

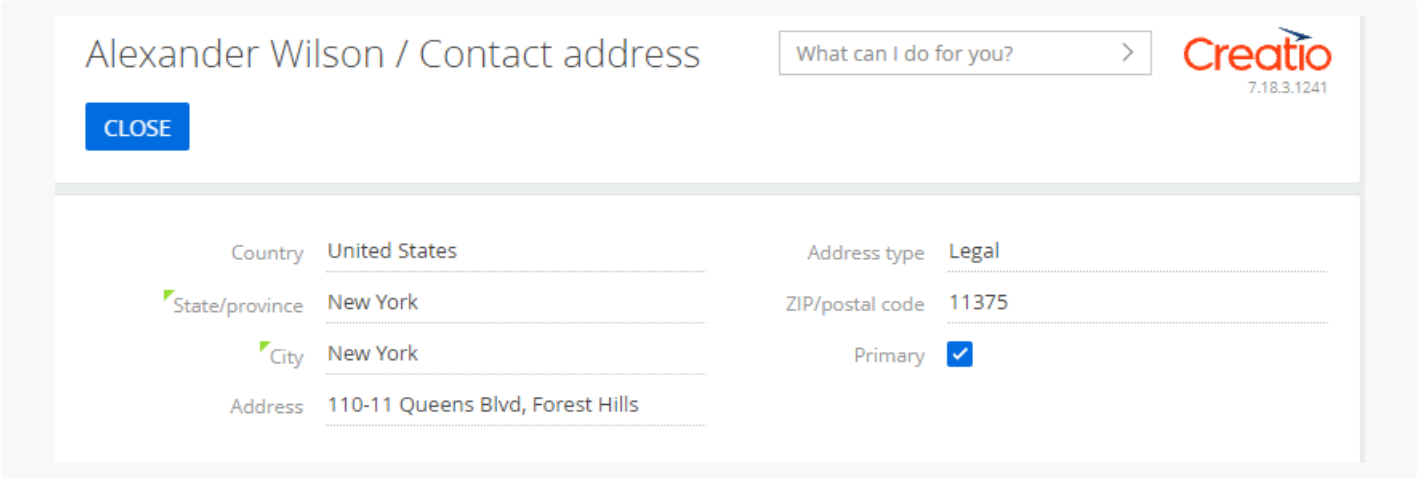
Пример схемы `ContactPageV2` замещающей модели представления страницы записи, на которой размещена деталь с редактируемым реестром `UsrRegDocumentFieldsDetail` приведен ниже.

Пример схемы замещающей модели представления

```
define("ContactPageV2", [], function() {
    return {
        entitySchemaName: "Contact",
        details: /**SCHEMA_DETAILS*/ {
            /* Добавление детали с полями. */
            "UsrRegDocumentFieldsDetail": {
                /* Название клиентской схемы детали. */
                "schemaName": "UsrRegDocumentFieldsDetail",
                /* Фильтрация записей детали текущего контакта (физ. лица). */
                "filter": {
                    /* Колонка объекта детали. */
                    "detailColumn": "UsrContact",
                    /* Колонка идентификатора контакта. */
                    "masterColumn": "Id"
                }
            }
        } /**SCHEMA_DETAILS*/ ,
        diff: /**SCHEMA_DIFF*/ [{
            /* Добавление нового элемента. */
            "operation": "insert",
            /* Название элемента. */
            "name": "UsrRegDocumentFieldsDetail",
            /* Конфигурационный объект значений. */
            "values": {
                /* Тип элемента. */
                "itemType": Terrasoft.ViewItemType.DETAIL
            },
            /* Имя элемента-контейнера. */
            "parentName": "HistoryTab",
            /* Имя свойства элемента-контейнера, который содержит коллекцию вложенных элемент
            "propertyName": "items",
            /* Индекс добавляемого в коллекцию элемента. */
            "index": 0
        }] /**SCHEMA_DIFF*/
    };
});
```

Реализовать деталь со страницей добавления

Деталь со страницей добавления позволяет вводить и редактировать данные на странице детали. Деталь со страницей добавления является подвидом детали с реестром. Функциональность базовой детали с реестром реализована в схеме `BaseGridDetailV2` пакета `NUI`. Примером детали со страницей добавления является деталь [*Адреса*] ([*Addresses*]) страницы контакта. Данные каждого адреса вводятся и редактируются на странице [*Адрес контакта*] ([*Contact address*]).



Чтобы реализовать деталь со страницей добавления с использованием **мастера деталей**, используйте общий алгоритм реализации детали с использованием мастера деталей.

Чтобы реализовать деталь со страницей добавления с использованием **Creatio IDE**:

1. **Создайте пользовательскую деталь**.

   a. Создайте схему объекта детали.

   - В качестве родительского объекта выберите `BaseEntity`.
   - В схему объекта добавьте необходимые колонки.

   d. Создайте схему модели представления детали со страницей добавления.

   - В качестве родительского объекта выберите `BaseGridDetailV2`.
   - На панели инструментов в контекстном меню узла [*Локализуемые строки*] ([*Localizable strings*]) выберите локализуемую строку [*Caption*] и в свойстве [*Значение*] ([*Value*]) задайте название детали.

   g. Создайте схему модели представления страницы записи детали. Для этого воспользуйтесь инструкцией, которая приведена в статье Разработка конфигурационных элементов.

   - В качестве родительского объекта выберите `BasePageV2`.
   - В массив модификаций `diff` добавьте конфигурационный объект модели представления детали.

   Пример схемы `UsrCourierDetailPage` модели представления страницы записи детали `UsrCourierInOrder` приведен ниже.

   Пример схемы модели представления страницы записи детали

```
define("UsrCourierDetailPage", [], function() {
    return {
        /* Название схемы объекта детали. */
        entitySchemaName: "UsrCourierInOrder",
        details: /**SCHEMA_DETAILS*/{}/**SCHEMA_DETAILS*/,
```

```
        /* Массив модификаций. */
        diff: /**SCHEMA_DIFF*/[
            /* Метаданные для добавления поля [Заказ]. */
            {
                "operation": "insert",
                /* Название поля. */
                "name": "Order",
                "values": {
                    /* Настройка расположения поля на странице записи. */
                    "layout": {
                        "colSpan": 12,
                        "rowSpan": 1,
                        "column": 0,
                        "row": 0,
                        "layoutName": "Header"
                    },
                    /* Привязка к колонке [Order] схемы объекта. */
                    "bindTo": "UsrOrder"
                },
                "parentName": "Header",
                "propertyName": "items",
                "index": 0
            },
            /* Метаданные для добавления поля [Контакт]. */
            {
                "operation": "insert",
                /* Название поля. */
                "name": "Contact",
                "values": {
                    /* Настройка расположения поля на странице записи. */
                    "layout": {
                        "colSpan": 12,
                        "rowSpan": 1,
                        "column": 12,
                        "row": 0,
                        "layoutName": "Header"
                    },
                    /* Привязка к колонке [Contact] схемы объекта. */
                    "bindTo": "UsrContact"
                },
                "parentName": "Header",
                "propertyName": "items",
                "index": 1
            }
        ]/**SCHEMA_DIFF*/,
        methods: {},
        rules: {}
    };
});
```

j. Зарегистрируйте деталь в базе данных.

a. Зарегистрируйте связь между схемой объекта детали и схемой реестра детали. Для этого выполните SQL-запрос к таблице `[SysDetails]` базы данных.

SQL-запрос

```
DECLARE
-- Название схемы детали.
@DetailSchemaName NCHAR(100) = 'UsrИмяСхемыДетали',
-- Название схемы объекта детали.
@EntitySchemaName NVARCHAR(100) = 'UsrИмяСхемыОбъектаДетали',
-- Название детали.
@DetailCaption NVARCHAR(100) = 'ИмяДетали'

INSERT INTO SysDetail(
    ProcessListeners,
    Caption,
    DetailSchemaUId,
    EntitySchemaUId
)
VALUES (
    0,
    @DetailCaption,
    (SELECT TOP 1 UId
    FROM SysSchema
    WHERE name = @DetailSchemaName),
    (SELECT TOP 1 UId
    FROM SysSchema
    WHERE name = @EntitySchemaName)
)
```

b. Зарегистрируйте связь между схемой объекта детали и схемой страницы записи детали. Для этого выполните SQL-запрос к таблицам `[SysModuleEntity]` и `[SysModuleEdit]` базы данных.

SQL-запрос

```
DECLARE
-- Название схемы страницы детали.
@CardSchemaName NCHAR(100) = 'UsrИмяСхемыСтраницыДетали',
-- Название схемы объекта детали.
@EntitySchemaName NVARCHAR(100) = 'UsrИмяСхемыОбъектаДетали',
-- Название страницы детали.
@PageCaption NVARCHAR(100) = 'ИмяСтраницыДетали',
-- Пустая строка.
@Blank NCHAR(100) = ''
```

```
INSERT INTO SysModuleEntity(
    ProcessListeners,
    SysEntitySchemaUId
)
VALUES (
    0,
    (SELECT TOP 1 UId
    FROM SysSchema
    WHERE Name = @EntitySchemaName
    )
)

INSERT INTO SysModuleEdit(
    SysModuleEntityId,
    UseModuleDetails,
    Position,
    HelpContextId,
    ProcessListeners,
    CardSchemaUId,
    ActionKindCaption,
    ActionKindName,
    PageCaption
)
VALUES (
    (SELECT TOP 1 Id
    FROM SysModuleEntity
    WHERE SysEntitySchemaUId = (
        SELECT TOP 1 UId
        FROM SysSchema
        WHERE Name = @EntitySchemaName
        )
    ),
    1,
    0,
    @Blank,
    0,
    (SELECT TOP 1 UId
     FROM SysSchema
     WHERE name = @CardSchemaName
    ),
    @Blank,
    @Blank,
    @PageCaption
)
```

Регистрация детали выполняется, чтобы деталь стала видимой для мастера разделов и мастера деталей.

k. Для применения изменений перезапустите приложение в IIS.

2. **Добавьте пользовательскую деталь на страницу записи**.

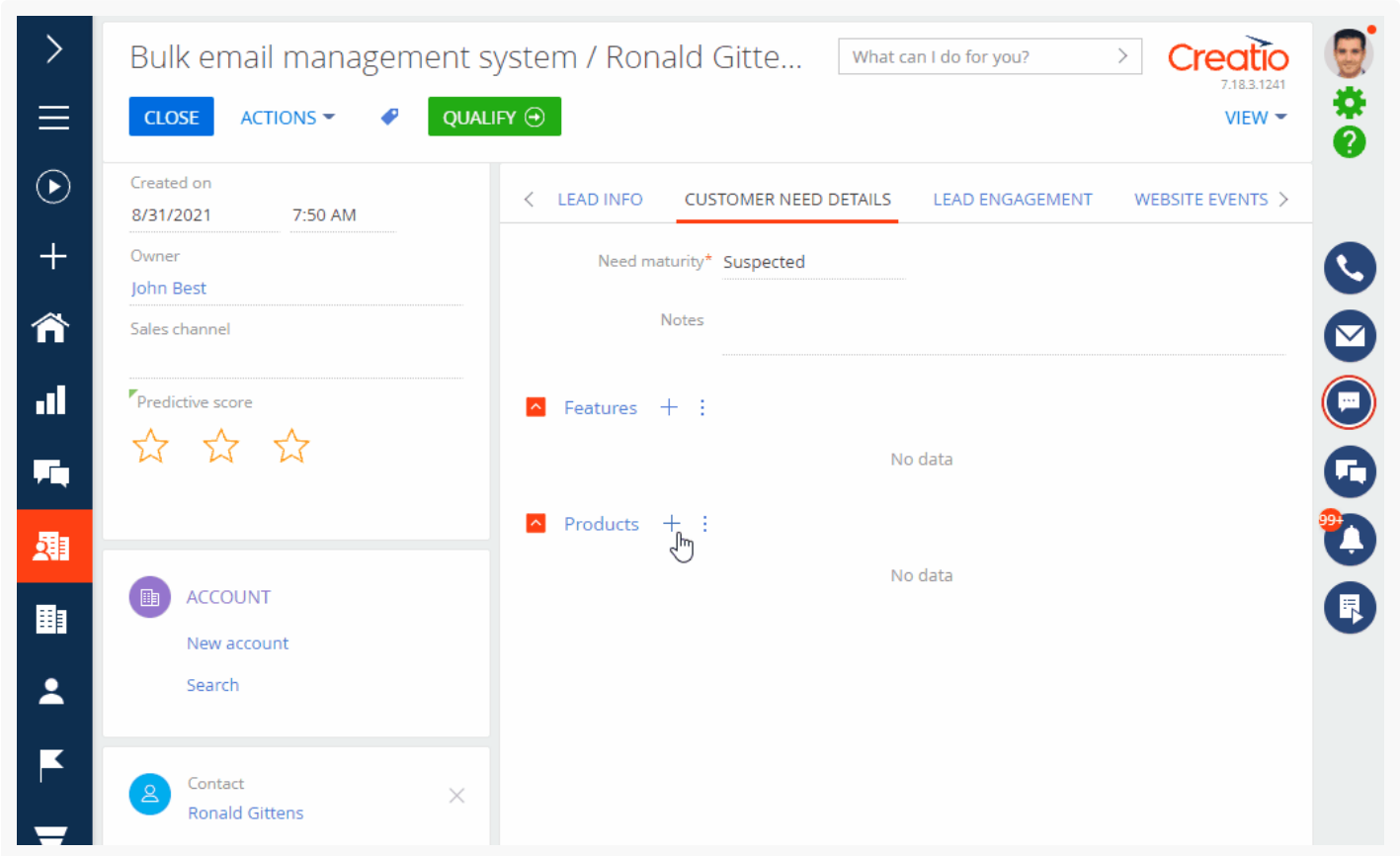
   Создайте схему замещающей модели представления страницы записи, на которой будет размещена деталь со страницей добавления.

   - В качестве родительского объекта выберите схему модели представления, которую необходимо заместить.
   - В свойство `details` добавьте деталь.
   - В массив модификаций `diff` добавьте конфигурационный объект модели представления детали.

3. Выполните шаг 4 общего алгоритма реализации детали с использованием Creatio IDE. Без выполнения этого шага деталь отображается на странице записи, но не содержит записей, поскольку не указаны колонки для отображения.

Реализовать деталь с выбором из справочника

Деталь с выбором из справочника позволяет выбирать данные из справочника, который отображается в модальном окне. Деталь с выбором из справочника является подвидом детали с реестром. Функциональность базовой детали с реестром реализована в схеме `BaseGridDetailV2` пакета `NUI`. Примером детали с выбором из справочника является деталь [*Продукты*] ([*Products*]) страницы лида. Выбор продукта выполняется в модальном окне [*Выбор: Продукты*] ([*Select: Products*]).



Использовать мастер деталей для реализации детали с выбором из справочника

1. **Создайте пользовательскую деталь**.

   - Добавьте на деталь колонку типа [*Справочник*] ([*Lookup*]).
   - Настройте вид справочника. Для этого в свойстве [*Вид справочника*] ([*Lookup view*]) выберите значение "`Всплывающее окно`" ("`Selection window`").

2. Выполните шаг 2 общего алгоритма реализации детали с использованием мастера деталей.

Использовать Creatio IDE для реализации детали с выбором из справочника

1. **Создайте пользовательскую деталь**.

   a. Создайте схему объекта детали.

   - В качестве родительского объекта выберите `BaseEntity`.
   - В схему объекта добавьте колонку типа [*Справочник*] ([*Lookup*]) и другие необходимые колонки.

   d. Создайте схему модели представления детали с выбором из справочника.

   - В качестве родительского объекта выберите `BaseGridDetailV2`.
   - На панели инструментов в контекстном меню узла [*Локализуемые строки*] ([*Localizable strings*]) выберите локализуемую строку [*Caption*] и в свойстве [*Значение*] ([*Value*]) задайте название детали.

2. **Добавьте пользовательскую деталь на страницу записи**.

   Создайте схему замещающей модели представления страницы записи, на которой будет размещена деталь с выбором из справочника.

   - В качестве родительского объекта выберите схему модели представления, которую необходимо заместить.
   - В зависимости схемы модели представления страницы записи добавьте схему модуля `ConfigurationEnums`.
   - В свойство `methods` добавьте методы:
     - `onDocumentInsert()` — обрабатывает событие добавления записей в реестр детали.
     - `onCardSaved()` — обрабатывает событие сохранения страницы записи с деталью.
     - `openDocumentLookup()` — вызывает модальное окно справочника.
     - Вспомогательные методы управления данными.
   - В массив модификаций `diff` добавьте конфигурационный объект модели представления детали.

   Пример схемы `UsrCourierCertDetail` модели представления страницы записи, на которой размещена деталь с выбором из справочника `UsrCourierCertInOrder` приведен ниже.

