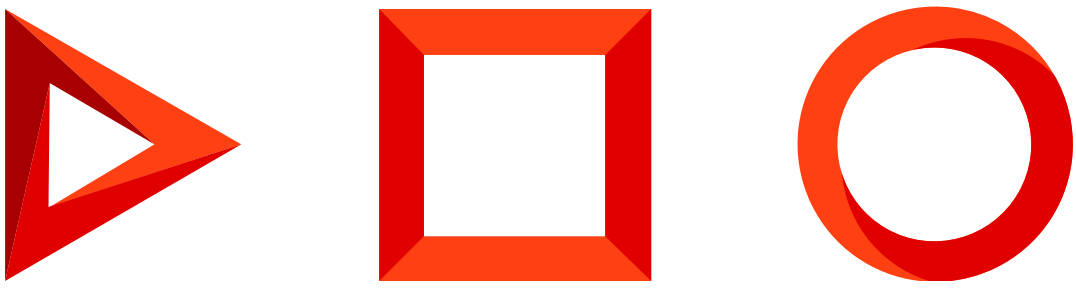
Creatio Academy

# Back-end разработка

## Фабрика замещающих классов

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

# Фабрика замещающих классов

Сложный

Инстанцирование замещающего класса выполняется через **фабрику объектов замещающих классов**. У фабрики запрашивается экземпляр замещаемого типа. В результате возвращается экземпляр соответствующего замещающего типа, который вычисляется фабрикой по дереву зависимостей типов в схемах исходных кодов.

Чтобы **создать замещающий конфигурационный элемент** соответствующего типа, воспользуйтесь инструкцией, которая приведена в статье Разработка конфигурационных элементов.

## Атрибут [Override]

Тип атрибута `[Override]` принадлежит пространству имен `Terrasoft.Core.Factories` и является прямым наследником базового типа `System.Attribute`. Пространство имен `Terrasoft.Core.Factories` описано в Библиотеке .NET классов. Базовый тип `System.Attribute` описан в официальной документации Microsoft.

Атрибут `[Override]` применяется только к классам. **Назначение атрибута** `[Override]` — определение классов, которые должны учитываться при построении дерева зависимостей замещающих и замещаемых типов фабрики.

Ниже рассмотрено применение атрибута `[Override]` к замещающему классу `MySubstituteClass`. `SubstitutableClass` — замещаемый класс.

Шаблон применения атрибута [Override]

```
[Override]
public class ИмяЗамещающегоКласса : ИмяЗамещаемогоКласса
{
    /* Реализация класса. */
}
```

Пример применения атрибута [Override]

```
[Override]
public class MySubstituteClass : SubstitutableClass
{
    /* Реализация класса. */
}
```

# Класс `ClassFactory`

**Назначение** статического класса `ClassFactory` — реализация фабрики по созданию замещающих объектов Creatio. Фабрика использует open-source фреймворк внедрения зависимостей Ninject. Статический класс `ClassFactory` описан в Библиотеке .NET классов.

**Инициализация фабрики** осуществляется в момент первого обращения к ней, то есть при первой попытке получить экземпляр замещающего типа. При инициализации фабрика собирает информацию обо всех замещаемых типах конфигурации.

**Алгоритм работы фабрики**:

1. Поиск замещающих типов. Выполняется фабрикой путем анализа типов конфигурационной сборки. Класс, помеченный атрибутом `[Override]`, интерпретируется фабрикой как замещающий, а родитель этого класса — как замещаемый.
2. Формирование дерева зависимостей типов в виде списка пар значений `Замещаемый тип –> Замещающий тип`. При этом в дереве иерархии замещения не учитываются промежуточные типы.
3. Замещение исходного класса последним наследником в иерархии замещения.
4. Выполнение привязки типов замещения в соответствии с построенным деревом зависимостей типов. Используется фреймворк Ninject.

Рассмотрим работу фабрики на примере иерархии классов, которая приведена ниже.

**Пример иерархии классов**

```
/* Исходный класс. */
public class ClassA { }

/* Класс, который замещает ClassA. */
[Override]
public class ClassB : ClassA { }

/* Класс, который замещает ClassB. */
[Override]
public class ClassC : ClassB { }
```

По иерархии классов будет построено дерево зависимостей, которое приведено ниже.

