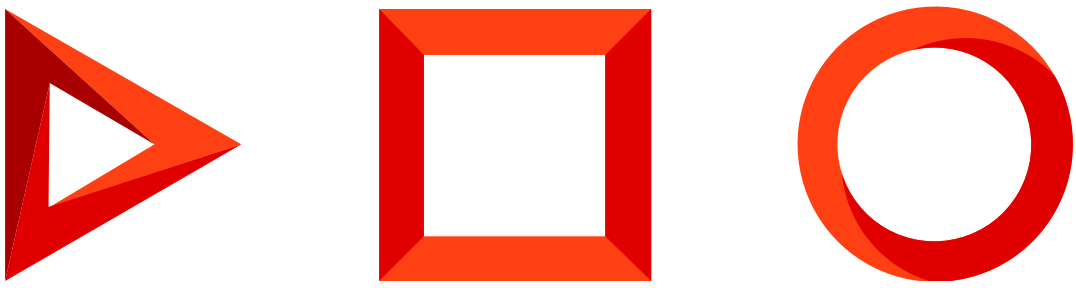
Creatio Academy

# Виды модулей

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

# Виды модулей

Легкий

## Базовые модули

В Creatio реализованы **базовые модули**:

- `ext-base` — реализует функциональность фреймворка `ExtJs`.
- `terrasoft` пространства имен и объектов `Terrasoft` — реализует доступ к системным операциям, переменным ядра и т. д.
- `sandbox` — реализует механизм обмена сообщениями между модулями.

**Доступ к модулям `ext-base`, `terrasoft` и `sandbox`**

```
// Определение модуля и получение ссылок на модули-зависимости.
define("ExampleModule", ["ext-base", "terrasoft", "sandbox"],
    // Ext – ссылка на объект, дающий доступ к возможностям фреймворка ExtJs.
    // Terrasoft – ссылка на объект, дающий доступ к системным переменным, переменным ядра и т.д
    // sandbox – используется для обмена сообщениями между модулями.
    function (Ext, Terrasoft, sandbox) {
});
```

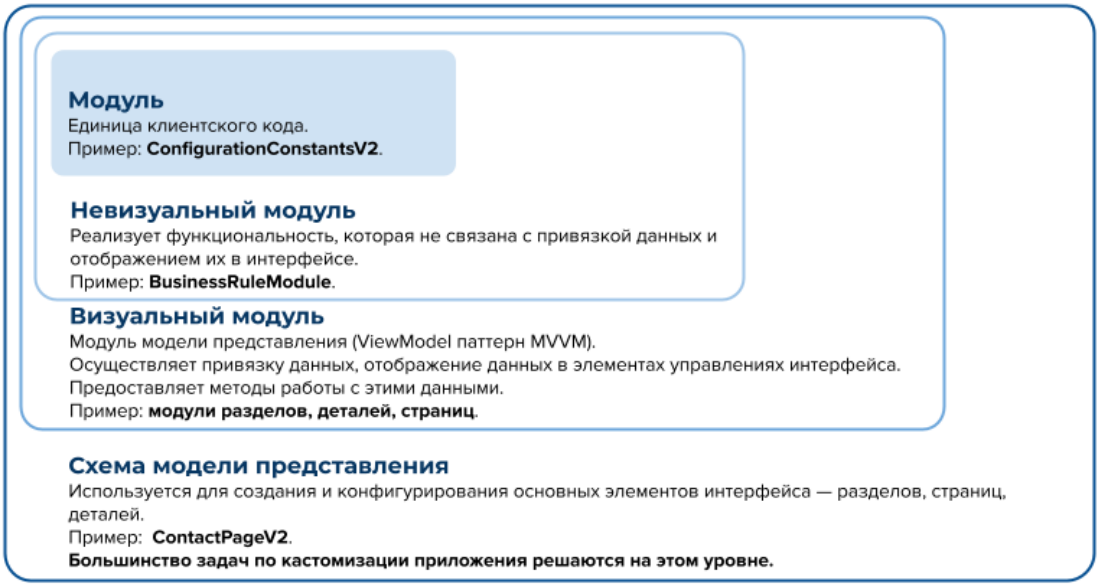
Базовые модули используются в большинстве клиентских модулей. Указывать базовые модули в зависимостях `["ext-base", "terrasoft", "sandbox"]` не обязательно. После создания объекта класса модуля объекты `Ext`, `Terrasoft` и `sandbox` будут доступны как свойства объекта `this.Ext`, `this.Terrasoft`, `this.sandrenderbox`.