   Пример схемы замещающей модели представления

```
/* Определение схемы и установка ее зависимостей от других модулей. */
define("UsrCourierCertDetail", ["ConfigurationEnums"],
    function(configurationEnums) {
        return {
            /* Название схемы объекта детали. */
            entitySchemaName: "UsrCourierCertInOrder",
            /* Методы схемы детали. */
            methods: {
                /* Возвращает колонки, которые выбираются запросом. */
                getGridDataColumns: function() {
                    return {
                        "Id": {path: "Id"},
                        "Document": {path: "UsrDocument"},
                        "Document.Number": {path: "UsrDocument.Number"}
                    };
                },

                /* Конфигурирует и отображает модальное окно справочника. */
                openDocumentLookup: function() {
                    /* Конфигурационный объект. */
                    var config = {
                        /* Название схемы объекта, записи которого будут отображены в справоч
                        entitySchemaName: "Document",
                        /* Возможность множественного выбора. */
                        multiSelect: true,
                        /* Колонки, которые будут использованы в справочнике, например, для с
                        columns: ["Number", "Date", "Type"]
                    };
                    var OrderId = this.get("MasterRecordId");
                    if (this.Ext.isEmpty(OrderId)) {
                        return;
                    }
                    /* Экземпляр класса EntitySchemaQuery. */
                    var esq = this.Ext.create("Terrasoft.EntitySchemaQuery", {
                        /* Установка корневой схемы. */
                        rootSchemaName: this.entitySchemaName
                    });
                    /* Добавление колонки Id. */
                    esq.addColumn("Id");
                    /* Добавление колонки Id из схемы Document. */
                    esq.addColumn("Document.Id", "DocumentId");
                    /* Создание и добавление фильтров в коллекцию запроса. */
                    esq.filters.add("filterOrder", this.Terrasoft.createColumnFilterWithParam
                        this.Terrasoft.ComparisonType.EQUAL, "UsrOrder", OrderId));
                    /* Получение всей коллекции записей и отображение ее в модальном окне спр
                    esq.getEntityCollection(function(result) {
                        var existsDocumentsCollection = [];
                        if (result.success) {
```

```
                    result.collection.each(function(item) {
                        existsDocumentsCollection.push(item.get("DocumentId"));
                    });
                }
                /* Добавление фильтра в конфигурационный объект. */
                if (existsDocumentsCollection.length > 0) {
                    var existsFilter = this.Terrasoft.createColumnInFilterWithParamet
                        existsDocumentsCollection);
                    existsFilter.comparisonType = this.Terrasoft.ComparisonType.NOT_E
                    existsFilter.Name = "existsFilter";
                    config.filters = existsFilter;
                }
                /* Вызов модального окна справочника. */
                this.openLookup(config, this.addCallBack, this);
            }, this);
        },

        /* Обработчик события сохранения страницы записи. */
        onCardSaved: function() {
            this.openDocumentLookup();
        },

        /* Открывает справочник документов в случае если страница заказа была ранее с
        addRecord: function() {
            var masterCardState = this.sandbox.publish("GetCardState", null, [this.sa
            var isNewRecord = (masterCardState.state === configurationEnums.CardState
            masterCardState.state === configurationEnums.CardStateV2.COPY);
            if (isNewRecord === true) {
                var args = {
                    isSilent: true,
                    messageTags: [this.sandbox.id]
                };
                this.sandbox.publish("SaveRecord", args, [this.sandbox.id]);
                return;
            }
            this.openDocumentLookup();
        },

        /* Добавление выбранных продуктов. */
        addCallBack: function(args) {
            /* Экземпляр класса пакетного запроса BatchQuery. */
            var bq = this.Ext.create("Terrasoft.BatchQuery");
            var OrderId = this.get("MasterRecordId");
            /* Коллекция выбранных в справочнике документов. */
            this.selectedRows = args.selectedRows.getItems();
            /* Коллекция, передаваемая в запрос. */
            this.selectedItems = [];
            /* Копирование необходимых данных. */
```

```
                this.selectedRows.forEach(function(item) {
                    item.OrderId = OrderId;
                    item.DocumentId = item.value;
                    bq.add(this.getDocumentInsertQuery(item));
                    this.selectedItems.push(item.value);
                }, this);
                /* Выполнение пакетного запроса, если он не пустой. */
                if (bq.queries.length) {
                    this.showBodyMask.call(this);
                    bq.execute(this.onDocumentInsert, this);
                }
            },

            /* Возвращает запрос на добавление текущего объекта. */
            getDocumentInsertQuery: function(item) {
                var insert = Ext.create("Terrasoft.InsertQuery", {
                    rootSchemaName: this.entitySchemaName
                });
                insert.setParameterValue("UsrOrder", item.OrderId, this.Terrasoft.DataVal
                insert.setParameterValue("UsrDocument", item.DocumentId, this.Terrasoft.D
                return insert;
            },

            /* Метод, вызываемый при добавлении записей в реестр детали. */
            onDocumentInsert: function(response) {
                this.hideBodyMask.call(this);
                this.beforeLoadGridData();
                var filterCollection = [];
                response.queryResults.forEach(function(item) {
                    filterCollection.push(item.id);
                });
                var esq = Ext.create("Terrasoft.EntitySchemaQuery", {
                    rootSchemaName: this.entitySchemaName
                });
                this.initQueryColumns(esq);
                esq.filters.add("recordId", Terrasoft.createColumnInFilterWithParameters(
                /* Создание модели представления. */
                esq.on("createviewmodel", this.createViewModel, this);
                esq.getEntityCollection(function(response) {
                    this.afterLoadGridData();
                    if (response.success) {
                        var responseCollection = response.collection;
                        this.prepareResponseCollection(responseCollection);
                        this.getGridData().loadAll(responseCollection);
                    }
                }, this);
            },

            /* Метод, вызываемый при удалении выбранных записей детали. */
```

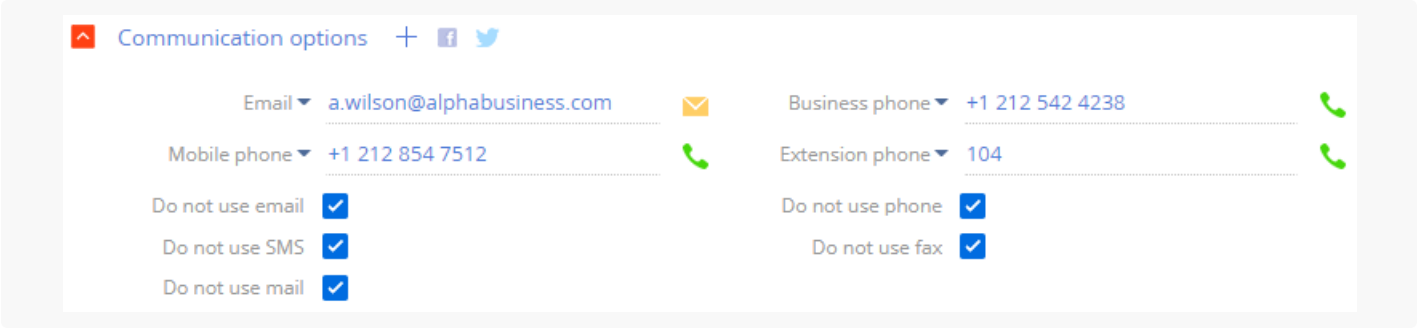
```
                deleteRecords: function() {
                    var selectedRows = this.getSelectedItems();
                    if (selectedRows.length > 0) {
                        this.set("SelectedRows", selectedRows);
                        this.callParent(arguments);
                    }
                },

                /* Скрыть пункт меню Копировать. */
                getCopyRecordMenuItem: Terrasoft.emptyFn,
                /* Скрыть пункт меню Изменить. */
                getEditRecordMenuItem: Terrasoft.emptyFn,
                /* Возвращает имя колонки по умолчанию для фильтра. */
                getFilterDefaultColumnName: function() {
                    return "UsrDocument";
                }
            },
            /* Массив модификаций. */
            diff: /**SCHEMA_DIFF*/[
                {
                    /* Тип операции — слияние. */
                    "operation": "merge",
                    /* Название элемента схемы, над которым производится действие. */
                    "name": "DataGrid",
                    /* Объект, свойства которого будут объединены со свойствами элемента схем
                    "values": {
                        "rowDataItemMarkerColumnName": "UsrDocument"
                    }
                },
                {
                    /* Тип операции — слияние. */
                    "operation": "merge",
                    /* Название элемента схемы, над которым производится действие. */
                    "name": "AddRecordButton",
                    /* Объект, свойства которого будут объединены со свойствами элемента схем
                    "values": {
                        "visible": {"bindTo": "getToolsVisible"}
                    }
                }
            ]/**SCHEMA_DIFF*/
        };
    }
);
```

3. Выполните шаг 4 общего [алгоритма реализации детали с использованием Creatio IDE](). Без выполнения этого шага деталь отображается на странице записи, но не содержит записей, поскольку не указаны колонки для отображения.

Реализовать деталь с полями

Деталь с полями позволяет вводить и редактировать данные непосредственно в полях детали. Может содержать несколько групп полей. Примером детали с полями является деталь [*Средства связи*] ([*Communication options*]) страницы контакта.



Действия, которые позволяет выполнять деталь с полями:

- Добавить записи на деталь без сохранения страницы, которая содержит текущую деталь.
- Работать с деталью, как со страницей записи.
- Использовать базовую валидацию полей.
- Реализовать пользовательскую валидацию полей.
- Добавить виртуальную запись.
- Расширить логику поведения записей.

Реализовать деталь с полями невозможно исключительно в мастере деталей, поскольку по умолчанию через мастер деталей создается деталь с реестром. Для реализации необходимо использовать комбинацию мастера деталей и Creatio IDE.

Реализация детали с полями для продуктов линейки Financial Services Creatio имеет свои особенности. Функциональность базовой детали с полями продуктов линейки Financial Services Creatio реализована в схеме `BaseFieldsDetail` пакета `BaseFinance`. Модель представления записи детали с полями реализована в схеме `BaseFieldRowViewModel` пакета `BaseFinance`.

Использовать комбинацию мастера деталей и Creatio IDE для реализации детали с полями

1. Для реализации детали с полями в CRM продуктах Creatio скачайте пакет `sdkFieldsDetailPackage`.
2. Для реализации детали с полями в CRM продуктах Creatio импортируйте пакет в пользовательское приложение. Для этого воспользуйтесь инструкцией, которая приведена в статье Перенести пакеты.
3. Для реализации детали с полями в CRM продуктах Creatio добавьте пакет `sdkFieldsDetailPackage` в зависимости пользовательского пакета. Для этого воспользуйтесь инструкцией, которая приведена в статье Создать пользовательский пакет.
4. Выполните шаг 1 общего алгоритма реализации детали с использованием мастера деталей. При необходимости, выполните настройку колонки детали, использовав Creatio IDE.
5. Используя Creatio IDE, замените родительский объект объекта детали на `BaseFieldsDetail`.

6. Выполните шаг 2 общего алгоритма реализации детали с использованием мастера деталей.

Использовать Creatio IDE для реализации детали с полями

1. Для реализации детали с полями в CRM продуктах Creatio выполните шаги 1-3 алгоритма реализации детали с полями с помощью комбинации мастера деталей и Creatio IDE.
2. **Создайте пользовательскую деталь**.

   a. Создайте схему объекта детали.

   - В качестве родительского объекта выберите `BaseEntity`.
   - В схему объекта добавьте необходимые колонки.

   d. Создайте схему модели представления детали с полями.

   - В качестве родительского объекта выберите `BaseFieldsDetail`.
   - На панели инструментов в контекстном меню узла [*Локализуемые строки*] ([*Localizable strings*]) выберите локализуемую строку [*Caption*] и в свойстве [*Значение*] ([*Value*]) задайте название детали.
   - В свойство `methods` добавьте метод `getDisplayColumns`, который возвращает массив с названиями колонок, отображающихся как поля в детали.

   h. Добавьте пользовательские стили детали (опционально).

   a. Создайте схему модуля, в которой определите стили. Для этого воспользуйтесь инструкцией, которая приведена в статье Разработка конфигурационных элементов.

   b. Укажите наследуемый класс.

   - Для реализации детали с полями в CRM продуктах Creatio укажите `UsrBaseFieldRowViewModel`.
   - Для реализации детали с полями в продуктах линейки Financial Services Creatio укажите `BaseFieldRowViewModel`.

   e. В зависимости схемы модели представления реестра детали добавьте схему модуля с реализацией стилей.

   f. В свойство `methods` добавьте методы переопределения базовых CSS-классов стилей:

   - `getRowViewModelClassName()` — возвращает имя класса модели представления записи на детали.
   - `getLeftRowContainerWrapClass()` — возвращает массив строк с названиями CSS-классов, используемых для генерации представления контейнеров, содержащих подписи полей записей.

   i. Зарегистрируйте деталь в базе данных. Для этого выполните SQL-запрос к таблице `[SysDetails]` базы данных.

SQL-запрос

```
DECLARE
```

```
-- Название схемы детали.
@ClientUnitSchemaName NVARCHAR(100) = 'UsrИмяСхемыДетали',
-- Название схемы объекта детали.
@EntitySchemaName NVARCHAR(100) = 'UsrИмяСхемыОбъектаДетали',
-- Название детали.
@DetailCaption NVARCHAR(100) = 'ИмяДетали'

INSERT INTO SysDetail(
    Caption,
    DetailSchemaUId,
    EntitySchemaUId
)
VALUES(
    @DetailCaption,
    (SELECT TOP 1 UId
    from SysSchema
    WHERE Name = @ClientUnitSchemaName),
    (SELECT TOP 1 UId
    from SysSchema
    WHERE Name = @EntitySchemaName)
)
```

Регистрация детали выполняется, чтобы деталь стала видимой для мастера разделов и мастера деталей.

3. **Добавьте пользовательскую деталь на страницу записи**.

   Создайте схему замещающей модели представления страницы записи, на которой будет размещена деталь с полями.

   - В качестве родительского объекта выберите схему модели представления, которую необходимо заместить.
   - В свойство `details` добавьте деталь.
   - В массив модификаций `diff` добавьте конфигурационный объект модели представления детали.

   Пример схемы `ContactPageV2` замещающей модели представления страницы записи, на которой размещена деталь с полями `UsrRegDocumentFieldsDetail` приведен ниже.

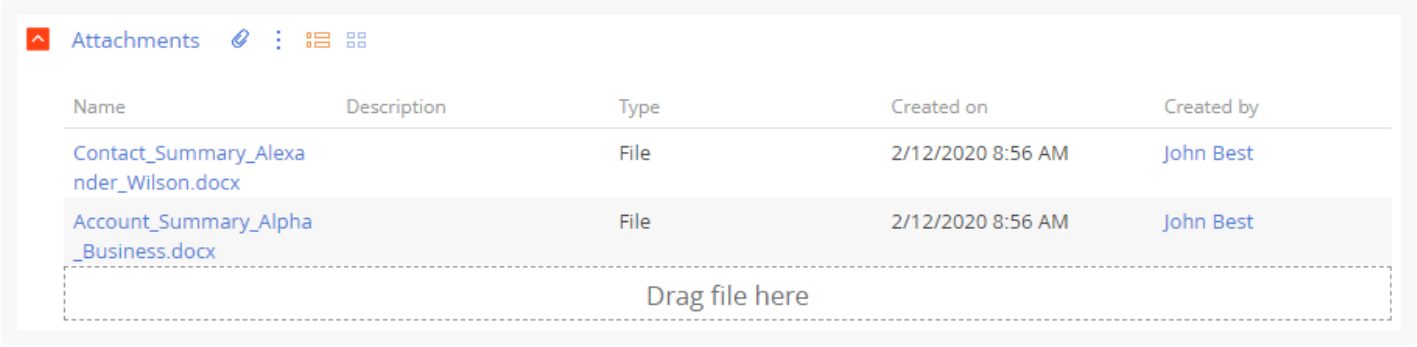
   Пример схемы замещающей модели представления

```
define("ContactPageV2", [], function() {
    return {
        entitySchemaName: "Contact",
        details: /**SCHEMA_DETAILS*/ {
            /* Добавление детали с полями. */
            "UsrRegDocumentFieldsDetail": {
                /* Название клиентской схемы детали. */
                "schemaName": "UsrRegDocumentFieldsDetail",
```

```
                /* Фильтрация записей детали текущего контакта (физ. лица). */
                "filter": {
                    /* Колонка объекта детали. */
                    "detailColumn": "UsrContact",
                    /* Колонка идентификатора контакта. */
                    "masterColumn": "Id"
                }
            }
        } /**SCHEMA_DETAILS*/ ,
        diff: /**SCHEMA_DIFF*/ [{
            /* Добавление нового элемента. */
            "operation": "insert",
            /* Название элемента. */
            "name": "UsrRegDocumentFieldsDetail",
            /* Конфигурационный объект значений. */
            "values": {
                /* Тип элемента. */
                "itemType": Terrasoft.ViewItemType.DETAIL
            },
            /* Имя элемента-контейнера. */
            "parentName": "HistoryTab",
            /* Имя свойства элемента-контейнера, который содержит коллекцию вложенных элемент
            "propertyName": "items",
            /* Индекс добавляемого в коллекцию элемента. */
            "index": 0
        }] /**SCHEMA_DIFF*/
    };
});
```

Реализовать деталь типа [*Файлы и ссылки*] ([*Attachments*])

Деталь типа [*Файлы и ссылки*] ([*Attachments*]) позволяет хранить внешние файлы, ссылки на веб-ресурсы и статьи базы знаний. Доступна во всех разделах приложения. Функциональность базовой детали типа [*Файлы и ссылки*] ([*Attachments*]) реализована в схеме `FileDetailV2` пакета `UIv2`. Деталь типа [*Файлы и ссылки*] ([*Attachments*]) описана в статье Файлы и примечания. Примером детали типа [*Файлы и ссылки*] ([*Attachments*]) является деталь [*Файлы и ссылки*] ([*Attachments*]) страницы контакта.



Реализовать деталь типа [*Файлы и ссылки*] ([*Attachments*]) невозможно исключительно в мастере деталей, поскольку по умолчанию через мастер деталей создается деталь с реестром. Для реализации необходимо использовать комбинацию мастера деталей и Creatio IDE.

Чтобы реализовать деталь типа [*Файлы и ссылки*] ([*Attachments*]) с использованием **комбинации мастера деталей и Creatio IDE**:

1. **Создайте пользовательскую деталь**.

   a. Настройте объект детали типа [*Файлы и ссылки*] ([*Attachments*]).

      - Для этого в свойстве [*По какому объекту создать деталь?*] ([*How to create detail?*]) выберите "`Существующему объекту`" ("`Based on existing object`").
      - Для этого в свойстве [*Объект*] ([*Object*]) выберите "`Файл и ссылка объекта [ИмяПользовательскогоРаздела]`" ("`[CustomSectionName] attachment`").

   d. Используя Creatio IDE, замените родительский объект страницы записи детали на `FileDetailV2`.

   e. Добавьте пользовательские стили детали (опционально).

      a. Создайте схему модуля, в которой определите стили. Для этого воспользуйтесь инструкцией, которая приведена в статье Разработка конфигурационных элементов.

      b. В зависимости схемы модели представления детали добавьте схему модуля с реализацией стилей.