**Пример дерева зависимостей**

```
ClassA –> ClassC
ClassB –> ClassC
```

При построении дерева зависимостей не будут учтены промежуточные типы.

**Пример иерархии замещения типов**

```
ClassA –> ClassB –> ClassC
```

Здесь `ClassA` замещается типом `ClassC`, а не промежуточным типом `ClassB`. Это связано с тем, что `ClassC` — последний наследник в иерархии замещения. Таким образом, при запросе экземпляра типа `ClassA` или `ClassB` фабрика возвращает экземпляр `ClassC`.

## Экземпляр замещаемого типа

Чтобы **получить экземпляр замещаемого типа**, необходимо использовать публичный статический параметризованный метод `Get<T>`. Этот метод предоставляет фабрика `ClassFactory`. В качестве обобщенного параметра метода выступает замещаемый тип. Метод `Get<T>` описан в Библиотеке .NET классов.

**Пример получения экземпляра замещаемого типа**

```
var substituteObject = ClassFactory.Get<SubstitutableClass>();
```

В результате будет создан экземпляр класса `MySubstituteClass`. Нет необходимости явно указывать тип создаваемого экземпляра, поскольку, благодаря предварительной инициализации, фабрика определяет, какой именно тип замещает запрашиваемый тип и создает соответствующий ему экземпляр.

В качестве параметров метод `Get<T>` может принимать массив объектов `ConstructorArgument`. Каждый объект массива представляет собой аргумент конструктора класса, который создан с помощью фабрики. Таким образом, фабрика позволяет инстанцировать замещающие объекты с параметризованными конструкторами. При этом ядро фабрики самостоятельно разрешает все зависимости, необходимые для создания или работы объекта.

Рекомендуется, чтобы конструкторы замещающего класса имели сигнатуру, соответствующую сигнатуре замещаемого класса. Если логика реализации замещающего класса требует объявления конструктора с пользовательской сигнатурой, необходимо соблюдать правила, которые приведены ниже.

**Правила создания и вызова конструкторов замещаемого и замещающего классов**:

- Если замещаемый класс **не имеет явно реализованного параметризованного конструктора** (имеет только конструктор по умолчанию), то в замещающем классе допускается явная реализация своего конструктора без каких-либо ограничений. При этом соблюдается стандартный порядок вызовов конструкторов родительского (замещаемого) и дочернего (замещающего) классов. В этом случае необходимо помнить, что при инстанцировании замещаемого класса через фабрику ей необходимо передавать корректные параметры для инициализации свойств замещающего класса.
- Если замещаемый класс **имеет параметризованный конструктор**, то в замещающем классе необходимо реализовать конструктор. Конструктор замещающего класса должен явно вызывать параметризованный конструктор родительского (замещаемого) класса, которому передаются параметры для корректной инициализации родительских свойств. При этом конструктор

замещающего класса может выполнять инициализацию своих свойств либо оставаться пустым.

Несоблюдение правил приводит к ошибке времени выполнения. Ответственность за корректность инициализации свойств замещающего и замещаемого классов лежит на разработчике.

# Примеры работы с замещающими классами

 Сложный

## Пример 1

Замещаемый класс `SubstitutableClass` имеет один конструктор по умолчанию.

**Замещаемый класс** `SubstitutableClass`

```
/* Объявление замещаемого класса. */
public class SubstitutableClass
{
    /* Свойство класса, инициализация которого будет выполняться в конструкторе. */
    public int OriginalValue { get; private set; }

    /* Конструктор по умолчанию, который инициализирует свойство OriginalValue значением 10. */
    public SubstitutableClass()
    {
        OriginalValue = 10;
    }

    /* Метод, который возвращает значение OriginalValue, умноженное на 2. Этот метод может быть
    public virtual int GetMultipliedValue()
    {
        return OriginalValue * 2;
    }
}
```

В замещающем классе `SubstituteClass` объявлены два конструктора — конструктор по умолчанию и параметризированный. Замещающий класс переопределяет родительский метод `GetMultipliedValue()`.

