

# Бизнес-логика элементов управления

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>Понятие элемента управления</b>	<b>4</b>
<b>Создать элемент управления</b>	<b>4</b>
1. Создать модуль	4
2. Создать класс элемента управления	5
3. Добавить элемент управления в интерфейс приложения	7
4. Проверить результат выполнения примера	10
<b>Создать элемент управления для редактирования исходного кода</b>	<b>11</b>
1. Подключить миксин	11
2. Реализовать абстрактные методы миксина	11
3. Вызвать метод openSourceCodeBox()	12
4. Реализовать метод удаления миксина	12
Полный исходный код примера	12
<b>Класс SourceCodeEditMixin</b>	<b>13</b>
Свойства	13
Методы	14

# Понятие элемента управления



**Элементы управления** — это объекты, используемые для организации взаимодействия между пользователем и приложением. Это, например, кнопки, поля ввода, элементы выбора и т. д.

Все элементы управления Creatio наследуются от класса `Terrasoft.controls.Component`. Описание классов, которые реализуют компоненты, содержится в документации [Библиотека JS классов](#).

В соответствии с принципом [открытости-закрытости](#), невозможно добавить пользовательскую логику непосредственно в существующий элемент управления.

**Алгоритм реализации** пользовательской логики элемента управления:

1. Создайте клиентский модуль.
2. В модуле объявите класс, наследующий функциональность существующего элемента управления.
3. В классе-наследнике реализуйте требуемую функциональность.
4. Добавьте новый элемент в интерфейс приложения.