2. Выполните шаг 2 общего алгоритма реализации детали с использованием мастера деталей.

Реализовать множественное добавление записей на деталь

По умолчанию деталь позволяет добавлять только одну запись. **Назначение** миксина `LookupMultiAddMixin` — расширение действия по добавлению записи на деталь. Использование миксина позволяет пользователю выбирать несколько записей из справочника за один раз.

Чтобы **реализовать множественное добавление записей на деталь**:

1. Создайте схему замещающей модели представления детали. Для этого воспользуйтесь инструкцией, которая приведена в статье Разработка конфигурационных элементов.
2. В свойство `mixins` добавьте миксин `LookupMultiAddMixin`.
3. В свойстве `methods`:

   - Переопределите методы.

     - `init()` — реализует логику, выполняемую при загрузке модуля. В методе выполните инициализацию миксина `LookupMultiAddMixin`. Метод `init()` описан в статье Виды модулей.
     - `getAddRecordButtonVisible()` — отвечает за отображение кнопки добавления.
     - `onCardSaved()` — отвечает за сохранение страницы детали. В переопределенном методе используйте метод `openLookupWithMultiSelect()`, который вызывает справочное окно для множественного выбора.

- `addRecord()` — отвечает за добавление записи на деталь. Как и для метода `onCardSaved()`, в переопределенном методе используйте метод `openLookupWithMultiSelect()`. Значение `true` указывает на необходимость выполнения проверки является ли запись новой.

- Реализуйте метод `getMultiSelectLookupConfig()`, который связан с методом `openLookupWithMultiSelect()`. Метод `getMultiSelectLookupConfig()` выполняет конфигурирование справочного окна и возвращает объект конфигурации для справочного окна.

Удалить деталь

Важно. Удаление детали невозможно без доступа к конфигурации системы и базе данных.

Чтобы **удалить деталь**:

1. В SVN-хранилище снимите блокировку с файлов детали, которую необходимо удалить.
2. Удалите записи из базы данных. Для этого выполните SQL-запрос в базу данных.

   SQL-запрос

   ```
   DECLARE @Caption nvarchar(max);
   SET @Caption = 'UsrИмяСхемыДетали';
   DECLARE @UId UNIQUEIDENTIFIER;
   select @UId = EntitySchemaUId from SysDetail
   where Caption = @Caption
   delete from SysDetail where EntitySchemaUId = @UId
   ```

3. Перейдите в раздел [*Конфигурация*] ([*Configuration*]) и удалите схему модели представления детали и схему объекта детали.

Настроить деталь с полями

Сложный

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.

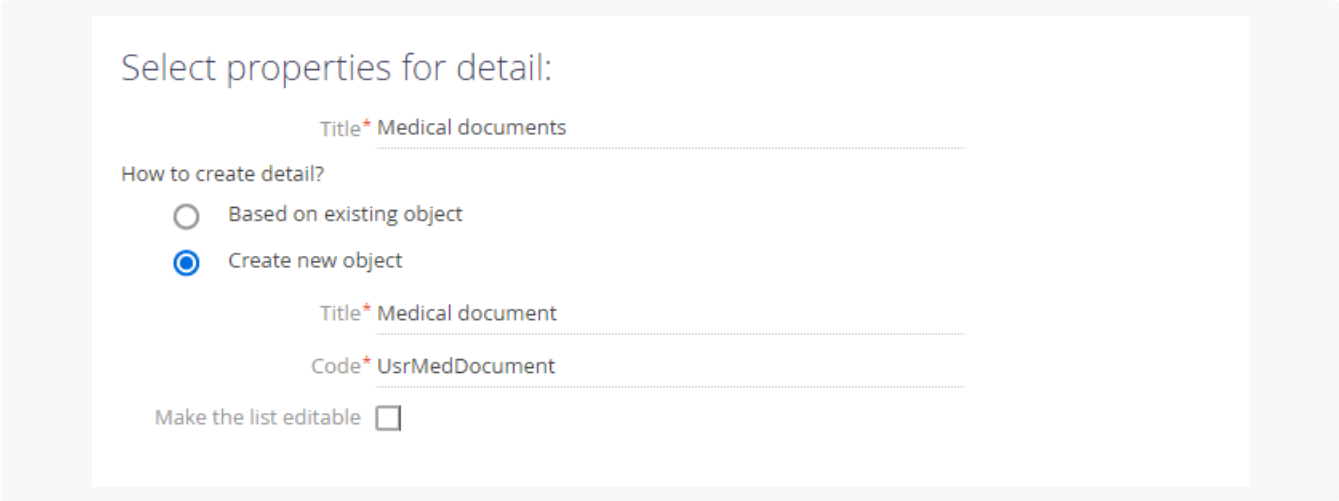
1. Создать пользовательскую деталь

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`".

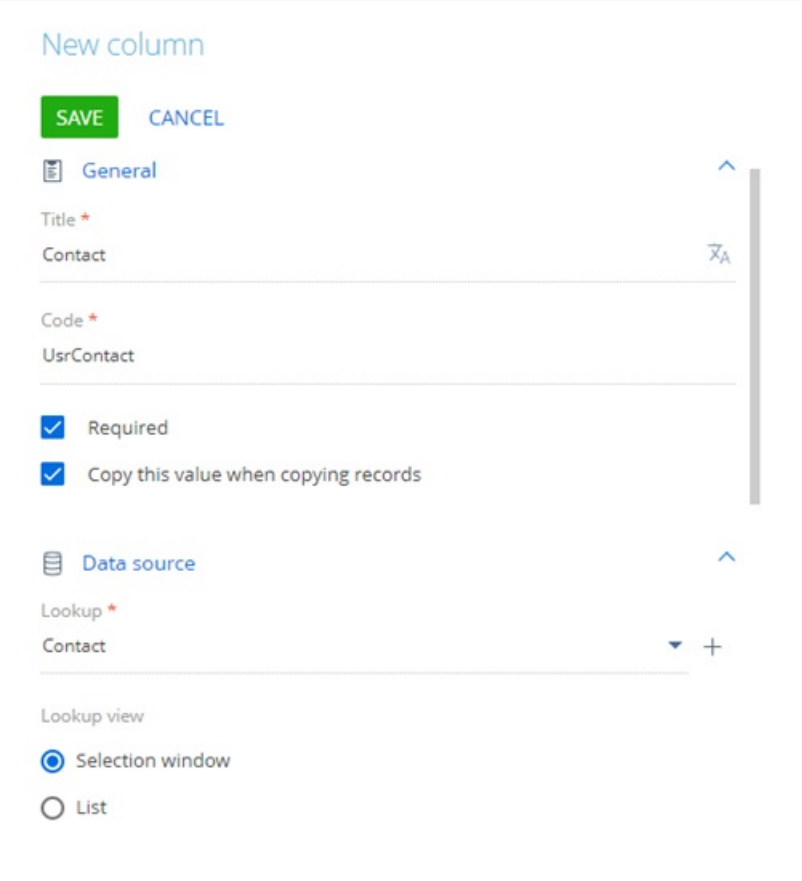


После сохранения в конфигурации будут созданы:

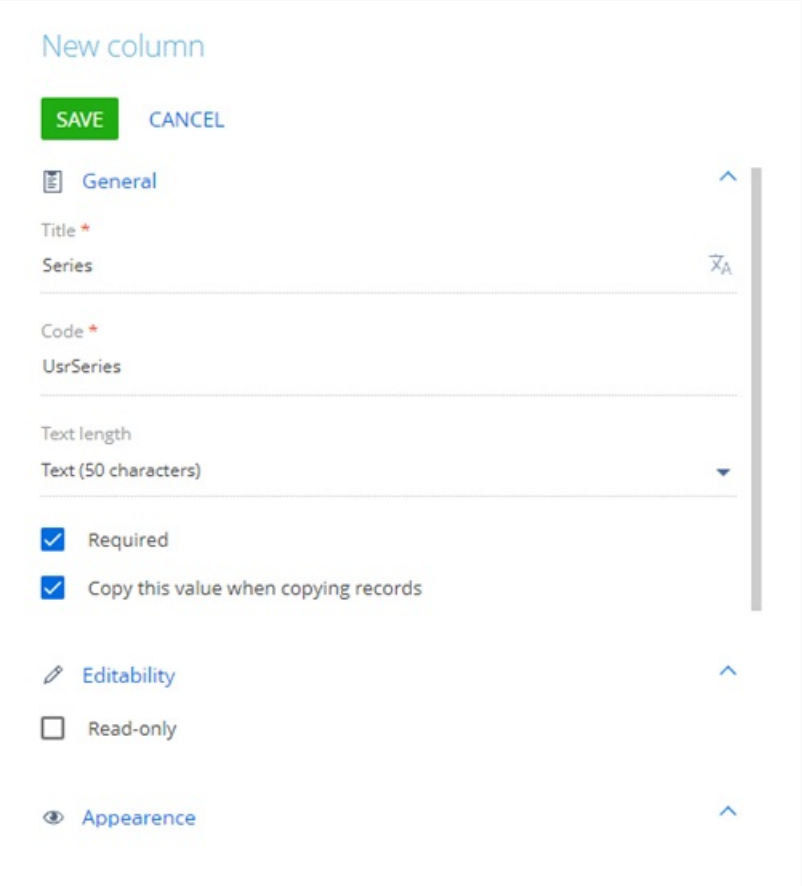
- Схема `UsrMedDocument` объекта детали.
- Схема `UsrSchemac6fd3fd0Detail` модели представления реестра детали [*Медицинские документы*] ([*Medical documents*]).
- Схема `UsrUsrMedDocument4988cee4Page` модели представления страницы записи детали [*Медицинские*

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`").



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

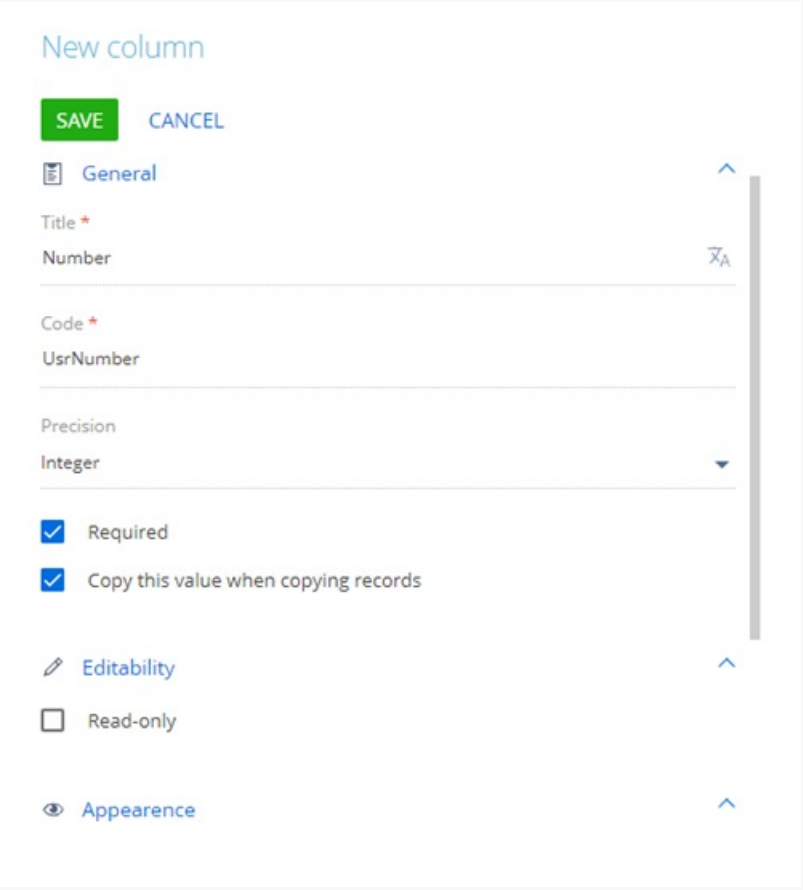


c. Заполните свойства поля [*Number*] типа [*Целое число*] ([*Integer*]).

   a. [*Заголовок*] ([*Title*]) — "Номер" ("Number").

   b. [*Код*] ([*Code*]) — "UsrNumber".

   c. Установите признак [*Обязательное*] ([*Required*]).

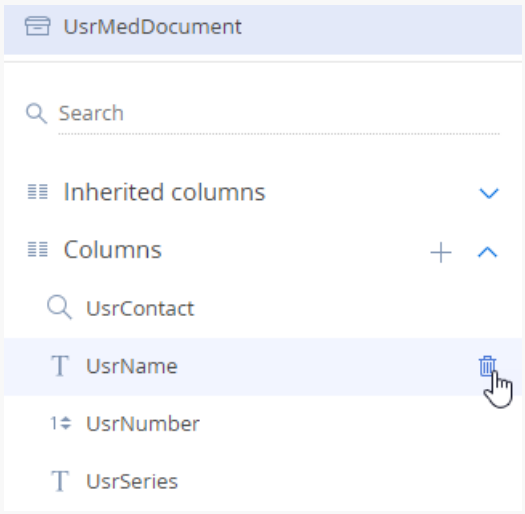


7. При необходимости, измените расположение полей детали.
8. На панели инструментов мастера деталей нажмите [*Сохранить*] ([*Save*]).

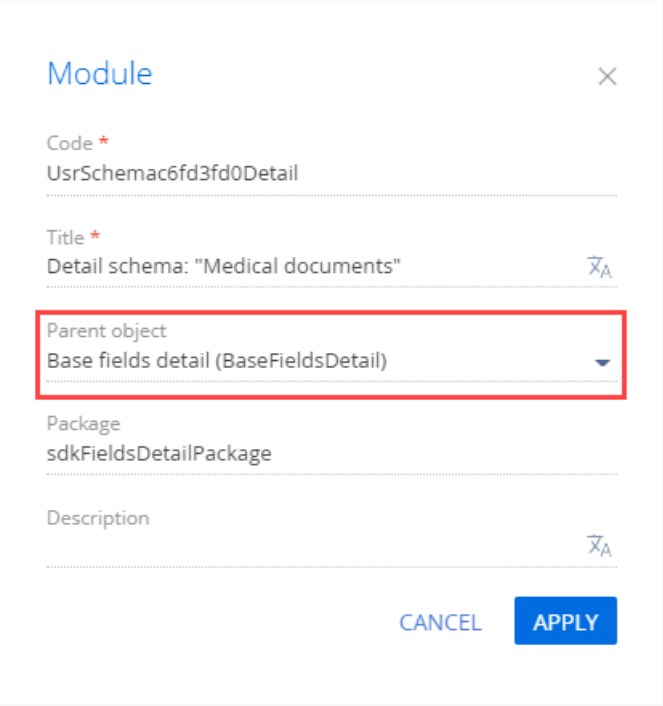
После сохранения в конфигурации будет модифицирована схема `UsrUsrMedDocument4988cee4Page` модели представления страницы записи детали [*Медицинские документы*] ([*Medical documents*]).