**Замещающий класс** `SubstituteClass`

```
/* Объявление класса, замещающего SubstitutableClass. */
[Terrasoft.Core.Factories.Override]
public class SubstituteClass : SubstitutableClass
{
    /* Свойство класса SubstituteClass. */
```

```
    public int AdditionalValue { get; private set; }

    /* Конструктор по умолчанию, который инициализирует свойство AdditionalValue значением 15. З
    public SubstituteClass()
    {
        AdditionalValue = 15;
    }

    /* Конструктор с параметром, который инициализирует свойство AdditionalValue значением, пере
    public SubstituteClass(int paramValue)
    {
        AdditionalValue = paramValue;
    }

    /* Замещение родительского метода. Метод будет возвращать значение AdditionalValue, умноженн
    public override int GetMultipliedValue()
    {
        return AdditionalValue * 3;
    }
}
```

**Пример.** Инстанцировать и вызвать метод `GetMultipliedValue()` замещающего класса `SubstituteClass` с конструктором по умолчанию и параметризированным конструктором.

Примеры получения экземпляра замещающего класса `SubstituteClass` через фабрику представлены ниже.

**Примеры получения экземпляра замещающего класса через фабрику**

```
/* Получение экземпляра класса, который замещает SubstitutableClass. Фабрика вернет экземпляр Su
var substituteObject = ClassFactory.Get<SubstitutableClass>();

/* Переменная будет содержать значение 10. Свойство OriginalValue инициализировано родительским
var originalValue = substituteObject.OriginalValue;

/* Здесь будет вызван метод замещающего класса, который будет возвращать значение AdditionalValu
var additionalValue = substituteObject.GetMultipliedValue();

/* Получение экземпляра замещающего класса, который инициализирован параметризированным конструк
var substituteObjectWithParameters = ClassFactory.Get<SubstitutableClass>(
    new ConstructorArgument("paramValue", 20));

/* Переменная будет содержать значение 10. */
var originalValueParametrized = substituteObjectWithParameters.OriginalValue;

/* Переменная будет содержать значение 60, так как свойство AdditionalValue инициализировано зна
```

```
var additionalValueParametrized = substituteObjectWithParameters.GetMultipliedValue();
```

# Пример 2

Замещаемый класс `SubstitutableClass` имеет один параметризированный конструктор.

**Замещаемый класс `SubstitutableClass`**

```
/* Объявление замещаемого класса. */
public class SubstitutableClass
{
    /* Свойство класса, инициализация которого будет выполняться в конструкторе. */
    public int OriginalValue { get; private set; }

    /* Параметризированный конструктор, который инициализирует свойство OriginalValue переданным
    public SubstitutableClass(int originalParamValue)
    {
        OriginalValue = originalParamValue;
    }

    /* Метод, который возвращает значение OriginalValue, умноженное на 2. Этот метод может быть
    public virtual int GetMultipliedValue()
    {
        return OriginalValue * 2;
    }
}
```

Замещающий класс `SubstituteClass` также имеет один параметризированный конструктор.

**Замещающий класс `SubstituteClass`**

```
/* Объявление класса, замещающего SubstitutableClass. */
[Terrasoft.Core.Factories.Override]
public class SubstituteClass : SubstitutableClass
{
    /* Свойство класса SubstituteClass. */
    public int AdditionalValue { get; private set; }

    /* Конструктор с параметром, который инициализирует свойство AdditionalValue значением, пере
    public SubstituteClass(int paramValue) : base(paramValue + 8)
    {
        AdditionalValue = paramValue;
    }

    /* Замещение родительского метода. Метод будет возвращать значение AdditionalValue, умноженн
```

```
    public override int GetMultipliedValue()
    {
        return AdditionalValue * 3;
    }
}
```

**Пример.** Создать и использовать экземпляр замещающего класса `SubstituteClass`.

Пример создания и использования экземпляра замещающего класса `SubstituteClass` через фабрику представлен ниже.