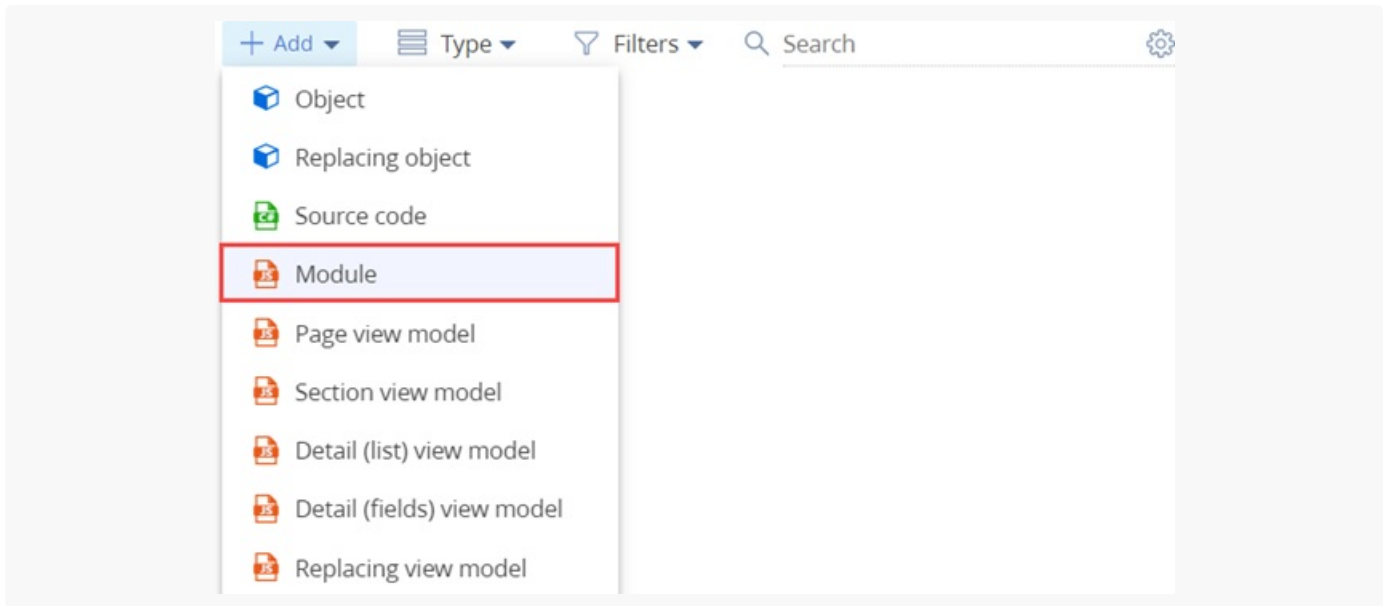
## Создать элемент управления



**Пример.** Создать элемент управления, который позволяет вводить только целые числа в заданном диапазоне. При нажатии кнопки Enter проверять введенное значение и выводить соответствующее сообщение, если число выходит за диапазон заданных значений. В качестве родительского использовать элемент управления `Terrasoft.controls.IntegerEdit`.

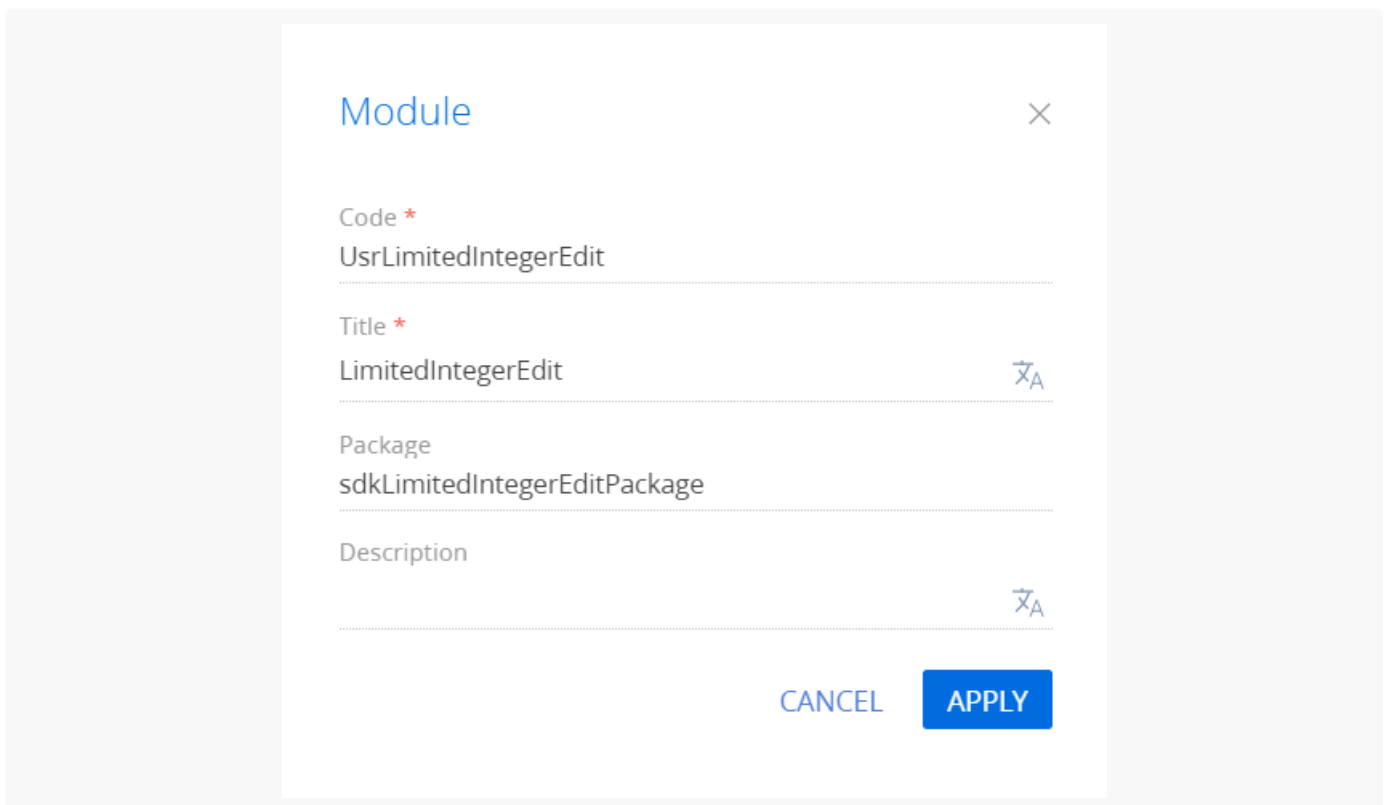
### 1. Создать модуль

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 ]) — "UsrLimitedIntegerEdit".
- [ Заголовок ] ([ Title ]) — "LimitedIntegerEdit".