## Клиентские модули

### Виды клиентских модулей

- Невизуальный модуль (схема модуля).
- Визуальный модуль (схема модели представления).
- Модуль расширения и замещающий клиентский модуль (схема замещающей модели представления).

Иерархия клиентских модулей представлена на рисунке ниже.




Разработка клиентских модулей описана в статье Разработка конфигурационных элементов.

## Невизуальный модуль

**Назначение** невизуального модуля — реализация функциональности системы, которая, как правило, не сопряжена с привязкой данных и отображением их в интерфейсе. Примерами невизуальных модулей в системе являются модули бизнес-правил (`BusinessRuleModule`) и утилитные модули, которые реализуют служебные функции.

Разработка невизуального модуля описана в статье Схема модуля.

## Визуальный модуль

К визуальным относятся модули, которые реализуют в системе модели представления (`ViewModel`) согласно шаблону MVVM.

**Назначение** визуального модуля — инкапсуляция данных, которые отображаются в элементах управления графического интерфейса, и методов работы с данными. Примерами визуальных модулей в системе являются модули разделов, деталей, страниц.

Разработка визуального модуля описана в статье Схема модели представления.

## Модуль расширения и замещающий клиентский модуль

**Назначение** модулей расширения и замещающих клиентских модулей — расширение функциональности базовых модулей.

Разработка модуля расширения и замещающего клиентского модуля описана в статье Схема замещающей модели представления.

# Особенности клиентских модулей

## Методы клиентских модулей

- `init()` — метод реализует логику, выполняемую при загрузке модуля. При загрузке модуля клиентское ядро автоматически вызывает этот метод первым. Как правило, в методе `init()` выполняется подписка на события других модулей и инициализация значений.
- `render(renderTo)` — метод реализует логику визуализации модуля. При загрузке модуля и наличии метода клиентское ядро автоматически его вызывает. Для корректного отображения данных перед их визуализацией должен отработать механизм связывания представления (`View`) и модели представления (`ViewModel`). Поэтому, как правило, в методе `render()` выполняется запуск механизма — вызов у объекта представления метода `bind()`. Если модуль загружается в контейнер, то в качестве аргумента метода `render()` будет передана ссылка на этот контейнер. Метод `render()` в обязательном порядке должен реализовываться визуальными модулями.

## Вызов одного модуля из другого. Утилитные модули

Несмотря на то что модуль является изолированной программной единицей, он может использовать функциональность других модулей. Для этого достаточно в качестве зависимости импортировать тот модуль, функциональность которого предполагается использовать. Доступ к экземпляру модуля–зависимости осуществляется через аргумент фабричной функции.

Так, в процессе разработки вспомогательные и служебные методы общего назначения можно группировать в отдельные **утилитные модули** и затем импортировать их в те модули, в которых необходима эта функциональность.

## Работа с ресурсами

**Ресурсы** — дополнительные свойства схемы. Ресурсы добавляются в клиентскую схему на панели свойств дизайнера (кнопка +). Чаще всего используемые ресурсы — локализуемые строки (свойство [ *Локализуемые строки* ] ([ *Localizable strings* ])) и изображения (свойство [ *Изображения* ] ([ *Images* ])). Ресурсы содержатся в специальном модуле с именем `[ИмяКлиентскогоМодуля]Resources`, который автоматически генерирует ядро приложения для каждого клиентского модуля.

Чтобы получить доступ к модулю ресурсов из клиентского модуля, необходимо в качестве зависимости импортировать модуль ресурсов в клиентский модуль. В коде модуля необходимо использовать локализуемые ресурсы, а не строковые литералы или константы.

## Использование модулей расширения

Модули расширения базовой функциональности не поддерживают наследование в традиционном его представлении.

**Особенности** создания расширяющего модуля:

1. Скопировать программный код расширяемого модуля.
2. Внести изменения в функциональность.

