

Back-end разработка

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

Фоновое выполнение операций	10
Зарегистрировать фоновую операцию	11
1. Создать класс для объекта активности	11
2. Создать класс для добавления активности	12
3. Создать бизнес-процесс для запуска фоновой операции	14
Результат выполнения примера	17
Фабрика замещающих классов	17
Атрибут [Override]	18
Класс ClassFactory	18
Экземпляр замещаемого типа	19
Примеры работы с замещающими классами	20
Пример 1	20
Пример 2	22
Пример 3	24
Создать замещающий класс	25
1. Реализовать замещаемый класс	25
2. Реализовать замещающий класс	27
3. Реализовать пользовательский веб-сервис	28
Результат выполнения примера	31
Понятие локализуемых ресурсов	31
Отобразить локализуемые ресурсы	32
Хранить локализуемые ресурсы	35
Установить привязанные локализуемые ресурсы	38
Прямой доступ к данным	39
Получить данные из базы данных	39
Добавить данные в базу данных	40
Изменить данные в базе данных	42
Удалить данные из базы данных	43
Использовать многопоточность при работе с базой данных	43
Получить данные из базы данных	45
Пример 1	45
Пример 2	46
Пример 3	47
Пример 4	48
Пример 5	48
Пример 6	49

Пример 7	50
Добавить данные в базу данных	50
Пример 1	50
Пример 2	51
Добавить данные в базу данных с помощью подзапросов	51
Пример 1	52
Пример 2	52
Изменить данные в базе данных	53
Пример 1	53
Пример 2	53
Удалить данные из базы данных	54
Пример	54
Класс Select	55
Конструкторы	55
Свойства	56
Методы	57
Класс Insert	67
Конструкторы	67
Свойства	68
Методы	68
Класс InsertSelect	70
Конструкторы	71
Свойства	71
Методы	72
Класс Update	73
Конструкторы	74
Свойства	74
Методы	75
Класс UpdateSelect	78
Конструкторы	78
Свойства	78
Методы	78
Класс Delete	79
Конструкторы	79
Свойства	79
Методы	80
Класс QueryFunction	82
Класс QueryFunction	83
Класс AggregationQueryFunction	86

Класс IsNullQueryFunction	89
Класс CreateGuidQueryFunction	91
Класс CurrentDateTimeQueryFunction	91
Класс CoalesceQueryFunction	92
Класс DatePartQueryFunction	94
Класс DateAddQueryFunction	96
Класс DateDiffQueryFunction	98
Класс CastQueryFunction	100
Класс UpperQueryFunction	101
Класс CustomQueryFunction	103
Класс DataLengthQueryFunction	105
Класс TrimQueryFunction	107
Класс LengthQueryFunction	108
Класс SubstringQueryFunction	110
Класс ConcatQueryFunction	111
Класс WindowQueryFunction	113
Операции с локализуемыми ресурсами	115
Использовать Creatio IDE для выполнения операций с локализуемыми ресурсами	115
Использовать базу данных для выполнения операций с локализуемыми ресурсами	118
Использовать систему контроля версий SVN для выполнения операций с локализуемыми ресурсами	124
Использовать файловую систему для выполнения операций с локализуемыми ресурсами	126
Получить локализуемые ресурсы из базы данных	127
Пример 1	127
Пример 2	128
Добавить локализуемую колонку	130
1. Создать объект	130
2. Добавить локализуемую колонку	131
Результат выполнения примера	131
Локализовать представление в базе данных	132
1. Создать схему объекта для представления	133
2. Добавить колонки	134
3. Создать представления в базе данных	135
Результат выполнения примера	136
Класс LocalizableValue<T>	140
Свойства	141
Методы	141
Доступ к данным через ORM	142
Сформировать путь к колонке относительно корневой схемы	143
Получить данные из базы данных	144

Управлять сущностью базы данных	148
Управлять сущностями базы данных	148
Пример 1	148
Пример 2	149
Пример 3	149
Пример 4	150
Пример 5	150
Пример 6	151
Пример 7	151
Пример 8	151
Получить данные из базы данных с учетом прав пользователя	152
Пример 1	152
Пример 2	153
Пример 3	153
Пример 4	154
Пример 5	155
Класс EntitySchemaQuery	156
Конструкторы	156
Свойства	156
Методы	160
Класс Entity	171
Конструкторы	171
Свойства	171
Методы	175
События	186
Класс EntityMapper	189
Класс EntityMapper	190
Класс EntityResult	191
Класс MapConfig	191
Класс DetailMapConfig	192
Класс RelationEntityMapConfig	193
Класс EntityFilterMap	193
Класс EntitySchemaQueryFunction	194
Класс EntitySchemaQueryFunction	195
Класс EntitySchemaAggregationQueryFunction	196
Класс EntitySchemasNullQueryFunction	200
Класс EntitySchemaCoalesceQueryFunction	202
Класс EntitySchemaCaseNotNullQueryFunctionWhenItem	203
Класс EntitySchemaCaseNotNullQueryFunctionWhenItems	204

Класс EntitySchemaCaseNotNullQueryFunction	205
Класс EntitySchemaSystemValueQueryFunction	206
Класс EntitySchemaCurrentDateTimeQueryFunction	207
Класс EntitySchemaBaseCurrentDateQueryFunction	208
Класс EntitySchemaCurrentDateQueryFunction	208
Класс EntitySchemaDateToCurrentYearQueryFunction	209
Класс EntitySchemaStartOfCurrentWeekQueryFunction	210
Класс EntitySchemaStartOfCurrentMonthQueryFunction	212
Класс EntitySchemaStartOfCurrentQuarterQueryFunction	213
Класс EntitySchemaStartOfCurrentHalfYearQueryFunction	214
Класс EntitySchemaStartOfCurrentYearQueryFunction	215
Класс EntitySchemaBaseCurrentDateTimeQueryFunction	216
Класс EntitySchemaStartOfCurrentHourQueryFunction	216
Класс EntitySchemaCurrentTimeQueryFunction	218
Класс EntitySchemaCurrentUserQueryFunction	219
Класс EntitySchemaCurrentUserContactQueryFunction	220
Класс EntitySchemaCurrentUserAccountQueryFunction	221
Класс EntitySchemaDatePartQueryFunction	222
Класс EntitySchemaUpperQueryFunction	224
Класс EntitySchemaCastQueryFunction	226
Класс EntitySchemaTrimQueryFunction	227
Класс EntitySchemaLengthQueryFunction	228
Класс EntitySchemaConcatQueryFunction	230
Класс EntitySchemaWindowQueryFunction	231
Класс EntitySchemaQueryOptions	233
Конструкторы	233
Свойства	233
Пользовательские веб-сервисы	234
Разработать пользовательский веб-сервис	235
Вызвать пользовательский веб-сервис	240
Перенести существующий пользовательский веб-сервис на платформу .NET Core	241
Разработать пользовательский веб-сервис с аутентификацией на основе cookies	243
1. Создать схему Исходный код	243
2. Создать класс сервиса	244
3. Реализовать метод класса	245
Результат выполнения примера	246
Разработать пользовательский веб-сервис с анонимной аутентификацией	247
1. Создать схему Исходный код	247
2. Создать класс сервиса	248

3. Реализовать метод класса	249
4. Зарегистрировать пользовательский веб-сервис с анонимной аутентификацией	250
5. Настроить пользовательский веб-сервис с анонимной аутентификацией для работы по протоколам http и https	251
6. Настроить доступ к пользовательскому веб-сервису с анонимной аутентификацией для всех пользователей	251
7. Перезапустить приложение в IIS	252
Результат выполнения примера	252
Вызвать пользовательский веб-сервис из front-end части	253
1. Создать пользовательский веб-сервис	253
2. Создать замещающую страницу записи контакта	254
3. Добавить кнопку на страницу записи контакта	255
Результат выполнения примера	257
Вызвать пользовательский веб-сервис с помощью Postman	258
1. Создать коллекцию запросов	258
2. Настроить аутентификационный запрос	259
3. Выполнить аутентификационный запрос	262
4. Настроить запрос к пользовательскому веб-сервису с аутентификацией на основе cookies	263
5. Выполнить запрос к пользовательскому веб-сервису с аутентификацией на основе cookies	265
Результат выполнения примера	265
Сложные Select-запросы	266
Отдельный пул запросов	266
Read-only реплика	269
Выполнить сложный Select-запрос	269
Бизнес-логика объектов	270
Механизм событийного слоя Entity	270
Асинхронность в событийном слое Entity	273
Хранилища данных и кэш	275
Хранилище данных	276
Хранилище кэша	277
Объектная модель хранилищ	277
Доступ к хранилищам данных и кэшам из UserConnection	278
Использование кэша в EntitySchemaQuery	279
Прокси-классы хранилища данных и кэша	280
Особенности использования хранилищ данных и кэша	284
Копирование иерархических данных	286
Структура и алгоритм работы копирования иерархических данных	286
Кастомизировать копирование иерархических данных	289
Вызвать копирование иерархических данных	291

Автоматическая привязка данных	293
Структура автоматической привязки данных	293
Алгоритм работы автоматической привязки данных	294
Добавить пользовательскую структуру для привязки объекта	296
Вызвать автоматическую привязку данных	296
API для работы с файлами	298
Местоположение файла и файловые локаторы	298
Файлы и файловые хранилища	298
Получить файл (IFileFactory)	299
Реализовать и зарегистрировать новый тип файлового хранилища	299
Исключения при работе с файлами	300
Настройка активного хранилища	301
Примеры работы с файлами	301
Пример реализации файлового хранилища контента	304
Интерфейс IFile	306
Свойства	306
Методы	307
Интерфейс IFileContentStorage	308
Методы	308
Интерфейс IFileFactory	309
Свойства	309
Методы	309
Класс EntityFileLocator	309
Конструкторы	310
Свойства	310
Класс EntityFileMetadata	310
Конструкторы	310
Свойства	311
Методы	311
Класс FileFactoryUtils	311
Методы	312
Класс FileMetadata	312
Свойства	313
Методы	313
Класс FileUtils	314
Методы	314
Библиотека .NET классов	315

Фоновое выполнение операций



Средний

Фоновое выполнение операций позволяет в фоновом режиме выполнять операции, которые требуют длительного времени выполнения, без задержек в работе пользовательского интерфейса.

Для запуска фоновых операций в классе `Terrasoft.Core.Tasks.Task` реализованы методы `StartNew()` и `StartNewWithUserConnection()`. В качестве параметров методов можно использовать базовые типы данных в .NET (например, `string`, `int`, `Guid` и т. д.) или пользовательские типы. **Отличие** метода `StartNewWithUserConnection()` — запуск фоновой операции, которая требует использования пользовательского соединения `UserConnection`.