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

## 2. Создать класс элемента управления

В дизайнере схем добавьте исходный код.

**Исходный код модуля**

```

define("UsrLimitedIntegerEdit", [], function () {
    // Объявление класса элемента управления.
    Ext.define("Terrasoft.controls.UsrLimitedIntegerEdit", {
        // Базовый класс.
        extend: "Terrasoft.controls.IntegerEdit",
        // Псевдоним (сокращенное название класса).
        alternateClassName: "Terrasoft.UsrLimitedIntegerEdit",
        // Наименьшее допустимое значение.
        minLimit: -1000,
        // Наибольшее допустимое значение.
        maxLimit: 1000,
        // Метод проверки на вхождение в диапазон допустимых значений.
        isOutOfLimits: function (numericValue) {
            if (numericValue < this.minLimit || numericValue > this.maxLimit) {
                return true;
            }
            return false;
        },
        // Переопределение метода-обработчика события нажатия клавиши Enter.
        onEnterKeyPressed: function () {
            // Вызов базовой функциональности.
            this.callParent(arguments);
            // Получение введенного значения.
            var value = this.getTypedValue();
            // Приведение к числовому типу.
            var numericValue = this.parseNumber(value);
            // Проверка на вхождение в диапазон допустимых значений.
            var outOfLimits = this.isOutOfLimits(numericValue);
            if (outOfLimits) {
                // Формирование предупреждающего сообщения.
                var msg = "Value " + numericValue + " is out of limits [" + this.minLimit + ", "
                // Изменение конфигурационного объекта для показа предупреждающего сообщения.
                this.validationInfo.isValid = false;
                this.validationInfo.invalidMessage = msg;
            }
            else{
                // Изменение конфигурационного объекта для скрытия предупреждающего сообщения.
                this.validationInfo.isValid = true;
                this.validationInfo.invalidMessage = "";
            }
            // Вызов логики отображения предупреждающего сообщения.
            this.setMarkOut();
        },
    });
});

```

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

**На заметку.** Логику, аналогичную логике метода `onEnterKeyPressed()`, можно использовать в обработчике события потери фокуса `onBlur()`.

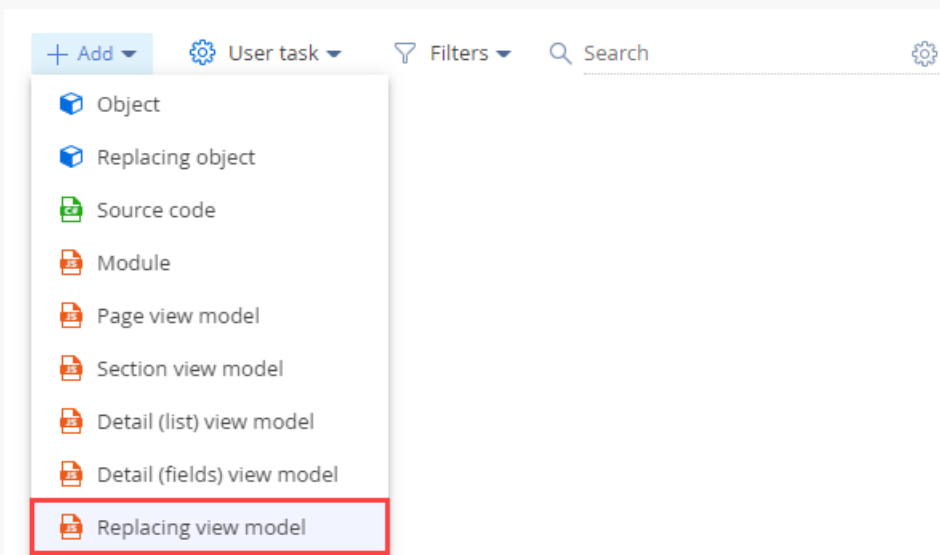
Кроме стандартных свойств `extend` и `alternateClassName`, в класс добавлены свойства `minLimit` и `maxLimit`, которые задают диапазон допустимых значений. Для этих свойств указаны значения по умолчанию.

