

# Клиентская схема

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>Клиентская схема</b>	<b>4</b>
Разработка клиентской схемы	4
Свойства клиентской схемы	5
<b>Переопределить метод миксина</b>	<b>22</b>
1. Создать миксин	22
2. Подключить миксин	24
3. Переопределить метод миксина	25
<b>Примеры объявления методов</b>	<b>26</b>
<b>Пример использования массива модификаций</b>	<b>27</b>
<b>Пример использования механизма alias при многократном замещении схемы</b>	<b>28</b>
<b>Свойство attributes</b>	<b>31</b>
Основные свойства	31
Дополнительные свойства	34
<b>Свойство messages</b>	<b>36</b>
Свойства	37
<b>Свойства rules и businessRules</b>	<b>37</b>
Основные свойства	37
Дополнительные свойства	41
<b>Свойство diff</b>	<b>43</b>
Свойства	43

# Клиентская схема



**Клиентская схема модели представления** — это схема визуального модуля, с помощью которой реализуется front-end часть приложения. Клиентская схема модели представления является конфигурационным объектом для генерации представления и модели представления генераторами Creatio `ViewGenerator` и `ViewModelGenerator`. Типы модулей и их особенности описаны в статье [Виды клиентских модулей](#).

## Разработка клиентской схемы

**Способы разработки** клиентских схем модели представления:

- Раздел [ *Конфигурация* ] ([ *Configuration* ]). Разработка с помощью раздела [ *Конфигурация* ] описана в статье [Схема модели представления](#).
- Мастер разделов. Разработка раздела с помощью мастера разделов описана в статье [Добавить новый раздел](#).
- Мастер деталей. Разработка деталей с помощью мастера деталей описана в статье [Создать новую деталь](#).

**Структурные элементы** клиентской схемы:

- Автоматически сгенерированный код — содержит описание схемы, ее зависимостей, локализованных ресурсов и сообщений.
- Стили, используемые для визуализации (присутствуют не во всех типах клиентских схем).
- Исходный код схемы — синтаксически правильный код на языке JavaScript, который является определением модуля.

**Маркерные комментарии** необходимо использовать в исходном коде схемы для свойств `diff`, `modules`, `details` и `businessRules`.

**Назначение** маркерных комментариев — однозначное определение свойств клиентской схемы. При открытии мастера выполняется валидация наличия маркерных комментариев, которые представлены в таблице ниже.

Тип схемы	Необходимые маркерные комментарии
<b>Схема представления модели страницы записи</b> EditViewModelSchema	<pre> details: /**SCHEMA_DETAILS*/{}/**SCHEMA_DETAILS*/, modules: /**SCHEMA_MODULES*/{}/**SCHEMA_MODULES*/, diff: /**SCHEMA_DIFF*/[]/**SCHEMA_DIFF*/, businessRules: /**SCHEMA_BUSINESS_RULES*/{}/**SCHEMA_BUSI </pre>
<b>Схема представления модели раздела</b> ModuleViewModelSchema	<pre> modules: /**SCHEMA_MODULES*/{}/**SCHEMA_MODULES*/, diff: /**SCHEMA_DIFF*/[]/**SCHEMA_DIFF*/ </pre>
<b>Схема представления модели детали с полями</b> EditControlsDetailViewModelSchema	
<b>Схема представления модели детали</b> DetailViewModelSchema	
<b>Схема представления модели детали с реестром</b> GridDetailViewModelSchema	

## Свойства клиентской схемы

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

### Структура исходного кода клиентской схемы

```

define("ExampleSchema", [], function() {
  return {
    entitySchemaName: "ExampleEntity",
    mixins: {},
    attributes: {},
    messages: {},
    methods: {},
    rules: {},
    businessRules: /**SCHEMA_BUSINESS_RULES*/{}/**SCHEMA_BUSINESS_RULES*/,
    modules: /**SCHEMA_MODULES*/{}/**SCHEMA_MODULES*/,
    diff: /**SCHEMA_DIFF*/[]/**SCHEMA_DIFF*/
  };
});

```

После загрузки модуля выполняется вызов анонимной функции-фабрики, которая возвращает конфигурационный объект схемы. **Свойства** конфигурационного объекта схемы:

- `entitySchemaName` — имя схемы объекта, который используется текущей клиентской схемой.
- `mixins` — конфигурационный объект, который содержит объявление миксинов.
- `attributes` — конфигурационный объект, который содержит атрибуты схемы.
- `messages` — конфигурационный объект, который содержит сообщения схемы.
- `methods` — конфигурационный объект, который содержит методы схемы.
- `rules` — конфигурационный объект, который содержит бизнес-правила схемы.
- `businessRules` — конфигурационный объект, который содержит бизнес-правила схемы, созданные или измененные мастером разделов или мастером деталей. Маркерные комментарии `/**SCHEMA_BUSINESS_RULES*/` обязательны к использованию для работы мастеров.
- `modules` — конфигурационный объект, который содержит модули схемы. Маркерные комментарии `/**SCHEMA_MODULES*/` обязательны к использованию для работы мастеров.

**На заметку.** Для загрузки детали на страницу используется свойство `details`. Поскольку деталь является модулем, то правильным будет использование свойства `modules`.

- `diff` — массив конфигурационных объектов, который содержит описание представления схемы. Маркерные комментарии `/**SCHEMA_DIFF*/` обязательны к использованию для работы мастеров.
- `properties` — конфигурационный объект, который содержит свойства схемы модели представления.
- `$-свойства` — автоматически сгенерированные свойства для атрибутов схемы модели представления.

## Имя схемы (`entitySchemaName`)

Для реализации имени схемы объекта необходимо использовать обязательное свойство `entitySchemaName`. Достаточно указать его в одной из схем иерархии наследования.

### Пример объявления свойства `entitySchemaName`

```
define("ClientSchemaName", [], function () {
    return {
        /* Схема объекта (модель). */
        entitySchemaName: "EntityName",
        /* ... */
    };
});
```

## Миксины (`mixins`)

**Миксин** — это класс-примесь, предназначенный для расширения функциональности других классов. В JavaScript нет множественного наследования, а наличие миксинов позволяет расширить функциональность схемы без дублирования логики, часто используемой в методах схемы. В разных клиентских схемах приложения Creatio может использоваться один и тот же набор действий. Чтобы не дублировать код в каждой схеме, необходимо создать миксин. **Особенность** миксинов в сравнении с другими модулями, подключаемыми в список зависимостей, — способ вызова методов из схемы модуля (к методам миксина можно обращаться напрямую, как к методам схемы). Для реализации миксинов необходимо использовать свойство `mixins`.

**Алгоритм** работы с миксинами:

1. Создайте миксин.
2. Присвойте миксину имя.
3. Подключите соответствующее пространство имен.
4. Реализуйте функциональность миксина.
5. Используйте миксин в клиентской схеме.

## Создать миксин

Создание миксина аналогично созданию [схемы объекта](#).

## Присвоить миксину имя

При именовании миксинов в названии схемы необходимо использовать суффикс "-able" (например, `Serializable` — миксин, который добавляет компонентам способность сериализоваться). Если нет возможности сформулировать имя миксина в описанной выше форме, необходимо в название схемы добавить окончание "Mixin".

**Важно.** Использование слов `Utilities`, `Extension`, `Tools` и т. д. является недопустимым, поскольку не позволяет четко формализовать функциональность, которая скрывается за названием миксина.

## Подключить пространство имен

Для миксина необходимо подключить соответствующее пространство имен (для конфигурации — `Terrasoft.configuration.mixins`, для ядра — `Terrasoft.core.mixins`).

## Реализовать функциональность миксина

Миксины не должны зависеть от внутренней реализации схемы, к которой они будут применены. Это должен быть самодостаточный механизм, который принимает набор параметров, выполняет с ними действие и, при необходимости, возвращает результат. Миксины оформляются в виде модулей, которые необходимо подключить в список зависимостей схемы при ее объявлении функцией `define()`.

Структура миксина представлена ниже.

### Структура миксина

```
define("ИмяМиксина", [], function() {
    Ext.define("Terrasoft.configuration.mixins.ИмяМиксина", {
        alternateClassName: "Terrasoft.ИмяМиксина",
        /* Функциональность миксина. */
    });
    return Ext.create(Terrasoft.ИмяМиксина);
})
```

## Использовать миксин

Миксин реализует функциональность, необходимую в клиентской схеме. Чтобы получить набор действий миксина, необходимо указать его в свойстве `mixins` клиентской схемы.