В замещающем модуле нельзя использовать ресурсы замещаемого модуля — все ресурсы должны

заново создаваться в замещающей схеме.

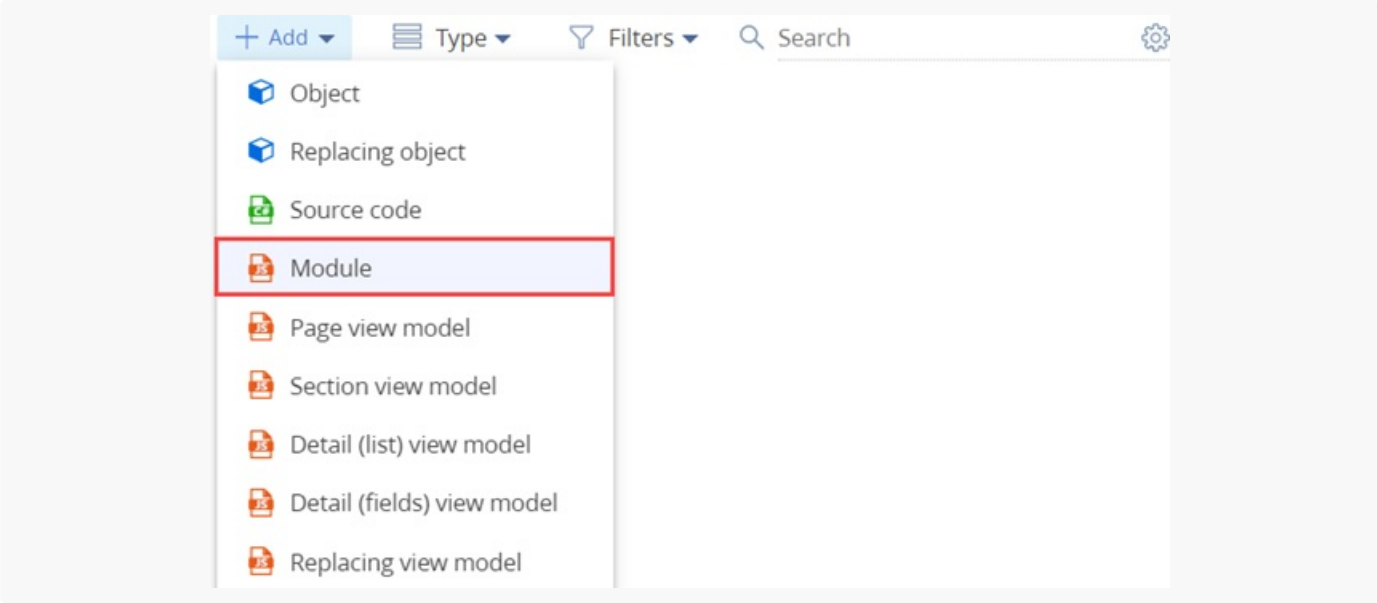
# Создать стандартный модуль

Средний

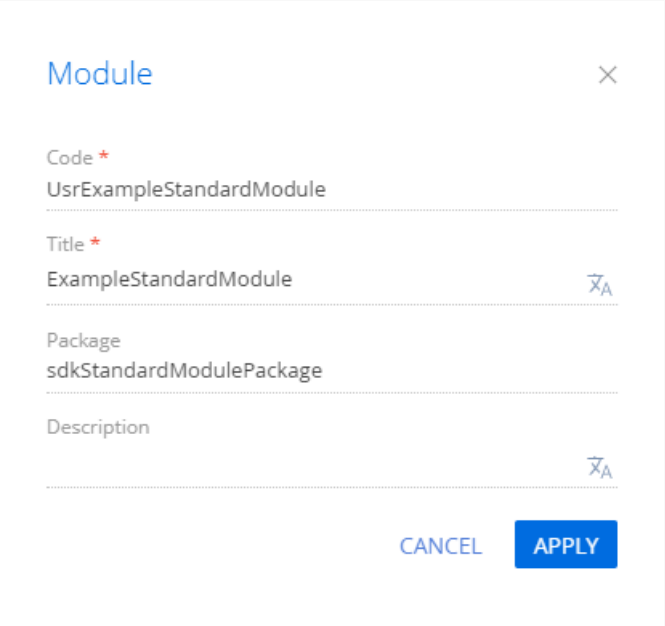
**Пример.** Создать стандартный модуль, который содержит методы `init()` и `render()`. Каждый метод должен отображать информационное сообщение. При загрузке модуля на клиент ядро сначала вызовет метод `init()`, а затем — метод `render()`, о чем должны оповестить соответствующие информационные сообщения.