Пользовательская логика элемента управления реализована в переопределенном методе `onEnterKeyPressed`. После вызова базовой логики, в которой выполняется генерация событий изменения значения, введенное значение проверяется на допустимость. Если число недопустимо, то в поле ввода отображается соответствующее предупреждающее сообщение. Для проверки вхождения введенного значения в диапазон допустимых значений предусмотрен метод `isOutOfLimits`.

Несмотря на вывод соответствующего предупреждения, при текущей реализации введенное значение все равно сохраняется и передается в модель представления схемы, в которой будет использован компонент.

### 3. Добавить элемент управления в интерфейс приложения

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



3. В дизайнерах схем свойству [ *Родительский объект* ] ([ *Parent object* ]) задайте значение "Display schema - Contact card" пакета [ *ContactPageV2* ]. После подтверждения выбранного родительского

объекта остальные свойства будут заполнены автоматически.

**Module** [X]

Code  
ContactPageV2

Title \*  
Display schema - Contact card [Text alignment icon]

Parent object \*  
Display schema - Contact card (ContactPageV2) [Dropdown arrow]

Package  
sdkLimitedIntegerEditPackage

Description [Text alignment icon]

CANCEL APPLY

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

4. В дизайнере схем добавьте исходный код.

#### Исходный код модуля

```
// Объявление модуля. Обязательно следует указать как зависимость
// модуль, в котором объявлен класс элемента управления.
define("ContactPageV2", ["UsrLimitedIntegerEdit"],
function () {
return {
attributes: {
// Атрибут, связываемый со значением в элементе управления.
"ScoresAttribute": {
// Тип данных атрибута – целочисленный.
"dataValueType": this.Terrasoft.DataValueType.INTEGER,
// Тип атрибута – виртуальная колонка.
"type": this.Terrasoft.ViewModelColumnType.VIRTUAL_COLUMN,
// Значение по умолчанию.
"value": 0
}
```



```

    }
  },
  diff: /**SCHEMA_DIFF*/[
    {
      "operation": "insert",
      "parentName": "ProfileContainer",
      "propertyName": "items",
      "name": "Scores",
      "values": {
        "contentType": Terrasoft.ContentType.LABEL,
        "caption": {"bindTo": "Resources.Strings.ScoresCaption"},
        "layout": {
          "column": 0,
          "row": 6,
          "colSpan": 24
        }
      }
    }
  ],
  {
    // Операция добавления компонента на страницу.
    "operation": "insert",
    // Мета-имя родительского контейнера, в который добавляется поле.
    "parentName": "ProfileContainer",
    // Поле добавляется в коллекцию компонентов
    // родительского элемента.
    "propertyName": "items",
    // Имя колонки схемы, к которой привязан компонент.
    "name": "ScoresValue",
    "values": {
      // Тип элемента управления – компонент.
      "itemType": Terrasoft.ViewItemType.COMPONENT,
      // Название класса.
      "className": "Terrasoft.UsrLimitedIntegerEdit",
      // Свойство value компонента связано с атрибутом ScoresAttribute.
      "value": { "bindTo": "ScoresAttribute" },
      // Значения для свойства minLimit.
      "minLimit": -300,
      // Значения для свойства maxLimit.
      "maxLimit": 300,
      // Свойства расположения компонента в контейнере.
      "layout": {
        "column": 0,
        "row": 6,
        "colSpan": 24
      }
    }
  }
]/**SCHEMA_DIFF*/