### Использование миксина в клиентской схеме

```
/* ИмяМиксина — модуль, в котором реализован класс миксина. */
define("ИмяКлиентскойСхемы", ["ИмяМодуля"], function () {
    return {
        /* ИмяСхемы — название схемы сущности. */
        entitySchemaName: "ИмяСхемы",
        mixins: {
            /* Подключение миксина. */
            ИмяМиксина: "Terrasoft.ПространствоИмен.ИмяМиксина"
        },
        attributes: {},
        messages: {},
        methods: {},
        rules: {},
        modules: /**SCHEMA_MODULES*/{}/**SCHEMA_MODULES*/,
        diff: /**SCHEMA_DIFF*/[/]**SCHEMA_DIFF*/
    };
});
```

После подключения можно пользоваться методами, атрибутами и полями миксина в клиентской схеме так, как будто они являются частью данной клиентской схемы. При этом вызовы методов получаются более краткими, чем при использовании отдельной схемы. Например, `getDefaultImageResource` — функция миксина. Для вызова функции миксина `getDefaultImageResource` в пользовательской схеме, к которой подключен миксин, необходимо записать `this.getDefaultImageResource()`.

**На заметку.** При переопределении функции из миксина необходимо в клиентской схеме создать функцию с аналогичным именем. В результате при вызове будет использоваться функция схемы, а не миксина.



## Атрибуты (attributes)

Для реализации атрибутов необходимо использовать свойство `attributes`.

## Сообщения (messages)

**Назначение** сообщений — организация [обмена данными](#) между модулями. Для реализации сообщений необходимо использовать свойство `messages`. Используя перечисление `Terrasoft.MessageMode`, можно задать **режим** сообщения.

Типы режимов сообщения

Режим сообщения	Описание	Подключение
<b>Адресное</b>	Адресные сообщения принимаются только последним подписанным подписчиком.	Для установки сообщения в адресный режим необходимо свойству <code>mode</code> присвоить значение <code>this.Terrasoft.MessageMode.PTP</code> .
<b>Широковещательное</b>	Широковещательные сообщения принимаются всеми подписчиками.	Для установки сообщения в широковещательный режим необходимо свойству <code>mode</code> присвоить значение <code>this.Terrasoft.MessageMode.BROADCAST</code> .

Кроме режимов, также можно задать **направление** сообщения.

## Типы направлений сообщения

Направление сообщения	Описание	Подключение
<b>Публикация</b>	Сообщение может быть только опубликовано, т. е. является <b>исходящим</b> .	Для установки направления публикации сообщения необходимо свойству <code>direction</code> присвоить значение <code>this.Terrasoft.MessageDirectionType.PUBLISH</code> .
<b>Подписка</b>	На сообщение можно только подписаться, т. е. является <b>входящим</b> .	Для установки направления подписки на сообщение необходимо свойству <code>direction</code> присвоить значение <code>this.Terrasoft.MessageDirectionType.SUBSCRIBE</code> .
<b>Двунаправленное</b>	Позволяет публиковать и подписываться на одно и то же сообщение в разных экземплярах одного и того же класса или в рамках одной и той же иерархии наследования схем. В иерархии наследования схем не может быть объявлено одно и то же сообщение с разным направлением. Для таких случаев необходимо использовать двунаправленные сообщения, особенности которых описаны в статье <a href="#">Sandbox</a> .	Соответствует значению перечисления <code>Terrasoft.MessageDirectionType.BIDIRECTIONAL</code> .

## Публикация сообщения

В схеме, в которой необходимо осуществить публикацию сообщения, должно быть объявлено сообщение с направлением "Публикация".

**Пример объявления сообщения с направлением "Публикация"**

```
messages: {
  /* Имя сообщения. */
  "GetColumnsValues": {
```

```

    /* Режим сообщения – адресное. */
    mode: this.Terrasoft.MessageMode.PTP,
    /* Направление сообщения – публикация. */
    direction: this.Terrasoft.MessageDirectionType.PUBLISH
  }
}

```

Публикация осуществляется вызовом метода `publish` у экземпляра класса `sandbox`.

### Пример публикации сообщения

```

/* Метод получения результата публикации сообщения GetColumnsValues. */
getColumnsValues: function(argument) {
  /* Публикация сообщения.
  GetColumnsValues – имя сообщения.
  argument – аргумент, передаваемый в функцию-обработчик подписчика. Объект, содержащий параметры
  key – массив тегов для фильтрации сообщений. */
  return this.sandbox.publish("GetColumnsValues", argument, ["key"]);
}

```

**Важно.** Публикация сообщения может возвращать результат работы функции-обработчика только при установленном **адресном** режиме.

## Подписка на сообщение

В схеме-подписке должно быть объявлено сообщение с направлением "Подписка".

### Пример объявления сообщения с направлением "Подписка"

```

messages: {
  /* Имя сообщения. */
  "GetColumnsValues": {
    /* Режим сообщения – адресное. */
    mode: this.Terrasoft.MessageMode.PTP,
    /* Направление сообщения – подписка. */
    direction: this.Terrasoft.MessageDirectionType.SUBSCRIBE
  }
}

```

Подписка осуществляется вызовом метода `subscribe` у экземпляра класса `sandbox`.

### Пример подписки на сообщение

```

/* GetColumnsValues — имя сообщения.
messageHandler — функция-обработчик сообщения.
context — контекст выполнения функции-обработчика.
key — массив тегов для фильтрации сообщений. */
this.sandbox.subscribe("GetColumnsValues", messageHandler, context, ["key"]);

```

При **адресном** режиме метод `messageHandler` должен возвращать объект, который обрабатывается как результат публикации сообщения. При **широковещательном** режиме метод `messageHandler` ничего не возвращает.

Метод `messageHandler` (адресный режим)

```

methods: {
  messageHandler: function(args) {
    /* Возвращение объекта, который обрабатывается как результат публикации сообщения. */
    return { };
  }
}

```

Метод `messageHandler` (широковещательный режим)

```

methods: {
  messageHandler: function(args) {
  }
}

```

## Методы (methods)

Для реализации методов необходимо использовать свойство `methods`, которое содержит коллекцию методов, формирующих бизнес-логику схемы и влияющих на модель представления. По умолчанию контекст методов является контекстом модели представления.

**Назначение** свойства `methods`:

1. Создавать новые методы.
2. Расширять базовые методы родительских схем.

## Бизнес-правила (rules и businessRules)

**Бизнес-правила** — это механизмы приложения, которые пользовательскими средствами позволяют настраивать поведение полей на странице или детали. Для реализации бизнес-правил необходимо использовать свойства `rules` и `businessRules`. Свойство `businessRules` используется для бизнес-правил,

которые созданы или изменены мастером разделов или мастером деталей.

**Назначение** бизнес-правил:

- Скрытие или отображение полей.
- Блокировка или доступность редактирования полей.
- Обязательность или необязательность заполнения полей.
- Фильтрация справочных полей в зависимости от значений в других полях.

Функциональность бизнес-правил реализована в клиентском модуле `BusinessRuleModule`. Для использования функциональности бизнес-правил необходимо в список зависимостей схемы добавить модуль `BusinessRuleModule`.

#### Пример добавления модуля `BusinessRuleModule` в список зависимостей

```
define("CustomPageModule", ["BusinessRuleModule"],
  function(BusinessRuleModule) {
    return {
      /* Реализация клиентского модуля. */
    };
  });
```

Типы бизнес-правил определены в перечислении `RuleType` модуля `BusinessRuleModule`.

## Особенности бизнес-правил

**Особенности** объявления бизнес-правил:

- Все бизнес-правила описываются в свойстве `rules` схемы.
- Бизнес-правила применяются к колонкам модели представления, а не к элементам управления.
- Бизнес-правило имеет название.
- Параметры бизнес-правила задаются в конфигурационном объекте.

**Особенности** бизнес-правил, определенных в свойстве `businessRules`:

- Генерируются мастерами разделов или деталей.
- При создании нового бизнес-правила мастер генерирует его имя и добавляет его в клиентскую схему модели представления страницы записи.
- При описании сгенерированного бизнес-правила не используются перечисления модуля бизнес-правил `BusinessRuleModule`.
- Маркерные комментарии `/**SCHEMA_BUSINESS_RULES*/` обязательны к использованию для работы мастеров.
- При выполнении имеют более высокий приоритет.
- При отключении бизнес-правила свойству `enabled` конфигурационного объекта присваивается значение `false`.

