

# Front-end разработка Freedom UI

Версия 8.0



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

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

# Содержание

<b>Front-end архитектура Creatio</b>	<b>5</b>
Режим DesignTime	5
Режим RunTime	5
<b>Клиентская схема</b>	<b>11</b>
<b>Секция validators</b>	<b>13</b>
Базовые валидаторы	14
<b>Секция converters</b>	<b>15</b>
Базовые конвертеры	15
<b>Страница</b>	<b>16</b>
Контейнеры страницы	17
Структура страницы	18
<b>Реализовать валидацию значения поля на странице</b>	<b>18</b>
1. Настроить интерфейс страницы	18
2. Настроить валидацию значения поля	18
Результат выполнения примера	20
<b>Реализовать конвертацию значения поля на странице</b>	<b>21</b>
1. Настроить интерфейс страницы	21
2. Настроить конвертацию значения поля	21
Результат выполнения примера	22
<b>Настроить условие отображения поля на странице</b>	<b>23</b>
1. Настроить интерфейс страницы	23
2. Настроить условие отображения поля	24
Результат выполнения примера	26
<b>Настроить условие блокировки поля на странице</b>	<b>27</b>
1. Настроить интерфейс страницы	28
2. Настроить условие блокировки поля	31
Результат выполнения примера	32
<b>Настроить условие заполнения поля на странице</b>	<b>33</b>
1. Настроить интерфейс страницы	33
2. Настроить условие заполнения поля	34
Результат выполнения примера	35
<b>Настроить условие обязательности поля на странице</b>	<b>36</b>
1. Настроить интерфейс страницы	37
2. Настроить условие обязательности поля	40
Результат выполнения примера	41
<b>Отобразить значения системных переменных на странице</b>	<b>42</b>

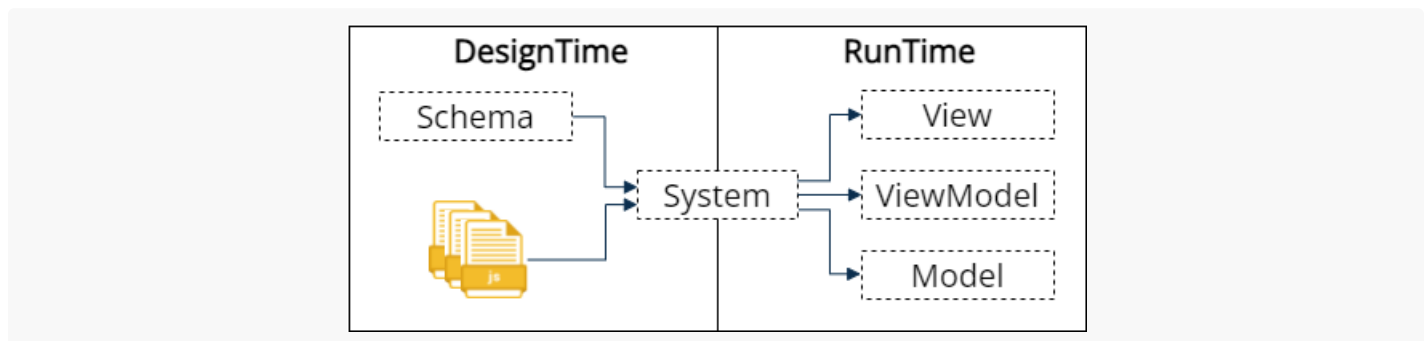
1. Настроить интерфейс страницы	43
2. Настроить получение значений системных переменных	44
Результат выполнения примера	47
<b>Компонент FlexContainer</b>	<b>47</b>
Свойства	47
Пример использования	51
<b>Компонент GridContainer</b>	<b>52</b>
Свойства	52
Пример использования	57
<b>Компонент DateTimePicker</b>	<b>58</b>
Свойства	58
Горячие клавиши	60
<b>Компонент Checkbox</b>	<b>61</b>
Свойства	61
События	62
<b>Компонент Input</b>	<b>63</b>
Свойства	63
<b>Компонент NumberInput</b>	<b>65</b>
Свойства	65

# Front-end архитектура Creatio

## Оснoвы

**Программная платформа** — среда, которая используется для разработки (режим `DesignTime`) и выполнения (режим `RunTime`) приложения.

Схема взаимодействия структурных элементов программной платформы Creatio представлена на рисунке ниже.



Структурные элементы режимов платформы Creatio взаимодействуют через системный слой `System`.

## Режим DesignTime

**Назначение** режима `DesignTime` — разработка, изменение и кастомизация приложения.

**Структурные элементы** режима `DesignTime` программной платформы Creatio:

- Слой `Schema` — слой метаданных. Содержит набор клиентских схем. Подробнее о клиентских схемах читайте в статье [Клиентская схема](#).
- Предварительно скомпилированный JavaScript-код.

## Режим RunTime

**Структурные элементы** режима `RunTime` программной платформы Creatio:

- Слой `View` — слой визуального представления информации.
- Слой `ViewModel` — слой бизнес-логики взаимодействия слоев `View` и `Model`.
- Слой `Model` — слой данных.

## Слой View

`View` — слой, который отвечает за визуальное представление информации. Представлен набором визуальных компонентов режима `DesignTime`.

Для front-end разработки на платформе Creatio Freedom UI появляется понятие декораторов.

**Декораторы** — элементы, которые отвечают за регистрацию и расширение функциональности элементов (компонентов, запросов, обработчиков запросов и т. д.) приложения. Использование декораторов позволяет универсализировать разработку путем использования специальных реестров типов, а также уйти от необходимости подключения функциональности вручную.

Слой `view` реализуют компоненты разных типов. Например, это могут быть компоненты для размещения вложенных компонентов (например, `GridContainerComponent`), компоненты для отображения информации (например, `LabelComponent`), компоненты для взаимодействия с пользователем (например, `InputComponent`) и т. д.

Компоненты могут использоваться в свойстве `viewConfigDiff` клиентской схемы. При использовании компонентов в клиентских схемах компоненты генерируются на основе метаданных схемы, что позволяет кастомизировать визуальное представление с использованием low-code / no-code инструментов Creatio.

## Слой ViewModel

`ViewModel` — слой, который отвечает за бизнес-логику взаимодействия слоев `View` и `Model`. Представлен типом `ViewModel`, который инкапсулирует в себе логику работы с атрибутами (инициализация данных, привязка к свойствам визуальных компонентов и отслеживание изменений). Creatio не предоставляет возможность создания новых типов `view model`.

**Типы** компонентов, которые позволяет реализовать слой `ViewModel`:

- **Валидаторы** — функции проверки корректности значения атрибута `ViewModel`. Подробнее читайте в пункте [Валидаторы](#).
- **Конвертеры** — функции модификации значения атрибута `ViewModel`, который привязан к свойству визуального компонента. Подробнее читайте в пункте [Конвертеры](#).
- **Запросы и обработчики запросов** — элементы механизма `HandlerChain`, который позволяет описывать бизнес-логику в формате запроса на действие и цепочки обработчиков запроса. Подробнее читайте в пункте [Запросы и обработчики запросов](#).

Каждому типу соответствует декоратор.

**Типы** атрибутов, которые реализуют слой `ViewModel`:

- Атрибут простого типа (`string`, `number`, `boolean`).
- Атрибут, который содержит вложенные `view model`.
- Ресурсный атрибут (`readonly`).

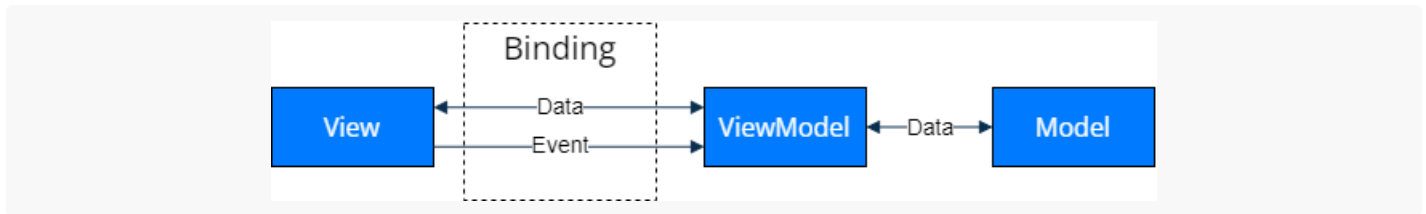
**Этапы** жизненного цикла, которые поддерживает слой `ViewModel` в Creatio:

- Инициализация экземпляра (`crt.HandleViewModelInitRequest`).
- Изменение значения атрибута (`crt.HandleViewModelAttributeChangeRequest`).
- Уничтожение экземпляра (`crt.HandleViewModelDestroyRequest`). На этом этапе необходимо выполнять только синхронный код, который уничтожает накопленные в процессе работы ресурсы.

Для обеспечения отображения данных в пользовательском интерфейсе приложения и синхронизации

этих данных необходимо установить привязку слоя `View` к `ViewModel`.

Схема привязки `View` к `ViewModel` представлена на рисунке ниже.



**Типы** привязок, которые предоставляет Creatio:

- Односторонняя привязка к атрибуту.
- Привязка к ресурсному атрибуту.
- Получение `CrtControl` экземпляра.

Пример использования разных типов привязок приведены ниже.

#### Пример использования привязок

```

schema{
  resources: {
    strings: {
      Title: {
        "en-US": "Example"
      }
    }
  }
  body: define("Example", [], () => {
    viewModelConfig: {
      attributes: {
        FirstName: {},
        Visible: {}
      }
    }
    viewConfigDiff: [{
      name: "Example",
      type: "crt.Input",
      control: "$FirstName", <== CrtControl
      title: "$Resources.Strings.Title", <== Resources
      visible: "$Visible" <== OneWay
    }]
  })
}
  
```

Механизм привязок можно расширить механизмом макросов, который необходимо использовать,

например, для привязки вложенных свойств объекта к ресурсам `ViewModel`. **Макрос** — элемент, который заменяет часть `view config` на значение ресурсов из `ViewModel`. В отличие от привязки, макрос срабатывает только один раз и в дальнейшем не синхронизируется при изменении `view model`. В Creatio 8.0 Atlas реализован только макрос `#ResourceString#`, в котором реализована работа со строками из ресурсов.

Пример установки значения из ресурсов в свойство `caption` элемента `Header` приведен ниже.

#### Пример использования макроса `#ResourceString#`

```
viewConfigDiff: /**SCHEMA_VIEW_CONFIG_DIFF*/[
  ...
  {
    "operation": "insert",
    "name": "Header",
    "values": {
      "type": "crt.Label",
      "caption": "#ResourceString(Header)#",
    },
    ...
  },
  ...
]/**SCHEMA_VIEW_CONFIG_DIFF*/
```

Поскольку у пользователей отсутствует возможность создания `ViewModel`, то в Creatio 8.0 Atlas бизнес-логику необходимо описывать в отдельных обработчиках запросов. Обработчики можно объединять в цепочки и определять необходимое время вызова соответствующего обработчика. При создании обработчика можно ограничить область его срабатывания путем добавления в свойство `scopes` имен схем, для которых он должен срабатывать.

## Валидаторы

**Валидаторы** — функции проверки корректности значения атрибута `ViewModel`. Примеры валидаторов:

`MaxLengthValidator`, `MinLengthValidator`, `RequiredValidator`.

Валидаторы применяются к атрибутам `ViewModel`, а не к визуальным элементам, но могут получить информацию о статусе валидности через `CrtControl`.

## Конвертеры

**Конвертеры** — функции модификации значения атрибута `ViewModel`, который привязан к свойству визуального компонента. Примеры конвертеров: `crt.invertBooleanValue`, `crt.toBoolean`.

**Особенности** использования конвертеров:

- В Creatio 8.0 Atlas применяются только в режиме `RunTime`.
- Используются только при установленной привязке, т. е. не работают с константами.



- Работают только в одну сторону. Поэтому недоступны к использованию с `CrtControl`.

Запросы и обработчики запросов

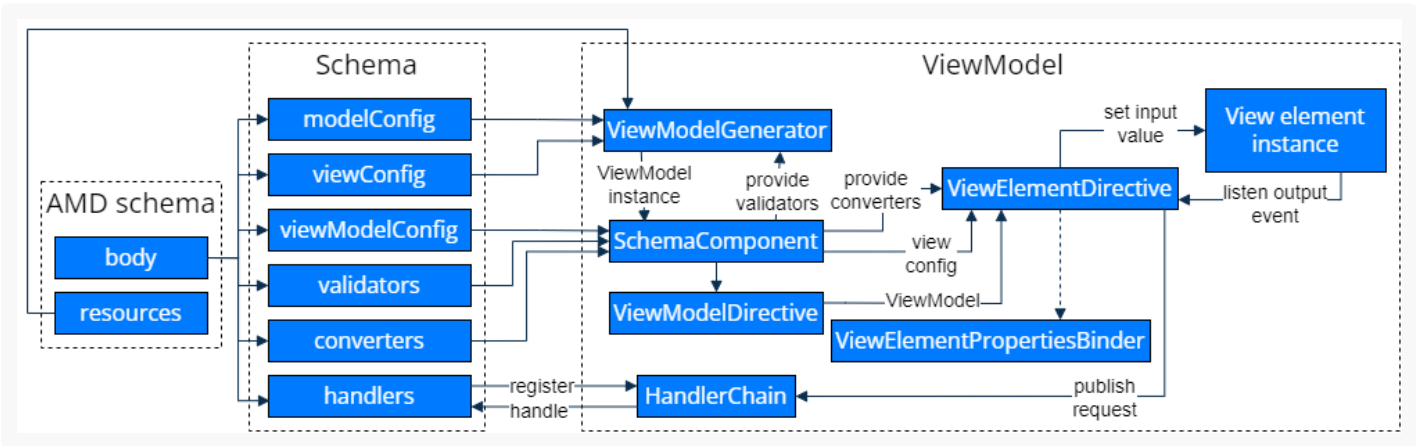
**Запросы и обработчики запросов** — элементы механизма `HandlerChain`, который позволяет описывать бизнес-логику в формате запроса на действие и цепочки обработчиков запроса. Примеры запросов: готовность страницы, загрузка и сохранение данных, запуск бизнес-процесса.

**Запросы**, кастомизацию которых может выполнять пользователь, приведены в таблице ниже.

Запросы, кастомизацию которых может выполнять пользователь

Тип запроса	Обработчик	Описание
Действия по открытию страниц	<code>crt.CreateRecordRequest</code>	Создать запись.
	<code>crt.UpdateRecordRequest</code>	Обновить запись.
	<code>crt.OpenPageRequest</code>	Открыть страницу.
Действия по работе с данными на странице	<code>crt.SaveRecordRequest</code>	Сохранить данные.
	<code>crt.CancelRecordChangesRequest</code>	Отменить изменение данных.
Другие действия в дизайнера интерфейсов	<code>crt.RunBusinessProcessRequest</code>	Запустить бизнес-процесс.
	<code>crt.ClosePageRequest</code>	Закрыть страницу.