```

```
};
});
```

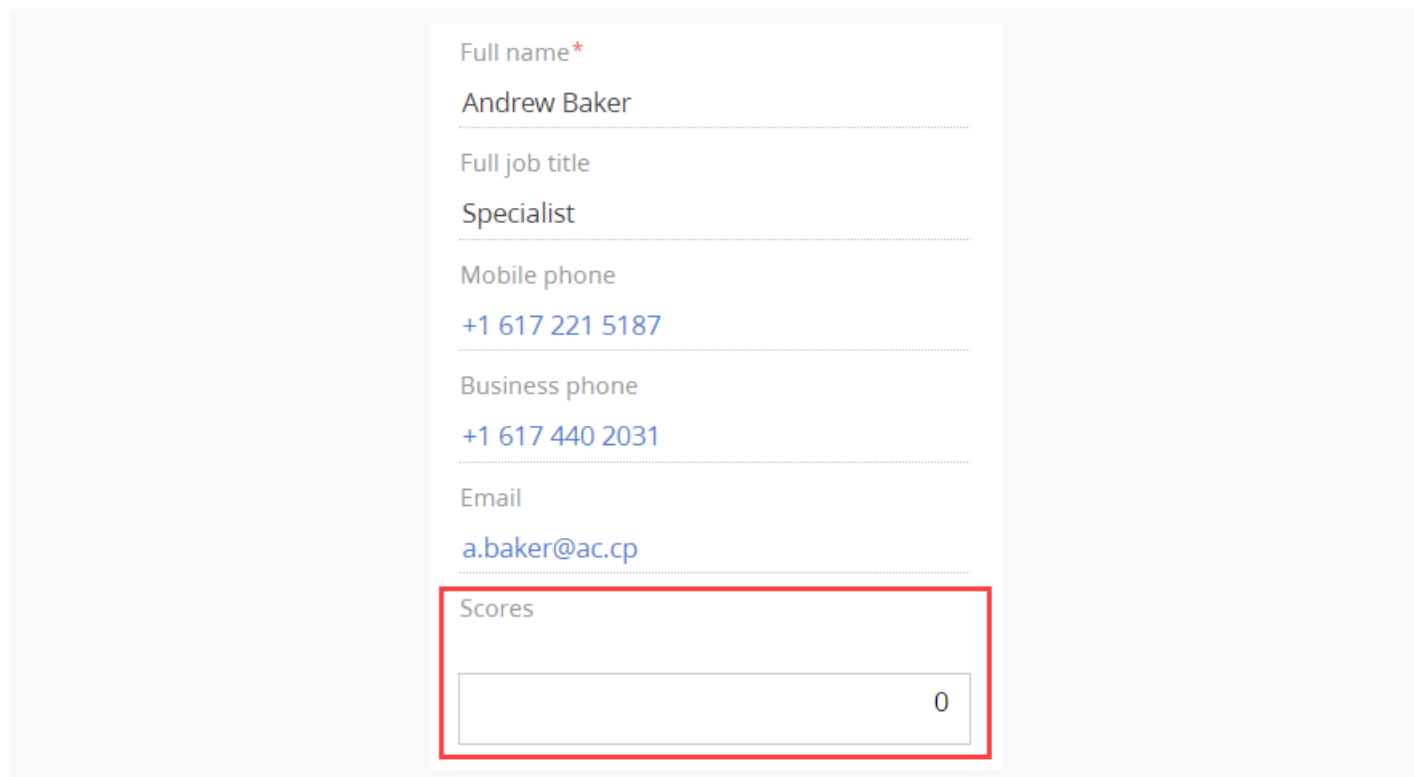
5. На панели инструментов дизайнера нажмите [ Сохранить ] ([ Save ]).

Значение атрибута `ScoresAttribute` связано со значением, введенным в поле ввода элемента управления. Вместо атрибута можно использовать целочисленную колонку объекта, связанного со схемой страницы редактирования записи.