- При удалении бизнес-правила конфигурационный объект остается в клиентской схеме модели представления страницы записи, но свойству `removed` присваивается значение `true`.

**Важно.** Не рекомендуется редактировать свойство `businessRules` клиентской схемы.

## Редактирование существующего бизнес-правила

После редактирования мастером созданного вручную бизнес-правила, остается неизменным конфигурационный объект бизнес-правила в свойстве `rules` модели представления страницы записи. При этом будет создана новая версия конфигурационного объекта бизнес-правила с тем же именем в свойстве `businessRules`.

При обработке бизнес-правила во время выполнения приложения приоритет отдается бизнес-правилу, определенному в свойстве `businessRules`. Поэтому последующие изменения этого бизнес-правила в свойстве `rules` никак не повлияют на систему.

**На заметку.** При удалении или отключении бизнес-правила более высокий приоритет имеют изменения, выполненные в конфигурационном объекте свойства `businessRules`.

## Модули (modules)

Для реализации модулей необходимо использовать свойство `modules`, конфигурационный объект которого отвечает за объявление и конфигурирование модулей и деталей, загружаемых на страницу. Маркерные комментарии `/**SCHEMA_MODULES*/` обязательны к использованию для работы мастеров.

**На заметку.** Для загрузки детали на страницу используется свойство `details`. Поскольку деталь по сути является модулем, то правильным будет использование свойства `modules`.

### Пример использования свойства `modules`

```
modules: /**SCHEMA_MODULES*/{
  /* Загрузка модуля.
  Заголовок модуля. Должен быть таким же, как свойство name в массиве diff. */
  "TestModule": {
    /* Опционально. Идентификатор загружаемого модуля. Если не указан, будет сгенерирован с
    "moduleId": "myModuleId",.
    /* Если параметр не указан, будет использован BaseSchemaModuleV2 для загрузки. */
    "moduleName": "MyTestModule",
    /* Конфигурационный объект. При загрузке модуля передается как instanceConfig. В нем хра
    "config": {
      "isSchemaConfigInitialized": true,
      "schemaName": "MyTestSchema",
      "useHistoryState": false,
```

```

/* Дополнительные параметры модуля. */
"parameters": {
  /* Параметры, передаваемые в схему при ее инициализации. */
  "viewModelConfig": {
    masterColumnName: "PrimaryContact"
  }
},

/* Загрузка детали.
Имя детали. */
"Project": {
  /* Название схемы детали. */
  "schemaName": "ProjectDetailV2",
  "filter": {
    /* Колонка схемы объекта раздела. */
    "masterColumn": "Id",
    /* Колонка схемы объекта детали. */
    "detailColumn": "Opportunity"
  }
}
}
}

```

## Массив модификаций (diff)

Для реализации массива модификаций необходимо использовать свойство `diff`, которое содержит массив конфигурационных объектов. **Назначение** массива модификаций — построение представления модуля в интерфейсе системы. Каждый элемент массива представляет собой метаданные, на основании которых генерируются различные элементы управления интерфейса. Маркерные комментарии `/**SCHEMA_DIFF*/` обязательны к использованию для работы мастеров.

## Механизм псевдонимов alias

При разработке новых версий периодически возникает необходимость переместить элементы страницы в новые зоны. В ситуациях, когда пользователи проводили кастомизацию страницы записи, подобные изменения могут привести к непредсказуемым последствиям. **Механизм псевдонимов** `alias` обеспечивает частичную обратную совместимость при изменении пользовательского интерфейса в новых версиях продукта путем взаимодействия с строителем `diff` — классом `json-applier`, осуществляющим слияние параметров базовой схемы и клиентских расширяющих схем.

Свойство `alias` содержит информацию о предыдущем названии элемента. Эта информация учитывается при построении массива модификаций `diff`, сообщая о необходимости учитывать элементы не только с новым именем, но и с именем, указанным в `alias`. Фактически, `alias` представляет собой конфигурационный объект, который связывает новый и старый элементы. При построении массива модификаций `diff` благодаря конфигурационному объекту `alias` можно исключить применение некоторых свойств и операций по отношению к элементу, в котором он объявлен. Объект `alias` может

быть добавлен в любой элемент массива модификаций `diff`.

## Связь между представлением и моделью

**Назначение** свойства `bindTo` — указание связи между атрибутом модели представления и свойством объекта представления.

Объявляется в свойстве `values` конфигурационных объектов массива `diff`.

Ниже представлен пример использования свойства `bindTo`.

### Пример использования свойства `bindTo`

```
diff: [
  {
    "operation": "insert",
    "parentName": "CombinedModeActionButtonCardLeftContainer",
    "propertyName": "items",
    "name": "MainContactButton",
    /* Свойства, передаваемые в конструктор компонента. */
    "values": {
      /* Тип добавляемого элемента — кнопка. */
      "itemType": Terrasoft.ViewItemType.BUTTON,
      /* Привязка заголовка кнопки к локализуемой строке схемы. */
      "caption": {bindTo: "Resources.Strings.OpenPrimaryContactButtonCaption"},
      /* Привязка метода-обработчика нажатия кнопки. */
      "click": {bindTo: "onOpenPrimaryContactClick"},
      /* Стиль отображения кнопки. */
      "style": Terrasoft.controls.ButtonEnums.style.GREEN,
      /* Привязка свойства доступности кнопки. */
      "enabled": {bindTo: "ButtonEnabled"}
    }
  }
]
```

**Вкладки** — объекты, которые в свойстве `propertyName` содержат значение `tabs`.

Для **заголовков вкладок** реализован альтернативный способ использования свойства `bindTo`.

### Альтернативный способ использования свойства `bindTo`

```
...
{
  "operation": "insert",
  "name": "GeneralInfoTab",
  "parentName": "Tabs",
  /* Указывает на то, что данный объект является вкладкой. */
  "propertyName": "tabs",
```



```

    "index": 0,
    "values": {
        /* $ заменяет использование bindTo: {...}. */
        "caption": "$Resources.Strings.GeneralInfoTabCaption",
        "items": []
    }
},
...

```

## Правила объявления свойства diff

- **Корректное использование конвертеров.**

**Конвертер** — это функция, которая выполняется в окружении `viewModel` и, получая значения свойства `viewModel`, должна вернуть результат соответствующего типа. Для корректной работы мастеров значение свойства `diff` должно быть представлено в формате JSON. Поэтому значением конвертера должно быть имя метода модели представления, а не `inline`-функция.

Пример правильного использования конвертера

```

methods: {
    someFunction: function(val) {
        /* ... */
    }
},

diff: /**SCHEMA_DIFF*/[
    {
        /* ... */
        "bindConfig": {
            "converter": "someFunction"
        }
        /* ...*/
    }
]/**SCHEMA_DIFF*/

```

Пример неправильного использования конвертера

```

diff: /**SCHEMA_DIFF*/[
    {
        /* ... */
        "bindConfig": {
            "converter": function(val) {
                /* ... */
            }
        }
    }
]

```

```

    }
  }
]/**SCHEMA_DIFF*/

```

Пример правильного использования генератора

```

methods: {
  someFunction: function(val) {
    /* ... */
  }
},

diff: /**SCHEMA_DIFF*/[
  {
    /* ... */
    "values": {
      "generator": "someFunction"
    }
    /* ... */
  }
]/**SCHEMA_DIFF*/

```

Пример неправильного использования генератора

```

diff: /**SCHEMA_DIFF*/[
  {
    /* ... */
    "values": {
      "generator": function(val) {
        /* ... */
      }
    }
  }
]/**SCHEMA_DIFF*/

```

- **Родительский элемент.**

**Родительский элемент (контейнер)** — это DOM-элемент, в который модуль отрисует свое представление. Для корректной работы мастерам необходимо, чтобы родительский контейнер содержал только один дочерний элемент.