Параметры, которые принимаются фоновой операцией, преобразуются в массив байт с помощью модуля `MessagePack-CSharp`. Реализация модуля `MessagePack-CSharp` содержится на [сайте GitHub](#). Если не удается сериализовать или десериализовать значение параметра, могут генерироваться исключения.

Важно. В фоновой операции не рекомендуется использовать бесконечные циклы, поскольку это приводит к невозможности запуска в приложении других задач.

Действие асинхронной операции описывается в отдельном классе, который должен реализовать интерфейс `IBackgroundTask<in TParameters>`.

Интерфейс `IBackgroundTask<in TParameters>`

```
namespace Terrasoft.Core.Tasks
{
    public interface IBackgroundTask<in TParameters>
    {
        void Run(TParameters parameters);
    }
}
```

Если для выполнения действия требуется пользовательское соединение, то класс должен дополнительно реализовать интерфейс `IUserConnectionRequired`.

Интерфейс `IUserConnectionRequired`

```
namespace Terrasoft.Core
{
    public interface IUserConnectionRequired
    {
        void SetUserConnection(UserConnection userConnection);
    }
}
```

```

    }
}

```

В классе, который реализует интерфейсы `IBackgroundTask<in TParameters>` и `IUserConnectionRequired` необходимо реализовать методы интерфейсов `Run` и `SetUserConnection`.

При **реализации методов** следует учесть:

- В метод `Run` не следует передавать `UserConnection`.
- В методе `Run` не следует вызывать метод `SetUserConnection` — ядро системы вызывает этот метод и инициализирует `UserConnection` при старте фоновой операции.
- В метод `Run` допустимо передавать структуры, состоящие только из простых типов данных. Если передавать сложные экземпляры классов, то с большой вероятностью произойдет ошибка сериализации параметров.

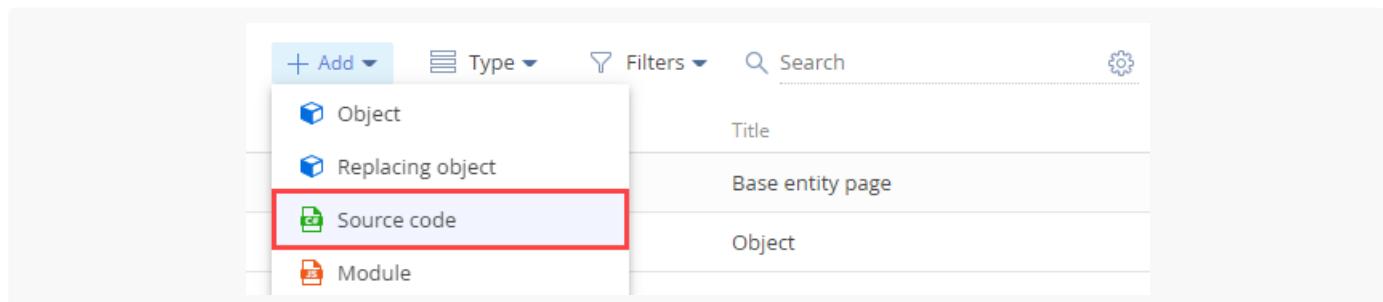
Зарегистрировать фоновую операцию

Средний

Пример. Создать бизнес-процесс, который регистрирует фоновую операцию. Фоновая операция выполняется около 30 секунд. По истечении этого времени в списочном представлении реестра раздела [Активности] ([Activities]) добавляется запись [*Activity created by background task*].

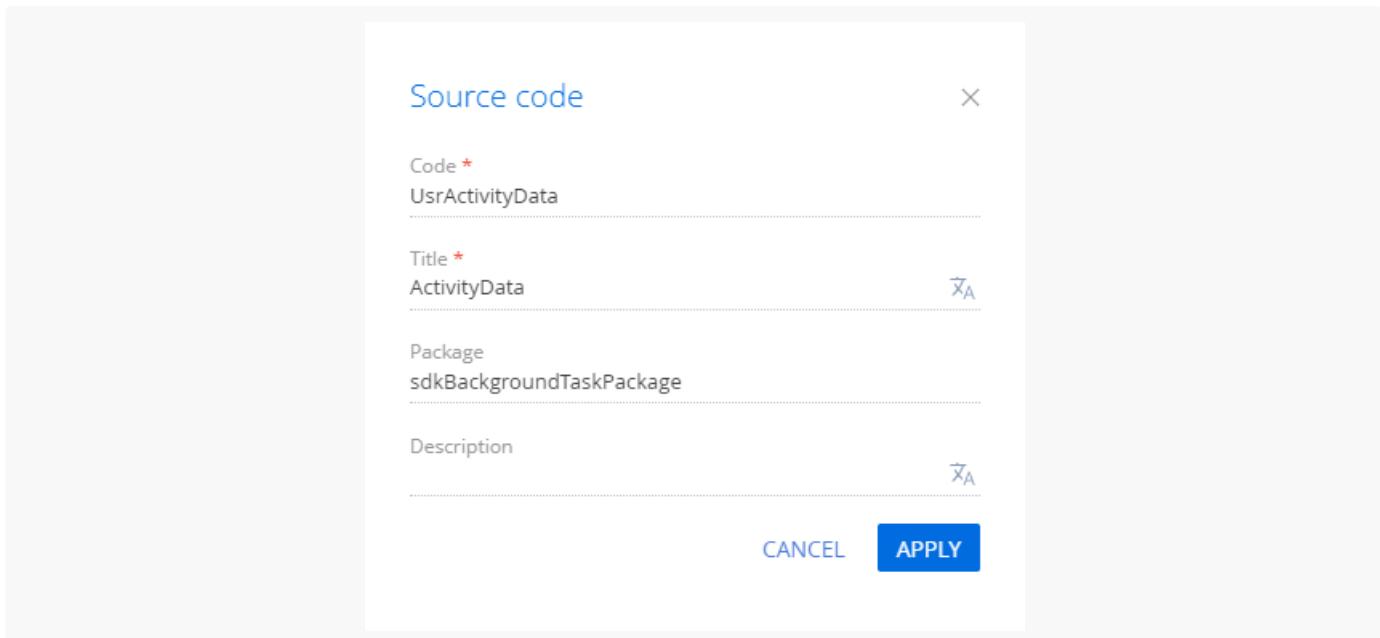
1. Создать класс для объекта активности

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]) — "UsrActivityData".
- [Заголовок] ([Title]) — "ActivityData".



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

4. В дизайнере схем добавьте исходный код.

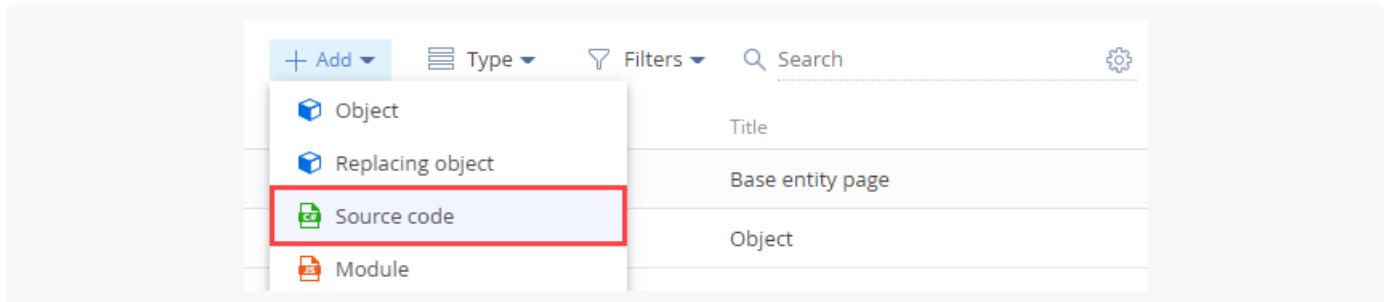
```
UsrActivityData

namespace Terrasoft.Configuration
{
    using System;
    using Terrasoft.Common;
    using Terrasoft.Core;
    using Terrasoft.Core.DB;
    public class UsrActivityData
    {
        public string Title { get; set; }
        public Guid TypeId { get; set; }
    }
}
```

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

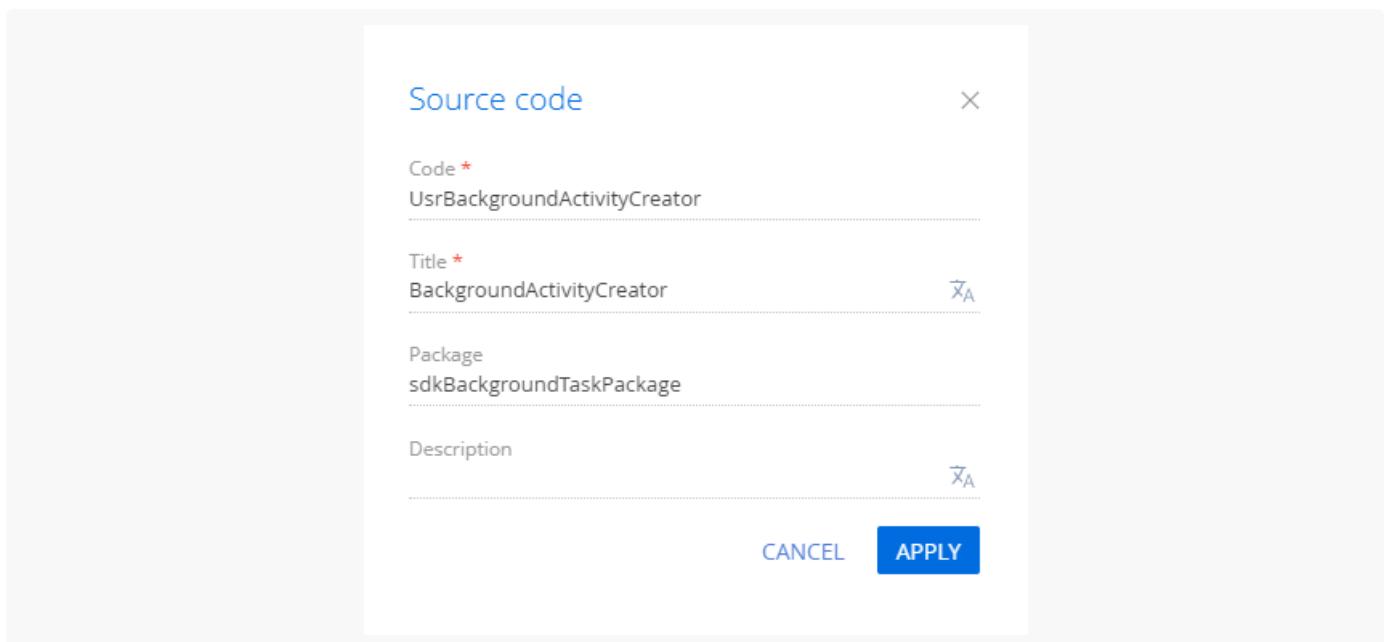
2. Создать класс для добавления активности

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]) — "UsrBackgroundActivityCreator".
- [Заголовок] ([Title]) — "BackgroundActivityCreator".