## 1. Создать визуальный 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* ]) — "`UsrExampleStandardModule`".
   - [ *Заголовок* ] ([ *Title* ]) — "`ExampleStandardModule`".




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

4. В дизайнере схем добавьте исходный код.

**Исходный код модуля**

```
// Объявление модуля с именем UsrExampleStandartModule. Модуль не имеет никаких зависимостей,
// поэтому в качестве второго параметра передается пустой массив.
define("UsrExampleStandardModule", [],
    // Функция-фабрика возвращает объект модуля с двумя методами.
    function () {
        return {
            // Метод будет вызван ядром самым первым, сразу после загрузки на клиент.
            init: function () {
                alert("Calling the init() method of the UsrExampleStandardModule module");
            },
            // Метод будет вызван ядром при загрузке модуля в контейнер. Ссылка на контейнер
            // в качестве параметра renderTo. В информационном сообщении будет выведен id эле
            // в котором должны отображаться визуальные данные модуля. По умолчанию – centerP
            render: function (renderTo) {
                alert("Calling the render() method of the UsrExampleStandardModule module. Th
            }
        };
    });
```

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

# 2. Проверить визуальный модуль

Базовая версия Creatio позволяет проверить визуальный модуль, выполнить его загрузку на клиент и

визуализацию. Для этого сформируйте адресную строку запроса.

Адресная строка запроса

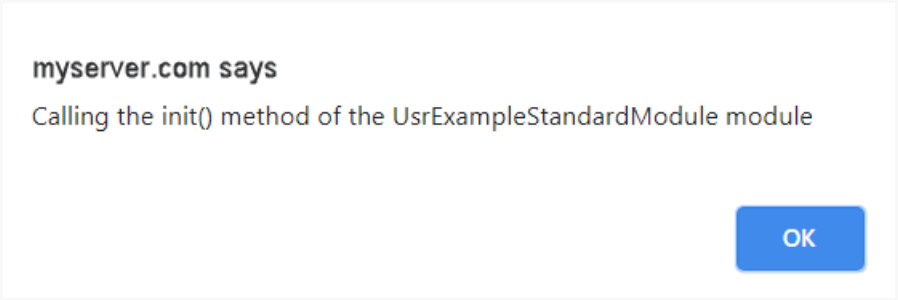
```
[АдресПриложения]/[НомерКонфигурации]/0/NUI/ViewModule.aspx#[ИмяМодуля]
```

Пример адресной строки запроса

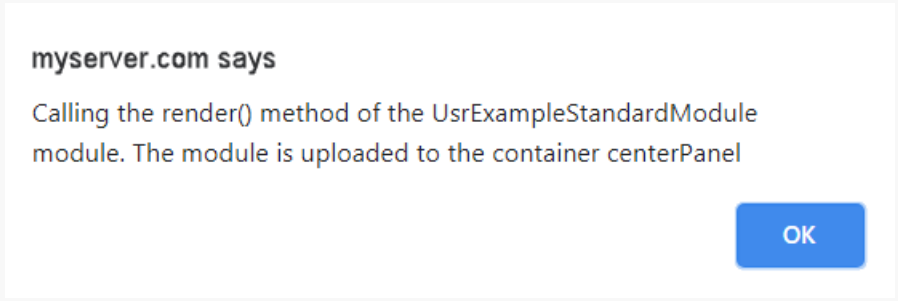
```
http://myserver.com/CreatioWebApp/0/NUI/ViewModule.aspx#UsrExampleStandardModule
```

На клиент будет возвращен модуль `UsrExampleStandardModule`.