Схема работы приложения представлена на рисунке ниже.

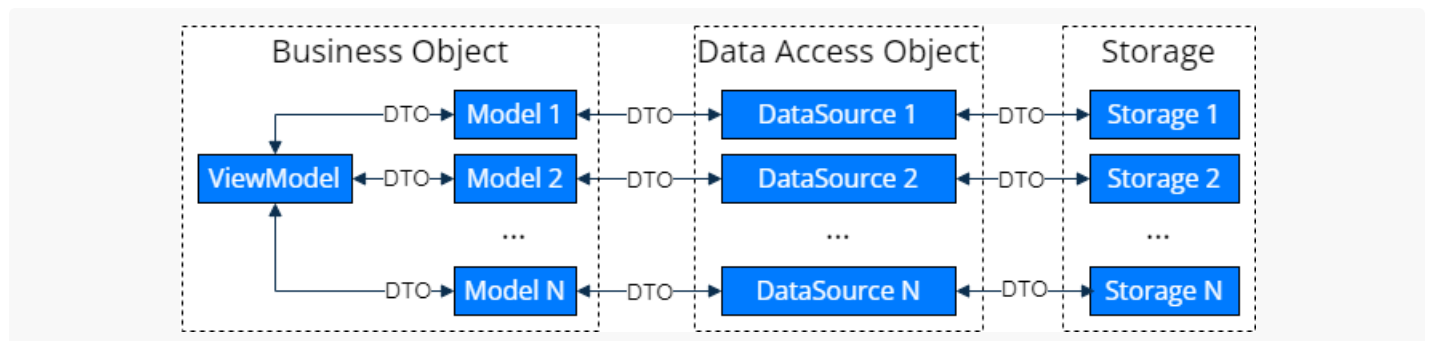


## Слой Model

**Model** — слой, который отвечает за работу с данными. Позволяет работать с источниками данных, схемой их данных и выполнять операции с данными (загрузка, сохранение, удаление, сортировка и т. д.). В Creatio 8.0 Atlas реализован тип **EntityDataSource** источника данных, который позволяет работать с данными **Entity**-объектов Creatio. Creatio 8.0 Atlas не предоставляет возможность расширения набора источников данных. В дальнейшем эта возможность будет предоставлена. Функциональность слоя **Model** используется слоем **ViewModel** для обеспечения данными слоя **View**.

Для хранения данных в Creatio используется паттерн **Data Access Object** (DAO). Подробнее о паттерне DAO читайте на [Википедии](#).

Схема работы DAO в Creatio представлена на рисунке ниже.

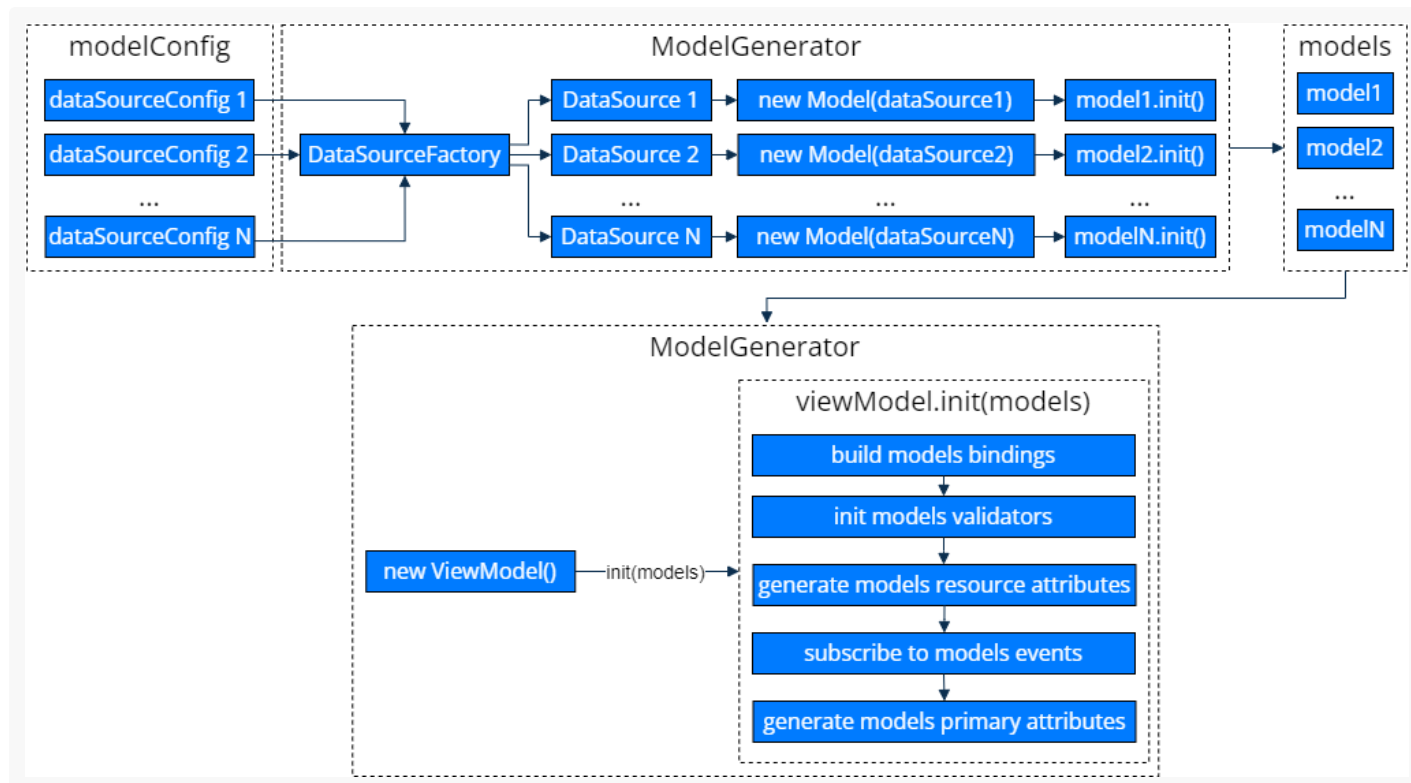


**Задачи** элементов группы **Data Access Object** :

- Обеспечить выполнение CRUD-операций.
- Предоставить права на операции с данными (создание, редактирование, удаление).
- Предоставить структуру данных (**DataSchema**).

В Creatio 8.0 Atlas реализован **EntityDataSource**, который работает с базой данных приложения.

Инициализация **Model** выполняется с **viewModel** по схеме, которая представлена ниже.



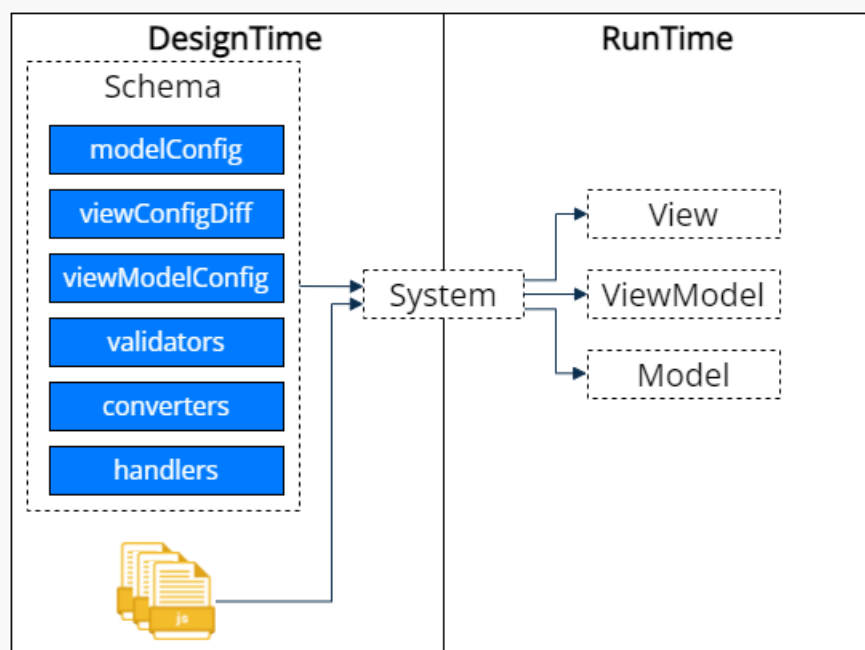
## Клиентская схема

### Оснoвы

**Клиентская схема** — это схема клиентского модуля, с помощью которой реализуется front-end часть приложения. Элемент пакетной подсистемы Creatio. **Назначение** клиентской схемы — сохранение и поставка метаданных части системы (приложения, раздела, модального окна и т. д.). Типы модулей и их особенности описаны в статье [Виды модулей](#).

Клиентская схема принадлежит к слою `Schema` (слой метаданных) режима `DesignTime` платформы Creatio и содержит настройки слоев режима `RunTime`, которые доступны к изменению путем использования по- code инструментов.

Схема слоя `Schema` и его взаимодействие с другими структурными элементами платформы Creatio представлены на рисунке ниже.



Подробнее о режимах `DesignTime` и `RunTime` читайте в статье [Front-end архитектура Creatio](#).

Исходный код клиентских схем имеет общую структуру, которая представлена ниже.

### Структура исходного кода клиентской схемы

```
define("ExampleSchema", [], function() {
    return {
        modelConfig: /**SCHEMA_MODEL_CONFIG*/{}/**SCHEMA_MODEL_CONFIG*/,
        viewConfigDiff: /**SCHEMA_VIEW_CONFIG_DIFF*/[]/**SCHEMA_VIEW_CONFIG_DIFF*/,
        viewModelConfig: /**SCHEMA_VIEW_MODEL_CONFIG*/{}/**SCHEMA_VIEW_MODEL_CONFIG*/,
        validators: /**SCHEMA_VALIDATORS*/ {} /**SCHEMA_VALIDATORS*/,
        converters: /**SCHEMA_CONVERTERS*/ {} /**SCHEMA_CONVERTERS*/,
        handlers: /**SCHEMA_HANDLERS*/[] /**SCHEMA_HANDLERS*/
    };
});
```

Маркерные комментарии к свойствам клиентской схемы обязательны к использованию.

Описание **свойств** конфигурационного объекта клиентской схемы представлено в таблице ниже.

## Свойства клиентской схемы

Свойство	Способ разработки	Описание
<code>modelConfig</code>	Кастомизация	Содержит описание источников данных.
<code>viewConfigDiff</code>	Кастомизация	Отвечает за формирование слоя <code>View</code> .
<code>viewModelConfig</code>	Кастомизация	Отвечает за формирование слоя <code>ViewModel</code> — бизнес-логики взаимодействия слоев <code>View</code> и <code>Model</code> .
<code>validators</code>	Замещение	Валидаторы.
<code>converters</code>	Замещение	Конвертеры.
<code>handlers</code>	Замещение	Обработчики.

Свойства клиентской схемы реализуются в формате `JSON`. В `JSON`-массиве свойств `validators`, `converters`, `handlers` может быть использован `JavaScript`.

Бизнес-логика реализуется в схемах страниц Freedom UI с использованием no-code и low-code инструментов. Подробнее читайте в статье [Front-end архитектура Creatio](#).

**Компоненты** бизнес-логики, JS-реализацию которых может содержать клиентская схема:

- Валидаторы.
- Конвертеры.
- События и обработчики событий.

**Бизнес-логика**, которой можно управлять в компонентах:

- Видимость элементов.
- Блокировка элементов.
- Обязательность элементов.
- Фильтрация элементов.
- Заполнение полей.
- Обращение к данным Creatio.
- Отправка http-запросов.
- Переход по страницам.

Также Creatio предоставляет возможность выстраивать обработчики запросов в цепочки выполнения, например, чтобы по запросу сохранения записи сначала отработал базовый обработчик сохранения, а потом выполнялась пользовательская логика страницы.

## Секция validators



**Валидаторы** — функции проверки корректности значения атрибута `ViewModel`. Например, проверка значения поля записи на соответствие установленным условиям. Пример использования валидатора приведен в статье [Реализовать валидацию значения поля на странице](#).

**Базовые валидаторы**, которые предоставляет Creatio 8 Atlas, представлены ниже.

## Базовые валидаторы

`crt.Required`

Устанавливает обязательность заполнения поля.

`crt.MinLength`

Устанавливает минимально допустимое значение длины строки.

### Параметры

`minLength`

INTEGER

Минимально допустимое значение длины строки.

`crt.MaxLength`

Устанавливает максимально допустимое значение длины строки.

### Параметры

`maxLength`

INTEGER

Максимально допустимое значение длины строки.

`crt.Min`

Устанавливает минимально допустимое значение.

### Параметры

`min`

INTEGER

Минимально допустимое значение.

`crt.Max`

Устанавливает максимально допустимое значение.

### Параметры

max

INTEGER

Максимально допустимое значение.

`crt.EmptyOrWhiteSpace`

Устанавливает обязательность заполнения поля. Не допускается заполнение поля пробелами.

## Секция converters JS

### Основы

**Конвертеры** — функции модификации значения атрибута `viewModel`, который привязан к свойству визуального компонента, в другое значение. Пример использования конвертера приведен в статье [Реализовать конвертацию значения поля на странице](#).

**Базовые конвертеры**, которые предоставляет Creatio 8 Atlas, представлены ниже.

## Базовые конвертеры

`crt.ToBoolean`

Конвертирует значения других типов в тип `BOOLEAN`.

### Значения

value	ANY	Входящее значение. Значение, которое не является типом <code>BOOLEAN</code> .
result	BOOLEAN	Исходящее значение. Конвертированное значение типа <code>BOOLEAN</code> .

`crt.InvertBooleanValue`

Конвертирует значение типа `BOOLEAN` в противоположное значение типа `BOOLEAN`. Например, если входящее значение — `true`, то возвращается значение `false` и наоборот.

### Значения

value	BOOLEAN	Входящее значение.
result	BOOLEAN	Исходящее значение.

`crt.ToEmailLink`

Конвертирует адрес электронной почты в ссылку, добавляя в начало адреса префикс `mailto:`.

#### Значения

value	TEXT	Входящее значение. Адрес электронной почты.
result	TEXT	Исходящее значение. Ссылка на адрес электронной почты.

`crt.ToObjectProp`

Получает значение указанного свойства объекта.

#### Значения

value	CUSTOM_OBJECT	Входящее значение. Имя свойства объекта.
result	ANY	Исходящее значение. Значение свойства объекта.

#### Параметры

prop <b>required</b>	TEXT	Имя свойства, которое необходимо получить.
defaultValue <b>optional</b>	ANY	Значение, которое возвращается, если свойства не существует или в ответе возвращено значение <code>false</code> .

`crt.ToPhoneLink`

Конвертирует номер телефона в ссылку, добавляя в начало адреса префикс `tel:`.

#### Значения

value	TEXT	Входящее значение. Номер телефона.
result	TEXT	Исходящее значение. Ссылка на номер телефона.