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

4. В дизайнере схем добавьте исходный код.

```
UsrBackgroundActivityCreator

namespace Terrasoft.Configuration
{
    using System;
    using Terrasoft.Common;
    using Terrasoft.Core;
    using Terrasoft.Core.DB;
    using Terrasoft.Core.Tasks;
    using System.Threading.Tasks;

    public class UsrBackgroundActivityCreator : IBackgroundTask<UsrActivityData>, IUserConnec
```

```

private UserConnection _userConnection;

/* Implement the Run method of the IBackgroundTask interface. */
public void Run(UsrActivityData data) {
    /* Forced 30-second delay. */
    System.Threading.Tasks.Task.Delay(TimeSpan.FromSeconds(30));
    /* Creating activity. */
    var activity = new Activity(_userConnection){
        UseAdminRights = false,
        Id = Guid.NewGuid(),
        TypeId = data.TypeId,
        Title = data.Title,

        /* Activity category is "To do". */
        ActivityCategoryId = new Guid("F51C4643-58E6-DF11-971B-001D60E938C6")
    };
    activity.SetDefColumnValues();
    activity.Save(false);
}

/* Implement the SetUserConnection method of the IUserConnectionRequired interface. */
public void SetUserConnection(UserConnection userConnection) {
    _userConnection = userConnection;
}
}
}

```

Класс `UsrBackgroundActivityCreator` реализует интерфейсы `IBackgroundTask<UsrActivityData>` и `IUserConnectionRequired`. В методе `Run()` после принудительной задержки в 30 секунд на основе предоставленного экземпляра `UsrActivityData` создается экземпляр объекта раздела [Активности] ([*Activities*]).

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

3. Создать бизнес-процесс для запуска фоновой операции

- [Перейдите в раздел \[Конфигурация \]](#) ([*Configuration*]) и выберите пользовательский [пакет](#), в который будет добавлена схема.
- На панели инструментов реестра раздела нажмите [Добавить] —> [Бизнес процесс] ([*Add*] —> [*Business process*]).

	Status	Type	Object	Modified on
entityData		Source code		9/7/2021, 2:27:12 PM
BackgroundActivityCrea		Source code		9/7/2021, 2:34:39 PM

3. В дизайнере процессов заполните свойства процесса:

- На панели настройки элементов заполните свойство [Заголовок] ([Title]) — "Background Task Example Process".
- На вкладке [Настройки] ([Settings]) панели настройки элементов заполните свойство [Имя] ([Code]) — "UsrBackgroundTaskExampleProcess".

Background Task Example Process

Process

SETTINGS **PARAMETERS** **METHODS**

Code
UsrBackgroundTaskExampleProcess

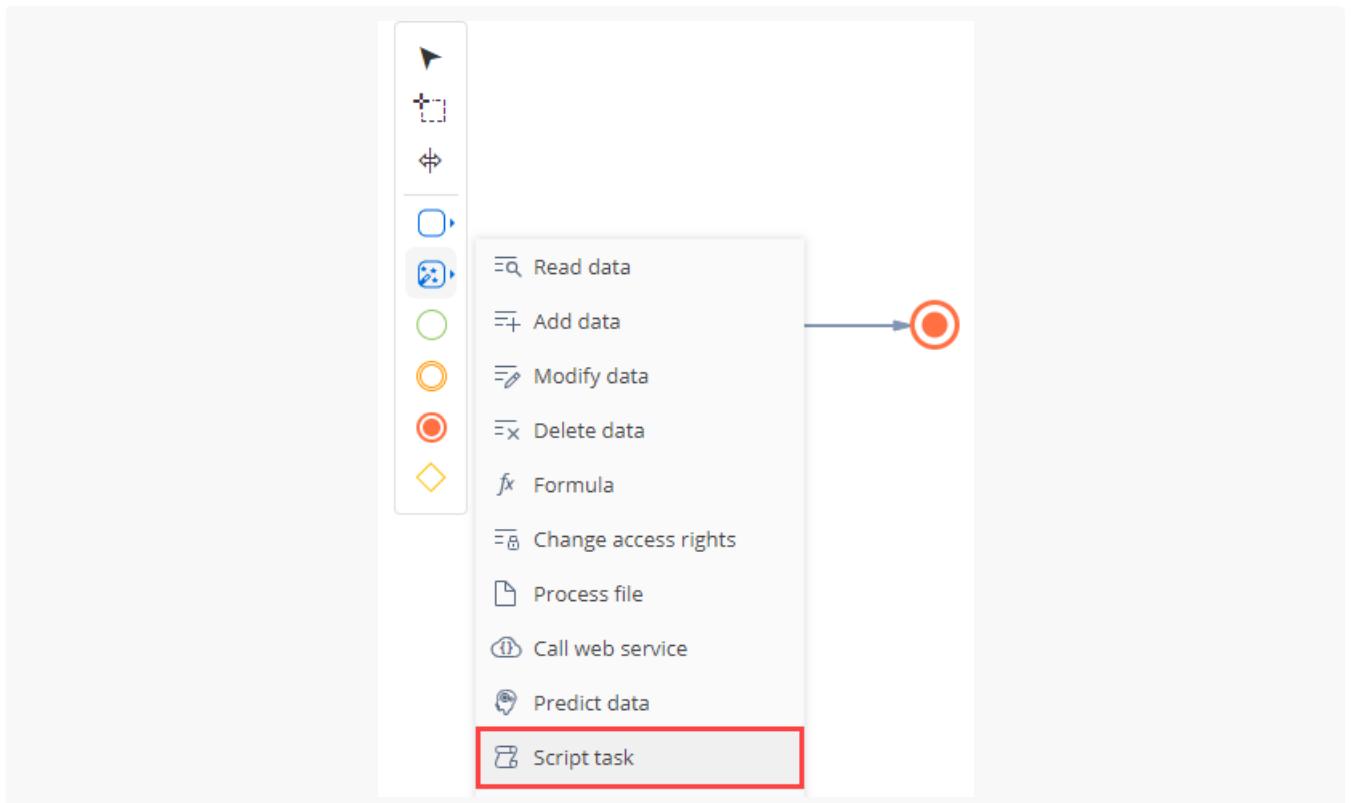
Version
0

Tag
Business Process

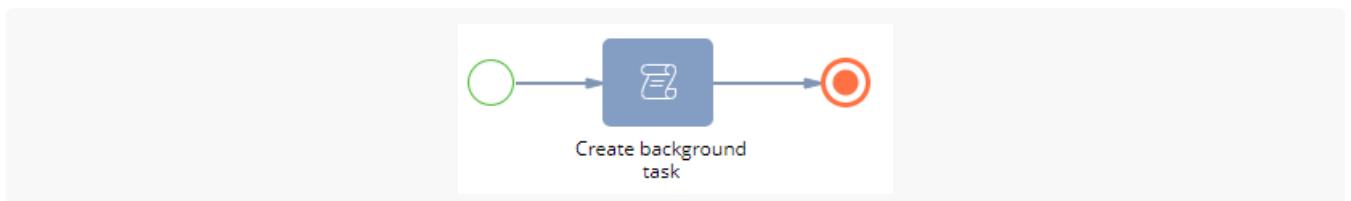
Process description

4. Реализуйте бизнес-процесс.

- В области элементов дизайнера нажмите [Действия системы] ([System actions]) и разместите элемент [Задание-сценарий] ([Script task]) в рабочей области дизайнера процессов между начальным событием [Простое] ([Simple]) и завершающим событием [Останов] ([Terminate]).



- b. Элементу [Задание-сценарий] ([*Script task*]) добавьте имя "Создать фоновую операцию" ("Create background task").



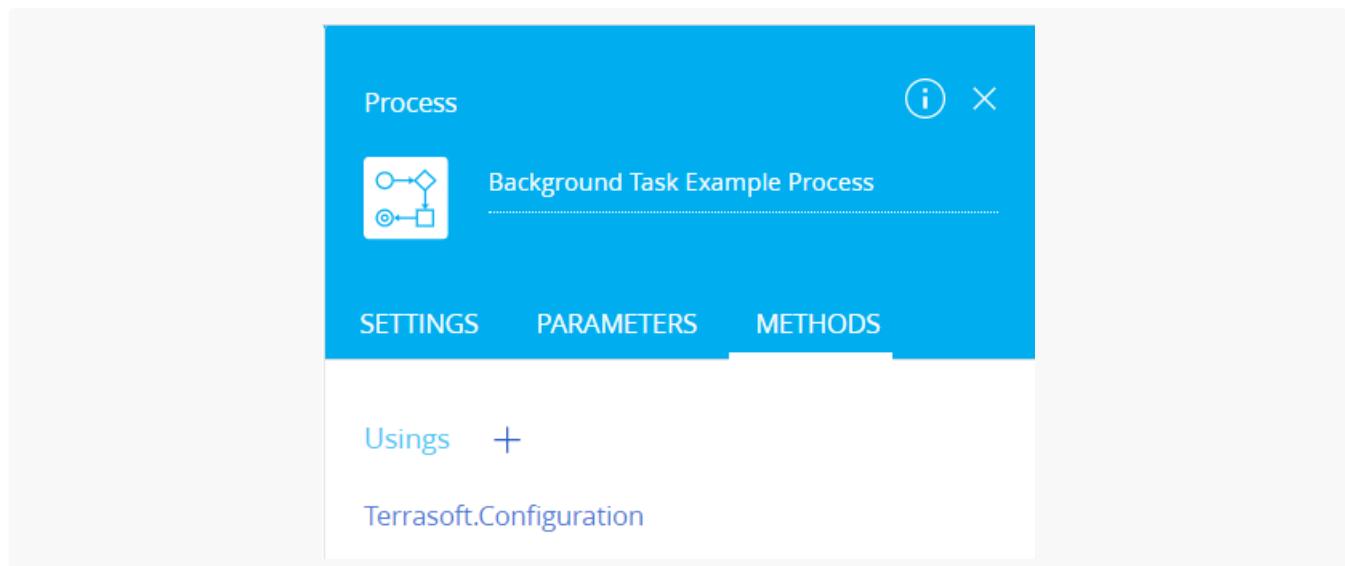
- c. Добавьте код элемента [Задание-сценарий] ([*Script task*]).