В свойство `diff` добавлен конфигурационный объект, определяющий значения свойств экземпляра элемента управления. Значение свойства `value` связано с атрибутом `ScoresAttribute`. Свойствам `minLimit` и `maxLimit` присвоены значения, указывающие допустимый диапазон ввода. Если в конфигурационном объекте явно не указать свойства `minLimit` и `maxLimit`, то по умолчанию будет использоваться диапазон допустимых значений `[-1000, 1000]`.

## 4. Проверить результат выполнения примера

В результате выполнения примера, на странице редактирования записи контакта будет добавлено поле ввода числовых значений.



Full name\*

Andrew Baker

Full job title

Specialist

Mobile phone

+1 617 221 5187

Business phone

+1 617 440 2031

Email

a.baker@ac.cp

Scores

0

При вводе в поле недопустимого значения отобразится предупреждающее сообщение.

# Создать элемент управления для редактирования исходного кода

 Средний

## 1. Подключить миксин

Для использования миксина в элементе управления добавьте его в свойстве `mixins`:

### Подключение миксина

```
mixins: {
  SourceCodeEditMixin: "Terrasoft.SourceCodeEditMixin"
},
```

## 2. Реализовать абстрактные методы миксина

Функциональность миксина получает значение, вызывая абстрактный **getter**-метод `getSourceCodeValue()`, задача которого — вернуть строку для редактирования. В каждом конкретном случае "подмешивания" функциональности должен быть реализован свой **getter**-метод:

### Реализация метода получения строкового значения

```
getSourceCodeValue: function () {
  // Метод getValue() реализован в базовом классе Terrasoft.BaseEdit.
  return this.getValue();
},
```

После завершения редактирования миксин вызовет абстрактный **setter**-метод `setSourceCodeValue()` для сохранения результата. В каждом конкретном случае "подмешивания" функциональности должен быть реализован свой **setter**-метод.

### Реализация метода установки результирующей строки

```
setSourceCodeValue: function (value) {
    // Метод setValue() реализован в базовом классе Terrasoft.BaseEdit.
    this.setValue(value);
},
```

### 3. Вызвать метод openSourceCodeBox()

Для открытия окна редактирования исходного кода вызовите метод миксина `openSourceCodeBox()`. Следует обратить внимание на то, что метод вызван в контексте экземпляра основного класса. Например, при вызове метода `onSourceButtonClick()` компонента.

#### Реализация вызова метода открытия окна редактора исходного кода

```
onSourceButtonClick: function () {
    this.mixins.SourceCodeEditMixin
        .openSourceCodeBox.call(this);
},
```

### 4. Реализовать метод удаления миксина

После завершения работы с экземпляром основного класса происходит его удаление из памяти. Так как `SourceCodeEditMixin` требует определенных ресурсов, их также необходимо освободить. Для этого вызывается метод миксина `destroySourceCode()` в контексте экземпляра основного класса.

```
onDestroy: function () {
    this.mixins.SourceCodeEditMixin
        .destroySourceCode.apply(this, arguments);
    this.callParent(arguments);
}
```

## Полный исходный код примера

#### SomeControl.js

```
// Добавление модуля миксина в зависимости.
define("SomeControl", ["SomeControlResources", "SourceCodeEditMixin"],
    function (resources) {
        Ext.define("Terrasoft.controls.SomeControl", {
            extend: "Terrasoft.BaseEdit",
            alternateClassName: "Terrasoft.SomeControl",
```

```

// Подключение миксина.
mixins: {
    SourceCodeEditMixin: "Terrasoft.SourceCodeEditMixin"
},

// Реализация метода получения строкового значения.
getSourceCodeValue: function () {
    // Метод getValue() реализован в базовом классе Terrasoft.BaseEdit.
    return this.getValue();
},

// Реализация метода установки результирующей строки.
setSourceCodeValue: function (value) {
    // Метод setValue() реализован в базовом классе Terrasoft.BaseEdit.
    this.setValue(value);
},

// Реализация вызова метода открытия окна редактора исходного кода.
onSourceButtonClick: function () {
    this.mixins.SourceCodeEditMixin
        .openSourceCodeBox.call(this);
},

// Реализация вызова очистки ресурсов миксина.
onDestroy: function () {
    this.mixins.SourceCodeEditMixin
        .destroySourceCode.apply(this, arguments);
    this.callParent(arguments);
}
});
});

```

## Класс SourceCodeEditMixin

 Сложный

Класс `SourceCodeEditMixin` предназначен для реализации элемента управления, предоставляющего возможность редактирования строкового значения, содержащего HTML, JavaScript или LESS код.