## Страница



**Страница** — элемент приложения, который позволяет управлять внешним видом элементов, работать с источниками данных и произвольным образом размещать компоненты на странице. Каждая страница представлена схемой [клиентского модуля](#). Например, домашняя страница Studio сконфигурирована в схеме `StudioHomePage` пакета `UIv2`. Функциональность базовой страницы реализована в схеме `BaseTemplate` пакета `UIv2`. Все схемы страниц записи должны наследовать страницу `BaseTemplate` или ее наследников. Примеры страницы: страница с островами слева (схема `PageWithLeftAreaTemplate` пакета `UIv2`), страница с реестром (схема `BaseGridSectionTemplate` пакета `UIv2`).

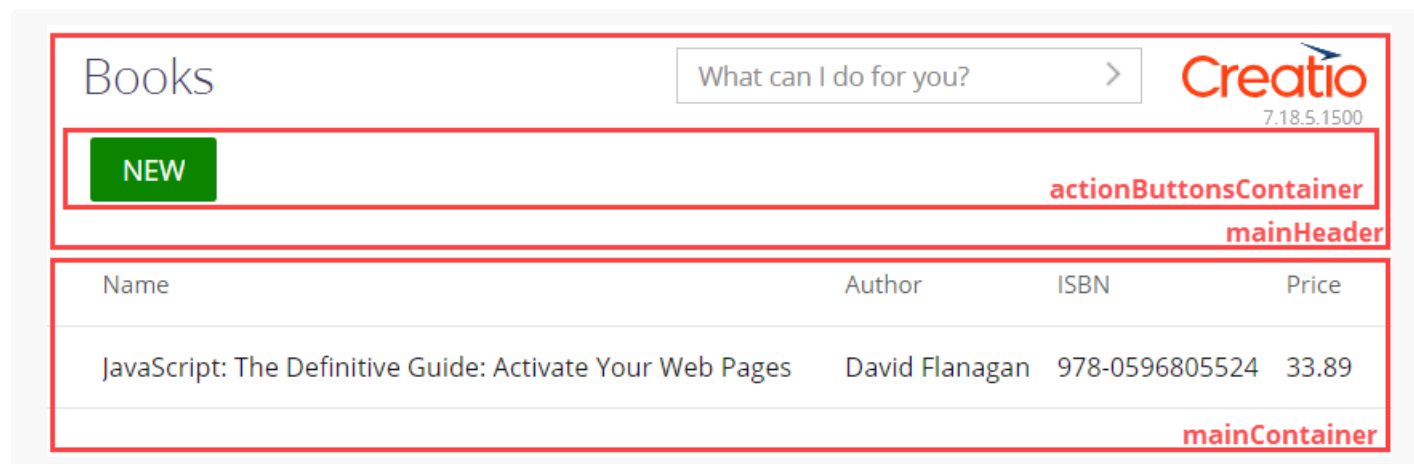
В зависимости от выбранного шаблона приложения, при создании приложение может содержать страницу с реестром и страницу записи с минимальным набором компонентов. Дальнейшая кастомизация выполняется в **дизайнере интерфейсов** с помощью no-code инструментов. Инструкция по настройке элементов в дизайнере интерфейсов содержится в статье [Дизайнер интерфейсов](#).

## Контейнеры страницы

Элементы пользовательского интерфейса приложения, которые относятся к странице, размещены в соответствующих контейнерах. Контейнеры конфигурируются в базовой схеме страницы или схеме замещающей страницы. Контейнеры не зависят от типа страницы.

**На заметку.** В приложении используются мета-имена html-контейнеров. На основании мета-имен приложение формирует фактические идентификаторы соответствующих html-элементов страницы.

Основные **контейнеры** страницы представлены на рисунке ниже.



- Контейнер заголовка страницы (`mainHeader`) — заголовок страницы и вложенный контейнер `actionButtonsContainer`.
- Контейнер действий страницы (`actionButtonsContainer`) — действия над страницей (например, сохранить, открыть и т. д.).
- Контейнер контента страницы (`mainContainer`) — контент страницы.

Создание страницы описано в статье [Настроить приложение](#).

## Структура страницы

**Структурные элементы** страницы в Creatio Freedom UI:

- **Данные.** Подробнее читайте в статье [Дизайнер интерфейсов](#).
- **Графики.** Подробнее читайте в статье [Дизайнер интерфейсов](#).
- **Компоненты** (кнопка, список, надпись, группы, меню управление группами, панель действий). Подробнее читайте в статье [Дизайнер интерфейсов](#).
- **Элементы разметки.**
  - `FlexContainer` — компонент разметки, который позволяет настроить расположение нескольких элементов последовательно в строку или колонку. Элементы могут изменять размер в зависимости от контента. Построен на базе компонента `CSS Flexible Box`.
  - `GridContainer` — компонент разметки, который позволяет настроить расположение нескольких элементов последовательно на сетке. Элементы могут изменять размер в зависимости от контента. Построен на базе `CSS Grid Layout`.


Подробнее читайте в статье [Дизайнер интерфейсов](#).

## Реализовать валидацию значения поля на странице

 Средний

**Пример.** На страницу записи пользовательского раздела [ *Validators* ] добавить валидатор, который проверяет, что в поле [ *Название* ] ([ *Name* ]) установлено любое значение за исключением значения `test`.

### 1. Настроить интерфейс страницы

1. Используя шаблон [ *Данные и бизнес-процессы* ] ([ *Records & business processes* ]), создайте пользовательское приложение `Validators`. Для этого воспользуйтесь инструкцией, которая приведена в статье [Создать пользовательское приложение](#).
2. В рабочей области страницы приложения `Validators` откройте страницу [ *Страница записи Validators* ] ([ *Validators form page* ]).  
Поле [ *Название* ] ([ *Name* ]) по умолчанию добавлено на страницу [ *Страница записи Validators* ] ([ *Validators form page* ]).
3. На панели действий Freedom UI дизайнера нажмите на кнопку . После сохранения настроек страницы открывается исходный код страницы Freedom UI.

### 2. Настроить валидацию значения поля

Бизнес-логика настраивается в дизайнере клиентского модуля. В этом примере настроим валидацию значения поля. Валидатор добавляем к полю [ *Название* ] ([ *Name* ]) страницы [ *Страница записи Validators* ] ([ *Validators form page* ]).

1. В секции `validators` реализуйте пользовательский валидатор `usr.MyValidator`.

#### Секция `validators`

```
validators: /**SCHEMA_VALIDATORS*/{
  /* Тип валидатора обязательно должен иметь вендорный префикс.
  Тип валидатора необходимо указывать в стиле PascalCase. */
  "usr.MyValidator": {
    "validator": function (config) {
      return function (control) {
        return control.value !== config.invalidName ? null: {
          "usr.MyValidator": { message: config.message }
        };
      };
    },
    "params": [
      {
        "name": "invalidName"
      },
      {
        "name": "message"
      }
    ],
    "async": false
  }
}/**SCHEMA_VALIDATORS*/
```

2. В секции `viewModelConfig` привяжите валидатор `MyValidator` к атрибуту `UserName` модели. В свойстве `invalidName` укажите значение "test". При вводе этого значения отображается сообщение об ошибке, которое указано в свойстве `message`.

#### Секция `viewModelConfig`

```
viewModelConfig: /**SCHEMA_VIEW_MODEL_CONFIG*/{
  "attributes": {
    "UserName": {
      ...,
      "validators": {
        /* Привязывает пользовательский валидатор к атрибуту. */
        "MyValidator": {
          "type": "usr.MyValidator",
          "params": {
            "invalidName": "test",
```

```

        "message": "Invalid name"
      }
    }
  },
  ...
}
}/**SCHEMA_VIEW_MODEL_CONFIG*/,

```

[Полный исходный код схемы страницы](#)

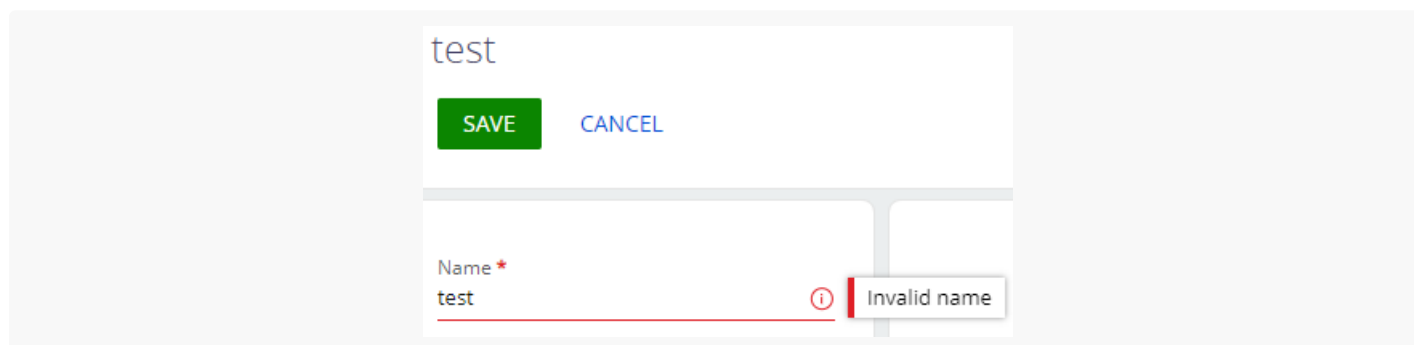
3. На панели инструментов дизайнера клиентского модуля нажмите [ *Сохранить* ] ([ *Save* ]).

## Результат выполнения примера

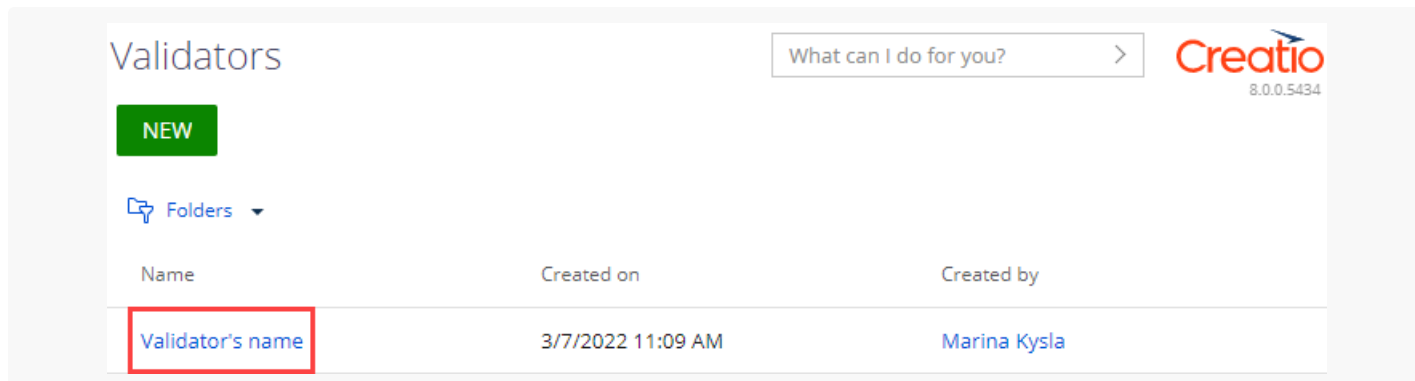
Чтобы **посмотреть результат выполнения примера**:

1. Перейдите на страницу приложения `Validators` и нажмите [ *Запустить приложение* ] ([ *Run app* ]).
2. На панели инструментов приложения `Validators` нажмите [ *Добавить* ] ([ *New* ]).
3. В поле [ *Название* ] ([ *Name* ]) введите значение "test".
4. На панели инструментов страницы валидатора нажмите [ *Сохранить* ] ([ *Save* ]).

В результате выполнения примера запись `test` не сохраняется и приложение выдает всплывающее уведомление об ошибке.



Сохранение другой записи (например, с именем `Validator's name`) выполняется корректно. Запись отображается в реестре раздела `Validators`.




# Реализовать конвертацию значения поля на странице

 Средний

**Пример.** На страницу записи пользовательского раздела [ *Converters* ] ([ *Records & business processes* ]), добавьте конвертер, который конвертирует значение поля [ *Название* ] ([ *Name* ]) в верхний регистр. Значение поля [ *Название* ] ([ *Name* ]) остается неизменным, а конвертированное значение отображается в компоненте типа [ *Надпись* ] ([ *Label* ]).

## 1. Настроить интерфейс страницы

- Используя шаблон [ *Данные и бизнес-процессы* ] ([ *Records & business processes* ]), создайте пользовательское приложение `Converters`. Для этого воспользуйтесь инструкцией, которая приведена в статье [Создать пользовательское приложение](#).
- В рабочей области страницы приложения `Converters` откройте страницу [ *Страница записи Converters* ] ([ *Converters form page* ]).  
Поле [ *Название* ] ([ *Name* ]) по умолчанию добавлено на страницу [ *Страница записи Converters* ] ([ *Converters form page* ]).
- В рабочую область Freedom UI дизайнера добавьте компонент типа [ *Надпись* ] ([ *Label* ]).
- На панели действий Freedom UI дизайнера нажмите на кнопку . После сохранения настроек страницы открывается исходный код страницы Freedom UI.

## 2. Настроить конвертацию значения поля

Бизнес-логика настраивается в дизайнера клиентского модуля. В этом примере настроим конвертацию значения поля. Конвертируем значение поля [ *Название* ] ([ *Name* ]) страницы [ *Страница записи Converters* ] ([ *Converters form page* ]).

- В секции `converters` реализуйте пользовательский конвертер `usr.ToUpperCase`.

**Секция** converters

```
converters: /**SCHEMA_CONVERTERS*/{
    /* Пользовательский конвертер. Конвертирует значение в верхний регистр. */
    "usr.ToUpperCase": function(value) {
        return value?.toUpperCase() ?? '';
    }
}/**SCHEMA_CONVERTERS*/,
```

2. В секции `viewConfigDiff` привяжите свойство `caption` элемента `Label` к атрибуту `$UserName` модели. `$UserName` — значение поля [ *Название* ] ([ *Name* ]). К атрибуту `$UserName` добавьте конвертер `usr.ToUpperCase`.