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("UsrSchemac6fd3fd0Detail", [], function() {
    return {
        entitySchemaName: "UsrMedDocument",
        details: /**SCHEMA_DETAILS*/{}/**SCHEMA_DETAILS*/,
        diff: /**SCHEMA_DIFF*/[]/**SCHEMA_DIFF*/,
        methods: {
            getDisplayColumns: function() {
                return ["UsrSeries", "UsrNumber"];
            }
        }
    };
});
```

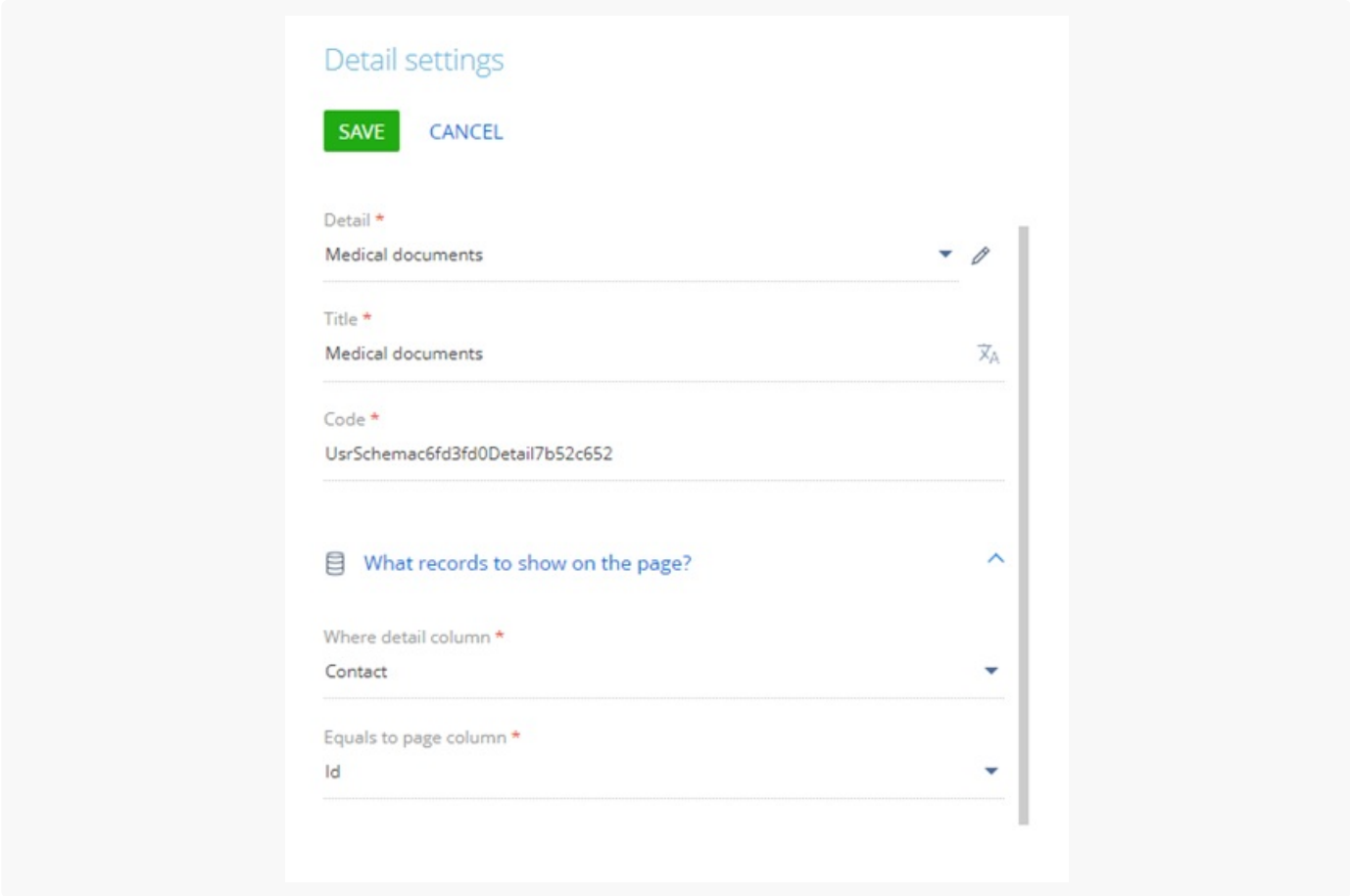
8. На панели инструментов дизайнера модуля нажмите [*Сохранить*] ([*Save*]).

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.

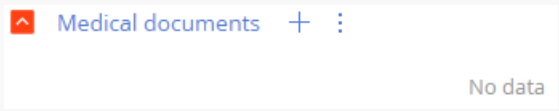
   - [*Деталь*] ([*Detail*]) — выберите "Медицинские документы" ("Medical documents"). Поля [*Заголовок*] ([*Title*]) и [*Код (на английском)*] ([*Code*]) заполнятся автоматически.
   - [*У которых колонка детали*] ([*Where detail column*]) — выберите "Физ. лицо" ("Contact").

     Значения остальных колонок оставьте без изменений.



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.



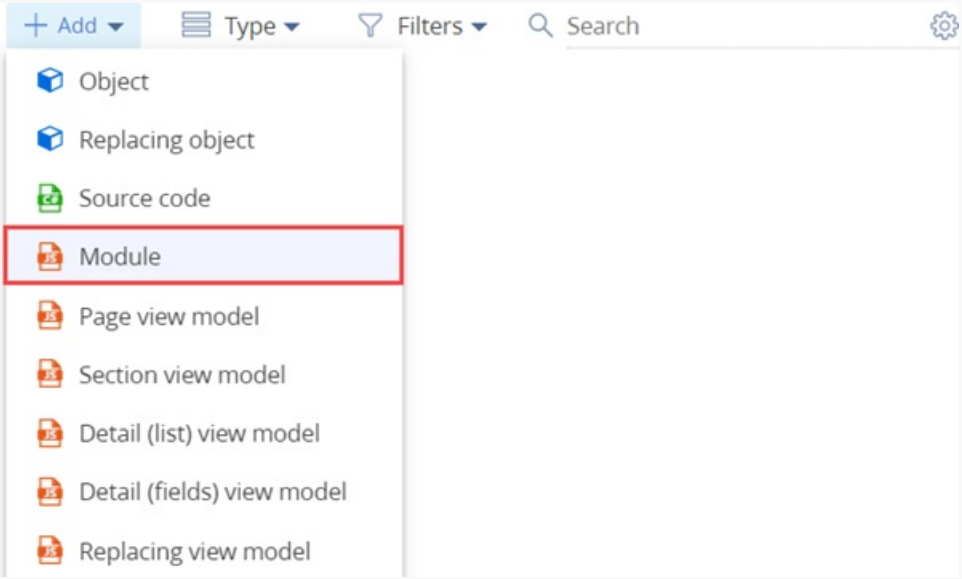
4. Добавить пользовательские стили детали

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.

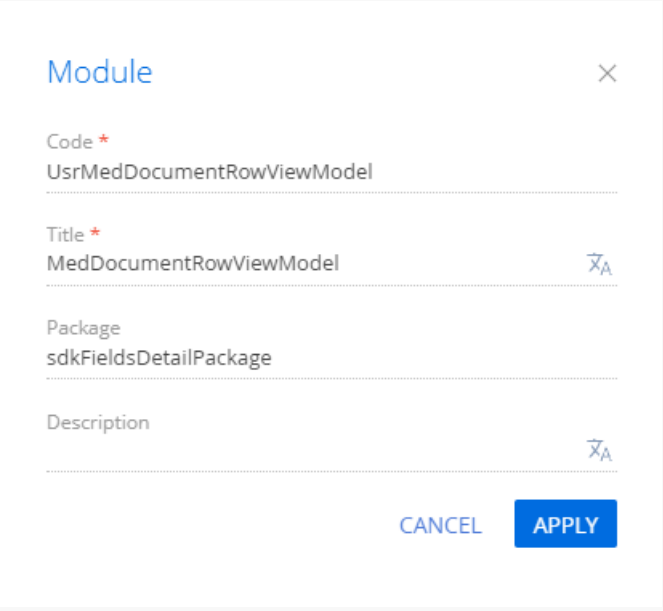
1. Создать схему модуля

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 дизайнере 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*]).
2. В зависимости схемы `UsrSchemac6fd3fd0Detail` добавьте модуль `UsrMedDocumentRowViewModel`.
3. 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.

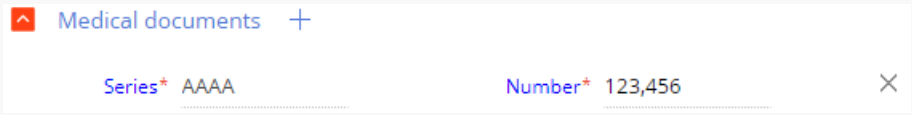
Исходный код модифицированной схемы представлен ниже.

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

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

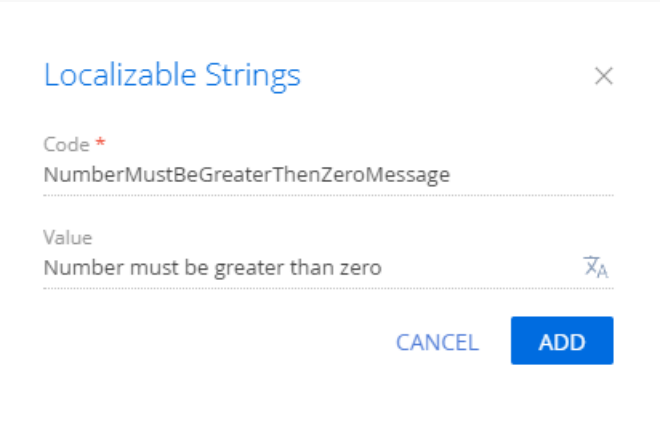
В результате названия полей детали [*Медицинские документы*] ([*Medical documents*]), которая была добавлена на вкладку [*История*] ([*History*]) страницы физического лица, отображаются синим цветом.



5. Добавить валидацию к полю детали

1. 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*]) — "NumberMustBeGreaterThenZeroMessage".
- [*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.

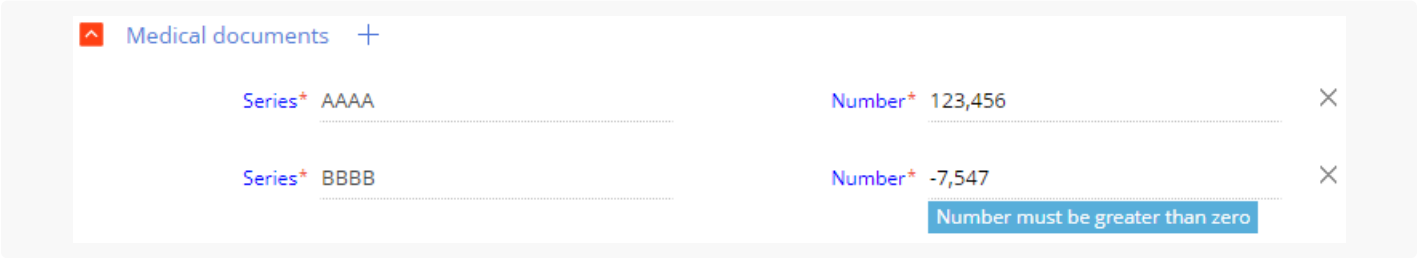
`UsrMedDocumentFieldsDetail`

```
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.NumberMustBeGreaterThenZeroMess
            }
            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.

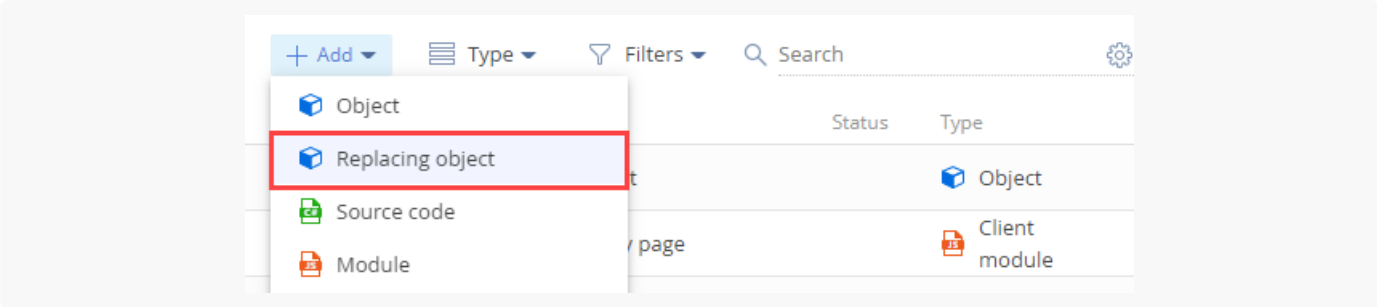
Добавить редактируемый реестр в деталь

Средний

Пример. На странице добавления продукта (деталь [*Продукты*] ([*Products*])) в разделе [*Заказы*] ([*Orders*]) в колонку [*Скидка, %*] ([*Discount, %*]), которая уже присутствует в схеме объекта продукта. добавить редактируемый реестр. Также добавить редактируемый реестр в пользовательскую колонку [*Пользовательская цена*] ([*Custom price*]).

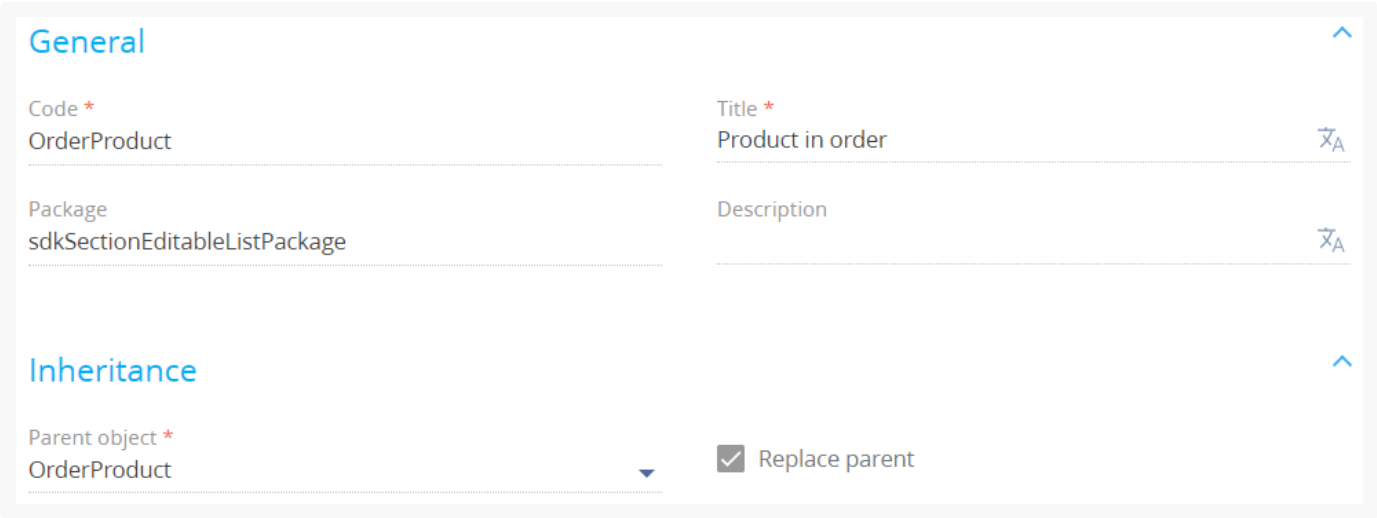
1. Создать схему замещающего объекта

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



3. Заполните **свойства схемы**.

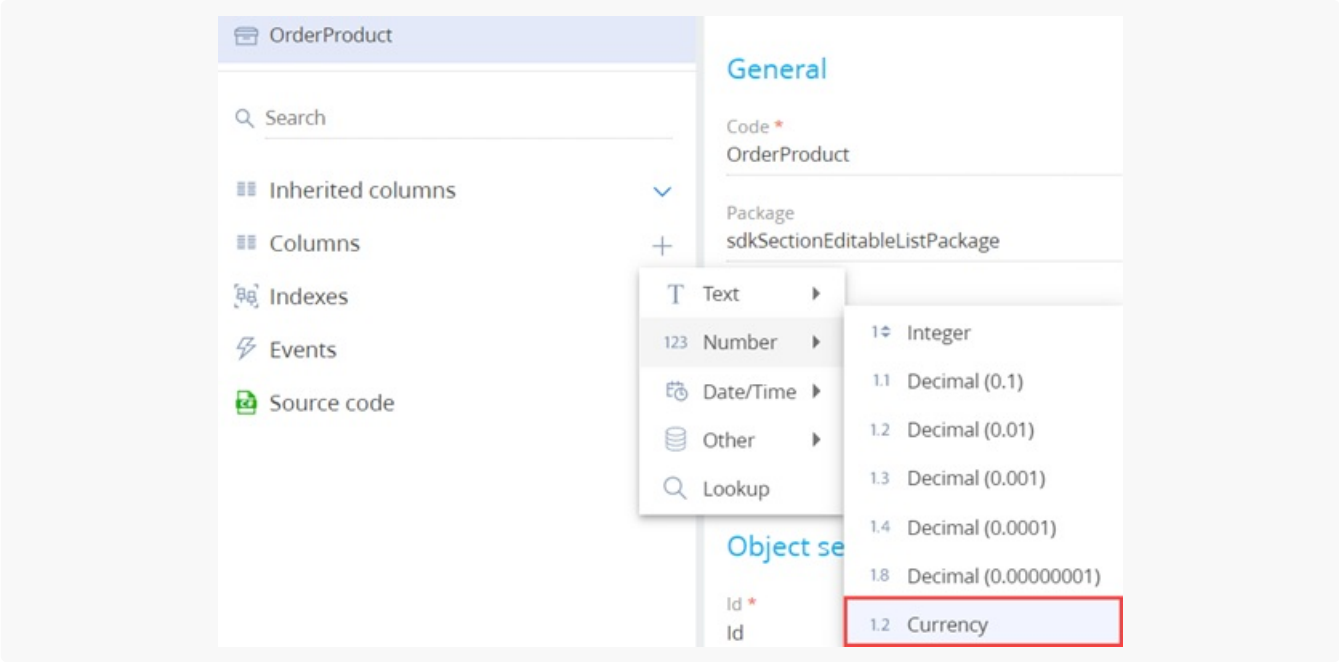
   - [*Код*] ([*Code*]) — "`OrderProduct`".
   - [*Заголовок*] ([*Title*]) — "`Продукт в заказе`" ("`Product in order`").
   - [*Родительский объект*] ([*Parent object*]) — выберите "`OrderProduct`".



4. В схему добавьте **колонку**.

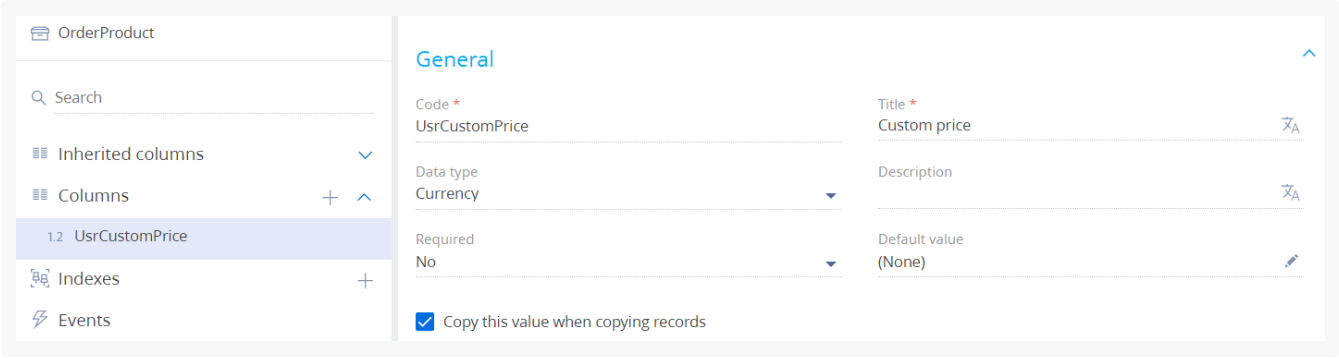
   a. В контекстном меню узла [*Колонки*] ([*Columns*]) структуры объекта нажмите +.

   b. В выпадающем меню нажмите [*Число*] —> [*Деньги*] ([*Number*] —> [*Currency*]).



c. Заполните **свойства добавляемой колонки**.