`SourceCodeEditMixin` — это [МИКСИН](#), который предназначен не для самостоятельного использования, а для обогащения других классов возможностью редактирования строки, используя удобный интерфейс. Концепция миксина напоминает концепцию множественного наследования.

### Свойства

---

```
sourceCodeEdit Terrasoft.SourceCodeEdit
```

Экземпляр элемента управления редактора исходного кода.

---

```
sourceCodeEditContainer Terrasoft.Container
```

Экземпляр контейнера, в котором размещен редактор исходного кода.

---

## Методы

---

```
openSourceCodeEditModalBox()
```

Метод, реализующий открытие модального окна редактирования исходного кода.

---

```
loadSourceCodeValue()
```

Абстрактный метод. Должен быть реализован в основном классе. Реализует логику получения значения для редактирования.

---

```
saveSourceCodeValue()
```

Абстрактный метод. Должен быть реализован в основном классе. Реализует логику сохранения результата редактирования в объект основного класса.

---

### Параметры

<code>{String} value</code>	Результат редактирования.
-----------------------------	---------------------------

---

```
destroySourceCodeEdit()
```

Метод, реализующий очистку ресурсов, используемых миксином.

---

```
getSourceCodeEditModalBoxStyleConfig()
```

Возвращает объект типа "ключ-значение", описывающий стили, которые будут установлены на модальное окно редактора исходного кода.

---

```
getSourceCodeEditStyleConfig()
```

Возвращает объект типа ключ-значение, описывающий стили, которые будут применены к элементам управления редактора исходного кода.

---

getSourceCodeEditConfig()

Возвращает объект типа "ключ-значение", описывающий свойства, с которыми будет создан экземпляр редактора исходного кода.

#### Свойства созданного объекта

{Boolean} showWhitespaces	Отображение невидимых строк. По умолчанию: <code>false</code> .
{SourceCodeEditEnums.Language} language	<p>Синтаксис языка. Выбирается из перечисления <code>SourceCodeEditEnums.Language</code>. По умолчанию: <code>SourceCodeEditEnums.Language.JAVASCRIPT</code>.</p> <p><b>Возможные значения</b> ( <code>SourceCodeEditEnums.Language</code> )</p> <hr/> <p>JAVASCRIPT</p> <p>JavaScript</p> <hr/> <p>CSHARP</p> <p>C#</p> <hr/> <p>LESS</p> <p>LESS</p> <hr/> <p>CSS</p> <p>CSS</p> <hr/> <p>SQL</p> <p>SQL</p> <hr/> <p>HTML</p> <p>HTML</p>
{SourceCodeEditEnums.Theme} theme	<p>Тема редактора. Выбирается из перечисления <code>SourceCodeEditEnums.Theme</code>. По умолчанию: <code>SourceCodeEditEnums.Theme.CRIMSON_EDITOR</code>.</p> <p><b>Возможные значения</b> ( <code>SourceCodeEditEnums.Theme</code> )</p>

---

`SQLSERVER`

Тема редактора SQL.

---

`CRIMSON_EDITOR`

Тема редактора Crimson.

<code>{Boolean} showLineNumbers</code>	Отображение номеров строк. По умолчанию: <code>true</code> .
<code>{Boolean} showGutter</code>	Установка зазора между столбцами. По умолчанию: <code>true</code> .
<code>{Boolean} highlightActiveLine</code>	Подсветка активной линии. По умолчанию: <code>true</code> .
<code>{Boolean} highlightGutterLine</code>	Подсветка линии в межстолбцовом промежутке. По умолчанию: <code>true</code> .