**Секция** viewConfigDiff

```
viewConfigDiff: /**SCHEMA_VIEW_CONFIG_DIFF*/[
    ...,
    {
        "operation": "insert",
        "name": "Label",
        "values": {
            ...,
            /* Привязывает конвертер usr.ToUpperCase к атрибуту $UserName. */
            "caption": "$UserName | usr.ToUpperCase",
            ...
        },
    },
    ...
]
```

[Полный исходный код схемы страницы](#)

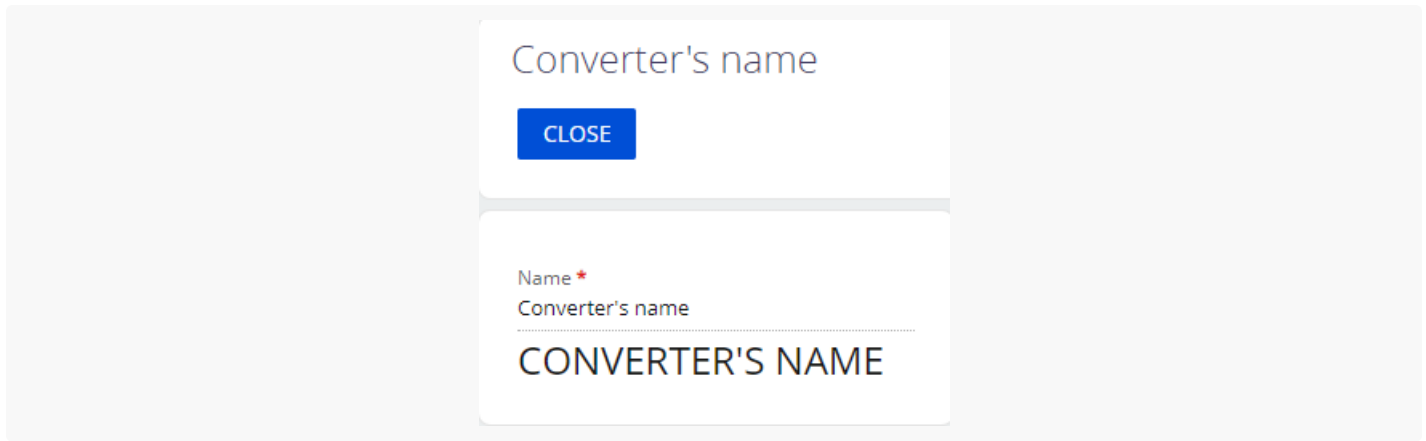
3. На панели инструментов дизайнера клиентского модуля нажмите [ *Сохранить* ] ([ *Save* ]).

## Результат выполнения примера

Чтобы **посмотреть результат выполнения примера**:

1. Перейдите на страницу приложения `Converters` и нажмите [ *Запустить приложение* ] ([ *Run app* ]).
2. На панели инструментов приложения `Converters` нажмите [ *Добавить* ] ([ *New* ]).
3. В поле [ *Название* ] ([ *Name* ]) введите значение "Converters's name".

В результате выполнения примера на странице конвертера при заполнении поля [ *Название* ] ([ *Name* ]) его значение конвертируется в верхний регистр и отображается в компоненте типа [ *Надпись* ] ([ *Label* ]). При этом значение поля [ *Название* ] ([ *Name* ]) не меняется.




# Настроить условие отображения поля на странице


 Средний


**Пример.** Настроить условие отображения поля [ *Количество доступных дней* ] ([ *Sick leave, days left* ]) на странице записи пользовательского раздела [ *Requests* ]. Поле отображается для заявки, которую инициировал сотрудник (т. е. в поле [ *Тип инициатора* ] ([ *Originator type* ]) заявки выбрано значение "Сотрудник" ("Employee")).

## 1. Настроить интерфейс страницы

1. Используя шаблон [ *Данные и бизнес-процессы* ] ([ *Records & business processes* ]), создайте пользовательское приложение `Requests`. Для этого воспользуйтесь инструкцией, которая приведена в статье [Создать пользовательское приложение](#).
2. В рабочей области страницы приложения `Requests` откройте страницу [ *Страница записи Requests* ] ([ *Requests form page* ]).  
Поле [ *Название* ] ([ *Name* ]) по умолчанию добавлено на страницу [ *Страница записи Requests* ] ([ *Requests form page* ]).
3. Добавьте **поле, которое содержит тип инициатора заявки**.
  - a. В рабочую область Freedom UI дизайнера добавьте новое поле типа [ *Выпадающий список* ] ([ *Dropdown* ]).
  - b. На панели действий Freedom UI дизайнера нажмите на кнопку  и на панели настройки заполните **свойства поля**:
    - [ *Заголовок* ] ([ *Title* ]) — "Тип инициатора" ("Originator type").
    - [ *Код (на английском)* ] ([ *Code* ]) — "UsrOriginatorType".
    - [ *Выбор объекта* ] ([ *Lookup* ]) — выберите "Тип контакта" ("Contact type").

4. Добавьте **поле, которое содержит количество доступных дней**.

- В рабочую область Freedom UI дизайнера добавьте новое поле типа [ Число ] ([ Number ]).
- На панели действий Freedom UI дизайнера нажмите на кнопку  и на панели настройки заполните **свойства поля**:
  - [ Заголовок ] ([ Title ]) — "Количество доступных дней" ("Sick leave, days left").
  - [ Код (на английском) ] ([ Code ]) — "UsrSickDaysLeft".

5. На панели действий Freedom UI дизайнера нажмите на кнопку . После сохранения настроек страницы открывается исходный код страницы Freedom UI.

## 2. Настроить условие отображения поля

Бизнес-логика настраивается в дизайнера клиентского модуля. В этом примере настроим условие отображения поля.

- В секцию `viewModelConfig` добавьте атрибут `IsRequestFromEmployee`, который хранит информацию о типе контакта, который инициировал заявку.



**Секция** `viewModelConfig`

```
viewModelConfig: /**SCHEMA_VIEW_MODEL_CONFIG*/{
  "attributes": {
    ...,
    /* Атрибут, который хранит тип инициатора заявки. */
    "IsRequestFromEmployee": {}
  }
}/**SCHEMA_VIEW_MODEL_CONFIG*/,
```

2. В секции `viewConfigDiff` привяжите свойство `visible` элемента `UsrSickDaysLeft` к атрибуту `IsRequestFromEmployee` модели. Если заявка инициирована контактом типа [ *Сотрудник* ] ([ *Employee* ]), то отображается поле [ *Количество доступных дней* ] ([ *Sick leave, days left* ]). Для другого типа контакта поле скрыто.

**Секция** `viewConfigDiff`

```
viewConfigDiff: /**SCHEMA_VIEW_CONFIG_DIFF*/[
  ...,
  {
    "operation": "insert",
    "name": "UsrSickDaysLeft",
    "values": {
      ...,
      /* Свойство, которое определяет видимость поля. Привязано к значению атрибута IsR
      "visible": "$IsRequestFromEmployee"
    },
    ...
  }
]/**SCHEMA_VIEW_CONFIG_DIFF*/,
```

3. В секции `handlers` добавьте пользовательскую реализацию обработчика системного запроса `crt.HandleViewModelAttributeChangeRequest`. Обработчик выполняется при изменении значения любого атрибута (в т. ч. при загрузке значений атрибутов из источника данных). Обработчик проверяет значение атрибута `UsrOriginatorType`. Если новое значение атрибута ссылается на значение "Сотрудник" ("Employee") справочника [ *Тип контакта* ] ([ *Contact type* ]), то для атрибута `IsRequestFromEmployee` устанавливается значение `true`, в другом случае — `false`. Уникальный идентификатор контакта типа [ *Сотрудник* ] ([ *Employee* ]), который установлен в качестве значения константы `employeeOriginatorTypeId`, содержится в соответствующей строке записи справочника [ *Тип контакта* ] ([ *Contact type* ]). В нашем примере идентификатор контакта типа [ *Сотрудник* ] ([ *Employee* ]) — "60733efc-f36b-1410-a883-16d83cab0980".

**Секция** `handlers`

```
handlers: /**SCHEMA_HANDLERS*/[
```

```

{
  request: "crt.HandleViewModelAttributeChangeRequest",
  /* Пользовательская реализация обработчика системного запроса. */
  handler: async (request, next) => {
    /* Проверяет тип инициатора заявки. */
    if (request.attributeName === 'UsrOriginatorType') {
      const employeeOriginatorTypeId = '60733efc-f36b-1410-a883-16d83cab0980';
      const selectedOriginatorType = await request.$context.UsrOriginatorType;
      const selectedOriginatorTypeId = selectedOriginatorType?.value;
      /* Если инициатор заявки – сотрудник, то атрибуту IsRequestFromEmployee присв
      request.$context.IsRequestFromEmployee = selectedOriginatorTypeId === employee
    }
    /* Вызываем следующий обработчик, если такой есть, и возвращаем его результат. */
    return next?.handle(request);
  }
}
]/**SCHEMA_HANDLERS*/,

```

[Полный исходный код схемы страницы](#)

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

## Результат выполнения примера

Чтобы **посмотреть результат выполнения примера**:

1. Перейдите на страницу приложения `Requests` и нажмите [ *Запустить приложение* ] ([ *Run app* ]).
2. На панели инструментов приложения `Requests` нажмите [ *Добавить* ] ([ *New* ]).
3. В поле [ *Название* ] ([ *Name* ]) введите значение "Sick leave".
4. В поле [ *Тип инициатора* ] ([ *Originator type* ]) выберите "Сотрудник" ("Employee").

В результате выполнения примера поле [ *Количество доступных дней* ] ([ *Sick leave, days left* ]) отображается на странице заявки для контакта типа [ *Сотрудник* ] ([ *Employee* ]).

Поле [ *Количество доступных дней* ] ([ *Sick leave, days left* ]) не отображается для другого типа сотрудника (например, [ *Клиент* ] ([ *Customer* ])).

## Настроить условие блокировки поля на странице

 Средний

Пример реализован на front-end стороне приложения. Реализация на back-end стороне описана в блоке статей [Управление доступом](#).

**Пример.** Настроить условие блокировки поля [ *Заявитель* ] ([ *Applicant* ]) на странице записи пользовательского раздела [ *Requests* ]. Поле заблокировано для выполненной заявки (т. е. в поле [ *Состояние* ] ([ *Status* ]) выбрано значение "Выполнена" ("Completed")).

# 1. Настроить интерфейс страницы


1. Используя шаблон [ *Данные и бизнес-процессы* ] ([ *Records & business processes* ]), создайте пользовательское приложение `Requests`. Для этого воспользуйтесь инструкцией, которая приведена в статье [Создать пользовательское приложение](#).

2. В рабочей области страницы приложения `Requests` откройте страницу [ *Страница записи Requests* ] ([ *Requests form page* ]).

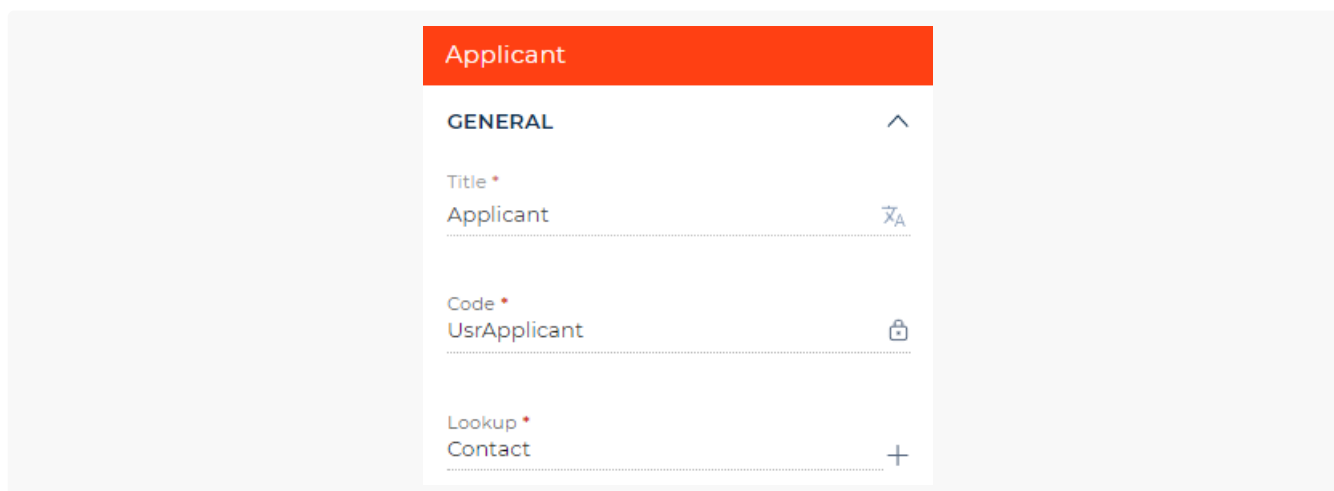
Поле [ *Название* ] ([ *Name* ]) по умолчанию добавлено на страницу [ *Страница записи Requests* ] ([ *Requests form page* ]).

3. Добавьте **поле, которое содержит заявителя**.

a. В рабочую область Freedom UI дизайнера добавьте новое поле типа [ *Выпадающий список* ] ([ *Dropdown* ]).

b. На панели действий Freedom UI дизайнера нажмите на кнопку  и на панели настройки заполните **свойства поля**:

- [ *Заголовок* ] ([ *Title* ]) — "Заявитель" ("Applicant").
- [ *Код (на английском)* ] ([ *Code* ]) — "UsrApplicant".
- [ *Выбор объекта* ] ([ *Lookup* ]) — выберите "Контакт" ("Contact").





The screenshot shows a configuration window for a field named 'Applicant'. The window has a red header with the title 'Applicant'. Below the header, there is a section titled 'GENERAL' with a collapse arrow. The configuration properties are as follows:

Property	Value	Icon
Title *	Applicant	✕A
Code *	UsrApplicant	🔒
Lookup *	Contact	+

4. Добавьте **поле, которое содержит состояние заявки**.

a. В рабочую область Freedom UI дизайнера добавьте новое поле типа [ *Выпадающий список* ] ([ *Dropdown* ]).

b. На панели действий Freedom UI дизайнера нажмите на кнопку  и на панели настройки заполните **свойства поля**:

- [ *Заголовок* ] ([ *Title* ]) — "Состояние" ("Status").
- [ *Код (на английском)* ] ([ *Code* ]) — "UsrStatus".
- [ *Выбор объекта* ] ([ *Lookup* ]) — нажмите на кнопку  и заполните **свойства справочника**:
  - [ *Заголовок* ] ([ *Title* ]) — "Состояние заявки" ("Request status").

- [ Код (на английском) ] ([ Code ]) — "UsrRequestStatusLookup".

New lookup

Title \*  
Request status

Code \*  
UsrRequestStatusLookup

Cancel Save

Для добавления справочника нажмите [ Сохранить ] ([ Save ]).

Status

GENERAL

Title \*  
Status

Code \*  
UsrStatus


Lookup \*  
Request status

h. На панели инструментов Freedom UI дизайнера нажмите на кнопку [ Сохранить ] ([ Save ]).

5. Заполните **справочник** [ Состояние заявки ] ([ Request status ]).

- Перейдите на страницу приложения `Requests` и нажмите [ Запустить приложение ] ([ Run app ]).
- Перейдите в дизайнер системы по кнопке . В блоке [ Настройка системы ] ([ System setup ]) перейдите по ссылке [ Справочники ] ([ Lookups ]).
- Для Creatio версии 8.0.0 зарегистрируйте **справочник**. Начиная с версии 8.0.1 справочник регистрируется автоматически.
  - На панели инструментов раздела [ Справочники ] ([ Lookups ]) нажмите [ Добавить справочник ] ([ New lookup ]) и заполните **свойства справочника**:
    - [ Название ] ([ Name ]) — "Состояние заявки" ("Request status").
    - [ Объект ] ([ Object ]) — выберите "Состояние заявки" ("Request status").