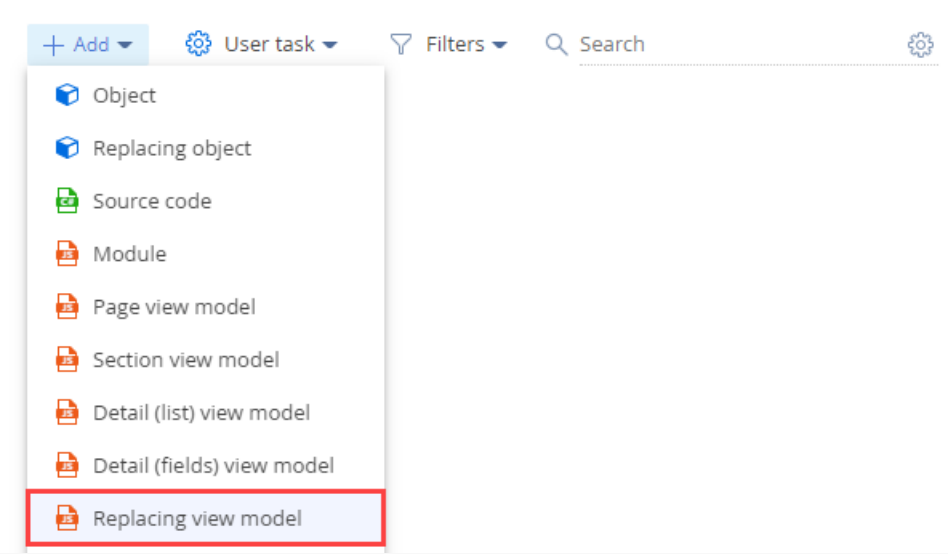
- [*Код*] ([*Code*]) — "`UsrCustomPrice`".
- [*Заголовок*] ([*Title*]) — "`Пользовательская цена`" ("`Custom price`").



5. На панели инструментов дизайнера объектов нажмите [*Сохранить*] ([*Save*]), а затем [*Опубликовать*] ([*Publish*]).

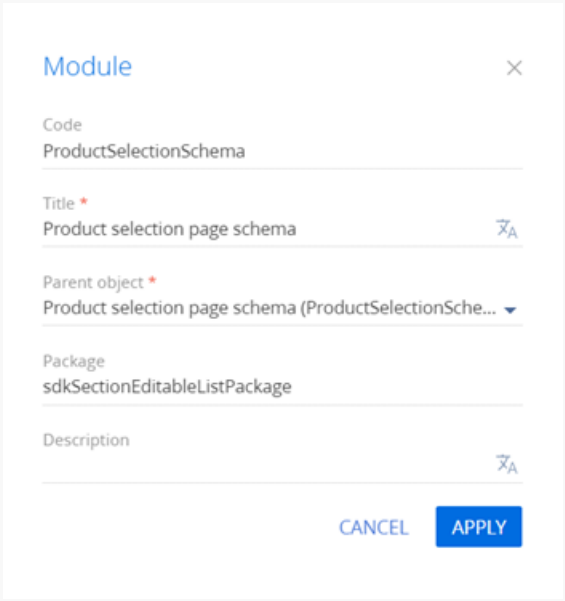
2. Создать схему замещающей модели представления раздела

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



4. Реализуйте **редактируемый реестр**. Для этого в свойстве `methods` реализуйте **методы**:

   - `getEditableColumns()` — получает массив редактируемых колонкок и добавляет пользовательскую колонку в массив.
   - `setColumnHandlers()` — привязывает обработчик события изменения пользовательской колонки.
   - `onCustomPriceChanged()` — метод-обработчик, который вызывается при изменении значения поля.

   Исходный код схемы замещающей модели представления раздела представлен ниже.

ProductSelectionSchema

```
define("ProductSelectionSchema", [], function() {
    return {
        methods: {
            getEditableColumns: function() {
                /* Получает массив редактируемых колонок. */
                var columns = this.callParent(arguments);
                /* Добавляет колонку [Скидка, %] в массив редактируемых колонок. */
                columns.push("DiscountPercent");
                /* Добавляет пользовательскую колонку. */
                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 вызывается при изменении значения поля. */
            onCustomPriceChanged: function(item, value) {
                window.console.log("Changed: ", item, value);
            }
        }
    };
});
```

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

Результат выполнения примера

Чтобы **посмотреть результат выполнения примера**:

1. Обновите страницу раздела [*Заказы*] ([*Orders*]).
2. Настройте **колонки страницы добавления продукта**.

   a. На панели инструментов раздела нажмите [*Вид*] —> [*Настроить колонки*] ([*View*] —> [*Select fields to display*]) и на странице настройки колонок перейдите в режим настройки плиточного представления реестра раздела ([*Плиточное представление*] ([*Tile view*])).

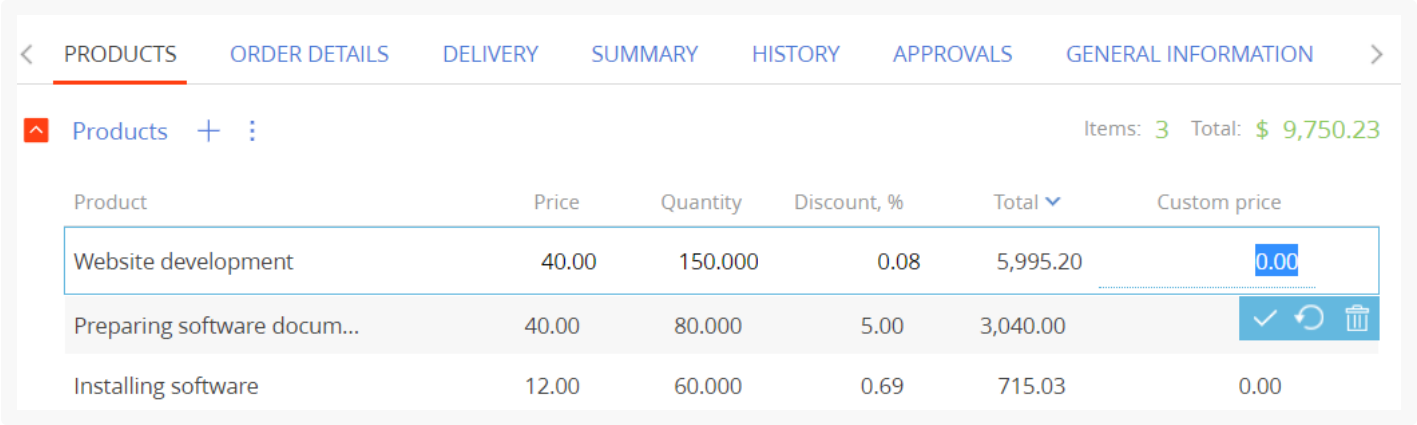
   b. Добавьте колонку в реестр раздела. Для этого нажмите на кнопку  . Затем нажмите на кнопку + и в поле [*Выберите объект*] ([*Select object*]) выберите объект [*Продукт в заказе*] ([*Product in order*]).

   c. В поле [*Колонка*] ([*Column*]) выберите колонку [*Скидка, %*] ([*Discount, %*]).

   d. Аналогично добавьте колонку [*Пользовательская цена*] ([*Custom price*]).

В результате выполнения примера на странице добавления продукта (деталь [*Продукты*] ([*Products*]))

в разделе [*Заказы*] ([*Orders*]) в реестр добавлены редактируемые колонки [*Скидка, %*] ([*Discount, %*]) и [*Пользовательская цена*] ([*Custom price*]).



Скрыть пункты меню детали с реестром

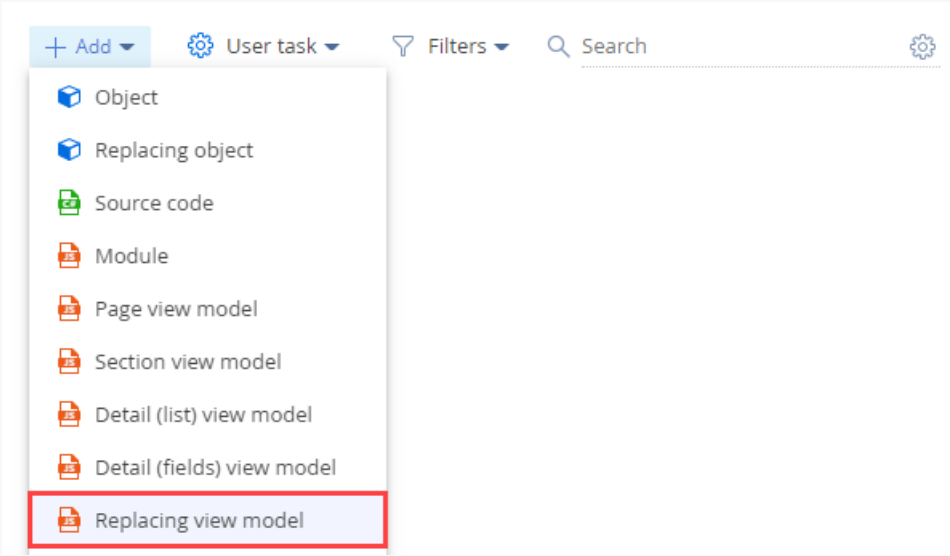
Средний

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.

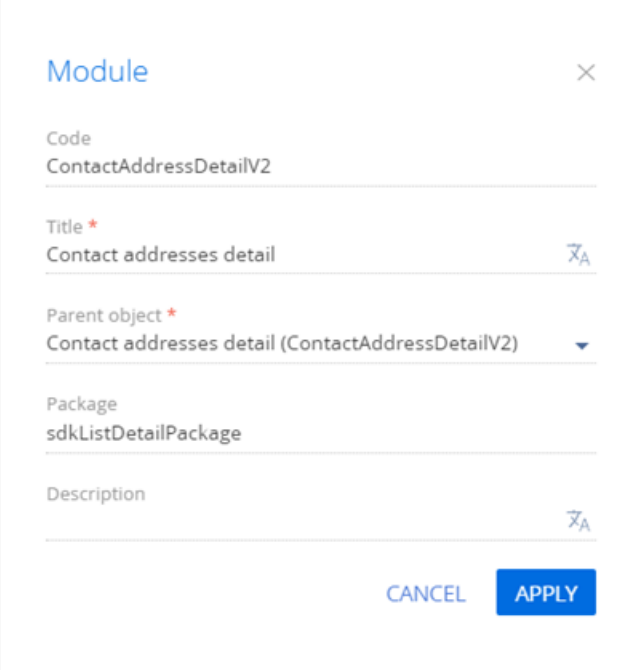
Создать схему замещающей модели представления реестра детали

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 дизайнере модуля добавьте исходный код.