Код элемента [Задание-сценарий] ([*Script task*])

```
var data = new UsrActivityData {
    Title = "Activity created by background task",
    TypeId = ActivityConsts.TaskTypeUID
};
Terrasoft.Core.Tasks.Task.StartNewWithUserConnection<UsrBackgroundActivityCreator, UsrActivity>(data);

return true;
```

- d. В дизайнере процессов на вкладке [Методы] ([*Methods*]) в блоке [*Usings*] нажмите кнопку + и добавьте пространство имен `Terrasoft.Configuration`. Это необходимо для использования в бизнес-процессе реализации [класса для объекта активности](#) и [класса для добавления активности](#).



5. На панели инструментов дизайнера нажмите [Сохранить] ([Save]).
6. Во всплывающем окне нажмите [Опубликовать] ([Publish]) для компиляции кода элемента [Задание-сценарий] ([Script task]).

Результат выполнения примера

Чтобы запустить бизнес-процесс **Background Task Example Process**, на панели инструментов дизайнера процессов нажмите [Запустить] ([Run]).

В результате выполнения бизнес-процесса **Background Task Example Process** в списочном представлении реестра раздела [Активности] ([Activities]) добавляется запись [Activity created by background task].

Owner	Due	Status	Category
Supervisor	9/8/2021 8:54 AM	Not started	To do

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
     * переопределен в производном классе.
    public virtual int GetMultipliedValue()
    {
        return OriginalValue * 2;
    }
}
```

В замещающем классе `SubstituteClass` объявлены два конструктора — конструктор по умолчанию и параметризованный. 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. Это значение
     * можно изменить при создании нового объекта.
    public SubstituteClass()
    {
        AdditionalValue = 15;
    }

    /* Конструктор с параметром, который инициализирует свойство AdditionalValue значением, переданным в
     * параметре.
    public SubstituteClass(int paramValue)
    {
        AdditionalValue = paramValue;
    }
}
```

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

Пример. Инстанцировать и вызвать метод `GetMultipliedValue()` замещающего класса `SubstituteClass` с конструктором по умолчанию и параметризованным 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 SubstituteObject */
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 AdditionalValue */
var additionalValue = substituteObject.GetMultipliedValue();

/* Получение экземпляра замещающего класса, который инициализирован параметризованным конструктором */
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 значением 3 */
var additionalValueParametrized = substituteObjectWithParameters.GetMultipliedValue();
```

Пример 2

Замещаемый класс `SubstitutableClass` имеет один параметризованный конструктор.

Замещаемый класс `SubstitutableClass`

```
/* Объявление замещаемого класса. */
public class SubstitutableClass
```

```

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

    /* Параметризованный 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 параметризованный 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, переданным в конструктором.
    public SubstituteClass(int paramInt) : base(paramInt + 8)
    {
        AdditionalValue = paramInt;
    }

    /* Замещение родительского метода. Метод будет возвращать значение AdditionalValue, умноженное на 3.
    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 параметризованным конструктором */
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 параметризованный конструктор.

Замещаемый класс `SubstitutableClass`

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

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

    /* Метод, который возвращает значение OriginalValue, умноженное на 2. Этот метод может быть переопределен в производных классах */
    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 */
    public SubstituteClass() : base(0)
    {
    }

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

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

Создать замещающий класс

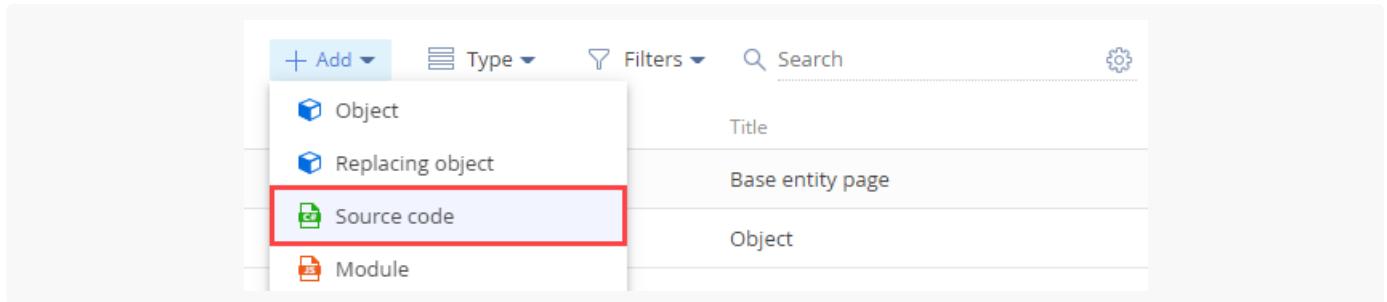


Сложный

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.

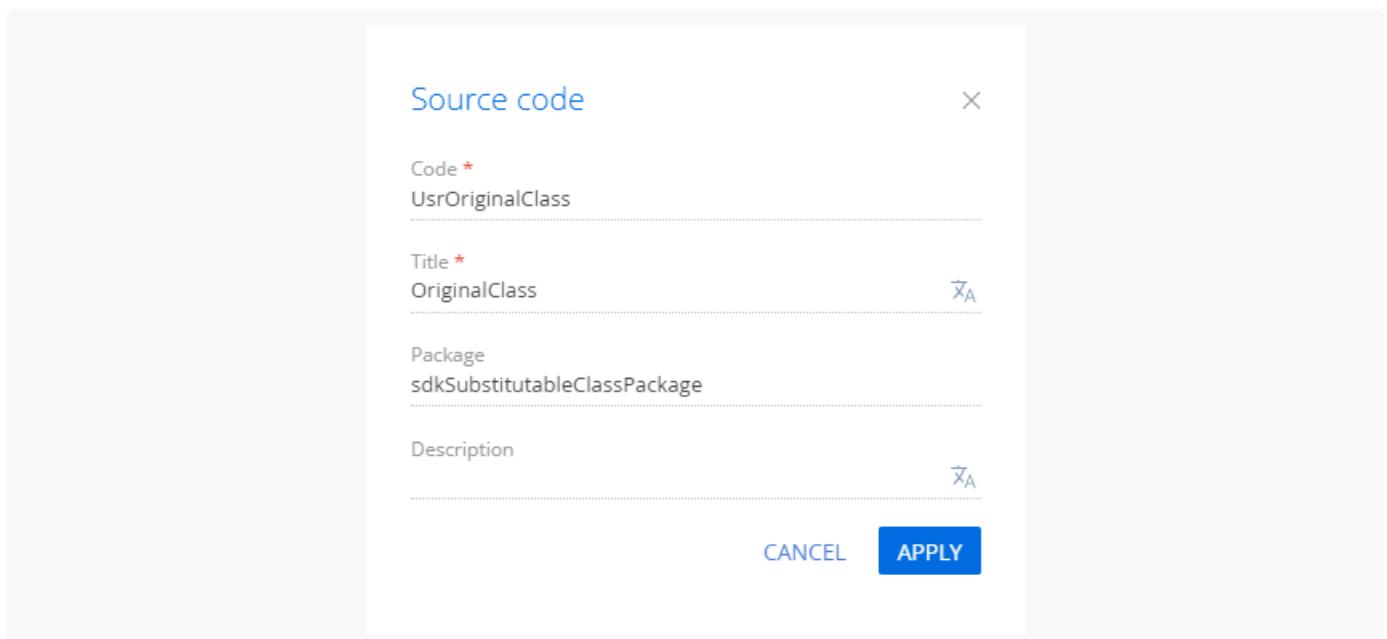
1. Реализовать замещаемый класс

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.