**Пример создания и использования экземпляра замещающего класса через фабрику**

```
/* Получение экземпляра замещающего класса, который инициализирован параметризированным конструк
var substituteObjectWithParameters = ClassFactory.Get<SubstitutableClass>(
    new ConstructorArgument("paramValue", 10));

/* Переменная будет содержать значение 18. */
var originalValueParametrized = substituteObjectWithParameters.OriginalValue;

/* Переменная будет содержать значение 30. */
var additionalValueParametrized = substituteObjectWithParameters.GetMultipliedValue();
```

# Пример 3

Замещаемый класс `SubstitutableClass` имеет один параметризированный конструктор.

**Замещаемый класс `SubstitutableClass`**

```
/* Объявление замещаемого класса. */
public class SubstitutableClass
{
    /* Свойство класса, инициализация которого будет выполняться в конструкторе. */
    public int OriginalValue { get; private set; }

    /* Параметризированный конструктор, который инициализирует свойство OriginalValue переданным
    public SubstitutableClass(int originalParamValue)
    {
        OriginalValue = originalParamValue;
    }

    /* Метод, который возвращает значение OriginalValue, умноженное на 2. Этот метод может быть
    public virtual int GetMultipliedValue()
```

```
    {
        return OriginalValue * 2;
    }
}
```

**Пример.** В замещающем классе `SubstituteClass` переопределить родительский метод `GetMultipliedValue()`.

В замещающем классе `SubstituteClass` переопределяется метод `GetMultipliedValue()`, который будет возвращать фиксированное значение. Класс `SubstituteClass` не требует первичной инициализации своих свойств, в нем должен быть явно объявлен конструктор, который вызывает родительский конструктор с параметрами для корректной инициализации родительских свойств.

**Замещающий класс `SubstituteClass`**

```
// Объявление класса, замещающего SubstitutableClass.
[Terrasoft.Core.Factories.Override]
public class SubstituteClass : SubstitutableClass
{
    /* Пустой конструктор по умолчанию, который явно вызывает конструктор родительского класса д
    public SubstituteClass() : base(0)
    {
    }

    /* Также можно использовать пустой конструктор с параметрами для того, чтобы передать эти па
    public SubstituteClass(int someValue) : base(someValue)
    {
    }

    /* Замещение родительского метода. Метод будет возвращать фиксированное значение. */
    public override int GetMultipliedValue()
    {
        return 111;
    }
}
```

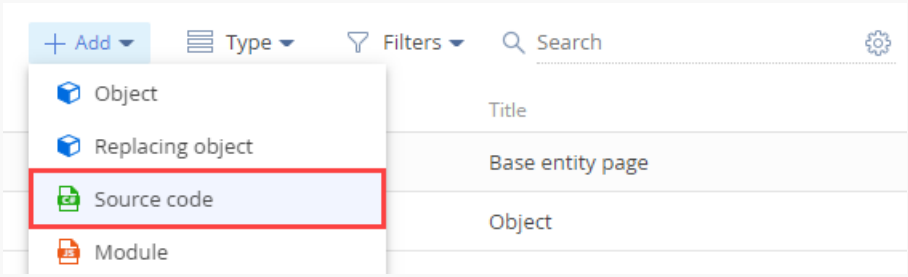
# Создать замещающий класс

Сложный

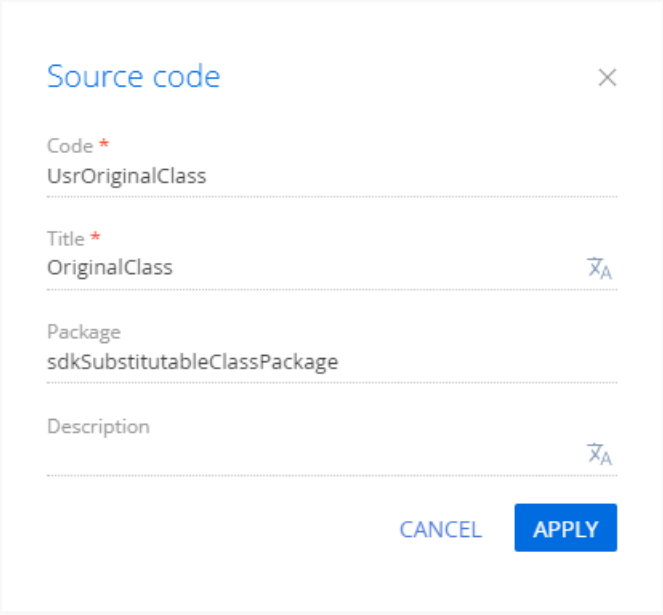
**Пример.** В пользовательском пакете создать замещаемый класс и пользовательский веб-сервис с аутентификацией на основе cookies. В другом пользовательском пакете создать замещающий класс. Вызвать пользовательский веб-сервис без замещения классов и с замещением классов.