- d. На панели инструментов страницы настройки справочника нажмите [ Сохранить ] ([ Save ]) для сохранения справочника.
- d. Откройте справочник [ Состояние заявки ] ([ Request status ]).
- e. На панели инструментов страницы настройки справочника нажмите [ Добавить ] ([ New ]) и заполните **значения справочника**:
- "Новая" ("New").
  - "Оценивается" ("Under evaluation").
  - "Выполняется" ("In progress").
  - "Отменена" ("Canceled").
  - "Выполнена" ("Completed").

6. Перейдите на страницу [ Страница записи Requests ] ([ Requests form page ]) и на панели действий Freedom UI дизайнера нажмите на кнопку . После сохранения настроек страницы открывается исходный код страницы Freedom UI.

## 2. Настроить условие блокировки поля

Бизнес-логика настраивается в дизайнере клиентского модуля. В этом примере настроим условие блокировки поля.

1. В секцию `viewModelConfig` добавьте атрибут `IsApplicantReadOnly`, который хранит информацию о разрешении контакта на редактирование поля [ Заявитель ] ([ *Applicant* ]).

### Секция `viewModelConfig`

```
viewModelConfig: /**SCHEMA_VIEW_MODEL_CONFIG*/{
  "attributes": {
    ...,
    /* Атрибут, который регулирует доступность поля [Заявитель] для редактирования. */
    "IsApplicantReadOnly": {}
  }
}/**SCHEMA_VIEW_MODEL_CONFIG*/,
```

2. В секции `viewConfigDiff` привяжите свойство `readonly` элемента `UsrApplicant` к атрибуту `IsApplicantReadOnly` модели. Если заявка выполнена, то заблокировано поле [ Заявитель ] ([ *Applicant* ])). Для другого статуса заявки поле доступно для изменения.

### Секция `viewConfigDiff`

```
viewConfigDiff: /**SCHEMA_VIEW_CONFIG_DIFF*/[
  ...,
  {
    "operation": "insert",
    "name": "UsrApplicant",
    "values": {
      ...,
      /* Свойство, которое определяет блокировку поля для редактирования. Привязано к з
      "readonly": "$IsApplicantReadOnly"
    },
    ...
  },
  ...
]/**SCHEMA_VIEW_CONFIG_DIFF*/,
```

3. В секции `handlers` добавьте пользовательскую реализацию обработчика системного запроса `crt.HandleViewModelAttributeChangeRequest`. Обработчик выполняется при изменении значения любого атрибута (в т. ч. при загрузке значений атрибутов из источника данных). Обработчик проверяет значение атрибута `UsrStatus`. Если новое значение атрибута ссылается на значение "Выполнена" ("Completed") справочника [ Состояние заявки ] ([ *Request status* ]), то для атрибута `IsApplicantReadOnly` устанавливается значение `true`, в другом случае — `false`. Уникальный идентификатор статуса выполненной заявки, который установлен в качестве значения константы `completedStatusId`,

содержится в соответствующей колонке строки записи справочника [ *Состояние заявки* ] ([ *Request status* ]). Чтобы отобразить колонку [ *Id* ] в реестре справочника [ *Состояние заявки* ] ([ *Request status* ]), воспользуйтесь инструкцией, которая описана в статье [Реестр раздела](#). В нашем примере идентификатор статуса выполненной заявки — "6d76b4e0-6507-4c34-902b-90e18df84153".

#### Секция `handlers`

```
handlers: /**SCHEMA_HANDLERS*/[
  {
    request: "crt.HandleViewModelAttributeChangeRequest",
    /* Пользовательская реализация обработчика системного запроса. */
    handler: async (request, next) => {
      /* Проверяет статус заявки. */
      if (request.attributeName === 'UsrStatus') {
        const completedStatusId = '6d76b4e0-6507-4c34-902b-90e18df84153';
        const selectedStatus = await request.$context.UsrStatus;
        const selectedStatusId = selectedStatus?.value;
        const isRequestCompleted = selectedStatusId === completedStatusId;
        /* Если статус заявки [Выполнена], то в атрибут IsApplicantReadonly записываем
        request.$context.IsApplicantReadonly = isRequestCompleted;
      }
      /* Вызываем следующий обработчик, если он присутствует, и возвращаем его результа
      return next?.handle(request);
    }
  }
]/**SCHEMA_HANDLERS*/,
```

[Полный исходный код схемы страницы](#)

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

## Результат выполнения примера

Чтобы **посмотреть результат выполнения примера**:

- Перейдите на страницу приложения `Requests` и нажмите [ *Запустить приложение* ] ([ *Run app* ]).
- На панели инструментов приложения `Requests` нажмите [ *Добавить* ] ([ *New* ]).
- В поле [ *Название* ] ([ *Name* ]) введите значение "Requests's name".
- В поле [ *Заявитель* ] ([ *Applicant* ]) выберите "Bruce Clayton".
- В поле [ *Состояние* ] ([ *Status* ]) выберите "Выполнена" ("Completed").

В результате выполнения примера поле [ *Заявитель* ] ([ *Applicant* ]) заблокировано для выполненной заявки.



Requests's name

SAVE CANCEL

Name \*  
Requests's name

Applicant \*  
Bruce Clayton

Status  
Completed

Поле [ Заявитель ] ([ Applicant ]) доступно для изменения для другого состояния заявки (например, "Новая" ("New")).

Requests's name

SAVE CANCEL

Name \*  
Requests's name

Applicant \*  
Bruce Clayton


Status  
New

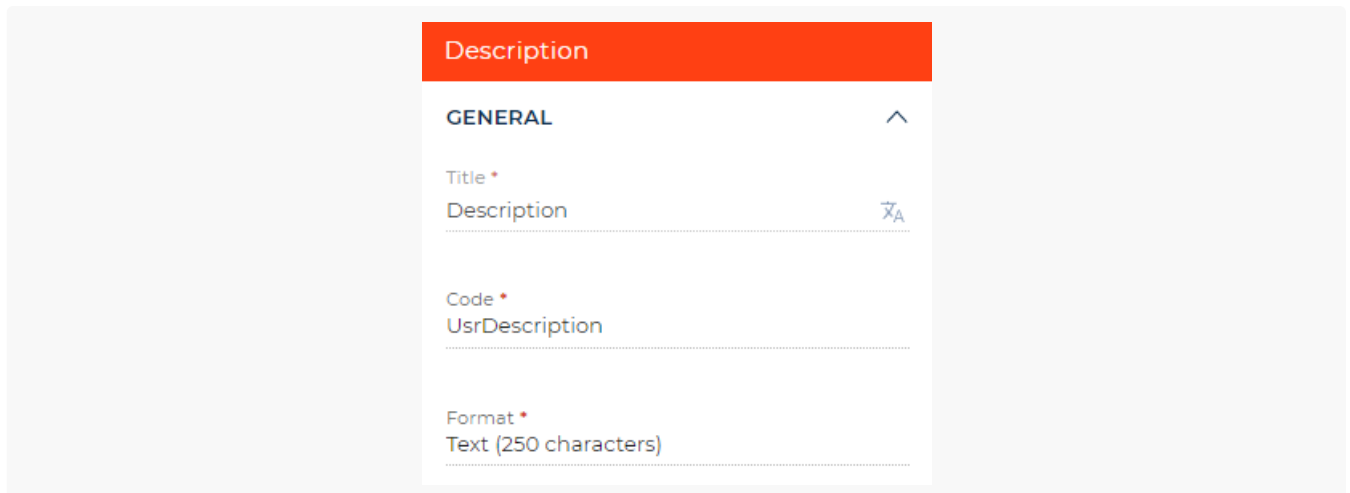
## Настроить условие заполнения поля на странице


 Средний

**Пример.** Настроить условие заполнения поля [ Описание ] ([ Description ]) на странице записи пользовательского раздела [ Requests ]. Если совпадают значения полей [ Название ] ([ Name ]) и [ Описание ] ([ Description ]), то поле [ Описание ] ([ Description ]) заполняется новым значением поля [ Название ] ([ Name ]). В другом случае значение поля [ Описание ] ([ Description ]) не меняется.

### 1. Настроить интерфейс страницы

1. Используя шаблон [ *Данные и бизнес-процессы* ] ([ *Records & business processes* ]), создайте пользовательское приложение `Requests`. Для этого воспользуйтесь инструкцией, которая приведена в статье [Создать пользовательское приложение](#).
  2. В рабочей области страницы приложения `Requests` откройте страницу [ *Страница записи Requests* ] ([ *Requests form page* ]).
- Поле [ *Название* ] ([ *Name* ]) по умолчанию добавлено на страницу [ *Страница записи Requests* ] ([ *Requests form page* ]).
3. Добавьте **поле, которое содержит описание заявки**.
    - a. В рабочую область Freedom UI дизайнера добавьте новое поле типа [ *Текст* ] ([ *Text* ]).
    - b. На панели действий Freedom UI дизайнера нажмите на кнопку  и на панели настройки заполните **свойства поля**:
      - [ *Заголовок* ] ([ *Title* ]) — "Описание" ("Description").
      - [ *Код (на английском)* ] ([ *Code* ]) — "UsrDescription".



4. На панели действий Freedom UI дизайнера нажмите на кнопку . После сохранения настроек страницы открывается исходный код страницы Freedom UI.

## 2. Настроить условие заполнения поля

Бизнес-логика настраивается в дизайнере клиентского модуля. В этом примере настроим условие заполнения поля.

1. В секции `handlers` добавьте пользовательскую реализацию обработчика системного запроса `crt.HandleViewModelAttributeChangeRequest`. Обработчик выполняется при изменении значения любого атрибута (в т. ч. при загрузке значений атрибутов из источника данных). Обработчик проверяет значение атрибута `UsrName`. Если старое значение атрибута совпадает со значением атрибута `UsrDescription`, то для атрибута `UsrDescription` устанавливаем такое же значение, как и новое значение атрибута `UsrName`.

**Секция** `handlers`

```
handlers: /**SCHEMA_HANDLERS*/[
  {
    request: "crt.HandleViewModelAttributeChangeRequest",
    /* Пользовательская реализация обработчика системного запроса. */
    handler: async (request, next) => {
      /* Если изменяется поле UsrName, то выполняются следующие шаги.*/
      if (request.attributeName === 'UsrName') {
        /* Проверяет равно ли старое значение поля UsrName значению поля UsrDescripti
        const isFieldsShouldBeSynchronized = request.oldValue === await request.$co
        if (isFieldsShouldBeSynchronized) {
          /* Присваивает новое значение поля UsrName полю UsrDescription. */
          request.$context.UsrDescription = await request.$context.UsrName;
        }
      }
      /* Вызываем следующий обработчик, если он присутствует, и возвращаем его результа
      return next?.handle(request);
    }
  }
]/**SCHEMA_HANDLERS*/,
```

[Полный исходный код схемы страницы](#)

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

## Результат выполнения примера

Чтобы **посмотреть результат выполнения примера для одинаковых значений полей** [ *Название* ] ([ *Name* ]) и [ *Описание* ] ([ *Description* ]):

1. Перейдите на страницу приложения `Requests` и нажмите [ *Запустить приложение* ] ([ *Run app* ]).
2. На панели инструментов приложения `Requests` нажмите [ *Добавить* ] ([ *New* ]).
3. В поле [ *Название* ] ([ *Name* ]) введите значение "Requests's name".
4. В поле [ *Описание* ] ([ *Description* ]) введите значение "Requests's name".
5. Измените значение поля [ *Название* ] ([ *Name* ]) на "Test".

В результате выполнения примера в поле [ *Описание* ] ([ *Description* ]) установлено значение "Test", как и в поле [ *Название* ] ([ *Name* ]).

Чтобы **посмотреть результат выполнения примера для разных значений полей** [ *Название* ] ([ *Name* ]) и [ *Описание* ] ([ *Description* ]):

1. Измените значение поля [ *Описание* ] ([ *Description* ]) на "Requests's description".
2. В поле [ *Название* ] ([ *Name* ]) введите значение "Test".



В результате выполнения примера значение поля [ *Описание* ] ([ *Description* ]) остается без изменений.

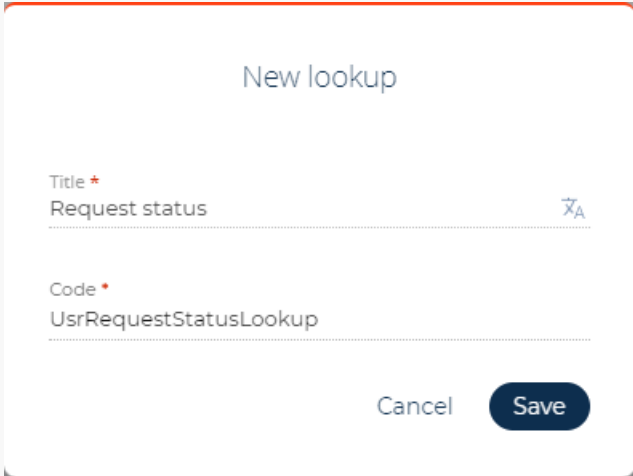
## Настроить условие обязательности поля на странице

 Средний

**Пример.** Настроить обязательность заполнения поля [ *Описание* ] ([ *Description* ]) на странице записи пользовательского раздела [ *Requests* ]. Поле обязательно для новой заявки (т. е. в поле [ *Состояние* ] ([ *Status* ]) выбрано значение "Новая" ("New")).

# 1. Настроить интерфейс страницы

1. Используя шаблон [ Данные и бизнес-процессы ] ([ Records & business processes ]), создайте пользовательское приложение `Requests`. Для этого воспользуйтесь инструкцией, которая приведена в статье [Создать пользовательское приложение](#).
2. В рабочей области страницы приложения `Requests` откройте страницу [ Страница записи Requests ] ([ Requests form page ]).  
Поле [ Название ] ([ Name ]) по умолчанию добавлено на страницу [ Страница записи Requests ] ([ Requests form page ]).
3. Добавьте **поле, которое содержит состояние заявки**.
  - a. В рабочую область Freedom UI дизайнера добавьте новое поле типа [ Выпадающий список ] ([ Dropdown ]).
  - b. На панели действий Freedom UI дизайнера нажмите на кнопку  и на панели настройки заполните **свойства поля**:
    - [ Заголовок ] ([ Title ]) — "Состояние" ("Status").
    - [ Код (на английском) ] ([ Code ]) — "UsrStatus".
    - [ Выбор объекта ] ([ Lookup ]) — нажмите на кнопку  и заполните **свойства справочника**:
      - [ Заголовок ] ([ Title ]) — "Состояние заявки" ("Request status").
      - [ Код (на английском) ] ([ Code ]) — "UsrRequestStatusLookup".