Вызов метода `init()` модуля `UsrExampleStandardModule`




Вызов метода `render()` модуля `UsrExampleStandardModule`




# Создать утилитный модуль

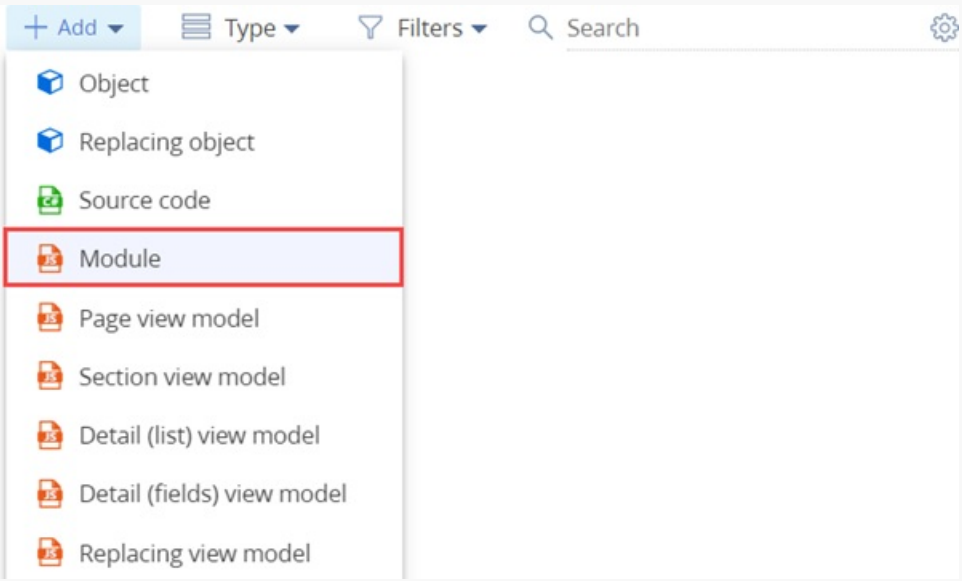
Средний

**Пример.** Создать стандартный модуль, который содержит методы `init()` и `render()`. Каждый метод должен отображать информационное сообщение. При загрузке модуля на клиент ядро сначала вызовет метод `init()`, а затем — метод `render()`, о чем должны оповестить соответствующие информационные сообщения. Метод отображения информационного окна

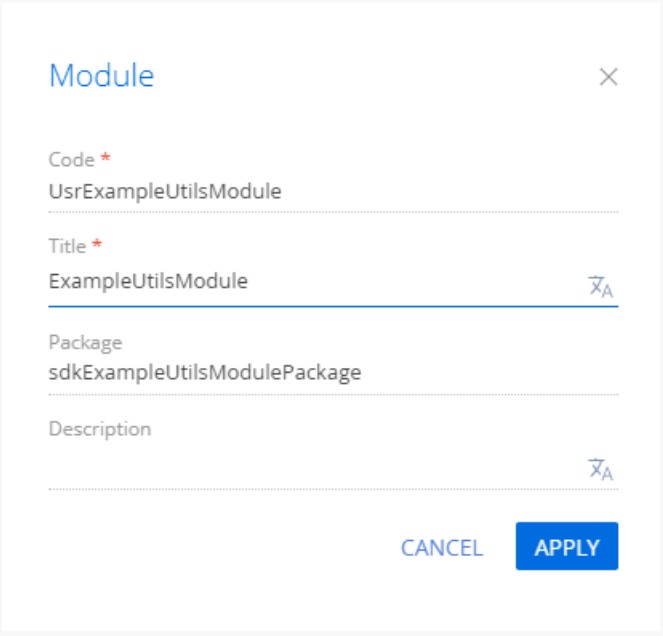
необходимо вынести в отдельный утилитный модуль.

# 1. Создать утилитный 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* ]) — "`UsrExampleUtilsModule`".
   - [ *Заголовок* ] ([ *Title* ]) — "`ExampleUtilsModule`".



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

4. В дизайнере схем добавьте исходный код.