# 1. Реализовать замещаемый класс

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




3. В дизайнере схем заполните свойства схемы:
   - [ *Код* ] ([ *Code* ]) — "`UsrOriginalClass`".
   - [ *Заголовок* ] ([ *Title* ]) — "`OriginalClass`".




4. Создайте замещаемый класс `UsrOriginalClass`, который содержит виртуальный метод `GetAmount(int, int)`. Метод выполняет суммирование двух значений, переданных в качестве параметров.

**`UsrOriginalClass`**

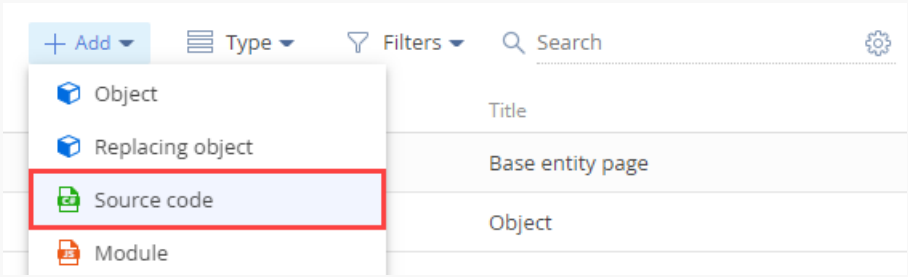
```
namespace Terrasoft.Configuration
```

```
{
    public class UsrOriginalClass
    {
        /* GetAmount() – виртуальный метод, который имеет свою реализацию и может быть переоп
        public virtual int GetAmount(int originalValue1, int originalValue2)
        {
            return originalValue1 + originalValue2;
        }
    }
}
```

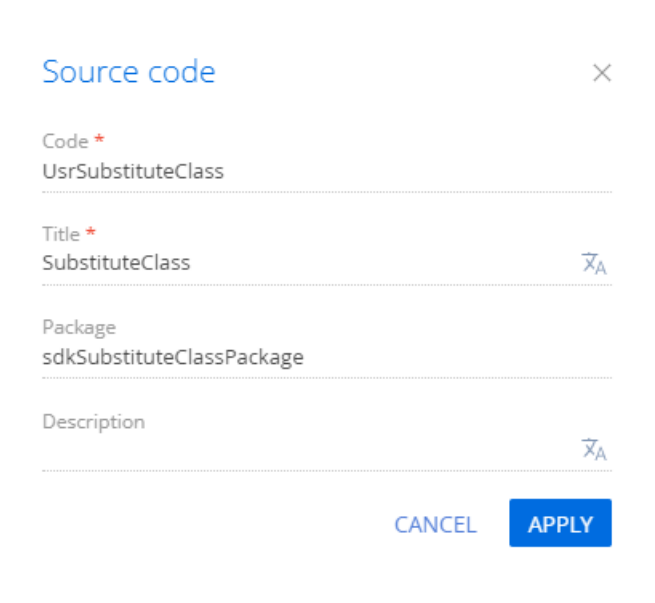
5. На панели инструментов дизайнера нажмите [ *Сохранить* ] ([ *Save* ]), а затем [ *Опубликовать* ] ([ *Publish* ]).

## 2. Реализовать замещающий класс

1. Перейдите в раздел [ *Конфигурация* ] ([ *Configuration* ]) и выберите пользовательский пакет, в который будет добавлена схема. Для замещающего класса используйте пакет, отличный от пакета, в котором реализован замещаемый класс `UsrOriginalClass`.
2. В зависимость пользовательского пакета с замещающим классом добавьте пользовательский пакет с замещаемым классом `UsrOriginalClass`, созданный на предыдущем шаге.
3. На панели инструментов реестра раздела нажмите [ *Добавить* ] —> [ *Исходный код* ] ([ *Add* ] —> [ *Source code* ]).