New lookup

Title \*  
Request status

Code \*  
UsrRequestStatusLookup

Cancel Save

Для добавления справочника нажмите [ Сохранить ] ([ Save ]).

h. На панели инструментов Freedom UI дизайнера нажмите на кнопку [ Сохранить ] ([ Save ]).

4. Заполните **справочник** [ Состояние заявки ] ([ Request status ]).

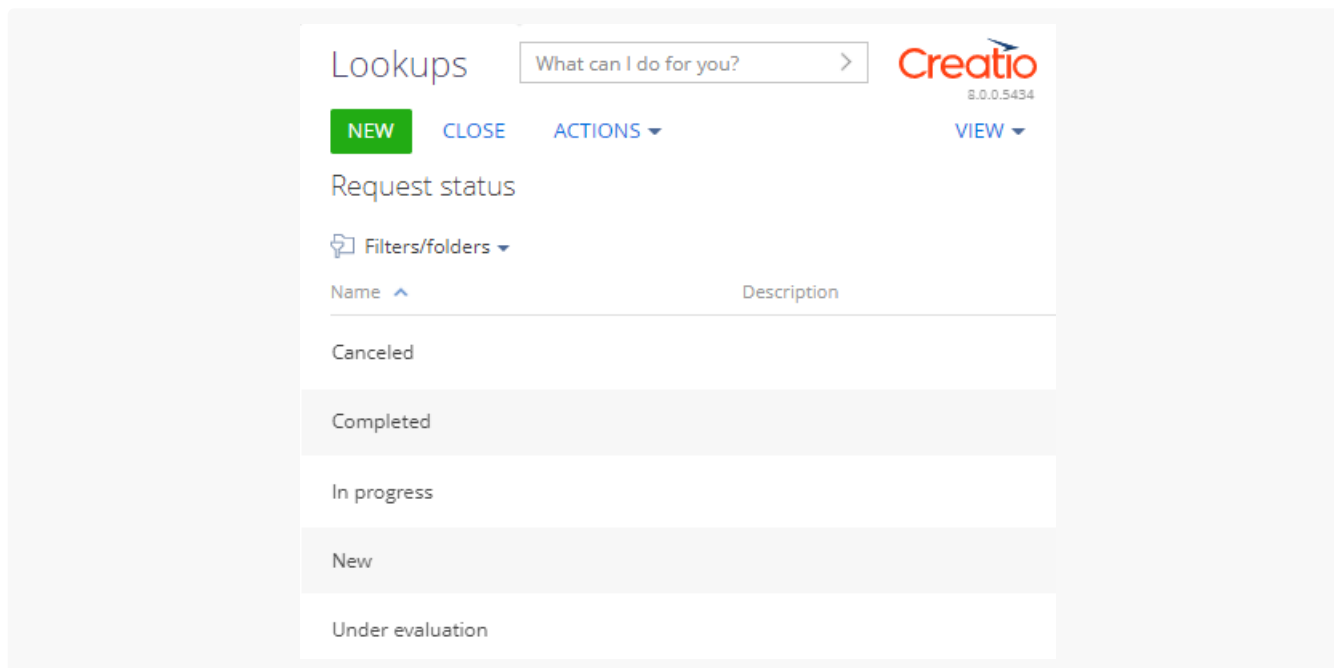
- a. Перейдите на страницу приложения **Requests** и нажмите [ Запустить приложение ] ([ Run app ]).
- b. Перейдите в дизайнер системы по кнопке . В блоке [ Настройка системы ] ([ System setup ]) перейдите по ссылке [ Справочники ] ([ Lookups ]).
- c. Для Creatio версии 8.0.0 зарегистрируйте **справочник**. Начиная с версии 8.0.1 справочник регистрируется автоматически.
  - a. На панели инструментов раздела [ Справочники ] ([ Lookups ]) нажмите [ Добавить справочник ] ([ New lookup ]) и заполните **свойства справочника**:
    - [ Название ] ([ Name ]) — "Состояние заявки" ("Request status").
    - [ Объект ] ([ Object ]) — выберите "Состояние заявки" ("Request status").

d. На панели инструментов страницы настройки справочника нажмите [ Сохранить ] ([ Save ]) для сохранения справочника.


d. Откройте справочник [ Состояние заявки ] ([ Request status ]).

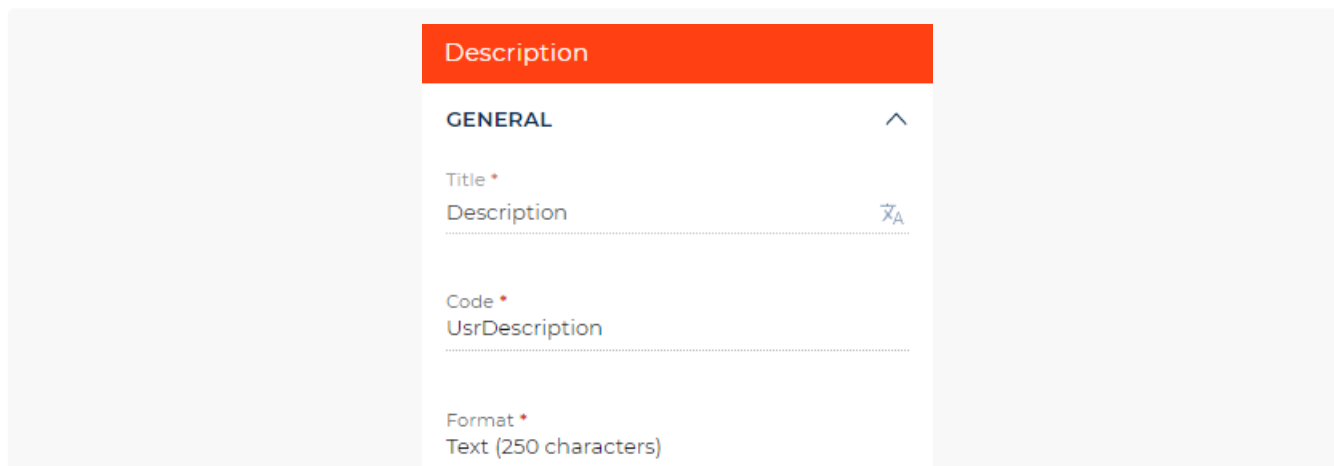
e. На панели инструментов страницы настройки справочника нажмите [ Добавить ] ([ New ]) и заполните **значения справочника**:


- "Новая" ("New").
- "Оценивается" ("Under evaluation").
- "Выполняется" ("In progress").
- "Отменена" ("Canceled").
- "Выполнена" ("Completed").



##### 5. Добавьте поле, которое содержит описание заявки.

- Перейдите на страницу [ Страница записи Requests ] ([ Requests form page ]) и в рабочую область Freedom UI дизайнера добавьте новое поле типа [ Текст ] ([ Text ]).
- На панели действий Freedom UI дизайнера нажмите на кнопку  и на панели настройки заполните **свойства поля**:
  - [ Заголовок ] ([ Title ]) — "Описание" ("Description").
  - [ Код (на английском) ] ([ Code ]) — "UsrDescription".



6. На панели действий Freedom UI дизайнера нажмите на кнопку . После сохранения настроек страницы открывается исходный код страницы Freedom UI.

## 2. Настроить условие обязательности поля

Бизнес-логика настраивается в дизайнера клиентского модуля. В этом примере настроим условие обязательности поля.

1. В секции `viewModelConfig` привяжите валидатор с типом `crt.Required` к атрибуту `UsrDescription` модели. Валидатор проверяет заполнение значения атрибута.

### Секция `viewModelConfig`

```
viewModelConfig: /**SCHEMA_VIEW_MODEL_CONFIG*/{
  "attributes": {
    ...
    "UsrDescription": {
      ...,
      /* Свойство, которое подключает валидаторы к атрибуту. */
      "validators": {
        /* Указывает на обязательность заполнения поля. */
        "required": {
          "type": "crt.Required"
        }
      }
    },
    ...
  }
}/**SCHEMA_VIEW_MODEL_CONFIG*/,
```

2. В секции `handlers` добавьте пользовательскую реализацию обработчика системного запроса `crt.HandleViewModelAttributeChangeRequest`. Обработчик выполняется при изменении значения любого атрибута (в т. ч. при загрузке значений атрибутов из источника данных). Обработчик проверяет значение атрибута `UsrStatus`. Если новое значение атрибута ссылается на значение "Новая" ("New") справочника [ *Состояние заявки* ] ([ *Request status* ]), то валидатор применяется, в другом случае — нет. Уникальный идентификатор статуса новой заявки, который установлен в качестве значения константы `newStatusId`, содержится в соответствующей колонке строки записи справочника [ *Состояние заявки* ] ([ *Request status* ]). Чтобы отобразить колонку [ *Id* ] в реестре справочника [ *Состояние заявки* ] ([ *Request status* ]), воспользуйтесь инструкцией, которая описана в статье [Реестр раздела](#). В нашем примере идентификатор статуса новой заявки — "3be636fa-12b4-40eb-a050-91b1d774a75f".

### Секция `handlers`

```
handlers: /**SCHEMA_HANDLERS*/[
```



```

{
  request: "crt.HandleViewModelAttributeChangeRequest",
  /* Пользовательская реализация обработчика системного запроса. */
  handler: async (request, next) => {
    if (request.attributeName === 'UsrStatus') {
      const newStatusId = '3be636fa-12b4-40eb-a050-91b1d774a75f';
      const selectedStatus = await request.$context.UsrStatus;
      const selectedStatusId = selectedStatus?.value;
      const isNewRequest = selectedStatusId === newStatusId;
      /* Проверяет статус заявки. */
      if (isNewRequest) {
        /* Если заявка новая, то применяет валидатор required к атрибуту UsrDescr
           request.$context.enableAttributeValidator('UsrDescription', 'required');
        */
      } else {
        /* Для других заявок (кроме новых) отключает применение валидатора requir
           request.$context.disableAttributeValidator('UsrDescription', 'required');
        */
      }
      /* Вызываем следующий обработчик, если он присутствует, и возвращаем его резу
        return next?.handle(request);
      */
    }
  }
}
]/**SCHEMA_HANDLERS*/,

```

[Полный исходный код схемы страницы](#)

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

## Результат выполнения примера

Чтобы **посмотреть результат выполнения примера**:

1. Перейдите на страницу приложения `Requests` и нажмите [ *Запустить приложение* ] ([ *Run app* ]).
2. На панели инструментов приложения `Requests` нажмите [ *Добавить* ] ([ *New* ]).
3. В поле [ *Название* ] ([ *Name* ]) введите значение "Requests's name".
4. В поле [ *Состояние* ] ([ *Status* ]) выберите "Новая" ("New").

В результате выполнения примера поле [ *Описание* ] ([ *Description* ]) обязательно для новой заявки.

The screenshot shows a web form titled "Requests". At the top, there are two buttons: a green "SAVE" button and a blue "CANCEL" button. Below the buttons, the form contains three input fields, each with a red asterisk indicating it is required:

- Name \***: The input field contains the text "Requests's name".
- Status \***: The input field contains the text "New".
- Description \***: The input field is empty.

Поле [ *Описание* ] ([ *Description* ]) необязательно для другого состояния заявки (например, "Выполнена" ("Completed")).

This screenshot shows the same "Requests" form, but with the status changed to "Completed". The input fields now contain:

- Name \***: "Requests's name"
- Status \***: "Completed"
- Description**: The field is empty, and the red asterisk is no longer present, indicating it is optional for this status.

## Отобразить значения системных переменных на странице


 Средний

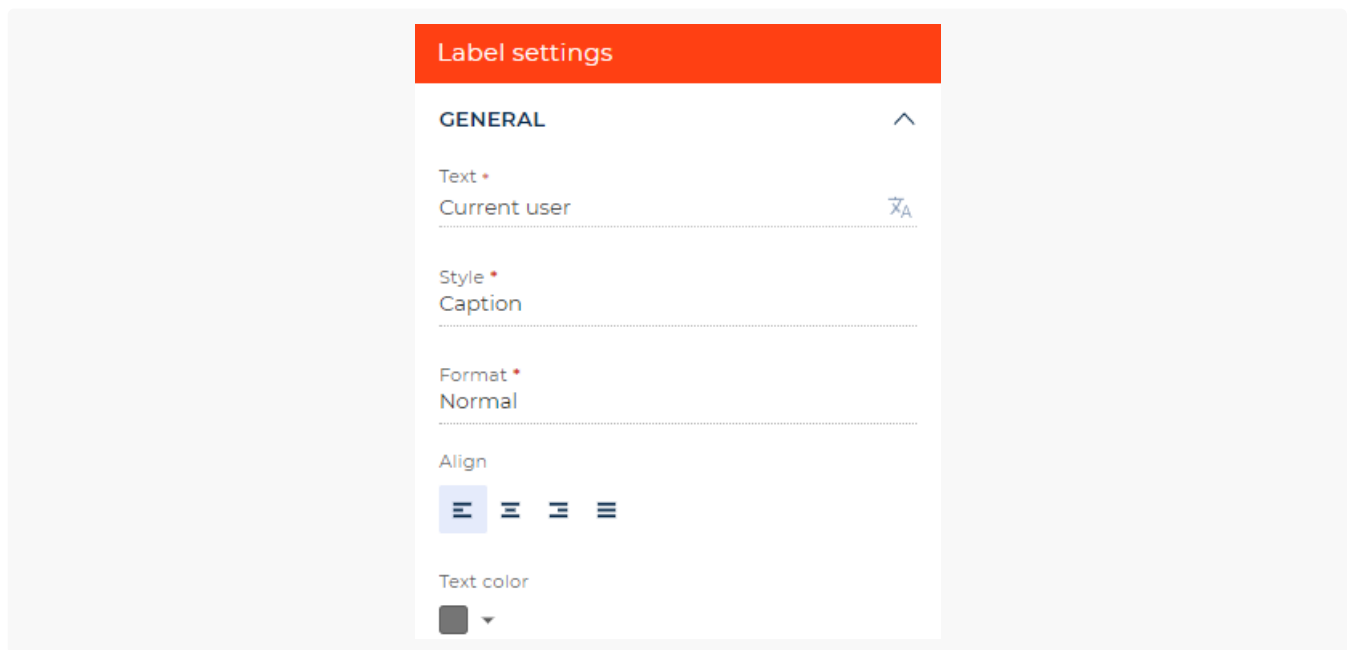
**Пример.** На странице записи пользовательского раздела [ *System variables* ] отобразить:

- Имя текущего пользователя.
- Значение разницы во времени (в часах) между временем в часовом поясе текущего контакта и универсальным временем (UTC).

В качестве значений используются значения соответствующих системных переменных.