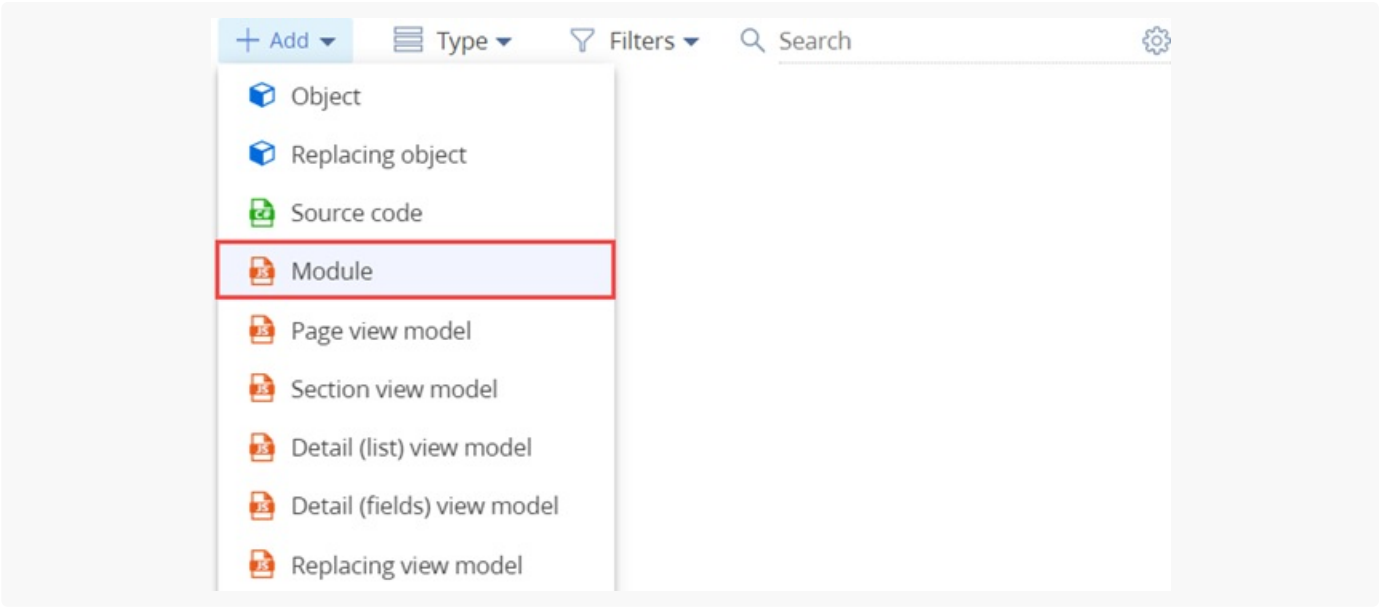
**Исходный код утилитного модуля**

```
// Объявление утилитного модуля. Модуль не имеет зависимостей и содержит один метод
// для отображения информационного сообщения.
define("UsrExampleUtilsModule", [],
    function () {
        return {
            // Метод, который отображает информационное окно с сообщением. Сообщение, выводим
            // передается в метод в качестве аргумента information.
            showInformation: function (information) {
                alert(information);
            }
        };
    });
```