Пример правильного размещения представления в родительском элементе

```
<div id="OpportunityPageV2Container" class="schema-wrap one-el" data-item-marker="Opportunity"
  <div id="CardContentWrapper" class="card-content-container page-with-left-el" data-item-m
</div>
```

Пример неправильного размещения представления в родительском элементе

```
<div id="OpportunityPageV2Container" class="schema-wrap one-el" data-item-marker="Opportunity"
  <div id="CardContentWrapper" class="card-content-container page-with-left-el" data-item-m
  <div id="DuplicateContainer" class="DuplicateContainer"></div>
</div>
```

При добавлении, изменении, перемещении элемента (операции `insert`, `merge`, `move`) в свойстве `diff` необходимо указать свойство `parentName` — имя родительского элемента.

Пример правильного определения элемента представления в свойстве `diff`

```
{
  "operation": "insert",
  "name": "SomeName",
  "propertyName": "items",
  "parentName": "SomeContainer",
  "values": {}
}
```

Пример неправильного определения элемента представления в свойстве `diff`

```
{
  "operation": "insert",
  "name": "SomeName",
  "propertyName": "items",
  "values": {}
}
```

Если отсутствует свойство `parentName`, то при открытии мастера отобразится ошибка о невозможности настройки страницы мастером.

Значение свойства `parentName` должно соответствовать имени родительского элемента в соответствующей базовой схеме страницы. Так, например, для страниц записи это `CardContentContainer`.

Если в качестве родительского элемента в свойстве `parentName` указать имя несуществующего элемента-контейнера, то возникнет ошибка "Схема не может иметь более одного корневого объекта" ("Schema cannot have more than one root object"), поскольку добавляемый элемент будет помещен в

корневой контейнер.

- **Уникальность имен.**

Каждый элемент в массиве `diff` должен иметь уникальное имя.

Пример правильного добавления элементов в массив `diff`

```
{
  "operation": "insert",
  "name": "SomeName",
  "values": { }
},
{
  "operation": "insert",
  "name": "SomeSecondName",
  "values": { }
}
```

Пример неправильного добавления элементов в массив `diff`

```
{
  "operation": "insert",
  "name": "SomeName",
  "values": { }
},
{
  "operation": "insert",
  "name": "SomeName",
  "values": { }
}
```

- **Размещение элементов представления.**

Чтобы иметь возможность настраивать и изменять элементы представления, они должны находиться на **сетке разметки**. В Creatio каждый ряд сетки разметки имеет 24 ячейки (столбца). Для размещения на сетке используется свойство `layout`.

**Свойства** элемента сетки:

- `column` — индекс левого столбца.
- `row` — индекс верхней строки.
- `colSpan` — количество занимаемых столбцов.
- `rowSpan` — количество занимаемых строк.

### Пример размещения элементов

```

{
  "operation": "insert",
  "parentName": "ParentContainerName",
  "propertyName": "items",
  "name": "ItemName",
  "values": {
    /* Размещение элемента. */
    "layout": {
      /* Начать с нулевого столбца. */
      "column": 0,
      /* Разместиться в пятой строке сетки. */
      "row": 5,
      /* Занять 12 столбцов в ширину. */
      "colSpan": 12,
      /* Занять один ряд в высоту. */
      "rowSpan": 1
    },
    "contentType": Terrasoft.ContentType.ENUM
  }
}

```

- **Количество операций.**

Если клиентская схема изменяется без помощи мастера, то для корректной работы мастера рекомендуется добавлять не более одной операции для одного элемента в редактируемой схеме.

## Свойства (properties)

Для реализации свойств необходимо использовать свойство `properties`, которое содержит JavaScript-объект.

Пример использования свойства `properties` в схеме `SectionTabsSchema` пакета `NUI` представлен ниже.

### Пример использования свойства `properties`

```

define("SectionTabsSchema", [],
function() {
  return {
    ...
    /* Объявление свойства properties. */
    properties: {
      /* Свойство parameters. Массив. */
      parameters: [],
      /* Свойство modulesContainer. Объект. */
      modulesContainer: {}
    },
    methods: {

```

```

...
/* Метод инициализации. Всегда выполняется первым. */
init: function(callback, scope) {
    ...
    /* Вызов метода, в котором используются свойства модели представления. */
    this.initContainers();
    ...
},
...
/* Метод, в котором используются свойства. */
initContainers: function() {
    /* Использование свойства modulesContainer. */
    this.modulesContainer.items = [];
    ...
    /* Использование свойства parameters. */
    this.Terrasoft.each(this.parameters, function(config) {
        config = this.applyConfigs(config);
        var moduleConfig = this.getModuleContainerConfig(config);
        var initConfig = this.getInitConfig();
        var container = viewGenerator.generatePartial(moduleConfig, initConfig)[
            this.modulesContainer.items.push(container);
        }, this);
    }, this);
},
...
    },
    ...
}
});