# 1. Настроить интерфейс страницы

1. Используя шаблон [ Данные и бизнес-процессы ] ([ Records & business processes ]), создайте пользовательское приложение `System variables`. Для этого воспользуйтесь инструкцией, которая приведена в статье [Создать пользовательское приложение](#).
2. В рабочей области страницы приложения `System variables` откройте страницу [ Страница записи `System variables` ] ([ `System variables form page` ]).
3. Удалите поле [ Название ] ([ `Name` ]), которое по умолчанию добавлено на страницу [ Страница записи `System variables` ] ([ `System variables form page` ]).
4. Добавьте **надпись текущего контакта**.
  - a. В рабочую область Freedom UI дизайнера добавьте компонент типа [ Надпись ] ([ `Label` ]).
  - b. На панели действий Freedom UI дизайнера нажмите на кнопку  и на панели настройки заполните **свойства надписи**:
    - [ Заголовок ] ([ `Title` ]) — "Текущий пользователь" ("Current user").
    - [ Стиль ] ([ `Style` ]) — выберите "Описание" ("Caption").
    - [ Цвет текста ] ([ `Text color` ]) — выберите серый цвет.




5. Аналогично добавьте **надписи**:
  - Значения имени текущего контакта из системной переменной.
  - Разницы во времени с UTC.
  - Значения разницы во времени с UTC из системной переменной.

Свойства надписей, которые необходимо добавить, приведены в таблице ниже.

Значения свойств надписей

Элемент	Свойство	Значение свойства
<b>Надпись, которая содержит значение имени текущего контакта из системной переменной</b>	[ Заголовок ] ([ Title ])	"Текущий пользователь (значение)" ("Current user (value)")
	[ Стиль ] ([ Style ])	Выберите "Обычный текст" ("Body text")
<b>Надпись разницы во времени с UTC</b>	[ Заголовок ] ([ Title ])	"Разница во времени с UTC" ("Contact time offset")
	[ Стиль ] ([ Style ])	Выберите "Описание" ("Caption")
	[ Цвет текста ] ([ Text color ])	Выберите серый цвет
<b>Надпись, которая содержит значение разницы во времени с UTC из системной переменной</b>	[ Заголовок ] ([ Title ])	"Разница во времени с UTC (значение)" ("Contact time offset (value)")
	[ Стиль ] ([ Style ])	Выберите "Обычный текст" ("Body text")

6. На панели действий Freedom UI дизайнера нажмите на кнопку . После сохранения настроек страницы открывается исходный код страницы Freedom UI.

## 2. Настроить получение значений системных переменных

Бизнес-логика настраивается в дизайнера клиентского модуля. В этом примере настроим получение значений системных переменных.

1. Подключите сервис системных переменных `sdk.SysValuesService`. Для этого добавьте в AMD-модуль зависимость `@creatio/sdk`.

### Зависимости AMD-модуля

```
/* Объявление AMD-модуля. */
define("UsrAppSystemvariable_FormPage", /**SCHEMA_DEPS*/["@creatio/sdk"] /**SCHEMA_DEPS*/, function() {
    return {
        ...
    };
});
```

```
};
});
```

2. В секцию `viewModelConfig` добавьте **атрибуты**:

- `CurrentUser` — атрибут, который хранит имя текущего контакта.
- `ContactTimezone` — атрибут, который хранит разницу во времени (в часах) между временем в часовом поясе текущего контакта и UTC.

#### Секция `viewModelConfig`

```
viewModelConfig: /**SCHEMA_VIEW_MODEL_CONFIG*/{
  "attributes": {
    ...,
    /* Атрибут, который хранит имя текущего контакта. */
    "CurrentUser": {},
    /* Атрибут, который хранит разницу во времени (в часах) между временем в часовом поясе
    "ContactTimezone": {}
  }
}/**SCHEMA_VIEW_MODEL_CONFIG*/,
```

3. В секции `viewConfigDiff` измените **значение свойства** `caption`:

- `$CurrentUser` — для элемента `CurrentUserValue`.
- `$ContactTimezone` — для элемента `ContactTimeOffsetValue`.

Свойство `caption` отвечает за текст, который содержит элемент.

#### Секция `viewConfigDiff`

```
viewConfigDiff: /**SCHEMA_VIEW_CONFIG_DIFF*/[
  ...,
  {
    "operation": "insert",
    "name": "CurrentUserValue",
    "values": {
      ...,
      /* Привязка атрибута CurrentUser к свойству caption. */
      "caption": "$CurrentUser",
      ...
    },
    ...
  },
  ...,
  {
    "operation": "insert",
```

```

    "name": "ContactTimeOffsetValue",
    "values": {
        ...,
        /* Привязка атрибута ContactTimezone к свойству caption. */
        "caption": "$ContactTimezone",
        ...
    },
    ...
}
]/**SCHEMA_VIEW_CONFIG_DIFF*/,

```

4. В секции `handlers` добавьте пользовательскую реализацию обработчика системного запроса `crt.HandlerViewModelInitRequest`. Обработчик выполняется при инициализации `View` модели.
  - a. Создайте экземпляр сервиса системных значений из `@creatio/sdk`.
  - b. Загрузите системные значения.
  - c. Установите имя контакта текущего пользователя в атрибут `CurrentUser`.
  - d. Переведите значение разницы в часовых поясах с минут в часы и запишите ее в атрибут `ContactTimezone`.

#### Секция `handlers`

```

handlers: /**SCHEMA_HANDLERS*/[
  {
    request: "crt.HandleViewModelInitRequest",
    /* Пользовательская реализация обработчика системного запроса. */
    handler: async (request, next) => {
      /* Создает экземпляр сервиса системных значений из @creatio/sdk. */
      const sysValuesService = new sdk.SysValuesService();
      /* Загружает системные значения. */
      const sysValues = await sysValuesService.loadSysValues();
      /* Устанавливает имя контакта текущего пользователя в атрибут CurrentUser. */
      request.$context.CurrentUser = sysValues.userContact.displayValue;
      /* Переводит разницу в часовых поясах с минут в часы и записываем ее в атрибут Co
      const offset = sysValues.userTimezoneOffset;
      const offsetDisplayValue = (offset > 0 ? '+' : '-') + (offset / 60) + 'h';
      request.$context.ContactTimezone = offsetDisplayValue;
      /* Вызывает следующий обработчик, если он присутствует, и возвращаем его результа
      return next?.handle(request);
    }
  }
] /**SCHEMA_HANDLERS*/,

```

#### Полный исходный код схемы страницы

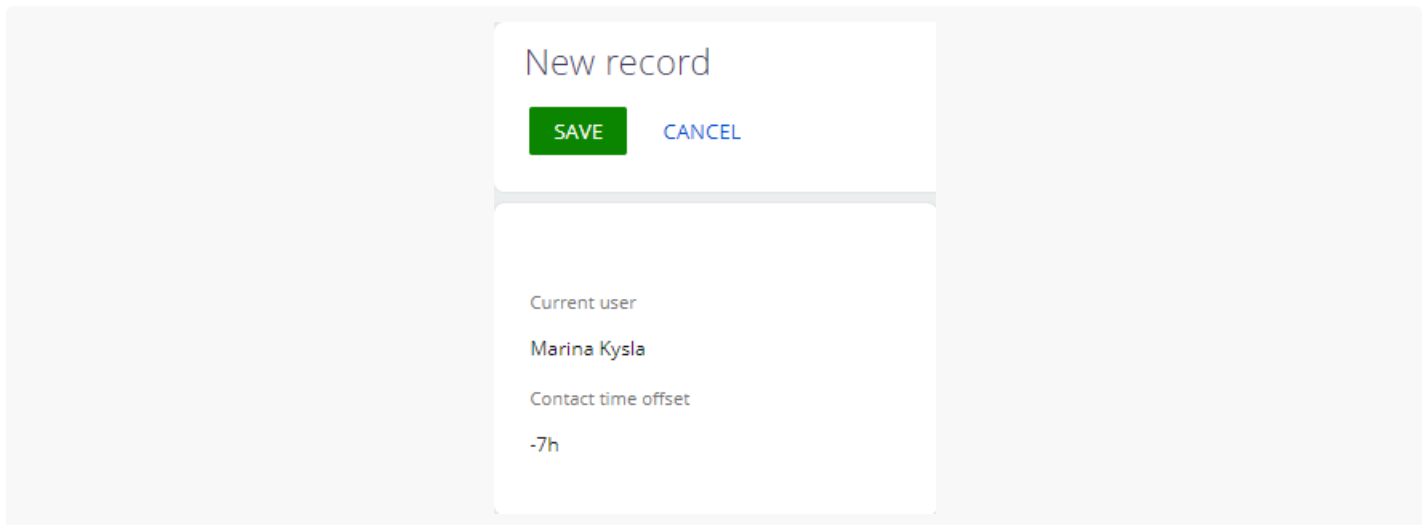
5. На панели инструментов дизайнера клиентского модуля нажмите [ *Сохранить* ] ([ *Save* ]).

## Результат выполнения примера

Чтобы **посмотреть результат выполнения примера**:

1. Перейдите на страницу приложения `System variables` и нажмите [ *Запустить приложение* ] ([ *Run app* ]).
2. На панели инструментов приложения `System variables` нажмите [ *Добавить* ] ([ *New* ]).

В результате выполнения примера на странице записи пользовательского раздела [ *System variables* ] отображаются имя текущего пользователя и значение разницы во времени (в часах) между временем в часовом поясе текущего контакта и UTC. При этом используются значения соответствующих системных переменных.



## Компонент FlexContainer JS

### Основы

`FlexContainer` — компонент разметки, который позволяет настроить расположение нескольких элементов последовательно в строку или колонку. Элементы могут изменять размер в зависимости от контента. Построен на базе компонента `CSS Flexible Box`.

**Действия**, которые позволяет выполнять компонент `FlexContainer` с элементами макета:

- Задаёт направление элементов.
- Выравнивает элементы.
- Распределяет пространство между элементами.

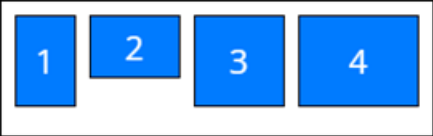
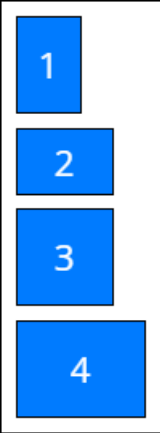

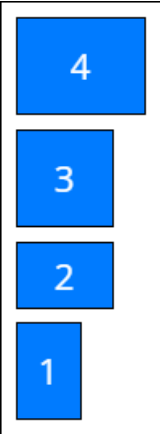
Подробнее о `Flexible Box` читайте в статье [Основные понятия Flexbox](#).

## Свойства

@Input direction

Указывает как flex-элементы располагаются во flex-контейнере относительно главной оси и направления.

Возможные значения




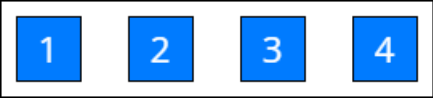
row	Блоки flex-контейнера размещаются в строку.	
column	Блоки flex-контейнера размещаются в столбец.	
row-reverse	Блоки flex-контейнера размещаются в строку, но в обратном направлении.	
column-reverse	Блоки flex-контейнера размещаются в столбец, но в обратном направлении.	

@Input justifyContent

Выравнивание вдоль главной оси. Указывает как браузер распределяет пространство между и вокруг элементов контента вдоль главной оси flex-контейнера.

Возможные значения

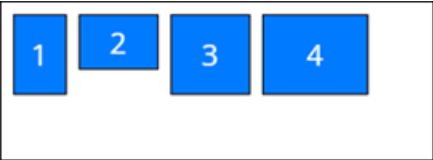
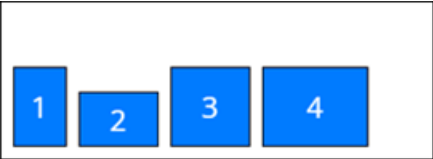
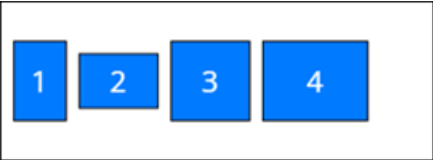


start	Выравнивание элементов по левому краю flex - контейнера.	
end	Выравнивание элементов по правому краю flex - контейнера.	
center	Выравнивание элементов по центру flex -контейнера.	
space-between	Блоки flex -контейнера равномерно размещаются по всей ширине. Первый элемент выровнен по левому краю, последний — по правому.	

@Input alignItems

Выравнивание по вертикали. Выравнивает элементы flex -контейнера, как и свойство @Input justifyContent , но в перпендикулярном направлении.

Возможные значения

flex-start	Выравнивание элементов по верхнему краю flex - контейнера.	
flex-end	Выравнивание элементов по нижнему краю flex - контейнера.	
center	Выравнивание элементов по центру flex -контейнера.	

@Input gap

Задаёт отступы между элементами flex -контейнера в столбцах и строках. Является сокращением для свойств row-gap (отступ между элементами строки) и column-gap (отступ между элементами

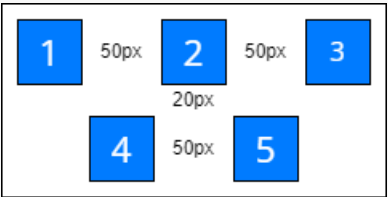
столбца).

Результат настройки свойств `row-gap` и `column-gap` приведен ниже.

**Результат настройки свойств `row-gap` и `column-gap`**

```
.container {
  display: flex;
  flex-wrap: wrap;
  column-gap: 50px;
  row-gap: 20px;
  justify-content: center;
}
```

Результат отображения представлен на рисунке ниже.



**@Input padding**

Задаёт внутренние отступы для контейнера. Может настраиваться как для каждой стороны отдельно, так и для всех сторон одновременно. Может принимать число, строку и возможные значения, которые приведены ниже.

**Возможные значения**

none	Внутренний отступ отсутствует.
small	Маленький размер отступа.
medium	Средний размер отступа.
large	Большой размер отступа.

Пример настройки свойства `padding` приведен ниже.

**Пример настройки свойства `padding`**

```
"padding": {
  "top": "6px",
```

```

    "right": 6,
    "bottom": "small",
    "left": "large"
  }

```

#### @Input border-radius

Задаёт радиус скругления углов для контейнера. Может настраиваться как для каждой стороны отдельно, так и для всех сторон одновременно. Может принимать число, строку и возможные значения, которые приведены ниже.

#### Возможные значения

none	Скругление углов отсутствует.
small	Маленький радиус скругления углов.
medium	Средний радиус скругления углов.
large	Большой радиус скругления углов.

Пример настройки свойства `border-radius` приведен ниже.

#### Пример настройки свойства `border-radius`

```
"borderRadius": "medium"
```

#### @Input color

Задаёт цвет контейнера. В качестве значения принимает код цвета.

Пример настройки свойства `color` приведен ниже.

#### Пример настройки свойства `color`

```
"color": "#FDAB06"
```

## Пример использования

Пример использования компонента `FlexContainer` в схеме представлен ниже.