ContactAddressDetailV2

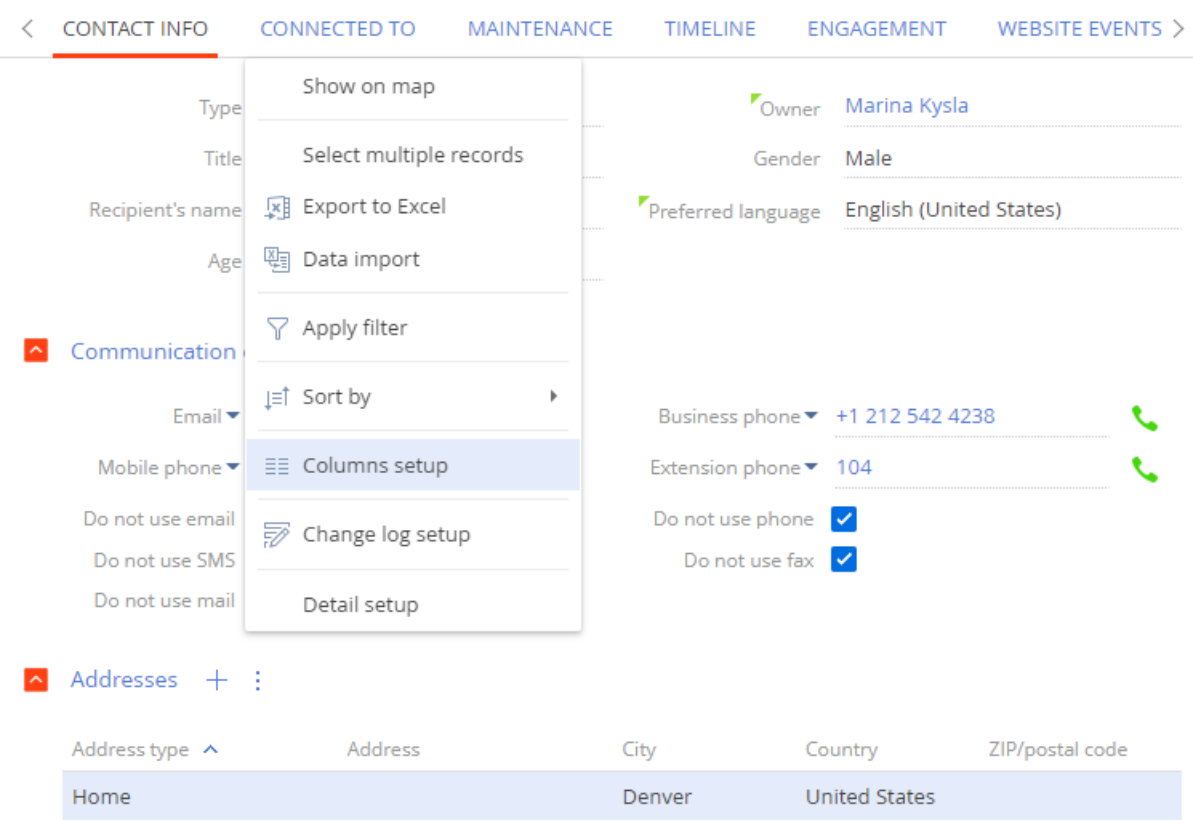
```
define("ContactAddressDetailV2", [], function() {
    return {
        entitySchemaName: "AccountAddress",
```

```
        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*]), [*Удалить*] ([*Delete*]) скрыты из меню детали [*Адреса*] ([*Addresses*]).



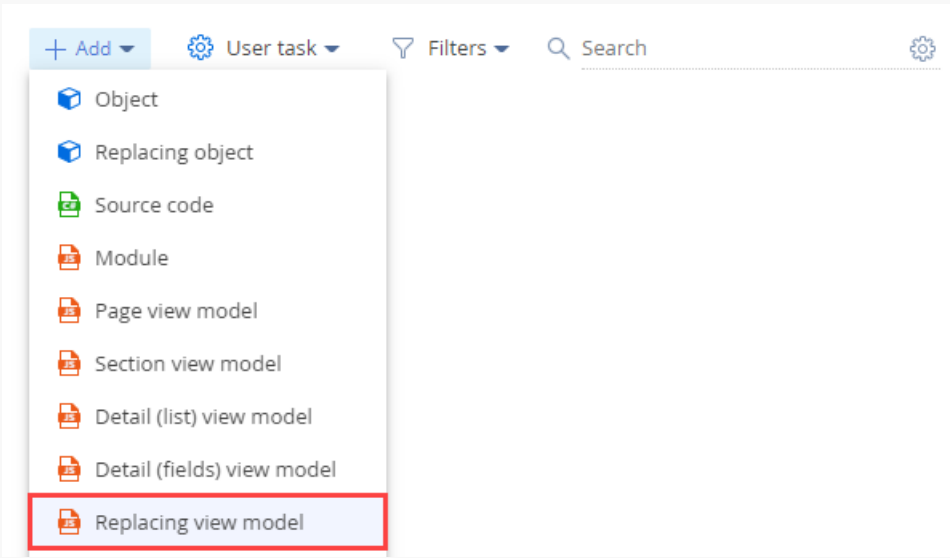
Реализовать множественное добавление записей на деталь

Сложный

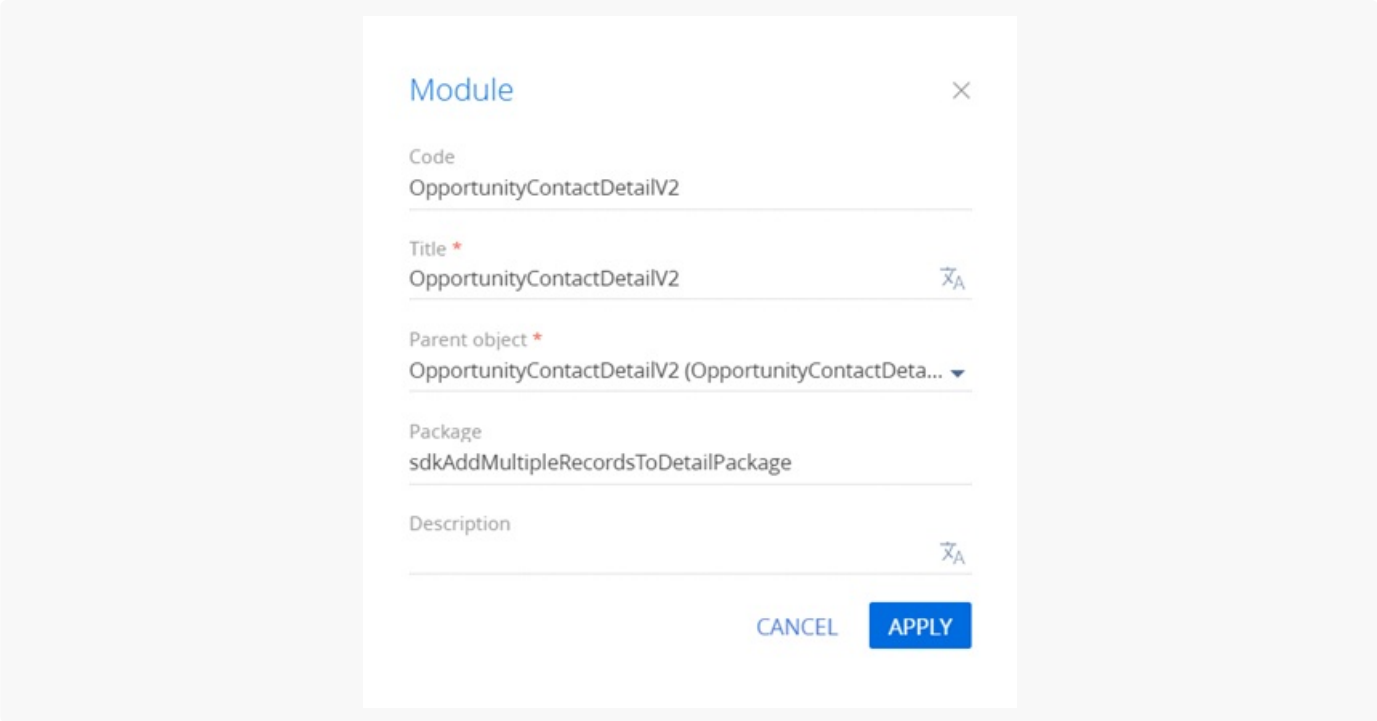
Пример. Реализовать множественное добавление записей на деталь [*Контакты*] ([*Contacts*]) страницы записи раздела [*Продажи*] ([*Opportunities*]).

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`. Метод `init()` описан 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

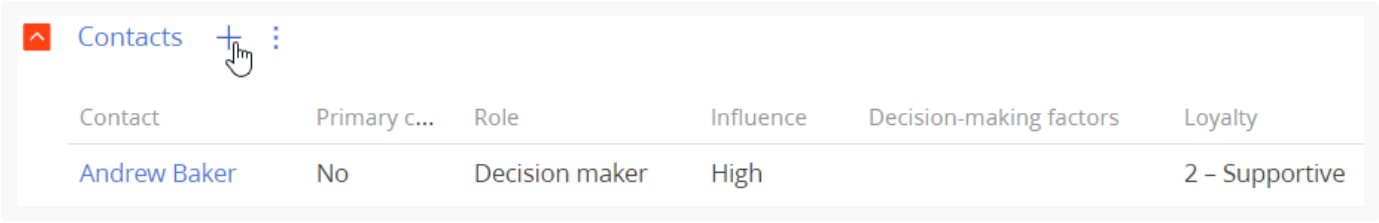
```
define("OpportunityContactDetailV2", ["LookupMultiAddMixin"], function() {
    return {
        mixins: {
            /* Подключение миксина к схеме. */
            LookupMultiAddMixin: "Terrasoft.LookupMultiAddMixin"
        },
        methods: {
            /* Переопределение базового метода инициализации 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"
                };
```

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

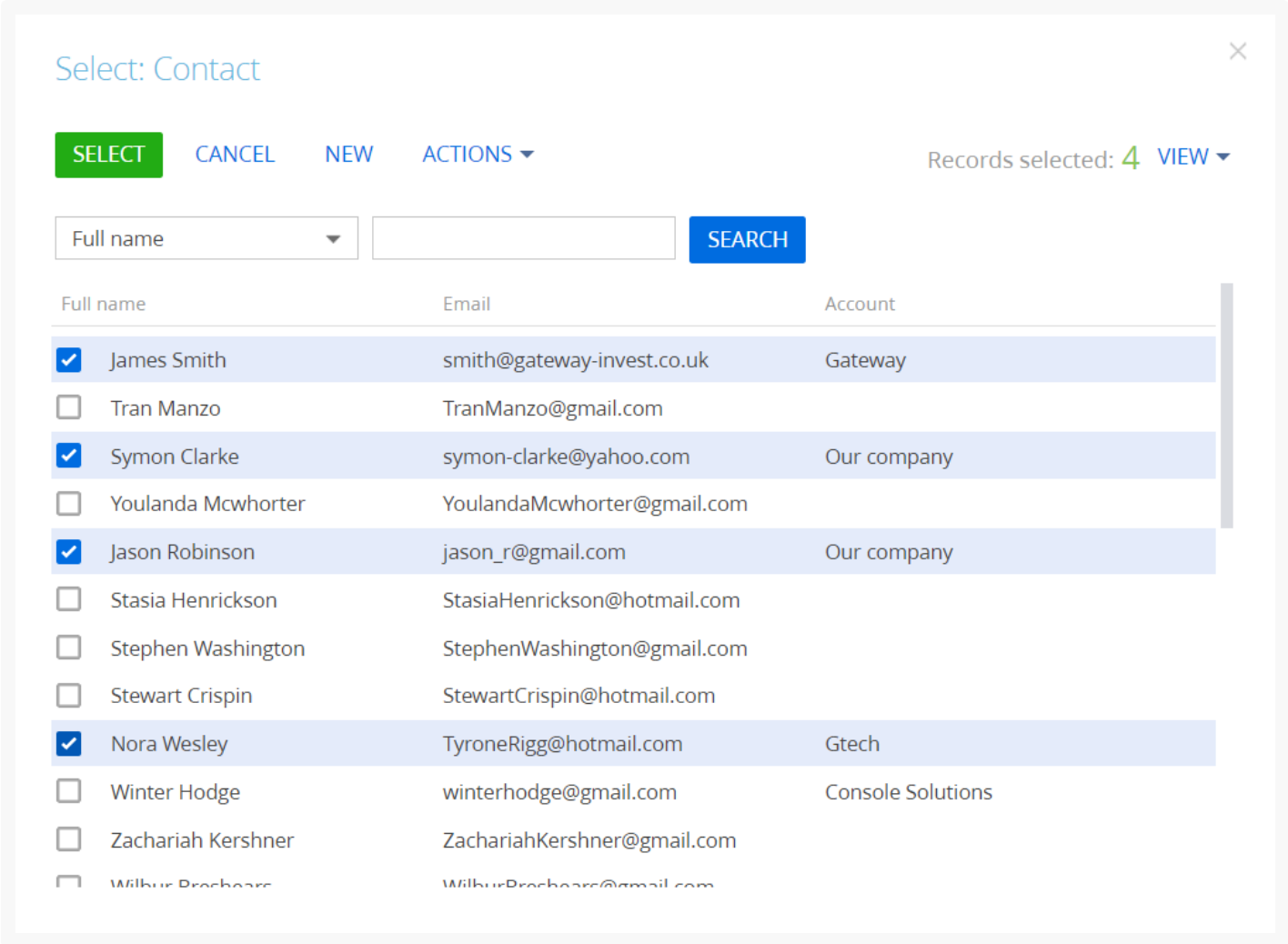
На панели инструментов дизайнера модуля нажмите [*Сохранить*] ([*Save*]).

Результат выполнения примера

1. Обновите страницу приложения.
2. На детали [*Контакты*] ([*Contacts*]) страницы записи раздела [*Продажи*] ([*Opportunities*]) нажмите кнопку +.



В результате выполнения примера приложение позволяет выбрать несколько записей из справочника.



После подтверждения выбранные записи добавляются на деталь [*Контакты*] ([*Contacts*]) на странице записи раздела [*Продажи*] ([*Opportunities*]).

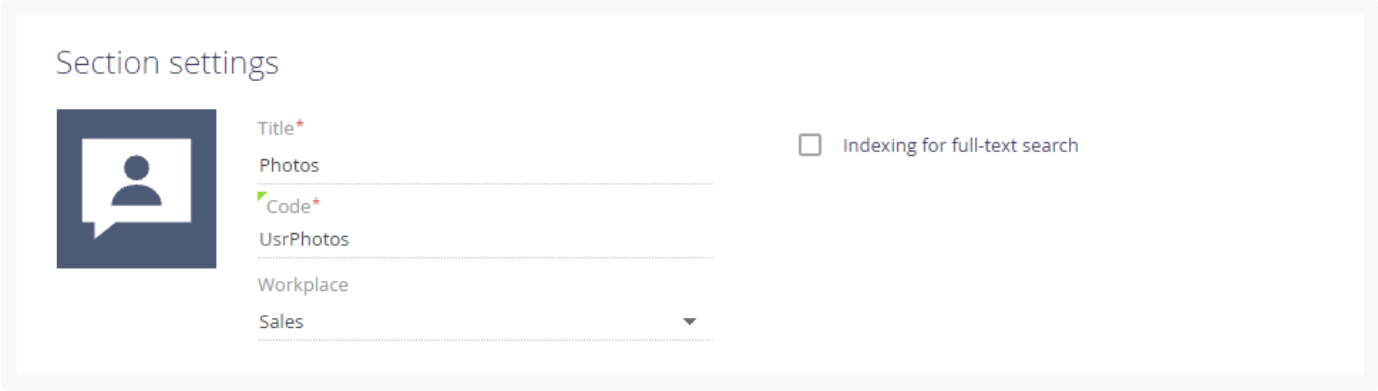
Реализовать деталь типа [Файлы и ссылки]

Сложный

Пример. На страницу записи пользовательского раздела [*Фотографии*] ([*Photos*]) добавить деталь [*Прикрепленные фотографии*] ([*Photos attachment*]) типа [*Файлы и ссылки*] ([*Attachments*]).

1. Создать пользовательский раздел

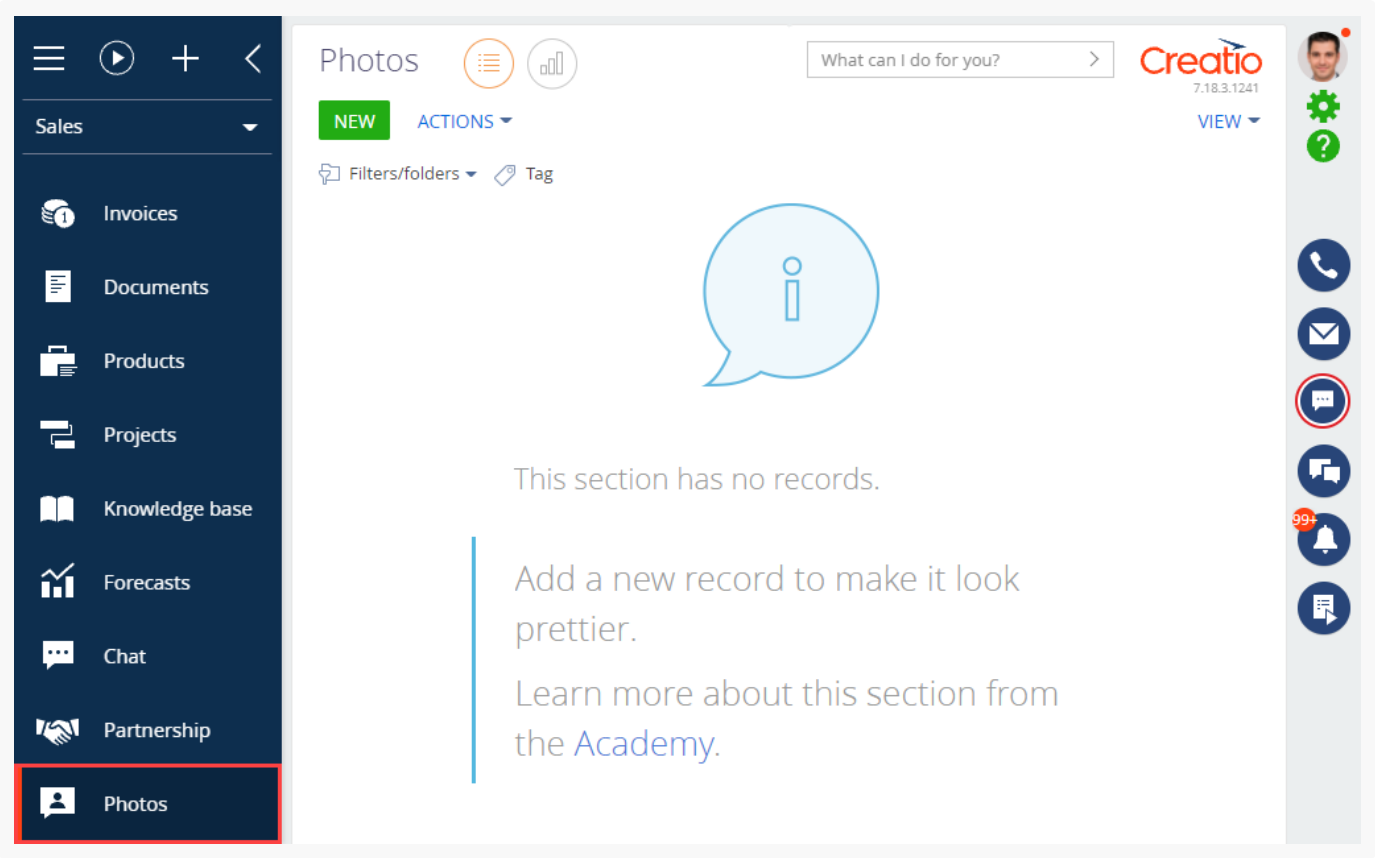
1. Создайте пользовательский пакет и установите его в качестве текущего. Подробнее читайте в статье Общие принципы работы с пакетами.
2. Перейдите в дизайнер системы по кнопке .
3. В блоке [*Настройка системы*] ([*System setup*]) перейдите по ссылке [*Мастер разделов*] ([*Section wizard*]).
4. Заполните настройки раздела:
   - [*Заголовок*] ([*Title*]) — "Фотографии" ("Photos").
   - [*Код (на английском)*] ([*Code*]) — "UsrPhotos".
   - [*Рабочее место*] ([*Workplace*]) — выберите "Продажи" ("Sales").



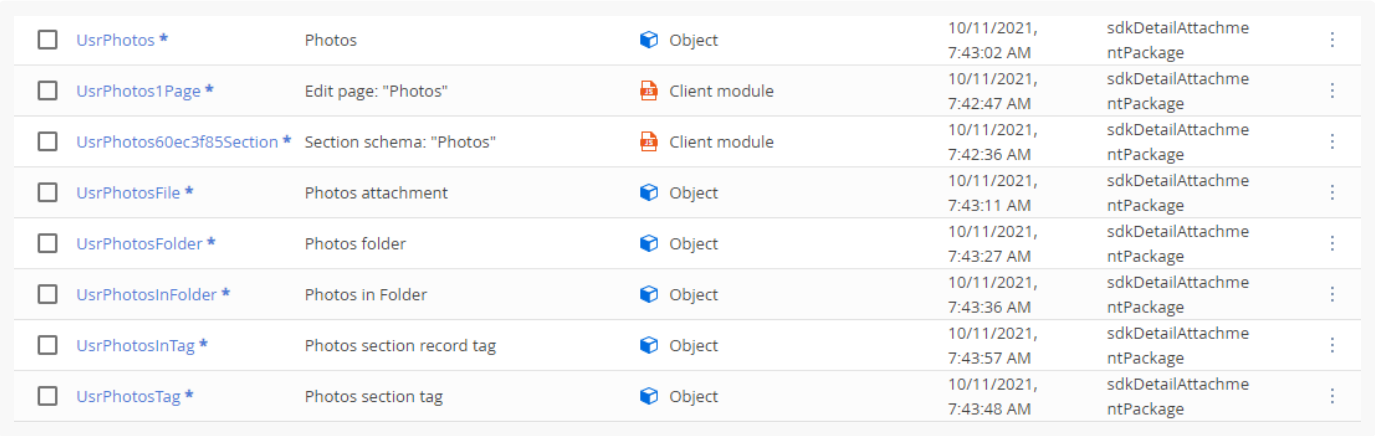
5. На панели инструментов мастера разделов нажмите [*Сохранить*] ([*Save*]).

В результате:

- Пользовательский раздел [*Фотографии*] ([*Photos*]) отображается в рабочем месте [*Продажи*] ([*Sales*]).



- В конфигурации созданы схемы раздела [*Фотографии*] ([*Photos*]).

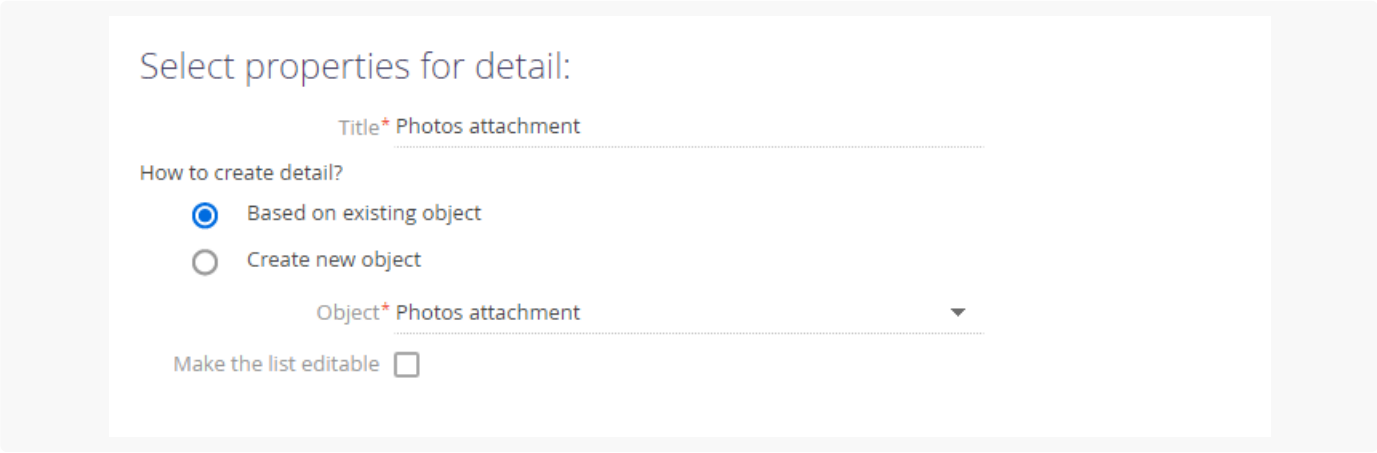


UsrPhotos *	Photos	Object	10/11/2021, 7:43:02 AM	sdkDetailAttachmentPackage
UsrPhotos1Page *	Edit page: "Photos"	Client module	10/11/2021, 7:42:47 AM	sdkDetailAttachmentPackage
UsrPhotos60ec3f85Section *	Section schema: "Photos"	Client module	10/11/2021, 7:42:36 AM	sdkDetailAttachmentPackage
UsrPhotosFile *	Photos attachment	Object	10/11/2021, 7:43:11 AM	sdkDetailAttachmentPackage
UsrPhotosFolder *	Photos folder	Object	10/11/2021, 7:43:27 AM	sdkDetailAttachmentPackage
UsrPhotosInFolder *	Photos in Folder	Object	10/11/2021, 7:43:36 AM	sdkDetailAttachmentPackage
UsrPhotosInTag *	Photos section record tag	Object	10/11/2021, 7:43:57 AM	sdkDetailAttachmentPackage