4. В дизайнере схем заполните свойства схемы:
   - [ *Код* ] ([ *Code* ]) — "`UsrSubstituteClass`".
   - [ *Заголовок* ] ([ *Title* ]) — "`SubstituteClass`".




5. Создайте замещающий класс `UsrSubstituteClass`, который содержит метод `GetAmount(int, int)`. Метод выполнит суммирование двух значений, переданных в качестве параметров, и умножит полученную сумму на значение, переданное в свойстве `Rate`. Первичная инициализация свойства `Rate` будет выполняться в конструкторе замещающего класса.

**UsrSubstituteClass**

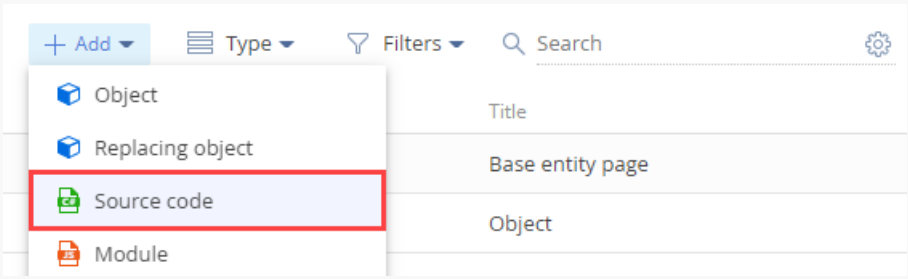
```
namespace Terrasoft.Configuration
{
    [Terrasoft.Core.Factories.Override]
    public class UsrSubstituteClass : UsrOriginalClass
    {
        /* Коэффициент. Значение свойства задается внутри класса. */
        public int Rate { get; private set; }

        /* В конструкторе выполняется первичная инициализация свойства Rate переданным значен
        public UsrSubstituteClass(int rateValue)
        {
            Rate = rateValue;
        }

        /* Замещение родительского метода пользовательской реализацией. */
        public override int GetAmount(int substValue1, int substValue2)
        {
            return (substValue1 + substValue2) * Rate;
        }
    }
}
```

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

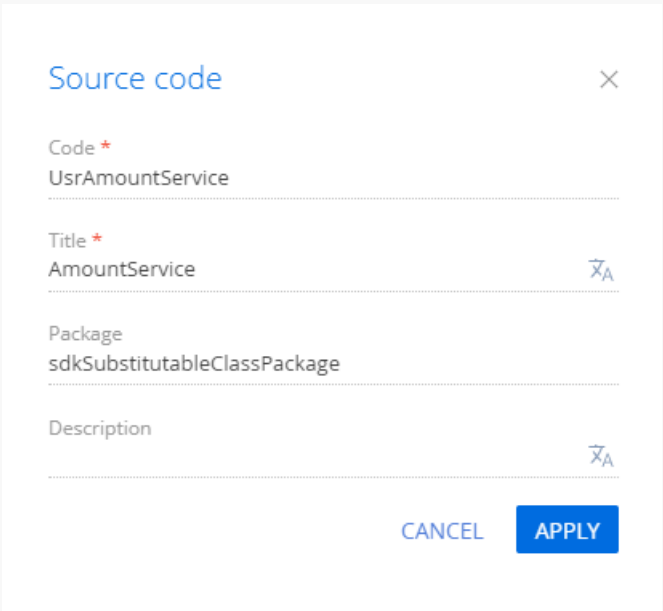
]).

# 3. Реализовать пользовательский веб-сервис

1. Перейдите в раздел [ *Конфигурация* ] ([ *Configuration* ]) и выберите пользовательский пакет, в который будет добавлена схема. Для пользовательского веб-сервиса используйте пакет, в котором реализован замещаемый класс `UsrOriginalClass`.
2. На панели инструментов реестра раздела нажмите [ *Добавить* ] —> [ *Исходный код* ] ([ *Add* ] —> [ *Source code* ]).




3. В дизайнере схем заполните свойства схемы:
   - [ *Код* ] ([ *Code* ]) — "`UsrAmountService`".
   - [ *Заголовок* ] ([ *Title* ]) — "`AmountService`".