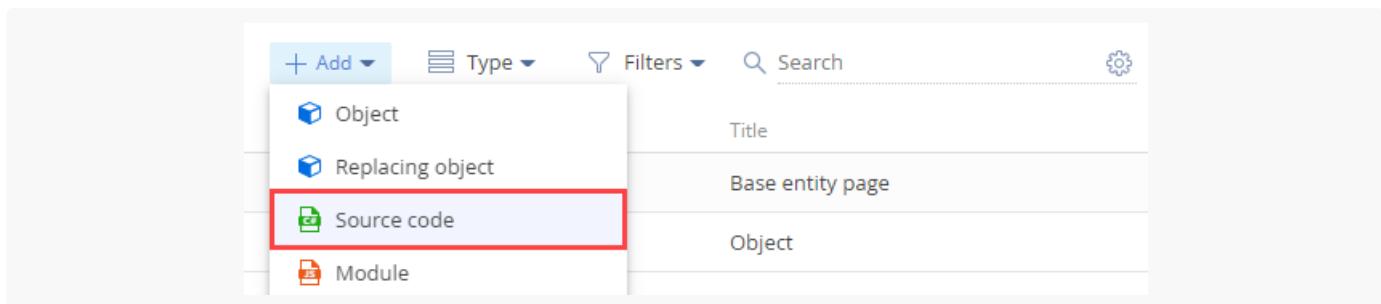
```
UsrOriginalClass

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

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

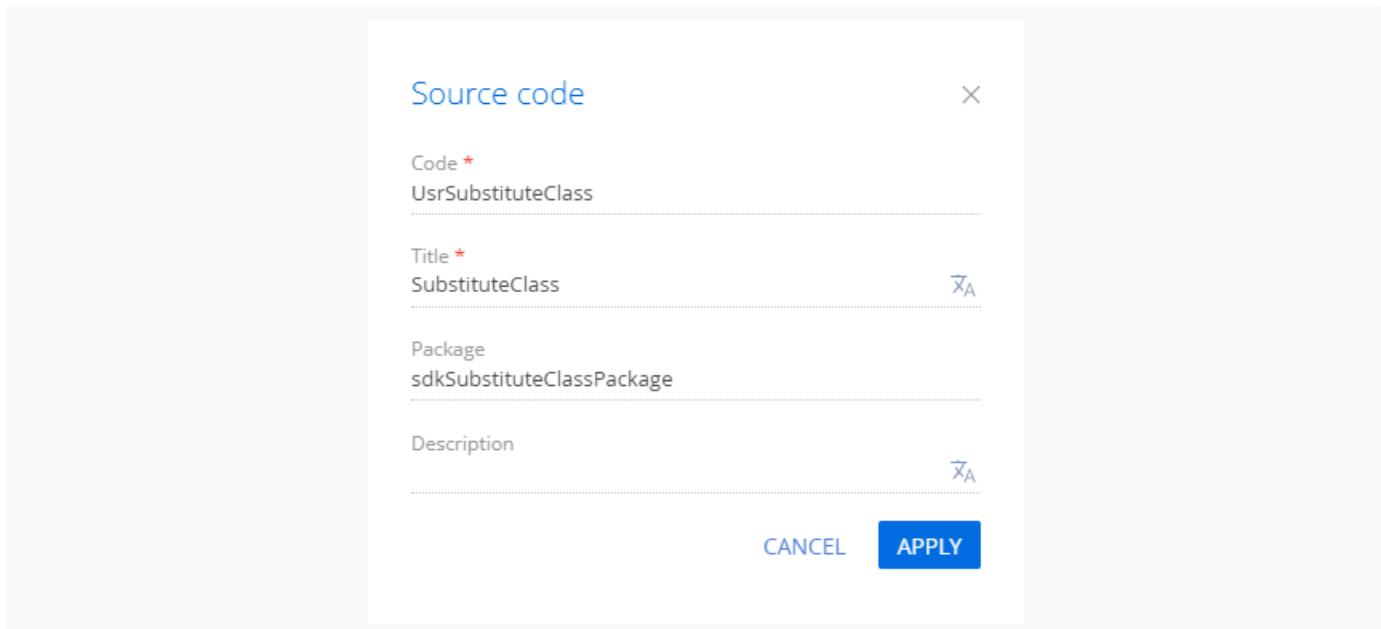
2. Реализовать замещающий класс

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; }

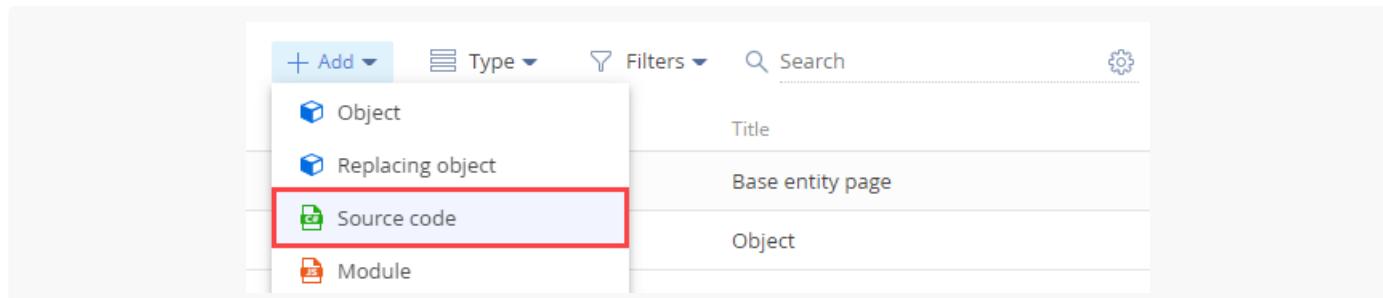
        /* В конструкторе выполняется первичная инициализация свойства Rate переданным значением */
        public UsrSubstituteClass(int rateValue)
        {
            Rate = rateValue;
        }

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

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

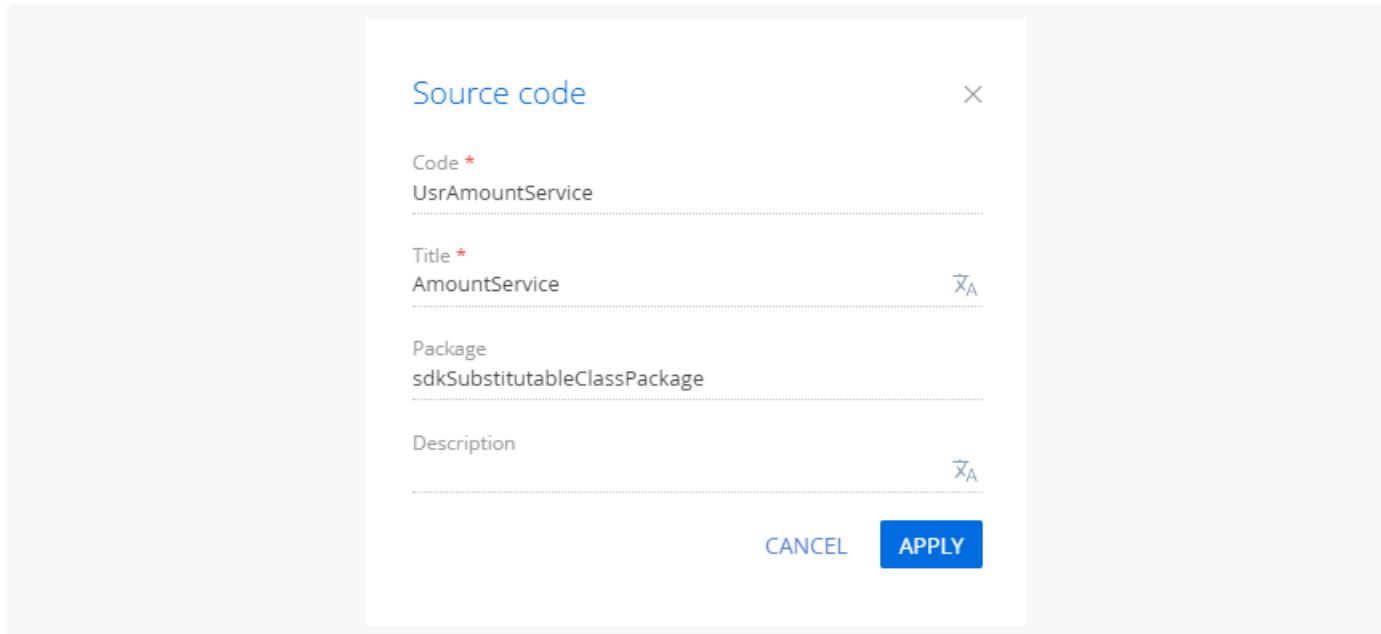
3. Реализовать пользовательский веб-сервис

- 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`.
- 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.

- В дизайнере схем добавьте пространство имен `Terrasoft.Configuration`.
- С помощью директивы `using` добавьте пространства имен, типы данных которых будут задействованы в классе.
- Добавьте название класса `UsrAmountService`, которое соответствует названию схемы (свойство [Код] ([Code])).
- В качестве родительского класса укажите класс `Terrasoft.Nui.ServiceModel.WebService.BaseService`.
- 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.Wrapper
    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 конструктора
        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 кл
        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()`.

- 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`.

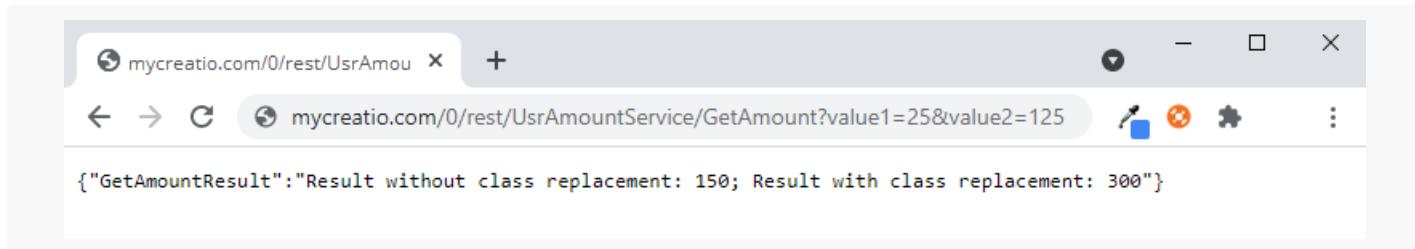
Результат выполнения примера

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.



Понятие локализуемых ресурсов

Сложный

Локализуемые ресурсы — ресурсы, которые используются для отображения интерфейса приложения Creatio в соответствии с языком, установленным в профиле пользователя. К локализуемым ресурсам относятся изображения и локализуемые строки.

Локализуемые ресурсы являются частью конфигурационных ресурсов приложения. Ресурсы приложения размещены в [пакетах](#) и привязаны к базовой схеме. При запросе ресурсов определенной схемы выполняется их сбор по иерархии с учетом уровней и позиций пакетов.

Для локализуемых ресурсов используются понятия основного и дополнительного языка.

Основной язык (основная языковая культура) — язык, который по умолчанию используется для отображения интерфейса приложения Creatio.

Дополнительный язык (дополнительная языковая культура) — язык, который используется для отображения интерфейса приложения Creatio, был изменен в профиле пользователя и отличается от основного языка.

Чтобы **изменить основной язык**, необходимо выполнить его активацию. Активация языка интерфейса описана в статье [Настроить язык интерфейса](#)

При настройке системы в мастерах разделов и деталей, дизайнерах процессов и кейсов, а также в разделе [Переводы] ([Translation]) используются все языки, которые установлены в системе. В других случаях используются только активированные языки (т. е. языки, для которых установлен признак [Активен] ([Active])). Такое разделение позволяет уменьшить время выполнения задач, например, вход в систему, открытие страницы записи и т. д. Подробнее читайте в статьях [Настроить язык интерфейса](#) и [Перевести элементы интерфейса в разделе \[Переводы\]](#).

Типы локализуемых ресурсов, которые реализованы в Creatio:

- Простые локализуемые ресурсы.
- Привязанные локализуемые ресурсы.

Отобразить локализуемые ресурсы

При отображении локализуемых ресурсов учитываются:

- Режим отображения.
- Способ отображения.

Режимы отображения локализуемых ресурсов

Режимы отображения локализуемых ресурсов, которые доступны в Creatio:

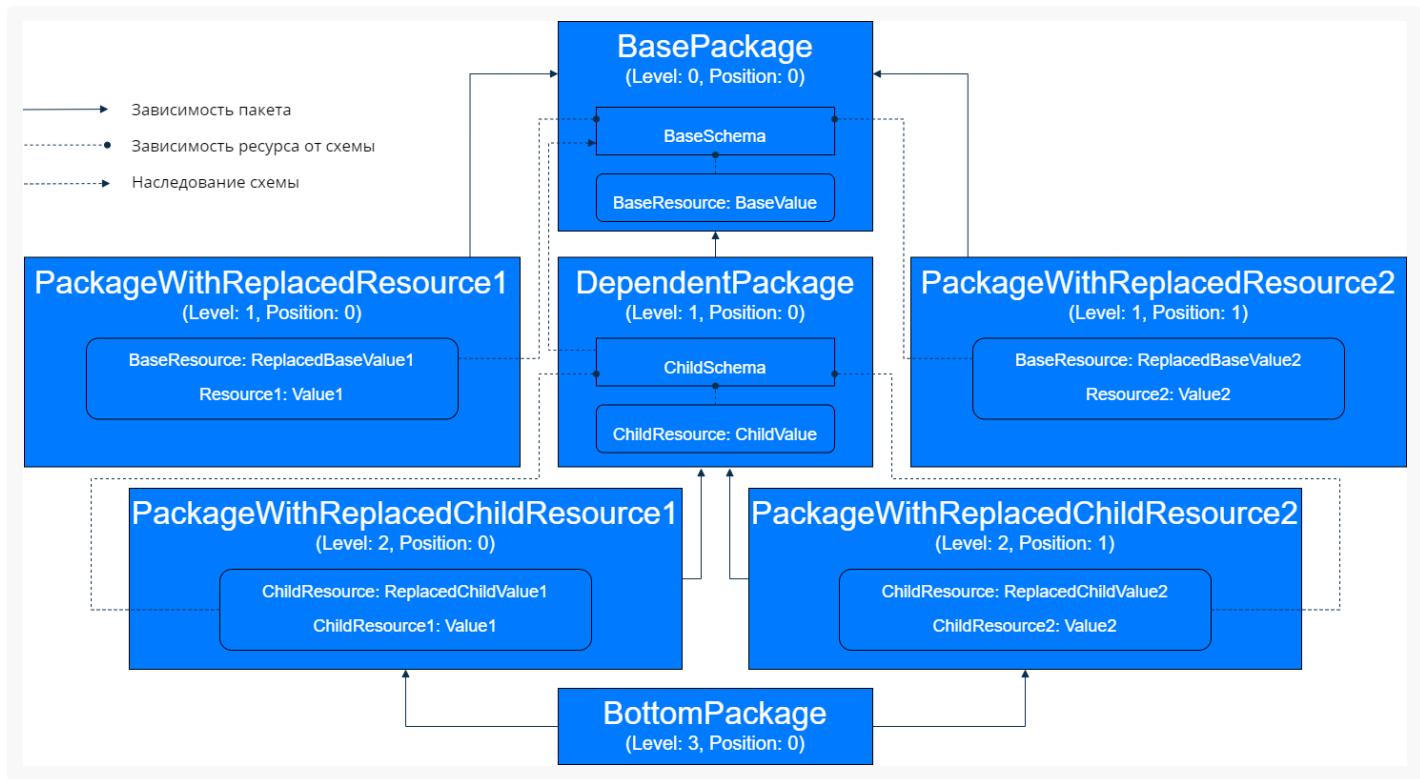
- Режим дизайна (Design-time).
- Режим выполнения приложения (Run-time).

При отображении локализуемых ресурсов учитывается иерархия пакетов.

Режим дизайна

Назначение режима дизайна — отображение локализуемых ресурсов в дизайнерах и мастерах. В этом режиме иерархия локализуемых ресурсов [схем](#) конфигурационных элементов строится только до уровня пакета, в котором содержится схема с запрашиваемыми ресурсами. При построении иерархии учитываются локализуемые ресурсы пакетов, которые попадают в иерархию [по прямым связям](#).

Рассмотрим отображение локализуемых ресурсов на примере ресурса `ChildResource: ChildValue` схемы `ChildSchema` пакета `DependentPackage`. Иерархия пакетов представлена на рисунке ниже.



Результирующие ресурсы для схемы `ChildSchema` в режиме дизайна:

- `BaseResource: BaseValue`.
- `ChildResource: ChildValue`.

В режиме дизайна в ресурсах не учитываются:

- Ресурсы пакетов `PackageWithReplacedResource1` И `PackageWithReplacedResource2`, поскольку эти пакеты не участвуют в построении иерархии.
- Ресурсы пакетов `PackageWithReplacedChildResource1` И `PackageWithReplacedChildResource2`, поскольку эти пакеты относительно запрашиваемой схемы находятся ниже в иерархии.

Если запрашивается схема с указанием стартового пакета (пакета, с которого необходимо начать сбор ресурсов), то результирующий набор ресурсов формируется до уровня пакета `BottomPackage`.

Результирующие ресурсы для схемы `ChildSchema` до уровня пакета `BottomPackage` в режиме дизайна:

- `BaseResource: BaseValue`.
- `ChildResource: ReplacedChildValue2`.
- `ChildResource1: Value1`.
- `ChildResource2: Value2`.

Здесь значение ресурса `ChildResource` изменилось на `ReplacedChildValue2`, поскольку в пакетах, которые по иерархии находятся на уровень ниже (`Level 2`), было выполнено замещение исходного ресурса. При этом учитывается позиция и имя пакета — преимущество отдается пакету с большим значением позиции. Для сортировки пакетов с одинаковым значением позиции используется алфавитный порядок.

Режим выполнения приложения

Назначение режима выполнения приложения — отображение локализуемых ресурсов в разделах приложения, за исключением дизайнеров и мастеров. **Отличие** режима выполнения приложения от режима дизайна — при запросе схемы в результирующем списке ресурсов учитываются ресурсы пакетов, которые не входят в иерархию.

Результирующие ресурсы для схемы `childSchema` в режиме выполнения приложения:

- `BaseResource: ReplacedBaseValue2`.
- `Resource1: Value1`.
- `Resource2: Value2`.
- `ChildResource: ReplacedChildValue2`.
- `ChildResource1: Value1`.
- `ChildResource2: Value2`.

Способы отображения локализуемых ресурсов

Способы отображения локализуемых ресурсов, которые реализованы в Creatio:

- Если в профиле пользователя **выбран основной язык**, то для отображения используется значение локализуемого ресурса на основном языке.
- Если в профиле пользователя **выбран дополнительный язык и присутствует значение локализуемого ресурса на дополнительном языке**, то для отображения используется значение локализуемого ресурса на дополнительном языке.
- Если в профиле пользователя **выбран дополнительный язык и отсутствует значение локализуемого ресурса на дополнительном языке**, то для отображения используется значение локализуемого ресурса на основном языке.

Классы, которые реализуют логику отображения локализуемых ресурсов:

- `Terrasoft.Common.LocalizableString` — работа с локализуемыми строками.
- `Terrasoft.Common.LocalizableImage` — работа с локализуемыми изображениями.

Свойства и методы, которые используются для получения значения локализуемого ресурса:

- `Value` — свойство, которое возвращает значение локализуемого объекта для текущего языка. Если не найдено значение локализуемого объекта для текущего языка, то будет возвращено значение для основного языка.
- `HasValue` — свойство, которое возвращает признак наличия значения локализуемого объекта для текущего языка. Если не найдено значение локализуемого объекта для текущего языка, то будет возвращено значение для основного языка.
- `GetCultureValue()` — метод, который возвращает значение локализуемого объекта для текущего языка. Если не найдено значение локализуемого объекта для текущего языка, то будет возвращено значение для основного языка.
- `HasCultureValue()` — метод, который возвращает признак наличия значения локализуемого объекта для запрашиваемого языка без учета основного языка.

Класс `Terrasoft.Common.LocalizableString` описан в [Библиотеке .NET классов](#). Класс `Terrasoft.Common.LocalizableImage` описан в [Библиотеке .NET классов](#).

Хранить локализуемые ресурсы

В зависимости от типа локализуемого ресурса, хранение имеет свои особенности.

Хранить простые локализуемые ресурсы

В качестве хранилищ простых локализуемых ресурсов используются:

- **База данных** — хранит ресурсы, которые необходимы для работы приложения. Основное хранилище локализуемых ресурсов.
- **Хранилище SVN** — хранит ресурсы, которые необходимо установить в приложение или перенести между [средами разработки](#). Предварительно необходимо выполнить экспорт локализуемых ресурсов в хранилище SVN.

Хранить простые локализуемые ресурсы в базе данных

Для хранения локализуемых ресурсов используется таблица `[SysLocalizableValue]` базы данных, в которой локализуемые ресурсы хранятся в текстовом формате в виде пар "ключ-значение". Каждая запись привязана к пакету и базовому идентификатору схемы.

Описание основных полей таблицы `[SysLocalizableValue]` базы данных приведено ниже.

Основные поля таблицы [SysLocalizableValue]

Колонка	Описание
[Id]	Идентификатор записи.
[SysPackageId]	Идентификатор пакета.
[SysSchemaId]	Идентификатор базовой схемы. Заполняется только для ресурсов конфигурации.
[ResourceManager]	Название менеджера ресурсов. Заполняется только для ресурсов ядра.
[SysCultureId]	Идентификатор языковой культуры.
[ResourceType]	Тип ресурса.
[IsChanged]	Признак изменения локализуемого ресурса пользователем.
[Key]	Ключ локализуемого ресурса.
[Value]	Значение строкового ресурса.
[ImageData]	Значение графического ресурса.

Если в профиле пользователя активной является дополнительная языковая культура, то при добавлении локализуемого ресурса в таблице [SysLocalizableValue] создается соответствующая запись. Пользователи других языковых культур имеют доступ к добавленному локализуемому ресурсу путем использования **механизма дублирования в основную языковую культуру**. Назначение механизма — создание аналогичной записи локализуемого ресурса со ссылкой на основную языковую культуру. Если в других языковых культурах не задано значение для текущего локализуемого ресурса, то будет отображаться значение, установленное в основной языковой культуре.

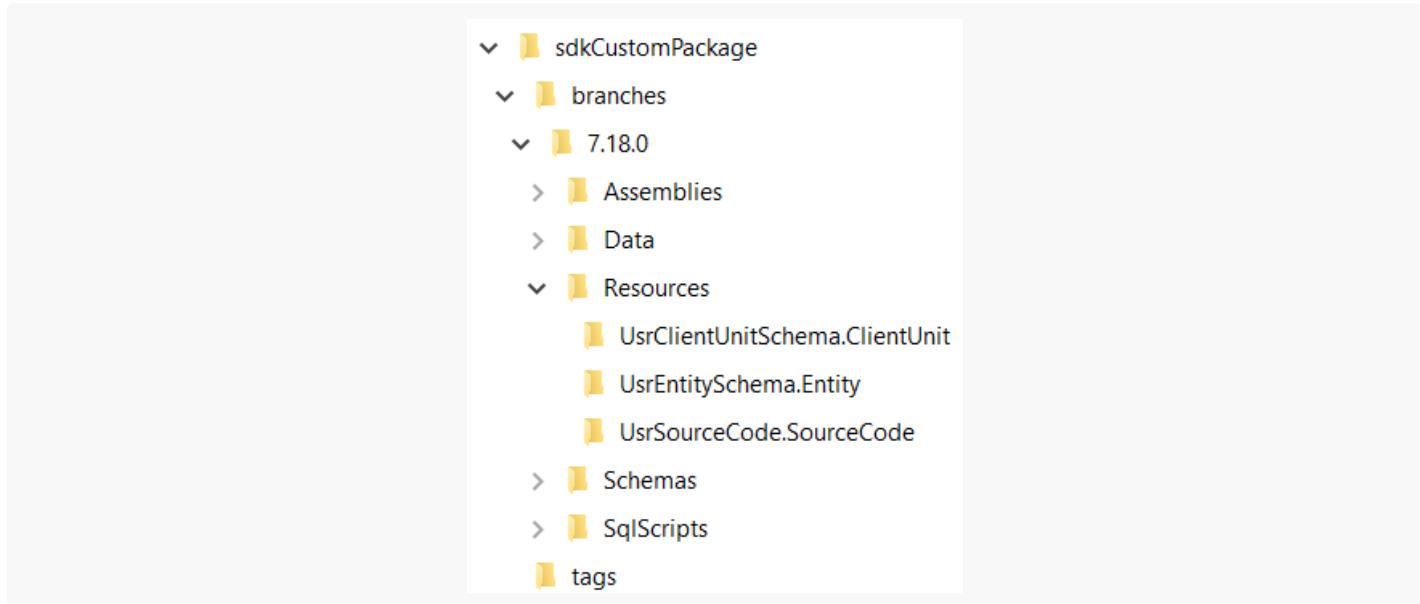
Для каждого набора ресурсов схемы в связке пакет — схема — культура в таблице [SysPackageResourceChecksum] базы данных хранится **контрольная сумма**, которая при обновлении пакета позволяет определить наличие изменений в его ресурсах. Благодаря использованию контрольной суммы ресурсы отделены от схем, что позволяет создавать пакеты переводов.

Хранить простые локализуемые ресурсы в хранилище SVN

Для хранения локализуемых ресурсов пакета в хранилище SVN предназначен каталог Resources, который используется для создания пакета переводов. **Пакет переводов** — пакет, который содержит только локализуемые ресурсы и не содержит схем конфигурационных элементов. Пакет переводов может содержать локализуемые ресурсы для схемы, которая находится в другом пакете.

Для хранения локализуемых ресурсов схем с одинаковыми именами, но с разными менеджерами (например, Entity и ClientUnit), в имена локализуемых ресурсов пакета добавляются имена менеджеров схем без префикса SchemaManager. В экспортированных схемах локализуемые ресурсы хранятся в

формате *.xml.



Хранить привязанные локализованные ресурсы

Хранилища привязанных локализуемых ресурсов идентичны [хранилищам простых локализуемых ресурсов](#).

Хранить привязанные локализуемые ресурсы в базе данных

Для хранения привязанных локализуемых ресурсов используется таблица `[SysPackageDataLcz]` базы данных.

Описание основных полей таблицы `[SysPackageDataLcz]` базы данных приведено ниже.

Основные поля таблицы `[SysPackageDataLcz]`

Колонка	Описание
<code>[Id]</code>	Идентификатор записи.
<code>[SysPackageSchemaDataId]</code>	Идентификатор привязки в таблице <code>[SysPackageSchemaData]</code> .
<code>[SysCultureId]</code>	Идентификатор языковой культуры.
<code>[Data]</code>	Привязанные локализуемые данные.

Особенности сохранения привязанных локализуемых ресурсов:

- Если схема **содержит привязанные локализуемые ресурсы**, то данные сохраняются в таблицу `[SysPackageDataLcz]` базы данных.