UsrPhotosTag *	Photos section tag	Object	10/11/2021, 7:43:48 AM	sdkDetailAttachmentPackage

2. Создать пользовательскую деталь

1. Перейдите в дизайнер системы по кнопке .
2. В блоке [*Настройка системы*] ([*System setup*]) перейдите по ссылке [*Мастер деталей*] ([*Detail wizard*]).
3. 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`").



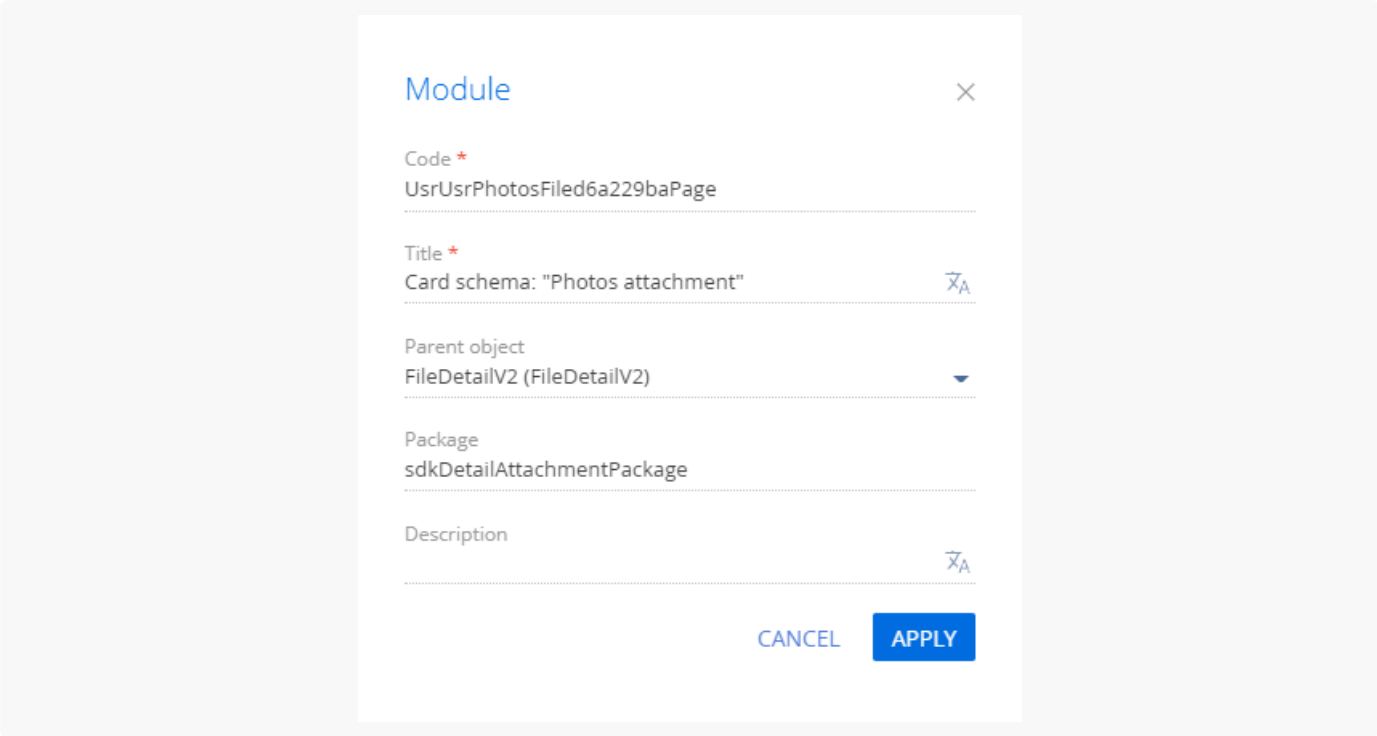
4. На панели инструментов мастера деталей нажмите [*Сохранить*] ([*Save*]).

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.

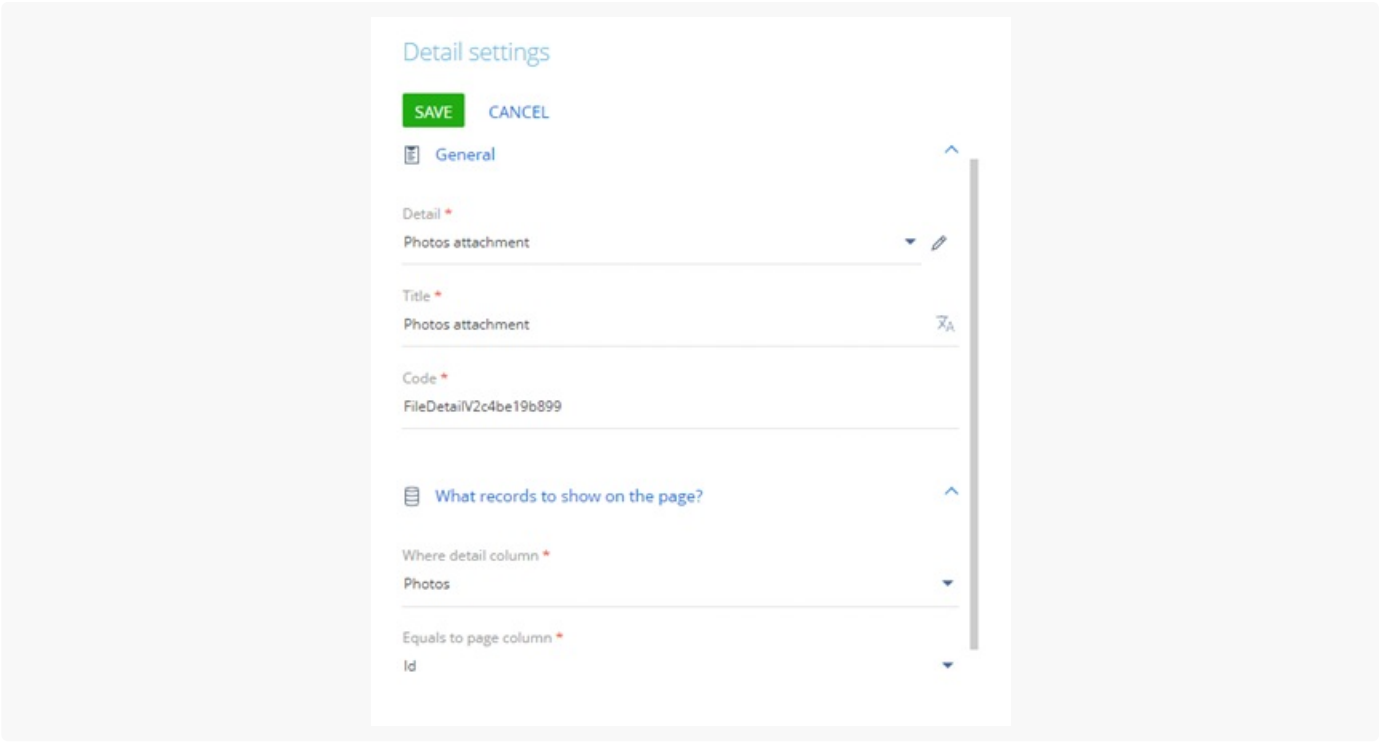


4. Для применения заданных свойств нажмите [*Применить*] ([*Apply*]).
5. На панели инструментов дизайнера нажмите [*Сохранить*] ([*Save*]).

4. Добавить деталь в раздел

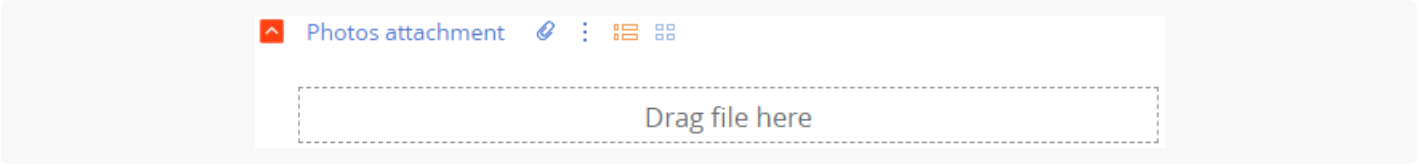
1. Перейдите в раздел [*Фотографии*] ([*Photos*]).
2. На панели инструментов кликните [*Вид*] —> [*Открыть мастер раздела*] ([*View*] —> [*Open section wizard*]).
3. В блоке [*Страницы раздела*] ([*Section pages*]) нажмите кнопку [*Редактировать страницу*] ([*Edit page*]).
4. В рабочей области мастера 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*]).



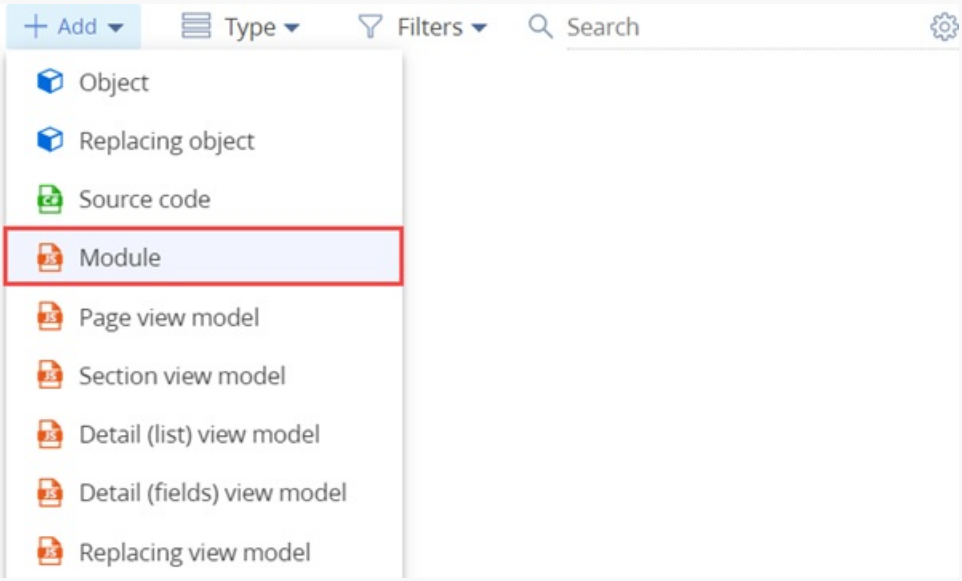
5. Добавить пользовательские стили детали

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.

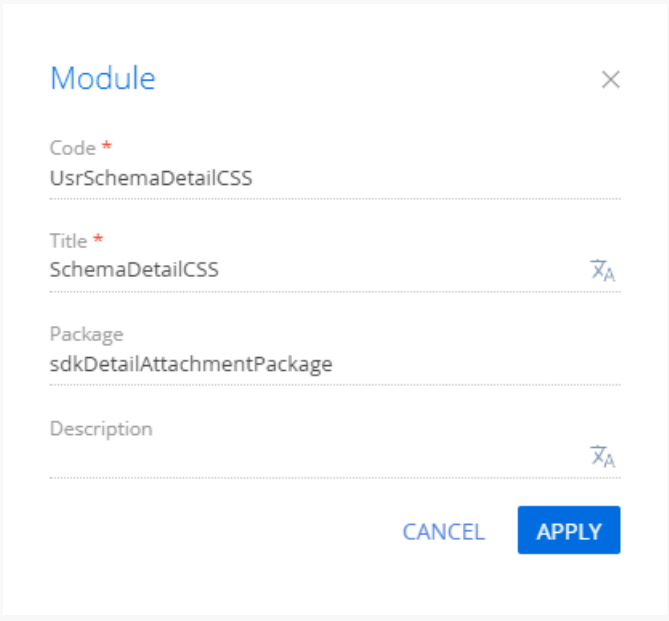
1. Создать схему модуля

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`".



   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*]).

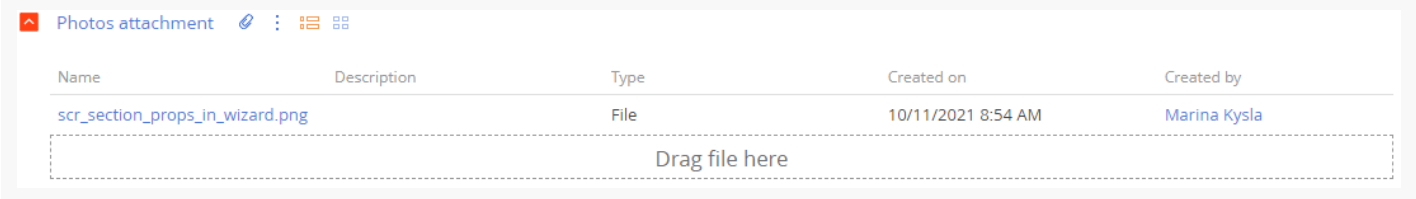


Схема BaseDetailV2 JS

Сложный

`BaseDetailV2` — базовая схема детали. Предоставляет базовую логику инициализации данных детали и взаимодействия детали со страницей записи. Реализована в пакете `NUI`. Схема является схемой модели представления. Описание свойств схемы содержится в статье Клиентская схема. Все схемы деталей должны наследовать схему `BaseDetailV2`.

Сообщения

Сообщения базовой детали

Название	Режим	Направление	Описание
`GetCardState`	Адресное	Публикация	Возвращает состояние страницы записи.
`IsCardChanged`	Адресное	Публикация	Сообщает об изменении страницы записи.
`SaveRecord`	Адресное	Публикация	Сообщает странице записи о необходимости сохранить данные.
`DetailChanged`	Адресное	Публикация	Сообщает странице записи об изменении данных детали.
`UpdateDetail`	Адресное	Подписка	Подписка на обновление страницы записи.
`OpenCard`	Адресное	Публикация	Открывает страницу записи.
`GetColumnsValues`	Адресное	Публикация	Возвращает запрашиваемые значения колонок.
`UpdateCardProperty`	Адресное	Публикация	Изменяет значение модели страницы записи.
`GetEntityInfo`	Адресное	Публикация	Запрашивает информацию о сущности основной записи.

Режимы сообщений представлены перечислением `Terrasoft.core.enums.MessageMode`, а направления сообщений — перечислением `Terrasoft.core.enums.MessageDirectionType`. Перечисление `MessageMode` описано в Библиотеке JS классов. Перечисление `MessageDirectionType` описано в Библиотеке JS классов.

Атрибуты

`CanAdd` `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.

Типы данных атрибутов представлены перечислением `Terrasoft.core.enums.DataValueType`.
Перечисление `DataValueType` описано в Библиотеке JS классов.

Методы

`init(callback, scope)`

Инициализирует страницу детали.

Параметры

`{Function} callback`	Функция обратного вызова.
`{Object} scope`	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

`{Object} config`	Конфигурационный объект, содержащий свойства детали.

`initData(callback, scope)`

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(isCollapsed)`

Обработчик сворачивания или разворачивания детали.

Параметры

`{Boolean} isCollapsed`	Признак свернутости 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.

Массив модификаций `diff`