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

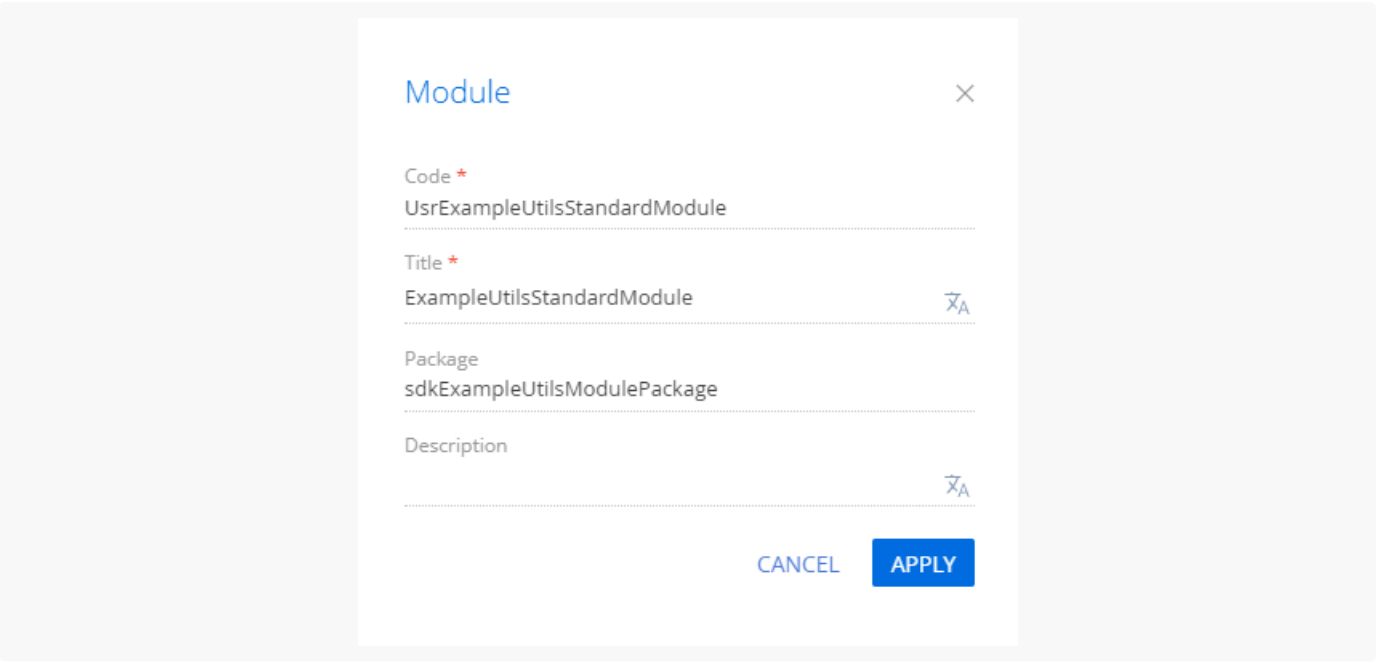
## 2. Создать визуальный модуль

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* ]) — "`UsrExampleUtilsStandardModule`".
- [ *Заголовок* ] ([ *Title* ]) — "`ExampleUtilsStandardModule`".




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

4. В дизайнере схем добавьте исходный код.

**Исходный код модуля**

```
// В модуль импортируется модуль-зависимость UsrExampleUtilsModule для доступа к утилитному м
// Аргумент функции-фабрики – ссылка на загруженный утилитный модуль.
define("UsrExampleUtilsStandardModule", ["UsrExampleUtilsModule"],
    function (UsrExampleUtilsModule) {
        return {
            // В функциях init() и render() вызывается утилитный метод для отображения информ
            // с сообщением, которое передается в качестве аргумента утилитному методу.
            init: function () {
                UsrExampleUtilsModule.showInformation("Calling the init() method of the UsrEx
            },
            render: function (renderTo) {
                UsrExampleUtilsModule.showInformation("Calling the render() method of the Usr
            }
        };
    });
```

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

# 3. Проверить визуальный модуль

Базовая версия Creatio позволяет проверить визуальный модуль, выполнить его загрузку на клиент и визуализацию. Для этого сформируйте адресную строку запроса.

Адресная строка запроса

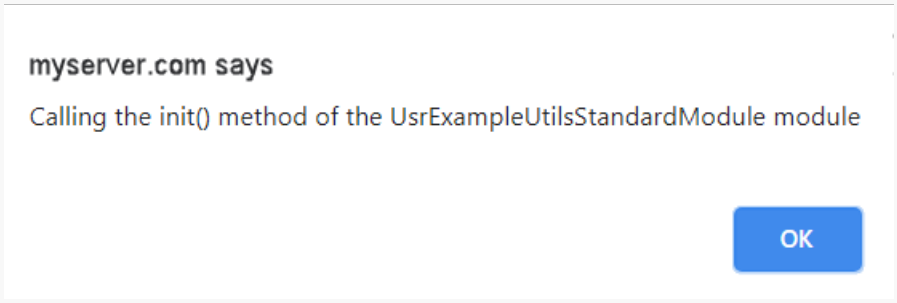
```
[АдресПриложения]/[НомерКонфигурации]/0/NUI/ViewModule.aspx#[ИмяМодуля]
```

Пример адресной строки запроса

```
http://myserver.com/CreatioWebApp/0/NUI/ViewModule.aspx#UsrExampleUtilsStandardModule
```

На клиент будет возвращен модуль `UsrExampleUtilsStandardModule`.

Вызов метода `init()` модуля `UsrExampleUtilsStandardModule`




Вызов метода `render()` модуля `UsrExampleUtilsStandardModule`