#### Пример использования компонента `FlexContainer` в схеме

```

"name": "FlexContainer",
"values": {
  "layoutConfig": {...},
  "type": "crt.FlexContainer",
  "direction": "column",
  "justifyContent": "start",
  "alignItems": "stretch",
  "wrap": "nowrap",
  "gap": "small",
  "items": []
}
...
{
  "operation": "insert",
  "name": "Button",
  "values": {
    "type": "crt.Button",
    "caption": "#ResourceString(Button_caption)#",
    "color": "primary",
  },
  "parentName": "FlexContainer",
  "propertyName": "Items",
  "index": 0
}

```

# Компонент GridContainer JS

## Основы

`GridContainer` — компонент разметки, который позволяет настроить расположение нескольких элементов последовательно на сетке. Элементы могут изменять размер в зависимости от контента. Построен на базе `CSS Grid Layout`.

Подробнее о `Grid Layout` читайте в статье [Основные понятия Grid Layout](#).

## Свойства

@Input rows, columns

Определяет ширину и высоту колонки сетки макета.

### Возможные значения

Константа	Размер колонки задается целочисленным значением.
-----------	--

Пример настройки свойств `columns` и `rows` приведен ниже.

#### Пример настройки свойств `columns` и `rows`

```
{
  ...
  "columns": [
    "298px",
    "minmax(64px, 1fr)"
  ],
  "rows": "minmax(max-content, 32px)",
  ...
}
```

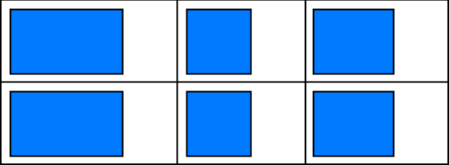
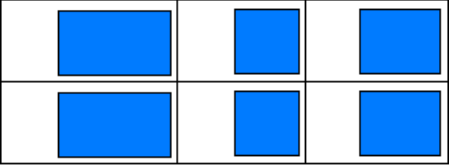
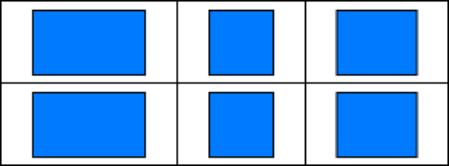
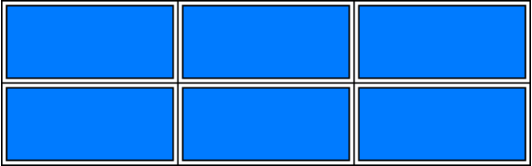
**Свойства** CSS Grid Layout, которые не поддерживает компонент `GridContainer`:

- `grid-line-name` — именованные линии.
- `fit-content()` — ограничивает ширину колонки. Представляет собой формулу `min(max-content, max(auto, argument))`, которая вычисляется, как свойство `auto` (т. е. `minmax(auto, max-content)`). Размер поля ограничен значением параметра `argument`, если он больше минимального значения, заданного в свойстве `auto`. В Creatio при любом заданном значении свойства `fit-content()` создается одна колонка шириной 32px.
- `repeat()` — повторяющийся фрагмент перечня строк, который в компактной форме позволяет записать строки с повторяющимся шаблоном.

#### @Input `justifyItems`

Выравнивание по горизонтали. Указывает как браузер распределяет пространство между и вокруг элементов контента вдоль строчной оси `grid`-контейнера.

#### Возможные значения

start	Выравнивание элементов внутри блока по левому краю.	
end	Выравнивание элементов внутри блока по правому краю.	
center	Выравнивание элементов внутри блока по центру.	
stretch	Выравнивание элементов внутри блока по ширине.	

Пример настройки свойства `justify-items` приведен ниже.

**Пример настройки свойства `justify-items`**

```
.container {
  justify-items: start | end | center | stretch;
}
```

@Input alignItems

Выравнивание по вертикали. Выравнивание элементов внутри блока вдоль соответствующей оси.

Возможные значения

start	Выравнивание элементов внутри блока по верхнему краю.	
end	Выравнивание элементов внутри блока по нижнему краю.	
center	Выравнивание элементов внутри блока по центру.	
stretch	Выравнивание элементов внутри блока по ширине.	

Пример настройки свойства `align-items` приведен ниже.

#### Пример настройки свойства `align-items`

```
.container {
  align-items: start | end | center | stretch;
}
```

#### @Input gap

Задаёт отступы между элементами `grid`-контейнера в столбцах и строках. Является сокращением для свойств `row-gap` (отступ между элементами строки) и `column-gap` (отступ между элементами столбца).

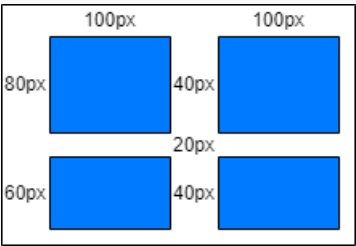
Результат настройки свойств `row-gap` и `column-gap` приведен ниже.

#### Результат настройки свойств `row-gap` и `column-gap`

```
.container {
  grid-template-columns: 100px auto;
  grid-template-rows: 80px 60px;
```

```
column-gap: 40px;  
row-gap: 20px;  
}
```

Результат отображения представлен на рисунке ниже.



@Input padding

Задаёт внутренние отступы для контейнера. Может настраиваться как для каждой стороны отдельно, так и для всех сторон одновременно. Может принимать число, строку и возможные значения, которые приведены ниже.

Возможные значения

none	Внутренний отступ отсутствует.
small	Маленький размер отступа.
medium	Средний размер отступа.
large	Большой размер отступа.

Пример настройки свойства `padding` приведен ниже.

Пример настройки свойства `padding`

```
"padding": {  
  "top": "6px",  
  "right": 6,  
  "bottom": "small",  
  "left": "large"  
}
```

@Input border-radius



Задаёт радиус скругления углов для контейнера. Может настраиваться как для каждой стороны отдельно, так и для всех сторон одновременно. Может принимать число, строку и возможные значения, которые приведены ниже.

Возможные значения

none	Скругление углов отсутствует.
small	Маленький радиус скругления углов.
medium	Средний радиус скругления углов.
large	Большой радиус скругления углов.

Пример настройки свойства `border-radius` приведен ниже.

Пример настройки свойства `border-radius`

```
"borderRadius": "medium"
```

@Input color

Задаёт цвет контейнера. В качестве значения принимает код цвета.

Пример настройки свойства `color` приведен ниже.

Пример настройки свойства `color`

```
"color": "#FDAB06"
```

Пример использования

Пример использования компонента `GridContainer` в схеме представлен ниже.

Пример использования компонента `GridContainer` в схеме

```
"name": "GridContainer",
"values": {
  "layoutConfig": {...},
  "type": "crt.GridContainer",
  "columns": [
    "minmax(32px, 1fr)",
    "minmax(32px, 1fr)",
    "minmax(32px, 1fr)",
```

```

        "minmax(32px, 1fr)"
    ],
    "rows": "minmax(max-content, 32px)",
    "gap": {
        "columnGap": "large"
    },
    "items": []
}
...
{
    "operation": "insert",
    "name": "Button",
    "values": {
        "layoutConfig": {
            "column": 1,
            "row": 1,
            "colSpan": 3,
            "rowSpan": 1
        },
        "type": "crt.Button",
        "caption": "#ResourceString(Button_caption)#",
        "color": "default",
    },
    "parentName": "GridContainer",
    "propertyName": "Items",
    "index": 0
}

```

# Компонент DateTimePicker

## Основы

`DateTimePicker` — компонент, который позволяет настроить поле типа [ *Дата/Время* ] ([ *Date/Time* ]) на странице приложения. Атрибут `aria-label` генерируется автоматически и указывает на родительский элемент `<input>`.

## Свойства

`id` `string`

Уникальный идентификатор. Служебное поле.

`control` `FormControl`

Элемент управления.

---

`label string`

Метка.

---

`ariaLabel string`

Свойство, которое связано с атрибутом `aria-label` элемента.

---

`value string`

Значение поля типа [ *Дата/Время* ] ([ *Date/Time* ]).

---

`disabled boolean`

Признак, который управляет доступностью изменения поля типа [ *Дата/Время* ] ([ *Date/Time* ]).

---

`readonly boolean`

Признак, который устанавливает доступность поля типа [ *Дата/Время* ] ([ *Date/Time* ]) только для чтения. По умолчанию — `false`.

---

`rowModeSizePx number`

Ширина поля типа [ *Дата/Время* ] ([ *Date/Time* ]). По умолчанию — `320`.

---

`multiYearSelector boolean`

Признак, который управляет возможностью просмотра списка годов при нажатии на год. По умолчанию — `false`.

---

`twelvehour boolean`

Если свойство установлено в `true`, то используется 12-часовой формат времени. По умолчанию — `false`.

---

`startView string`

Свойство, которое устанавливает период для просмотра данных. По умолчанию — `month`.

---

**Возможные значения**

month	Месяц.
year	Год.
hour	Час.
multi-year	Несколько лет.

mode `string`

Режим отображения поля типа [ *Дата/Время* ] ([ *Date/Time* ]). По умолчанию — `auto`.

Возможные значения

auto	Автоматический режим.
portrait	Вертикальный режим.
landscape	Горизонтальный режим.

timeInterval `number`

Длительность задержки поиска в миллисекундах. По умолчанию — `1`.

preventSameDateTimeSelection `boolean`

Запретить пользователю выбирать текущую выбранную дату и время. По умолчанию — `false`.

## Горячие клавиши

LEFT\_ARROW

Установить фокус на предыдущий день.

RIGHT\_ARROW

Установить фокус на следующий день.

UP\_ARROW

Установить фокус на предыдущую неделю.

DOWN\_ARROW

Установить фокус на следующую неделю.

---

HOME

Установить фокус на первый день месяца.

---

END

Установить фокус на последний день месяца.

---

PAGE\_UP

Установить фокус на предыдущий месяц.

---

PAGE\_DOWN

Установить фокус на следующий месяц.

---

ALT + PAGE\_UP

Установить фокус на предыдущий год.

---

ALT + PAGE\_DOWN

Установить фокус на следующий год.

---

ENTER

Выбрать элемент, на котором установлен фокус.

## Компонент Checkbox

### Основы

`Checkbox` — компонент, который позволяет настроить чекбокс на странице приложения.

## Свойства

---

`id string`

Уникальный идентификатор. Служебное поле.

---

`value` `boolean`

Признак, который устанавливает чекбокс. По умолчанию — `false`.

---

`disabled` `boolean`

Признак, который управляет доступностью изменения чекбокса. По умолчанию — `false`.

---

`inversed` `boolean`

Признак, который инвертирует стиль чекбокса. По умолчанию — `false`.

---

`label` `string`

Заголовок чекбокса.

---

`ariaLabel` `string`

Свойство чекбокса, которое связано с атрибутом `aria-label`. Описание атрибута `aria-label` содержится в официальной [документации Mozilla](#).

---

`labelPosition` `string`

Устанавливает место отображения заголовка чекбокса. По умолчанию — `auto`.

---

#### Возможные значения

<code>auto</code>	Автоматический перенос заголовка чекбокса при изменении доступного места для элемента ввода на экране. По умолчанию — слева, при сжатии — сверху.
<code>left</code>	Заголовок отображается слева от чекбокса.
<code>right</code>	Заголовок отображается справа от чекбокса.
<code>hidden</code>	Заголовок чекбокса скрыт.
<code>above</code>	Заголовок отображается сверху чекбокса.

## События

---

`valueChange` `boolean`

Событие, которое генерируется при изменении значения чекбокса.

# Компонент Input JS

## Основы

`Input` — компонент, который позволяет настроить поле ввода на странице приложения. Компонент `Input` использует собственные элементы `<input>` и `<label>` для доступности поля по умолчанию. Атрибут `aria-label` генерируется автоматически и указывает на родительский элемент `<input>`.

## Свойства

`id` `string`

Уникальный идентификатор. Служебное поле.

`control` `FormControl`

Элемент управления.

`label` `string`

Заголовок поля ввода.

`inputType` `string`

Тип поля ввода. По умолчанию — `text`.

### Возможные значения

<code>text</code>	Тип поля ввода — текст.
<code>password</code>	Тип поля ввода — текст, который отображается символами <code>*</code> . Есть кнопка просмотра введенного текста.

`placeholder` `string`

Текст подсказки, который отображаемый в поле до начала ввода текста в него.

`appearance` `enum`

Внешний вид поля ввода. По умолчанию — `legacy`.

## Возможные значения

<code>legacy</code>	Отображается только нижняя граница поля ввода.
<code>outline</code>	Отображаются все границы поля ввода.

---

`value` `string`

Значение поля ввода.

---

`disabled` `boolean`

Признак, который управляет доступностью изменения значения поля ввода.

---

`readonly` `boolean`

Признак, который устанавливает доступность поля ввода только для чтения. По умолчанию — `false`.

---

`autocomplete` `string`

Устанавливает разрешение для браузера на автоматическое заполнение поля ввода. По умолчанию — `off`.

---

`rowModeSizePx` `number`

Ширина поля ввода — число, при котором компоненты `Input` и `Checkbox` перестраиваются с вертикального в горизонтальный вид при установленном режиме `auto`. По умолчанию — `320`.

---

`labelPosition` `string`

Устанавливает место отображения заголовка поля ввода. По умолчанию — `auto`.

## Возможные значения



auto	Автоматический перенос заголовка поля ввода при изменении доступного места для элемента ввода на экране. По умолчанию — слева, при сжатии — сверху.
left	Заголовок отображается слева от поля ввода.
right	Заголовок отображается справа от поля ввода.
hidden	Заголовок поля ввода скрыт.
above	Заголовок отображается сверху поля ввода.

## Компонент NumberInput

### Основы

`NumberInput` — компонент, который позволяет настроить числовое поле на странице приложения. Компонент `NumberInput` использует собственные элементы `<input>` и `<label>` для доступности поля по умолчанию. Атрибут `aria-label` генерируется автоматически и указывает на родительский элемент `<input>`.

## Свойства

`id` `string`

Уникальный идентификатор. Служебное поле.

`control` `FormControl`

Элемент управления.

`label` `string`

Заголовок числового поля.

`value` `string`

Значение числового поля.

`disabled` `boolean`

Признак, который управляет доступностью изменения числового поля.

`min number`

Минимальное допустимое значение числового поля.

---

`max number`

Максимальное допустимое значение числового поля.

---

`decimalPrecision number`

Количество знаков после запятой.

---

`readonly boolean`

Признак, который устанавливает доступность изменения значения числового поля. По умолчанию — `false`.

---

`autocomplete string`

Устанавливает разрешения для браузера на автоматическое заполнение числового поля. По умолчанию — `off`.

---

`rowModeSizePx number`

Ширина числового поля. По умолчанию — `320`.

---

`labelPosition string`

Устанавливает место отображения заголовка числового поля. По умолчанию — `auto`.

#### Возможные значения

<code>auto</code>	Автоматический перенос заголовка числового поля при изменении доступного места для элемента ввода на экране. По умолчанию — слева, при сжатии — сверху.
<code>left</code>	Заголовок отображается слева от числового поля.
<code>right</code>	Заголовок отображается справа от числового поля.
<code>hidden</code>	Заголовок числового поля скрыт.
<code>above</code>	Заголовок отображается сверху числового поля.