```

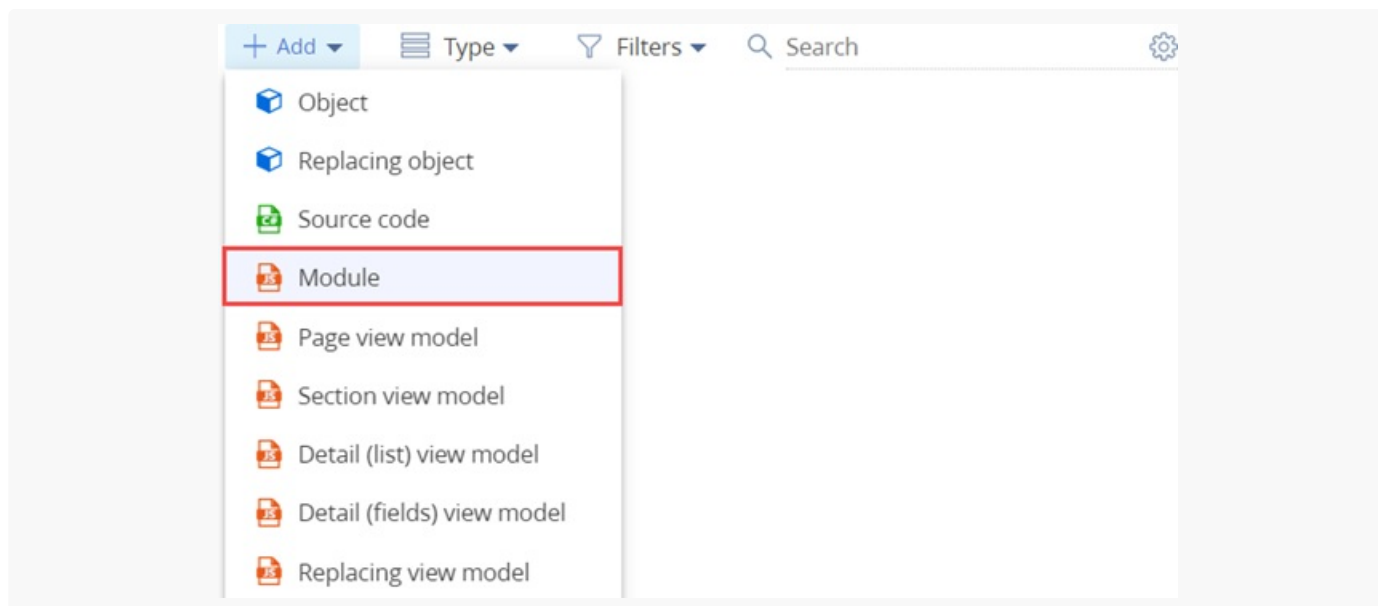
# Переопределить метод миксина

 Легкий

**Пример.** Подключить миксин `ContentImageMixin` к пользовательской схеме, переопределить метод миксина.

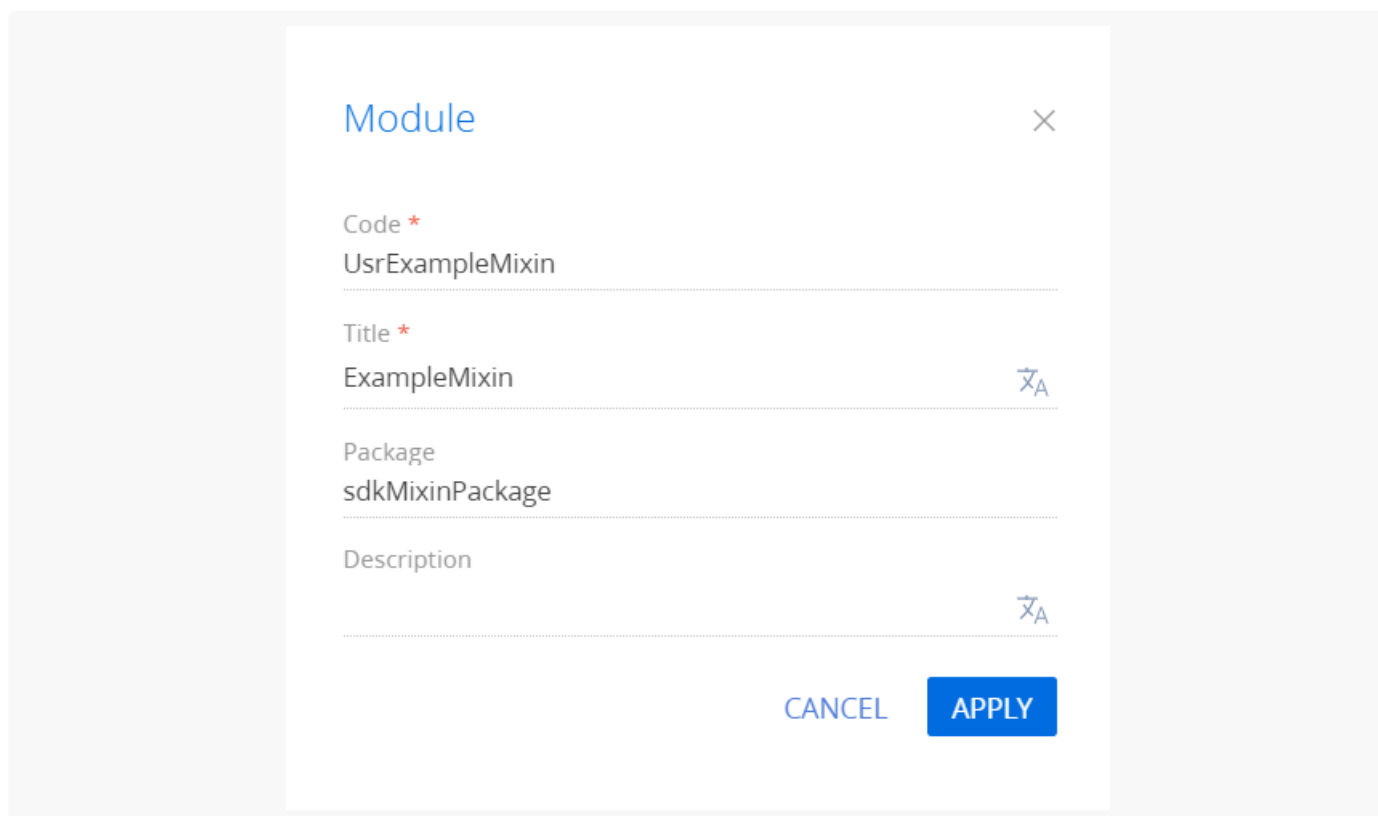
## 1. Создать миксин

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



3. В дизайнере схем заполните свойства схемы:

- [ Код ] ([ Code ]) — "UsrExampleMixin".
- [ Заголовок ] ([ Title ]) — "ExampleMixin".



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

4. В дизайнере схем добавьте исходный код.

### Исходный код модуля

```

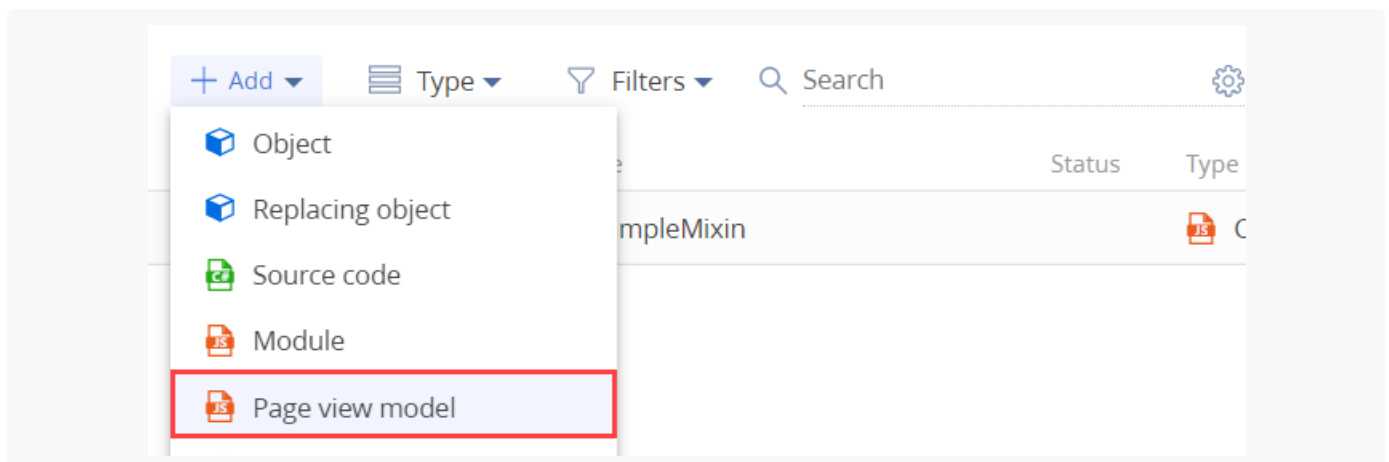
/* Определение модуля. */
define("ContentImageMixin", [ContentImageMixinV2Resources], function() {
    /* Определение класса ContentImageMixin. */
    Ext.define("Terrasoft.configuration.mixins.ContentImageMixin", {
        /* Псевдоним (сокращенное название класса). */
        alternateClassName: "Terrasoft.ContentImageMixin",
        /* Функциональность миксина. */
        getImageUrl: function() {
            var primaryImageColumnValue = this.get(this.primaryImageColumnName);
            if (primaryImageColumnValue) {
                return this.getSchemaImageUrl(primaryImageColumnValue);
            } else {
                var defImageResource = this.getDefaultImageResource();
                return this.Terrasoft.ImageUrlBuilder.getUrl(defImageResource);
            }
        }
    });
    return Ext.create(Terrasoft.ContentImageMixin);
});

```

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

## 2. Подключить МИКСИН

1. [Перейдите в раздел \[ Конфигурация \]](#) ([ Configuration ]) и выберите пользовательский [пакет](#), в который будет добавлена схема.
2. На панели инструментов реестра раздела нажмите [ Добавить ] —> [ Модель представления страницы ] ([ Add ] —> [ Page view model ]).



3. В дизайнере схем заполните свойства схемы:

- [ Код ] ([ Code ]) — "UsrExampleSchema".
- [ Заголовок ] ([ Title ]) — "ExampleSchema".



- [ Родительский объект ] ( [ Parent object ] ) — "BaseProfileSchema".

Module

×

Code \*

UsrExampleSchema

Title \*

ExampleSchema

✕<sub>A</sub>

Parent object \*

BaseProfileSchema (BaseProfileSchema)

▼

Package

sdkMixinPackage

Description

✕<sub>A</sub>

CANCEL

APPLY

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

Чтобы использовать миксин, подключите его в блоке `mixins` пользовательской схемы `ExampleSchema`.

### 3. Переопределить метод миксина

В дизайнере схем добавьте исходный код. В блоке `method` переопределите метод миксина `getReadImageURL()`. Используйте переопределенную функцию в блоке `diff`.

## Исходный код модуля

```
/* Объявление модуля. Обязательно укажите как зависимость модуль ContentImageMixin, в котором об
define("UsrExampleSchema", ["ContentImageMixin"], function() {
    return {
        entitySchemaName: "ExampleEntity",
        mixins: {
            /* Подключение миксина к схеме. */
            ContentImageMixin: "Terrasoft.ContentImageMixin"
        },
        details: /**SCHEMA_DETAILS*/{}/**SCHEMA_DETAILS*/,
```

```

diff: /**SCHEMA_DIFF*/[
  {
    "operation": "insert",
    "parentName": "AddRightsItemsHeaderImagesContainer",
    "propertyName": "items",
    "name": "AddRightsReadImage",
    "values": {
      "classes": {
        "wrapClass": ["rights-header-image"]
      },
      "getSrcMethod": "getReadImageUrl",
      "imageTitle": resources.localizableStrings.ReadImageTitle,
      "generator": "ImageCustomGeneratorV2.generateSimpleCustomImage"
    }
  }
]**SCHEMA_DIFF*/,
methods: {
  getReadImageUrl: function() {
    /* Пользовательская реализация. */
    console.log("Contains custom logic");
    /* Вызов метода миксина. */
    this.mixins.ContentImageMixin.getImageUrl.apply(this, arguments);
  },
  rules: {}
};
});

```

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

## Примеры объявления методов

 Средний

**Пример.** К базовой логике метода `setValidationConfig` класса `Terrasoft.configuration.BaseSchemaViewModel` добавьте логику проверки заполнения колонки [ Email ].