- Если схема **не содержит привязанные локализуемые ресурсы**, то данные сохраняются в

таблицу [SysPackageSchemaData] базы данных.

Запись таблицы [SysPackageDataLcz] содержит:

- Ссылку на соответствующий идентификатор записи в таблице [SysPackageSchemaData] базы данных.
- Ссылку на идентификатор соответствующей языковой культуры в таблице [SysCulture] базы данных.

Например, если в системе используются английская и русская языковые культуры, то каждой записи таблицы [SysPackageSchemaData] соответствуют две записи в таблице [SysPackageDataLcz].

Хранить привязанные локализуемые ресурсы в хранилище SVN

Для хранения привязанных локализуемых ресурсов пакета в хранилище SVN предназначен каталог Data .

Структура каталога Data :

- Файл data.json — нелокализуемые ресурсы.
- Каталог Localization — привязанные локализуемые ресурсы. Каталог содержит соответствующие файлы для языковых культур. Название файла: data.КодязыковойКультуры.json (например, data.en-US.json).

Структура хранения привязанных локализуемых ресурсов рассмотрена на примере схемы Periodicity пакета TryItPackage и отображена на рисунке ниже.

Name	Date modified	Type	Size
Localization	27.09.2021 9:31	File folder	1 KB
data.json	27.09.2021 9:31	JSON File	2 KB
descriptor.json	27.09.2021 9:31	JSON File	0 KB
filter.json	27.09.2021 9:31	JSON File	0 KB

Установить привязанные локализуемые ресурсы

Особенности установки привязанных локализуемых ресурсов:

- Если схема **не содержит привязанные локализуемые ресурсы**, то установка выполняется в основную таблицу объекта Entity .
- Если схема **содержит привязанные локализуемые ресурсы** (т. е. в таблице [SysPackageDataLcz] присутствуют соответствующие записи), то установка выполняется в основную таблицу базы данных соответствующей схемы и в таблицу локализации.

Шаблон формирования имени таблицы локализации: [SysИмяОсновнойТаблицыLcz] .

Например, при установке привязанных локализуемых ресурсов для схемы ContactType :

- Нелокализуемые данные будут установлены в таблицу [ContactType] базы данных.
- Локализуемые данные будут установлены в таблицу [ContactType] и в таблицу [SysContactTypeLcz] базы данных.

На заметку. Для таблицы локализации, которая соответствует системной таблице (т. е. название начинается с префикса `Sys`), префикс `Sys` повторно не добавляется. Например, для таблицы `[SysTestSchema]` базы данных таблица локализации называется `[SysTestSchemaLcz]`, а не `[SysSysTestSchemaLcz]`.

Прямой доступ к данным



Способы доступа к базе данных, которые предоставляют back-end компоненты ядра:

- Доступ через ORM-модель.
- Прямой доступ.

Для доступа к данным рекомендуется использовать ORM-модель, хотя прямой доступ к базе данных также реализован в back-end компонентах ядра. Выполнение запросов к базе данных через ORM-модель подробно описано в статье [Доступ к данным через ORM](#).

В этой статье будет рассмотрено выполнение запросов к базе данных через прямой доступ.

Классы, которые реализуют работу с данными через прямой доступ:

- `Terrasoft.Core.DB.Select` — построение запросов на получение записей из таблиц базы данных.
- `Terrasoft.Core.DB.Insert` — построение запросов на добавление записей в таблицы базы данных.
- `Terrasoft.Core.DB.InsertSelect` — построение запросов на добавление записей в таблицу базы данных на основании данных, полученных в запросах на получение записей из таблицы базы данных.
- `Terrasoft.Core.DB.Update` — построение запросов на изменение записей в таблице базы данных.
- `Terrasoft.Core.DB.UpdateSelect` — построение запросов на изменение записей в таблице базы данных на основании данных, полученных в запросах на получение записей из таблицы базы данных.
- `Terrasoft.Core.DB.Delete` — построение запросов на удаление записей в таблице базы данных.
- `Terrasoft.Core.DB.DBExecutor` — построение и выполнение сложных запросов (например, с несколькимиложенными фильтрациями, различными комбинациями `join`-ов и т. д.) к базе данных.

Получить данные из базы данных

Классы, которые реализуют получение данных из базы данных:

- `Terrasoft.Core.DB.Select` — получение данных из базы данных через прямой доступ.
- `Terrasoft.Core.Entities.EntitySchemaQuery` — получение данных из базы данных через ORM-модель.

Выполнение запросов к базе данных с использованием класса

`Terrasoft.Core.Entities.EntitySchemaQuery` подробно описано в статье [Доступ к данным через ORM](#).

Назначение класса `Terrasoft.Core.DB.Select` — построение запросов на выборку записей из таблиц базы данных. После создания и конфигурирования экземпляра класса будет построен `SELECT`-запрос к базе данных приложения. В запрос можно добавить колонки, фильтры и условия ограничений.

Особенности класса `Terrasoft.Core.DB.Select`:

- В результирующем запросе не учитываются права доступа текущего пользователя. Пользователь получает доступ ко всем таблицам и записям базы данных.
- В результирующем запросе не учитываются [данные из хранилища кэша](#).

Результат выполнения запроса — экземпляр, реализующий интерфейс `System.Data.IDataReader`, или скалярное значение соответствующего типа.

При необходимости построения запросов на выборку записей из базы данных с учетом прав доступа пользователя и данных из хранилища кэша, используйте класс `Terrasoft.Core.Entities.EntitySchemaQuery`.

Добавить данные в базу данных

Классы, которые реализуют добавление данных в базу данных:

- `Terrasoft.Core.DB.Insert`.
- `Terrasoft.Core.DB.InsertSelect`.

Массовое добавление данных

Назначение класса `Terrasoft.Core.DB.Insert` — построение запросов на добавление записей в таблицы базы данных. После создания и конфигурирования экземпляра класса будет построен `INSERT`-запрос к базе данных приложения.

Результат выполнения запроса — количество записей, которые были добавлены с помощью запроса.

Класс содержит реализацию функциональности многострочной вставки. Для этого предназначен метод `Values()`. При вызове метода `Values()` все последующие вызовы метода `Set()` попадают в новый экземпляр `ColumnValues`. Если коллекция `ColumnValuesCollection` содержит более одного набора данных, то будет построен запрос с несколькими блоками `Values()`.

Пример многострочной вставки