```
diff: /**SCHEMA_DIFF*/[
    /* Базовый контейнер для представления детали. */
    {
        "operation": "insert",
        "name": "Detail",
        "values": {
```

```
            ...
        }
    }
]/**SCHEMA_DIFF*/
```

Схема BaseGridDetailV2 JS

Сложный

`BaseGridDetailV2` — базовая схема детали с реестром. Предоставляет базовую логику работы с реестром (загрузка, фильтрация), удаление, добавление и редактирование записей на детали. Реализована в пакете `NUI`. Схема является схемой модели представления. Описание свойств схемы содержится в статье Клиентская схема. Является наследником схемы `BaseDetailV2`. Все схемы деталей с реестром должны наследовать схему `BaseGridDetailV2`.

Сообщения

Сообщения базовой детали с реестром

Название	Режим	Направление	Описание
`getCardInfo`	Адресное	Подписка	Возвращает информацию о странице записи — значения по умолчанию, название колонки типизации, значение колонки типизации.
`CardSaved`	Широковещательное	Подписка	Обрабатывает сообщение сохранения страницы записи.
`RerenderQuickFilterModule`	Адресное	Публикация	Публикует сообщение с применением фильтра.
`GetExtendedFilterConfig`	Адресное	Подписка	Публикует конфиг пользовательского фильтра.
`GetModuleSchema`	Адресное	Подписка	Возвращает информацию о

			сущности, которая работает с фильтром.
`UpdateFilter`	Широковещательное	Подписка	Обновляет фильтры в детали.
`LoadedFiltersFromStorage`	Широковещательное	Публикация	Фильтры, которые загружены с хранилища.
`InitFilterFromStorage`	Широковещательное	Подписка	Инициализирует фильтры, которые загружены с хранилища.
`GetColumnsValues`	Адресное	Публикация	Получает значения колонок модели страницы записи.
`IsCardChanged`	Адресное	Публикация	Сообщает об изменении страницы записи.
`ValidateCard`	Адресное	Публикация	Запрос на валидацию страницы записи.

Режимы сообщений представлены перечислением `Terrasoft.core.enums.MessageMode`, а направления сообщений — перечислением `Terrasoft.core.enums.MessageDirectionType`. Перечисление `MessageMode` описано в Библиотеке JS классов. Перечисление `MessageDirectionType` описано в Библиотеке JS классов.

Миксины

`GridUtilities` `Terrasoft.GridUtilities`

Миксин для работы с реестром.

`WizardUtilities` `Terrasoft.WizardUtilities`

Миксин для работы с мастером деталей.

Атрибуты

`ActiveRow` `GUID`

Значение первичной колонки активной записи реестра.

`IsGridEmpty` BOOLEAN

Признак пустого реестра.

`MultiSelect` BOOLEAN

Признак наличия разрешения 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.

`DetailFilters` COLLECTION

Коллекция фильтров детали.

`IsDetailWizardAvailable` BOOLEAN

Признак доступности мастера деталей.

Типы данных атрибутов представлены перечислением `Terrasoft.core.enums.DataValueType`.

Перечисление `DataValueType` описано в Библиотеке JS классов.

Методы

`init(callback, scope)`

Замещает метод класса `BaseDetailV2`. Вызывает логику метода `init` родителя, регистрирует сообщения, инициализирует фильтры.

Параметры

`{Function} callback`	Функция обратного вызова.
`{Object} scope`	Контекст выполнения метода.

`initData(callback, scope)`

Замещение метода класса `BaseDetailV2`. Вызывает логику метода `initData` родительского класса, инициализирует коллекцию данных представления реестра.

Параметры

`{Function} callback`	Функция обратного вызова.
`{Object} scope`	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.

`addRecord(editPageUId)`

Добавляет новую запись в реестр. Сохраняет страницу записи в случае необходимости.

Параметры

`{String} editPageUId`	Идентификатор типизированной страницы записи.

`copyRecord(editPageUId)`

Копирует запись и открывает страницу записи.

Параметры

`{String} editPageUId`	Идентификатор типизированной страницы записи.

`editRecord(record)`

Открывает страницу выбранной записи.

Параметры

`{Object} record`	Модель записи для редактирования.

`subscribeSandboxEvents()`

Подписывается на сообщения, которые необходимы для работы детали.

`updateDetail(config)`

Замещение метода класса `BaseDetailV2`. Вызывает логику метода `updateDetail` родителя, обновляет деталь.

Параметры

`{Object} config`	Конфигурационный объект, который содержит свойства детали.

`openCard(operation, typeColumnValue, recordId)`

Открывает страницу записи.

Параметры

`{String} operation`	Тип операции (добавление/редактирование).
`{String} typeColumnValue`	Значение колонки типизации записи.
`{String} recordId`	Идентификатор записи.

`onCardSaved()`

Обрабатывает событие сохранения страницы записи, в которой находится деталь.

`addToolsButtonMenuItems(toolsButtonMenu)`

Добавляет элементы в коллекцию выпадающего списка функциональной кнопки.

Параметры

`{Terrasoft. BaseViewModelCollection} toolsButtonMenu`	Коллекция выпадающего списка функциональной кнопки.

`initDetailFilterCollection()`

Инициализирует фильтр детали.

`setFilter(key, value)`

Устанавливает значение фильтров детали.

Параметры

`{String} key`	Тип фильтров.
`{Object} value`	Значение фильтров.

`loadQuickFilter(config)`

Загружает быстрый фильтр.

Параметры

`{Object} config`	Параметры загрузки модуля фильтров.

`destroy()`

Очищает данные, выгружает деталь.

Массив модификаций

В массиве модификаций `diff` базовой детали с реестром 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, который предоставляет логику работы с элементом управления "Реестр". Реализован в классе `Terrasoft.configuration.mixins.GridUtilities` пакета `NUI`. Элемент управления "Реестр" описан в статье Реестр раздела.

Миксин **позволяет**:

- Подписываться на сообщения.
- Загружать данные.
- Работать с реестром:
  - Выбирать записи (выполнять поиск активных записей).
  - Добавлять, удалять, редактировать записи.
  - Задавать фильтры.
  - Сортировать записи.
  - Экспортировать записи в файл.
  - Проверять права доступа к записям реестра.

Методы

`init()`

Выполняет 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(response)`

Обрабатывает события загрузки данных. Выполняется, когда сервер возвращает данные.

Параметры

`{Object} response`	Результат выборки данных из базы данных.

`addItemsToGridData(dataCollection, options)`

Добавляет коллекцию новых элементов в коллекцию реестра.

Параметры

`{Object} dataCollection`	Коллекция новых элементов.
`{Object} options`	Параметры добавления.

`initQueryOptions(esq)`

Инициализирует настройки (постраничность, иерархичность) экземпляра запроса.

Параметры

{Terrasoft.data.queries.EntitySchemaQuery} esq	Запрос, в котором будут инициализированы необходимые настройки.

`initQuerySorting(esq)`

Инициализирует колонки сортировки.

Параметры

{Terrasoft.data.queries.EntitySchemaQuery} esq	Запрос, в котором будут инициализированы необходимые настройки.

`prepareResponseCollection(collection)`

Модифицирует коллекцию данных перед загрузкой в реестр.

Параметры

{Object} collection	Коллекция элементов реестра.

`getFilters()`

Возвращает примененные в данной схеме фильтры. Переопределяется в наследниках.

`exportToFile()`

Экспортирует содержимое реестра в файл.

`sortGrid(tag)`

Выполняет сортировку в реестре.

Параметры

{String} tag	Ключ, который указывает, как пересортировать реестр.

`deleteRecords()`

Инициирует удаление выбранных записей.

`checkCanDelete(items, callback, scope)`

Проверяет возможность удаления записи.

Параметры

`{Array} items`	Идентификаторы выбранных записей.
`{Function} callback`	Функция обратного вызова.
`{Object} scope`	Контекст выполнения метода.

`onDeleteAccept()`

Выполняет удаление после подтверждения пользователем.

`getSelectedItems()`

Возвращает выбранные записи в реестре.

`removeGridRecords(records)`

Убирает из реестра удаленные записи.

Параметры

`{Array} records`	Удаленные записи.

`reloadGridData()`

Выполняет перезагрузку реестра.

Модуль ConfigurationGrid JS

Сложный

`ConfigurationGrid` — модуль, который предоставляет логику работы с элементом управления "Конфигурационный реестр". Реализован в классе `Terrasoft.controls.ConfigurationGrid` пакета `UIv2`.

Класс `Terrasoft.controls.ConfigurationGrid` является наследником класса `Terrasoft.Grid`.

Методы

`init()`

Инициализирует компонент. Осуществляет подписку на события.

`activateRow(id)`

Выделяет строку и добавляет элементы редактирования.

Параметры

`{String|Number} id`	Идентификатор строки реестра.

`deactivateRow(id)`

Снимает выделение строки и удаляет элементы редактирования.

Параметры

`{String|Number} id`	Идентификатор строки реестра.

`formatCellContent(cell, data, column, link)`

Форматирует данные ячейки строки.

Параметры

`{Object} cell`	Ячейка.
`{Object} data`	Данные.
`{Object} column`	Конфигурация ячейки.
`{Object} link`	Ссылка.

`onUpdateItem(item)`

Обработчик события обновления записи.

Параметры

`{Terrasoft.BaseViewModel} item`	Элемент коллекции.

`onDestroy(clear)`

Уничтожает реестр и его компоненты.

Параметры

`{Boolean} clear`	Очистка реестра и компонентов.

Модуль ConfigurationGridGenerator JS

Сложный

`ConfigurationGridGenerator` — модуль, который генерирует конфигурацию реестра. Реализован в классе `Terrasoft.configuration.ConfigurationGridGenerator` пакета `UIv2`. Класс `Terrasoft.configuration.ConfigurationGridGenerator` является наследником класса `Terrasoft.ViewGenerator`.

Методы

`addLinks`

Переопределенный метод класса `Terrasoft.ViewGenerator`. По умолчанию содержит значение `Terrasoft.emptyFn`. В редактируемый реестр не будут добавлены ссылки.

`generateGridCellValue(config)`

Переопределенный метод класса `Terrasoft.ViewGenerator`. Генерирует конфигурацию значения в ячейке.

Параметры

`{Object} config`	Конфигурация колонки.

Модуль ConfigurationGridUtilities JS

Сложный

`ConfigurationGridUtilities` — модуль, который содержит методы инициализации модели представления строки реестра, обработки действий активной записи и обработки горячих клавиш. Реализован в классе `Terrasoft.configuration.mixins.ConfigurationGridUtilities` пакета `UIv2`.

Свойства

`currentActiveColumnName` `String`

Имя текущей выделенной колонки.

`columnsConfig` `String`

Конфигурация колонок.

`systemColumns` `Array`

Коллекция названий системных колонок.

Методы

`onActiveRowAction(buttonTag, primaryColumnValue)`

Обрабатывает нажатие действия активной записи.

Параметры

`{String} buttonTag`	Тег выбранного действия.
`{String} primaryColumnValue`	Идентификатор активной записи.

`activeRowSaved(row, callback, scope)`

Обрабатывает результат сохранения записи.

Параметры

`{Object} row`	Строка реестра.
`{Function} [callback]`	Функция обратного вызова.
`{Object} scope`	Контекст вызова функции обратного вызова.

`initActiveRowKeyMap(keyMap)`

Инициализирует подписку на события нажатия кнопок в активной строке.

Параметры

{Array} keyMap	Описание событий.

---

getCellControlsConfig(entitySchemaColumn)

Возвращает конфигурацию элементов редактирования ячейки реестра.

Параметры

{Terrasoft.EntitySchemaColumn} entitySchemaColumn	Колонка ячейки реестра.

---

copyRow(recordId)

Копирует и добавляет запись в реестр.

Параметры

{String} recordId	Идентификатор копируемой записи.

---

initEditableGridRowViewModel(callback, scope)

Инициализирует классы элементов коллекции редактируемого реестра.

Параметры

{Function} callback	Функция обратного вызова.
{Object} scope	Контекст вызова функции обратного вызова.

---

saveRowChanges(row, callback, scope)

Сохраняет запись.

Параметры

{Object} row	Строка реестра.
{Function} [callback]	Функция обратного вызова.
{Object} [scope]	Контекст вызова функции обратного вызова.

Схема BasePageV2 JS

Средний

`BasePageV2` — базовая схема карточки. Реализована в пакете `NUI`. Схема является схемой модели представления. Описание свойств схемы содержится в статье Клиентская схема.

Сообщения

Сообщения базовой карточки

Название	Режим	Направление	Описание
`UpdatePage HeaderCaption`	Адресное	Публикация	Обновляет заголовок страницы.
`GridRowChanged`	Адресное	Подписка	Получает идентификатор выбранной в реестре записи при ее изменении.
`UpdateCard Property`	Адресное	Подписка	Изменяет значение параметра модели.
`UpdateCard Header`	Адресное	Подписка	Обновляет заголовок карточки.
`CloseCard`	Адресное	Публикация	Закрывает карточку.
`OpenCard`	Адресное	Подписка	Открывает карточку.
`OpenCardInChain`	Адресное	Публикация	Открывает цепочку карточек.
`GetCardState`	Адресное	Подписка	Возвращает состояние карточки.
`IsCardChanged`	Адресное	Публикация	Сообщает об изменении карточки.
`GetActiveView Name`	Адресное	Публикация	Получает имя активного представления.
`GetMiniPage MasterEntity Info`	Адресное	Подписка	Возвращает информацию об основной сущности мини-карточки.
`GetPageTips`	Адресное	Подписка	Возвращает подсказки страницы.

`GetColumnInfo`	Адресное	Подписка	Возвращает информацию колонки.
`GetEntityColumnChanges`	Широковещательное	Публикация	Отправляет информацию колонки сущности при ее изменении.
`ReloadSectionRow`	Адресное	Публикация	Перезагружает строку раздела в соответствии со значением основного столбца.
`ValidateCard`	Адресное	Подписка	Запускает проверку валидности карточки.
`ReInitializeActionsDashboard`	Адресное	Публикация	Запускает повторную инициализацию панели действий.
`ReInitializeActionsDashboard`	Адресное	Подписка	Обновляет конфиг панели действий.
`ReloadDashboardItems`	Широковещательное	Публикация	Перезагружает элементы дашбордов.
`ReloadDashboardItemsPTP`	Адресное	Публикация	Перезагружает элементы дашбордов для текущей страницы.
`CanChangeHistoryState`	Широковещательное	Подписка	Разрешает или запрещает изменение текущего состояния истории.
`IsEntityChanged`	Адресное	Подписка	Возвращает измененную сущность.
`IsDcmFilterColumnChanged`	Адресное	Подписка	Возвращает `true`, если изменены отфильтрованные колонки.
`UpdateParentLookupDisplayValue`	Широковещательное	Двунаправленное	Обновляет значение родительской записи справочника по конфигу.
`UpdateParentLookupDisplay`	Широковещательное	Двунаправленное	Указывает на необходимость

`DisplayValue`		необходимость перезагрузки данных при следующем запуске.

Режимы сообщений представлены перечислением `Terrasoft.core.enums.MessageMode`, а направления сообщений — перечислением `Terrasoft.core.enums.MessageDirectionType`. Перечисление `MessageMode` описано в Библиотеке JS классов. Перечисление `MessageDirectionType` описано в Библиотеке JS классов.

Атрибуты

`IsLeftModulesContainerVisible` `BOOLEAN`

Признак видимости контейнера `LeftModulesContainer`.

`IsActionDashboardContainerVisible` `BOOLEAN`

Признак видимости контейнера `ActionDashboardContainer`.

`HasActiveDcm` `BOOLEAN`

Признак видимости контейнера `DcmActionsDashboardContainer`.

`ActionsDashboardAttributes` `CUSTOM_OBJECT`

Пользовательские атрибуты контейнера `DcmActionsDashboardContainer`.

`IsPageHeaderVisible` `BOOLEAN`

Флаг видимости заголовка страницы.

`ActiveTabName` `TEXT`

Сохранить имя активной вкладки.

`GridDataViewName` `TEXT`

Имя представления `GridData`.

`AnalyticsDataViewName` `TEXT`

Имя представления `AnalyticsData`.

`IsCardOpenedAttribute` `STRING`

Атрибут тела карточки, когда карточки отображена или скрыта.

`IsMainHeaderVisibleAttribute` `STRING`

Атрибут тела карточки, когда основной заголовок отображен или скрыт.

`PageHeaderColumnNames` `CUSTOM_OBJECT`

Массив имен колонок заголовка страницы.

`IsNotAvailable` `BOOLEAN`

Признак недоступности страницы.

`CanCustomize` `BOOLEAN`

Признак возможности кастомизации страницы.

`Operation` `ENUM`

Операции карточки.

`EntityReloadScheduled` `BOOLEAN`

Признак необходимости перезагрузки сущности при следующем запуске.

Типы данных атрибутов представлены перечислением `Terrasoft.core.enums.DataValueType`.
Перечисление `DataValueType` описано в Библиотеке JS классов.

Методы

`onDiscardChangesClick(callback, scope)`

Обрабатывает нажатие кнопки [*Отменить*] ([*Discard*]).

Параметры

`{String} [callback]`	Функция обратного вызова.
`{Terrasoft.BaseViewModel} [scope]`	Контекст выполнения метода.

`addChangeDataViewOptions(viewOptions)`

Добавляет представления точек переключения в выпадающий список кнопки [*Вид*] ([*View*]).

Параметры

`{Terrasoft.BaseViewModelCollection} viewOptions`	Пункты выпадающего списка кнопки [*Вид*] ([*View*]).

`addSectionDesignerViewOptions(viewOptions)`

Добавляет пункт [*Открыть мастер раздела*] ([*Open section wizard*]) в выпадающий список кнопки [*Вид*] ([*View*]).

Параметры

`{Terrasoft.BaseViewModelCollection} viewOptions`	Пункты выпадающего списка кнопки [*Вид*] ([*View*]).

`getReportFilters()`

Возвращает коллекцию фильтров для запроса.

`initPageHeaderColumnNames()`

Инициализировать имена колонок заголовка страницы.

`getParameters(parameters)`

Возвращает значения параметров `ViewModel`.

Параметры

`{Array} parameters`	Имена параметров.

`setParameters(parameters)`

Задает параметры `ViewModel`.

Параметры

`{Object} parameters`	Значения параметров.

`getLookupModuleId()`

Возвращает идентификатор модуля страницы справочника.

`onReloadCard(defaultValues)`

Обработчик сообщения `ReloadCard`. Перезагружает данные сущности карточки.

Параметры

`{Object[]} defaultValues`	Массив значений по умолчанию.

`onGetColumnInfo(columnName)`

Возвращает информацию колонки.

Параметры

`{String} columnName`	Имя колонки.

`getTabsContainerVisible()`

Возвращает статус видимости вкладок контейнера.

`getPrintMenuItemVisible(reportId)`

Возвращает статус видимости пунктов выпадающего списка (т. е. отчетов) кнопки [*Печать*] ([*Print*]).

Параметры

`{String} reportId`	Идентификатор отчета.

`getDataViews()`

Получает представление раздела.

`runProcess(tag)`

Запустить бизнес-процесс с помощью кнопки запуска глобальных бизнес-процессов.

Параметры

`{Object} tag`	Идентификатор схемы бизнес-процесса.

`runProcessWithParameters(config)`

Запустить бизнес-процесс с параметрами.

Параметры

`{Object} config`	Конфигурационный объект.

`onCanBeDestroyed(cacheKey)`

Проверяет наличие несохраненных данных. Измените `config.result` из кэша, если данные изменены, но не сохранены.

Параметры

`{String} cacheKey`	Ключ конфигурационного объекта в кэше.

`onValidateCard()`

Отображается сообщение о некорректности, если карточка невалидна.

Схема BaseFieldsDetail JS

Средний

`BaseFieldsDetail` — базовая схема детали с полями. Реализована в пакете `BaseFinance` продуктов линейки Financial Services Creatio. Схема является схемой модели представления. Описание свойств схемы содержится в статье Клиентская схема.

Сообщения

Сообщения базовой детали с полями

Название	Режим	Направление	Описание
`LookupInfo`	Адресное	Подписка	Возвращает информацию о справочнике.
`UpdateCardProperty`	Адресное	Публикация	Изменяет значение модели страницы записи.
`CardSaved`	Широковещательное	Подписка	Получает информацию о сохранении родительской страницы.
`IsCardChanged`	Адресное	Публикация	Сообщает об изменении карточки.

Схема FileDetailV2 JS

Средний

`FileDetailV2` — базовая схема детали типа [*Файлы и ссылки*] ([*Attachments*]). Реализована в пакете `UIv2`. Схема является схемой модели представления. Описание свойств схемы содержится в статье Клиентская схема.

Атрибуты

`SchemaCardName` `TEXT`

Сохранить имя страницы записи.

`parentEntity` `CUSTOM_OBJECT`

Родительская сущность.

Типы данных атрибутов представлены перечислением `Terrasoft.core.enums.DataValueType`. Перечисление `DataValueType` описано в Библиотеке JS классов.

Методы

`getShowPreviewSettingsValue()`

Получить значение системных настроек `ShowPreview`.

`initParentEntity()`

Инициализировать родительскую сущность.

`itemsRendered()`

Обрабатывает событие `itemsRendered`, которое сработало в компоненте `ContainerList`.