Пример замещенного метода

```

methods: {
  /* Имя метода. */
  setValidationConfig: function() {
    /* Вызов логики метода setValidationConfig схемы родителя. */
    this.callParent(arguments);
    /* Установка валидации на колонку [Email]. */
  }
}

```

```

        this.addColumnValidator("Email", EmailHelper.getEmailValidator());
    }
}

```

Пример нового метода

```

methods: {
    /* Имя метода. */
    getBlankSlateHeaderCaption: function() {
        /* Получение значения колонки MasterColumnInfo. */
        var masterColumnInfo = this.get("MasterColumnInfo");
        /* Возвращение результата работы метода. */
        return masterColumnInfo ? masterColumnInfo.caption : "";
    },
    /* Имя метода. */
    getBlankSlateIcon: function() {
        /* Возвращение результата работы метода. */
        return this.Terrasoft.ImageUrlBuilder.getUrl(this.get("Resources.Images.BlankSlateIcon"))
    }
}

```

## Пример использования массива модификаций

 Средний

```

diff: /**SCHEMA_DIFF*/[
    {
        "operation": "insert",
        "name": "CardContentWrapper",
        "values": {
            "id": "CardContentWrapper",
            "itemType": Terrasoft.ViewItemType.CONTAINER,
            "wrapClass": ["card-content-container"],
            "items": []
        }
    },
    {
        "operation": "insert",
        "name": "CardContentContainer",
        "parentName": "CardContentWrapper",
        "propertyName": "items",
    }
]

```

```

    "values": {
      "itemType": Terrasoft.ViewItemType.CONTAINER,
      "items": []
    }
  },
  {
    "operation": "insert",
    "name": "HeaderContainer",
    "parentName": "CardContentContainer",
    "propertyName": "items",
    "values": {
      "itemType": Terrasoft.ViewItemType.CONTAINER,
      "wrapClass": ["header-container-margin-bottom"],
      "items": []
    }
  },
  {
    "operation": "insert",
    "name": "Header",
    "parentName": "HeaderContainer",
    "propertyName": "items",
    "values": {
      "itemType": Terrasoft.ViewItemType.GRID_LAYOUT,
      "items": [],
      "collapseEmptyRow": true
    }
  }
}
]/**SCHEMA_DIFF*/

```

## Пример использования механизма alias при многократном замещении схемы



Есть начальный элемент массива модификаций `diff` с именем "Name" и некоторым набором свойств. Элемент расположен в контейнере `Header`. Эта схема замещена несколько раз, при этом элемент с именем "Name" всячески модифицируется и перемещается.

Свойство `diff` базовой схемы

```

diff: /**SCHEMA_DIFF*/ [
  {
    /* Операция вставки. */
    "operation": "insert",
    /* Имя элемента-родителя, в который осуществляется вставка. */

```

```

    "parentName": "Header",
    /* Имя свойства элемента-родителя, с которым производится операция. */
    "propertyName": "items",
    /* Имя элемента. */
    "name": "Name",
    /* Объект значений свойств элемента. */
    "values": {
        /* Разметка. */
        "layout": {
            /* Номер колонки. */
            "column": 0,
            /* Номер строки. */
            "row": 1,
            /* Количество объединенных колонок. */
            "colSpan": 24
        }
    }
}
] /**SCHEMA_DIFF*/

```

Свойство diff после первого замещения базовой схемы

```

diff: /**SCHEMA_DIFF*/ [
  {
    /* Операция объединения свойств двух элементов. */
    "operation": "merge",
    "name": "Name",
    "values": {
      "layout": {
        "column": 0,
        /* Номер строки. Элемент перемещен. */
        "row": 8,
        "colSpan": 24
      }
    }
  }
] /**SCHEMA_DIFF*/

```

Свойство diff после второго замещения базовой схемы

```

diff: /**SCHEMA_DIFF*/ [
  {
    /* Операция перемещения. */
    "operation": "move",
    "name": "Name",

```

```

        /* Имя элемента-родителя, в который осуществляется перемещение. */
        "parentName": "SomeContainer"
    }
] /**SCHEMA_DIFF*/

```

В новой версии элемент с именем "Name" был перемещен из элемента `SomeContainer` в элемент `ProfileContainer` и должен там остаться, несмотря на кастомизации клиента. Для этого элемент получает новое имя "NewName" и к нему добавляется конфигурационный объект `alias`.

```

diff: /**SCHEMA_DIFF*/ [
    {
        /* Операция вставки. */
        "operation": "insert",
        /* Имя элемента-родителя, в который осуществляется вставка. */
        "parentName": "ProfileContainer",
        /* Имя свойства элемента-родителя, с которым производится операция. */
        "propertyName": "items",
        /* Новое имя элемента. */
        "name": "NewName",
        /* Объект значений свойств элемента. */
        "values": {
            /* Привязка к значению свойства или функции. */
            "bindTo": "Name",
            /* Разметка. */
            "layout": {
                /* Номер колонки. */
                "column": 0,
                /* Номер строки. */
                "row": 0,
                /* Количество объединенных колонок. */
                "colSpan": 12
            }
        },

        /* Конфигурационный объект alias. */
        "alias": {
            /* Старое имя элемента. */
            "name": "Name",
            /* Массив игнорируемых свойств пользовательского замещения. */
            "excludeProperties": [ "layout" ],
            /* Массив игнорируемых операций пользовательского замещения. */
            "excludeOperations": [ "remove", "move" ]
        }
    }
] /**SCHEMA_DIFF*/

```

В новом элементе добавился `alias`, изменился родительский элемент и его расположение на странице записи. В свойстве `excludeProperties` хранится набор свойств, которые будут проигнорированы при применении разницы, а внутри `excludeOperations` хранится набор операций, которые не будут применяться к этому элементу из замещений.

В данном примере исключены свойства `layout` всех наследников для элемента с именем "Name", а также запрещены операции `remove` и `move`. Это говорит о том, что элемент с именем "NewName" будет содержать только корневое свойство `layout` и все свойства элемента "Name" из замещений, кроме `Layout`. Это же касается и операций.

#### Результат для построителя массива модификаций diff

```
diff: /**SCHEMA_DIFF*/ [
  {
    /* Операция вставки. */
    "operation": "insert",
    /* Имя элемента-родителя, в который осуществится вставка. */
    "parentName": "ProfileContainer",
    /* Имя свойства элемента-родителя, с которым производится операция. */
    "propertyName": "items",
    /* Новое имя элемента. */
    "name": "NewName",
    /* Объект значений свойств элемента. */
    "values": {
      /* Привязка к значению свойства или функции. */
      "bindTo": "Name",
      /* Разметка. */
      "layout": {
        /* Номер колонки. */
        "column": 0,
        /* Номер строки. */
        "row": 0,
        /* Количество объединенных колонок. */
        "colSpan": 12
      },
    },
  },
], /**SCHEMA_DIFF*/
```

## Свойство attributes

 Легкий

Свойство `attributes` клиентской схемы содержит конфигурационный объект со своими свойствами.

## Основные свойства

## dataValueType

Тип данных атрибута. Будет использоваться при генерации представления. Доступные типы данных представлены перечислением `Terrasoft.DataValueType`.

### Возможные значения ( `DataValueType` )

GUID	0
TEXT	1
INTEGER	4
FLOAT	5
MONEY	6
DATE_TIME	7
DATE	8
TIME	9
LOOKUP	10
ENUM	11
BOOLEAN	12
BLOB	13
IMAGE	14
CUSTOM_OBJECT	15
IMAGELOOKUP	16
COLLECTION	17
COLOR	18
LOCALIZABLE_STRING	19
ENTITY	20
ENTITY_COLLECTION	21



ENTITY_COLUMN_MAPPING_COLLECTION	22
HASH_TEXT	23
SECURE_TEXT	24
FILE	25
MAPPING	26
SHORT_TEXT	27
MEDIUM_TEXT	28
MAXSIZE_TEXT	29
LONG_TEXT	30
FLOAT1	31
FLOAT2	32
FLOAT3	33
FLOAT4	34
LOCALIZABLE_PARAMETER_VALUES_LIST	35
METADATA_TEXT	36
STAGE_INDICATOR	37

type

Тип колонки. Необязательный параметр, который используется во внутренних механизмах `BaseViewModel`. Доступные типы колонок представлены перечислением `Terrasoft.ViewModelColumnType`.