4. Создайте класс сервиса.
   a. В дизайнере схем добавьте пространство имен `Terrasoft.Configuration`.
   b. С помощью директивы `using` добавьте пространства имен, типы данных которых будут задействованы в классе.
   c. Добавьте название класса `UsrAmountService`, которое соответствует названию схемы (свойство

[ *Код* ] ([ *Code* ])).

d. В качестве родительского класса укажите класс `Terrasoft.Nui.ServiceModel.WebService.BaseService`.

e. Для класса добавьте атрибуты `[ServiceContract]` и `[AspNetCompatibilityRequirements(RequirementsMode = AspNetCompatibilityRequirementsMode.Required)]`.

5. Реализуйте метод класса.

В дизайнере схем добавьте в класс метод `public string GetAmount(int value1, int value2)`, который реализует конечную точку пользовательского веб-сервиса. Метод `GetAmount(int, int)` выполнит суммирование двух значений, переданных в качестве параметров.

Исходный код пользовательского веб-сервиса `UsrAmountService` представлен ниже.

**UsrAmountService**

```
namespace Terrasoft.Configuration
{
    using System.ServiceModel;
    using System.ServiceModel.Activation;
    using System.ServiceModel.Web;
    using Terrasoft.Core;
    using Terrasoft.Web.Common;

    [ServiceContract]
    [AspNetCompatibilityRequirements(RequirementsMode = AspNetCompatibilityRequirementsMode.R
    public class UsrAmountService : BaseService
    {

        [OperationContract]
        [WebGet(RequestFormat = WebMessageFormat.Json, BodyStyle = WebMessageBodyStyle.Wrappe
        public string GetAmount(int value1, int value2) {
            /*
            // Создание экземпляра исходного класса через фабрику классов.
            var originalObject = Terrasoft.Core.Factories.ClassFactory.Get<UsrOriginalClass>(

            // Получение результата работы метода GetAmount(). В качестве параметров передают
            int result = originalObject.GetAmount(value1, value2);

            // Отображение на странице результата выполнения.
            return string.Format("The result value, retrieved after calling the replacement c
            */

            /*
            // Создание экземпляра замещающего класса через фабрику замещаемых объектов.
            // В качестве параметра метода фабрики передается экземпляр аргумента конструктор
            var substObject = Terrasoft.Core.Factories.ClassFactory.Get<UsrOriginalClass>(new

            // Получение результата работы метода GetAmount(). В качестве параметров передают
```

```
            int result = substObject.GetAmount(value1, value2);

            // Отображение на странице результата выполнения.
            return string.Format("The result value, retrieved after calling the replaceable c
            */

            /* Создание экземпляра замещающего класса через оператор new(). */
            var substObjectByNew = new UsrOriginalClass();

            /* Создание экземпляра замещающего класса через фабрику замещаемых объектов. */
            var substObjectByFactory = Terrasoft.Core.Factories.ClassFactory.Get<UsrOriginalC

            /* Получение результата работы метода GetAmount(). Будет вызван метод исходного к
            int resultByNew = substObjectByNew.GetAmount(value1, value2);

            /* Получение результата работы метода GetAmount(). Будет вызван метод класса UsrS
            int resultByFactory = substObjectByFactory.GetAmount(value1, value2);

            /* Отображение на странице результата выполнения. */
            return string.Format("Result without class replacement: {0}; Result with class re
        }
    }
}
```

В коде приведены примеры создания экземпляра замещающего класса и через фабрику замещаемых объектов, и через оператор `new()`.

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

В результате в Creatio появится пользовательский веб-сервис `UsrAmountService` типа REST с конечной точкой `GetAmount`.

## Результат выполнения примера

Чтобы вызвать пользовательский веб-сервис, из браузера обратитесь к конечной точке `GetAmount` веб-сервиса `UsrAmountService` и в качестве параметров `value1` и `value2` передайте 2 произвольных числа.

**Строка запроса**

```
http://mycreatio.com/0/rest/UsrAmountService/GetAmount?value1=25&value2=125
```

В свойстве `GetAmountResult` будет возвращен результат работы пользовательского веб-сервиса без замещения классов и с замещением классов.