Возможные значения ( `ViewModelColumnType` )

ENTITY_COLUMN	0
CALCULATED_COLUMN	1
VIRTUAL_COLUMN	2
RESOURCE_COLUMN	3

value

Значение атрибута. Значение этого параметра будет установлено в модель представления при ее создании. В атрибуте `value` можно задавать числовые, строковые и логические значения. Если тип атрибута подразумевает использование значения ссылочного типа (массив, объект, коллекция и т. д.), то его начальное значение необходимо инициализировать с помощью методов.

#### Пример использования

##### Пример использования основных свойств атрибутов

```
attributes: {
    /* Имя атрибута. */
    "NameAttribute": {
        /* Тип данных. */
        "dataValueType": this.Terrasoft.DataValueType.TEXT,
        /* Тип колонки. */
        "type": this.Terrasoft.ViewModelColumnType.VIRTUAL_COLUMN,
        /* Значение по умолчанию. */
        "value": "NameValue"
    }
}
```

## Дополнительные свойства

caption

Заголовок атрибута.

isRequired

Признак обязательности заполнения атрибута.

dependencies

Зависимость от другого атрибута модели. Например, установка атрибута в зависимости от значения другого атрибута. Используется для создания [вычисляемых полей](#).

## lookupListConfig

Свойство, отвечающее за свойства поля-справочника. Использование данного параметра описано в статье [Настроить фильтрацию значений справочного поля](#). Это конфигурационный объект, который может содержать в себе опциональные свойства.

### Опциональные свойства

columns	Массив имен колонок, которые будут добавлены к запросу дополнительно к колонке [ <i>Id</i> ] и первичной для отображения колонке.
orders	Массив конфигурационных объектов, которые определяют сортировку данных при отображении.
filter	Метод, возвращающий объект класса <code>Terrasoft.BaseFilter</code> или его наследника, который, в свою очередь, будет применен к запросу. Не может использоваться совместно со свойством <code>filters</code> .
filters	Массив фильтров (методов, возвращающих коллекции класса <code>Terrasoft.FilterGroup</code> ). Не может использоваться совместно со свойством <code>filter</code> .

### Пример использования

#### Пример использования дополнительных свойств атрибутов

```
attributes: {
  /* Имя атрибута. */
  "Client": {
    /* Заголовок атрибута. */
    "caption": { "bindTo": "Resources.Strings.Client" },
    /* Атрибут обязателен для заполнения. */
    "isRequired": true
  },

  /* Имя атрибута. */
  "ResponsibleDepartment": {
    lookupListConfig: {
      /* Дополнительные колонки. */
      columns: "SalesDirector",
      /* Колонка сортировки. */
      orders: [ { columnPath: "FromBaseCurrency" } ],
      /* Функция определения фильтра. */
    }
  }
}
```

```

        filter: function()
        {
            /* Возвращает фильтр по колонке [Type], которая равна константе Competitor
            return this.Terrasoft.createColumnFilterWithParameter(
            this.Terrasoft.ComparisonType.EQUAL,
            "Type",
            ConfigurationConstants.AccountType.Competitor);
        }
    },
    /* Имя атрибута. */
    "Probability": {
        /* Определение зависимости колонки. */
        "dependencies": [
            {
                /* Зависит от колонки [Stage]. */
                "columns": [ "Stage" ],
                /* Имя метода-обработчика изменения колонки [Stage].
                Метод setProbabilityByStage() определен в свойстве methods объекта схемы. */
                "methodName": "setProbabilityByStage"
            }
        ]
    },
    methods: {
        /* Метод-обработчик изменения колонки [Stage]. */
        setProbabilityByStage: function()
        {
            /* Получение значения колонки [Stage]. */
            var stage = this.get("Stage");
            /* Условие изменения колонки [Probability]. */
            if (stage.value && stage.value ===
            ConfigurationConstants.Opportunity.Stage.RejectedByUs)
            {
                /* Установка значения колонки [Probability]. */
                this.set("Probability", 0);
            }
        }
    }
}

```

## Свойство messages



Средний

Свойство `messages` клиентской схемы содержит конфигурационный объект со своими свойствами.

## Свойства

mode

Режим сообщения. Доступные режимы представлены перечислением `Terrasoft.MessageMode`.

Возможные значения ( `MessageMode` )

PTP	Адресное.
BROADCAST	Широковещательное.

direction

Направление сообщения. Доступные режимы представлены перечислением `Terrasoft.MessageDirectionType`.

Возможные значения ( `MessageDirectionType` )

PUBLISH	Публикация.
SUBSCRIBE	Подписка.
BIDIRECTIONAL	Двунаправленное.

## Свойства rules и businessRules



Легкий

Свойства `rules` и `businessRules` клиентской схемы содержат конфигурационный объект со своими свойствами.

## Основные свойства

ruleType

Тип правила. Определяется значением перечисления `BusinessRuleModule.enums.RuleType`.

Возможные значения ( `BusinessRuleModule.enums.RuleType` )

BINDPARAMETER	Тип бизнес-правила. Используется для настройки привязки свойств одной колонки к значениям различных параметров. Например, чтобы настроить видимость или доступность колонки в зависимости от значения другой колонки.
FILTRATION	Тип бизнес-правила. Используется для настройки фильтрации значений колонки модели представления. Например, для фильтрации определенной колонки с типом <code>LOOKUP</code> в зависимости от значения текущего состояния страницы.

### property

Используется для типа бизнес-правила `BINDPARAMETER`. Свойство элемента управления. Задается значением перечисления `BusinessRuleModule.enums.Property`.

**Возможные значения** ( `BusinessRuleModule.enums.Property` )

VISIBLE	Видимость колонки.
ENABLED	Доступность колонки.
REQUIRED	Обязательность для заполнения.
READONLY	Колонка доступна для чтения.

### conditions

Используется для типа бизнес-правила `BINDPARAMETER`. Массив условий применения правила. Каждое условие представляет собой конфигурационный объект.

**Свойства конфигурационного объекта**

**leftExpression**

Выражение левой части условия. Представляет собой конфигурационный объект.

[Свойства конфигурационного объекта](#)

**type**

Тип выражения. Задается значением из перечисления

`BusinessRuleModule.enums.ValueType`.

[Возможные значения](#) (`BusinessRuleModule.enums.ValueType`)

CONSTANT	Константа.
ATTRIBUTE	Значение колонки модели представления.
SYSSETTING	Системная настройка.
SYSVALUE	Системное значение. Элемент списка системных значений <code>Terrasoft.core.enums.SystemValueType</code> .

**attribute**

Название колонки модели.

**attributePath**

Мета-путь к колонке справочной схемы.

**value**

Значение для сравнения.

**comparisonType**

Тип сравнения. Задается значением из перечисления

`Terrasoft.core.enums.ComparisonType`.

**rightExpression**

Выражение правой части условия. Аналогично `leftExpression`.

**logical**

Используется для типа бизнес-правила `BINDPARAMETER`. Логическая операция объединения условий из свойства `conditions`. Задается значением из перечисления `Terrasoft.LogicalOperatorType`.

### autocomplete

Используется для типа бизнес-правила `FILTRATION`. Признак обратной фильтрации. Может принимать значения `true` или `false`.

---

### autoClean

Используется для типа бизнес-правила `FILTRATION`. Признак автоматической очистки значения при изменении колонки, по которой осуществляется фильтрация. Может принимать значения `true` или `false`.

---

### baseAttributePatch

Используется для типа бизнес-правила `FILTRATION`. Мета-путь к колонке справочной схемы, по которой будет выполняться фильтрация. При построении пути к колонке применяется принцип обратной связи так же, как в `EntitySchemaQuery`. Путь формируется относительно схемы, на которую ссылается колонка модели.

---

### comparisonType

Используется для типа бизнес-правила `FILTRATION`. Тип операции сравнения. Задается значением из перечисления `Terrasoft.ComparisonType`.

---

### type

Используется для типа бизнес-правила `FILTRATION`. Тип значения, с которым будет сравниваться `baseAttributePatch`. Задается значением из перечисления `BusinessRuleModule.enums.ValueType`.

---

### attribute

Используется для типа бизнес-правила `FILTRATION`. Имя колонки модели представления. Это свойство описывается в том случае, если указан тип значения (`type`) `ATTRIBUTE`.

---

### attributePath

Используется для типа бизнес-правила `FILTRATION`. Мета-путь к колонке схемы объекта. При построении пути к колонке применяется принцип обратной связи так же, как в `EntitySchemaQuery`. Путь формируется относительно схемы, на которую ссылается колонка модели.

---

### value

Используется для типа бизнес-правила `FILTRATION`. Значение для фильтрации. Это свойство описывается в том случае, если указан тип значения (`type`) `ATTRIBUTE`.



## Дополнительные свойства

Дополнительные свойства используются только для свойства `businessRules`.

`uId`

Уникальный идентификатор правила. Значение типа "GUID".

`enabled`

Признак активности. Может принимать значения `true` или `false`.

`removed`

Признак, указывающий, является ли правило удаленным. Может принимать значения `true` или `false`.

`invalid`

Признак, указывающий, является ли правило корректным. Может принимать значения `true` или `false`.

### Примеры использования

Пример созданного мастером бизнес-правила BINDPARAMETER

```
define("SomePage", [], function() {
  return {
    /* ... */
    businessRules: /**SCHEMA_BUSINESS_RULES*/{
      /* Набор правил для колонки Type модели представления. */
      "Type": {
        /* Сгенерированный мастером код правила. */
        "ca246daa-6634-4416-ae8b-2c24ea61d1f0": {
          /* Уникальный идентификатор правила. */
          "uId": "ca246daa-6634-4416-ae8b-2c24ea61d1f0",
          /* Признак активности. */
          "enabled": true,
          /* Признак, указывающий, является ли правило удаленным. */
          "removed": false,
          /* Признак, указывающий, является ли правило корректным. */
          "invalid": false,
          /* Тип правила. */
          "ruleType": 0,
          /* Код свойства, которое регулирует правило. */
          "property": 0,
```

```

/* Логическая взаимосвязь между несколькими условиями правила. */
"logical": 0,
/* Массив условий, при выполнении которых срабатывает правило.
Сравнивает значение Account.PrimaryContact.Type со значением в колонке Type
"conditions": [
  {
    /* Тип операции сравнения. */
    "comparisonType": 3,
    /* Выражение левой части условия. */
    "leftExpression": {
      /* Тип выражения – колонка (атрибут) модели представления. */
      "type": 1,
      /* Название колонки модели представления. */
      "attribute": "Account",
      /* Путь к колонке в справочной схеме Account, значение которой сра
      "attributePath": "PrimaryContact.Type"
    },
    /* Выражение правой части условия. */
    "rightExpression": {
      /* Тип выражения – колонка (атрибут) модели представления. */
      "type": 1,
      /* Название колонки модели представления. */
      "attribute": "Type"
    }
  }
]
}
}
}/**SCHEMA_BUSINESS_RULES*/
/* ... */
};
});

```

#### Пример созданного мастером бизнес-правила FILTRATION

```

define("SomePage", [], function() {
  return {
    /* ... */
    businessRules: /**SCHEMA_BUSINESS_RULES*/{
      /* Набор правил для колонки Type модели представления. */
      "Account": {
        /* Сгенерированный мастером код правила. */
        "a78b898c-c999-437f-9102-34c85779340d": {
          /* Уникальный идентификатор правила. */
          "uId": "a78b898c-c999-437f-9102-34c85779340d",
          /* Признак активности. */
          "enabled": true,

```

```

        /* Признак, указывающий, является ли правило удаленным. */
        "removed": false,
        /* Признак, указывающий, является ли правило корректным. */
        "invalid": false,
        /* Тип правила. */
        "ruleType": 1,
        /* Путь к колонке для фильтрации в справочной схеме Account, на которую ссы-
        "baseAttributePatch": "PrimaryContact.Type",
        /* Тип операции сравнения в фильтре. */
        "comparisonType": 3,
        /* Тип выражения – колонка (атрибут) модели представления. */
        "type": 1,
        /* Название колонки модели представления, по значению которой будет выполни-
        "attribute": "Type"
    }
}
}/**SCHEMA_BUSINESS_RULES*/
/* ... */
};
});

```

## Свойство diff JS

 Средний

Свойство `diff` клиентской схемы содержит массив конфигурационных объектов со своими свойствами.

### Свойства

operation

Операции с элементами.

[Возможные значения](#)

set	Значение элемента схемы устанавливается значением параметра <code>values</code> .
merge	Значения из родительских, замещаемых и замещающих схем сливаются вместе, при этом свойства из значения параметра <code>values</code> последнего наследника имеют приоритет.
remove	Элемент удаляется из схемы.
move	Элемент перемещается в другой родительский элемент.
insert	Элемент добавляется в схему.

name

Имя элемента схемы, с которым выполняется операция.

parentName

Имя родительского элемента схемы, в котором размещается элемент при операции `insert`, или в который перемещается элемент при операции `move`.

propertyName

Имя параметра родительского элемента при операции `insert`. Также используется при операции `remove`, если нужно удалить только определенные параметры элемента, а не весь элемент.

index

Индекс, в который перемещается или добавляется параметр. Параметр используется с операциями `insert` и `move`. Если параметр не указан, то вставка идет последним элементом массива.

values

Объект, свойства которого будут установлены либо объединены со свойствами элемента схемы. Используется при операциях `set`, `merge` и `insert`.

Набор основных элементов, которые можно отобразить на странице, представлены перечислением `Terrasoft.ViewItemType`.

Возможные значения ( `ViewItemType` )

GRID_LAYOUT	0	Элемент-сетка, включающий в себя размещение других элементов управления.
-------------	---	--

TAB_PANEL	1	Набор вкладок.
DETAIL	2	Деталь.
MODEL_ITEM	3	Элемент модели представления.
MODULE	4	Модуль.
BUTTON	5	Кнопка.
LABEL	6	Надпись.
CONTAINER	7	Контейнер.
MENU	8	Выпадающий список.
MENU_ITEM	9	Элемент выпадающего списка.
MENU_SEPARATOR	10	Разделитель выпадающего списка.
SECTION_VIEWS	11	Представления раздела.
SECTION_VIEW	12	Представление раздела.
GRID	13	Реестр.
SCHEDULE_EDIT	14	Планировщик.
CONTROL_GROUP	15	Группа элементов.
RADIO_GROUP	16	Группа переключателей.
DESIGN_VIEW	17	Настраиваемое представление.
COLOR_BUTTON	18	Цвет.
IMAGE_TAB_PANEL	19	Набор вкладок с иконками.
HYPERLINK	20	Гиперссылка.
INFORMATION_BUTTON	21	Информационная кнопка из всплывающей подсказкой.
TIP	22	Всплывающая подсказка.
COMPONENT	23	Компонент.
PROGRESS_BAR	30	Индикатор.

alias

Конфигурационный объект.

Свойства объекта `alias`

name	Имя элемента, с которым связан новый элемент. По этому имени будут находиться элементы в замещенных схемах и связываться с новым элементом. Значение свойства <code>name</code> элемента массива модификаций <code>diff</code> не должно быть равным свойству <code>alias.name</code> .
excludeProperties	Массив свойств объекта <code>values</code> элемента массива модификаций <code>diff</code> , которые не будут применяться при построении <code>diff</code> .
excludeOperations	Массив операций, которые не должны применяться к данному элементу при построении массива модификаций <code>diff</code> .

Пример использования

#### Пример использования объекта `alias`

```
/* Массив diff. */
diff: /**SCHEMA_DIFF*/ [
  {
    /* Операция, совершаемая с элементом. */
    "operation": "insert",
    /* Новое имя элемента. */
    "name": "NewElementName",
    /* Значения элемента. */
    "values": {
      /* ... */
    },
    /* Конфигурационный объект alias. */
    "alias": {
      /* Предыдущее имя элемента. */
      "name": "OldElementName",
      /* Массив исключаемых свойств. */
      "excludeProperties": [ "layout", "visible", "bindTo" ],
      /* Массив игнорируемых операций. */
      "excludeOperations": [ "remove", "move", "merge" ]
    }
  },
  /* ... */
]
```