```
new Insert(UserConnection)
    .Into("Table")
    .Values()
        .Set("Column1", Column.Parameter(1))
        .Set("Column2", Column.Parameter(1))
        .Set("Column3", Column.Parameter(1))
    .Values()
        .Set("Column1", Column.Parameter(2))
        .Set("Column2", Column.Parameter(2))
        .Set("Column3", Column.Parameter(2))
```

```
.Values()
    .Set("Column1", Column.Parameter(3))
    .Set("Column2", Column.Parameter(3))
    .Set("Column3", Column.Parameter(3))
.Execute();
```

В результате будет сформирован SQL-запрос.

SQL-запрос

```
-- Для MSSQL или PostgreSQL
INSERT INTO [dbo].[Table] (Column1, Column2, Column3)
VALUES (1, 1, 1),
       (2, 2, 2),
       (3, 3, 3)

-- Для Oracle
INSERT ALL
    into Table (column1, column2, column3) values (1, 1, 1)
    into Table (column1, column2, column3) values (2, 2, 2)
    into Table (column1, column2, column3) values (3, 3, 3)
SELECT * FROM dual
```

Особенности использования многострочного добавления данных:

- Ограничение количества параметров в MS SQL при использовании `Column.Parameter` в выражении `Set()` — 2100 параметров.
- При превышении допустимого количества параметров разбивку запроса на подзапросы должен выполнить разработчик, поскольку класс `Terrasoft.Core.DB.Insert` не предоставляет такую возможность.

Пример

```
IEnumerable<IEnumereable<ImportEntity>> GetImportEntitiesChunks(IEnumerable<ImportEntity> entities)
{
    var entitiesList = entities.ToList();
    var columnsList = keyColumns.ToList();
    var maxParamsPerChunk = Math.Abs(MaxParametersCountPerQueryChunk / columnsList.Count + 1);
    var chunksCount = (int)Math.Ceiling(entitiesList.Count / (double)maxParamsPerChunk);
    return entitiesList.SplitOnParts(chunksCount);
}

var entitiesList = GetImportEntitiesChunks(entities, importColumns);
entitiesList.AsParallel().AsOrdered()
    .ForAll(entitiesBatch => {
        try {
```

```

        var insertQuery = GetBufferedImportEntityInsertQuery();
        foreach (var importEntity in entitiesBatch) {
            insertQuery.Values();
            SetBufferedImportEntityInsertColumnValues(importEntity, insertQuery,
                importColumns);
            insertQuery.Set("ImportSessionId", Column.Parameter(importSessionId));
        }
        insertQuery.Execute();
    } catch (Exception e) {
        //...
    }
});

```

- Валидацию совпадения количества колонок и количества условий `Set()` должен выполнить разработчик, поскольку класс `Terrasoft.Core.DB.Insert` не предоставляет такую возможность. При несовпадении количества возникнет исключение на уровне работы СУБД.

Добавление данных из выборки

Назначение класса `Terrasoft.Core.DB.InsertSelect` — построение запросов на добавление записей в таблицы базы данных на основании данных, полученных в запросах на получение записей из таблицы базы данных. То есть в качестве источника данных запроса используется экземпляр класса `Terrasoft.Core.DB.Select`. После создания и конфигурирования экземпляра класса будет построен `INSERT INTO SELECT`-запрос к базе данных приложения.

Особенность класса `Terrasoft.Core.DB.InsertSelect` — в результирующем запросе для добавляемых записей не учитываются права доступа текущего пользователя. Пользовательское соединение используется только для доступа к таблице базы данных.

Результат выполнения запроса — добавление в таблицы базы данных записей, которые были получены в `Select`-запросе.

Изменить данные в базе данных

Классы, которые реализуют изменение данных в базе данных:

- `Terrasoft.Core.DB.Update`.
- `Terrasoft.Core.DB.UpdateSelect`.

Массовое изменение данных

Назначение класса `Terrasoft.Core.DB.Update` — построение запросов на изменение записей в таблицах базы данных. После создания и конфигурирования экземпляра класса будет построен `UPDATE`-запрос к базе данных приложения.

Изменение данных на основании выборки

Назначение класса `Terrasoft.Core.DB.UpdateSelect` — построение запросов на изменение записей в

таблицах базы данных на основании данных, полученных в запросах на получение записей из таблицы базы данных. То есть в качестве источника данных запроса используется экземпляр класса `Terrasoft.Core.DB.Select`. После создания и конфигурирования экземпляра класса будет построен `UPDATE FROM`-запрос к базе данных приложения.

Результат выполнения запроса — изменение в таблице базы данных записей, которые были получены в `Select`-запросе.

Удалить данные из базы данных

`Terrasoft.Core.DB.Delete` — класс, который реализует удаление данных из базы данных.

Назначение класса `Terrasoft.Core.DB.Delete` — построение запросов на удаление записей из таблиц базы данных. После создания и конфигурирования экземпляра класса будет построен `DELETE`-запрос к базе данных приложения.

Использовать многопоточность при работе с базой данных

Многопоточность — использование нескольких параллельных потоков при отправке запросов к базе данных через `UserConnection`.

`Terrasoft.Core.DB.DBExecutor` — класс, который позволяет использовать многопоточность. Реализует построение и выполнение нескольких запросов к базе данных в одной транзакции. Одному пользователю доступен только один экземпляр `DBExecutor`. Пользователь не имеет возможности создавать новые экземпляры.

Использование многопоточности может привести к проблемам синхронизации старта и подтверждения транзакций. Проблема возникает, даже если `DBExecutor` не используется напрямую, а используется, например, через `EntitySchemaQuery`.

Особенность класса `Terrasoft.Core.DB.DBExecutor` — создание экземпляра `DBExecutor` необходимо обернуть в оператор `using`. Это связано с тем, что для работы с базой данных используются неуправляемые (`unmanaged`) ресурсы. Также для освобождения ресурсов можно явно вызвать метод `Dispose()`. Использование оператора `using` подробно описано в официальной [документации Microsoft](#).

Транзакция начинается вызовом метода `dbExecutor.StartTransaction` и заканчивается вызовом `dbExecutor.CommitTransaction` или `dbExecutor.RollbackTransaction`. Если выполнение вышло за область видимости блока `using` и не был вызван метод `dbExecutor.CommitTransaction`, происходит автоматический откат транзакции.

Важно. При выполнении нескольких запросов в одной транзакции необходимо передавать `dbExecutor` в методы `Execute`, `ExecuteReader`, `ExecuteScalar`.

Ниже представлены фрагменты исходного кода с использованием `DBExecutor`. Нельзя выполнять вызов методов экземпляра `DBExecutor` в параллельных потоках.

Пример правильного использования `DBExecutor`

```

/* Первое использование экземпляра DBExecutor в основном потоке. */
using (DBExecutor dbExecutor = UserConnection.EnsureDBConnection()) {
    dbExecutor.StartTransaction();
    //...
    dbExecutor.CommitTransaction();
}
//...
var select = (Select)new Select(UserConnection)
    .Column("Id")
    .From("Contact")
    .Where("Name")
    .IsEqual(Column.Parameter("Supervisor"));
/* Повторное использование экземпляра DBExecutor в основном потоке. */
using (DBExecutor dbExecutor = UserConnection.EnsureDBConnection()) {
    using (IDataReader dataReader = select.ExecuteReader(dbExecutor)) {
        while (dataReader.Read()) {
            //...
        }
    }
}

```

Пример неправильного использования DBExecutor

```

/* Создание параллельного потока. */
var task = new Task(() => {

    /* Использование экземпляра DBExecutor в параллельном потоке. */
    using (DBExecutor dbExecutor = UserConnection.EnsureDBConnection()) {
        dbExecutor.StartTransaction();
        //...
        dbExecutor.CommitTransaction();
    }
});

/* Запуск асинхронной задачи в параллельном потоке. Выполнение программы в основном потоке пропускается. */
task.Start();
//...
var select = (Select)new Select(UserConnection)
    .Column("Id")
    .From("Contact")
    .Where("Name")
    .IsEqual(Column.Parameter("Supervisor"));

/* Использование экземпляра DBExecutor в основном потоке приведет к возникновению ошибки, поскольку*/
using (DBExecutor dbExecutor = UserConnection.EnsureDBConnection()) {

```

```

using (IDataReader dataReader = select.ExecuteReader(dbExecutor)) {
    while (dataReader.Read()) {
        //...
    }
}

```

Получить данные из базы данных



Сложный

На заметку. Примеры, приведенные в этой статье, реализованы в веб-сервисе. Пакет с реализацией веб-сервиса прикреплен в блоке "Ресурсы".

Ниже приведен метод `CreateJson`, который используется в примерах для обработки результата запросов.

Метод `CreateJson`

```

private string CreateJson(IDataReader dataReader)
{
    var list = new List<dynamic>();
    var cnt = dataReader.FieldCount;
    var fields = new List<string>();
    for (int i = 0; i < cnt; i++)
    {
        fields.Add(dataReader.GetName(i));
    }
    while (dataReader.Read())
    {
        dynamic exo = new System.Dynamic.ExpandoObject();
        foreach (var field in fields)
        {
            ((IDictionary<String, Object>)exo).Add(field, dataReader.GetColumnValue(field));
        }
        list.Add(exo);
    }
    return JsonConvert.SerializeObject(list);
}

```

Пример 1

Пример. Выбрать определенное количество записей из требуемой таблицы (схемы объекта).

Метод SelectColumns

```
public string SelectColumns(string tableName, int top)
{
    top = top > 0 ? top : 1;
    var result = "{}";
    var select = new Select(UserConnection)
        .Top(top)
        .Column(Column.Asterisk())
        .From(tableName);
    using (DBExecutor dbExecutor = UserConnection.EnsureDBConnection())
    {
        using (IDataReader dataReader = select.ExecuteReader(dbExecutor))
        {
            result = CreateJson(dataReader);
        }
    }
    return result;
}
```

Пример 2

Пример. Выбрать идентификатор, имя и дату рождения контактов, дата рождения которых позже требуемого года.

Метод SelectContactsYoungerThan

```
public string SelectContactsYoungerThan(string birthYear)
{
    var result = "{}";
    var year = DateTime.ParseExact(birthYear, "yyyy", CultureInfo.InvariantCulture);
    var select = new Select(UserConnection)
        .Column("Id")
        .Column("Name")
        .Column("BirthDate")
        .From("Contact")
        .Where("BirthDate").IsGreater(Column.Parameter(year))
        .Or("BirthDate").IsNull()
        .OrderByDesc("BirthDate")
        as Select;
```

```

using (DBExecutor dbExecutor = UserConnection.EnsureDBConnection())
{
    using (IDataReader dataReader = select.ExecuteReader(dbExecutor))
    {
        result = CreateJson(dataReader);
    }
}
return result;
}

```

Пример 3

Пример. Выбрать идентификатор, имя и дату рождения контактов, дата рождения которых позже заданного года и у которых указан контрагент.

Метод SelectContactsYoungerThanAndHasAccountId

```

public string SelectContactsYoungerThanAndHasAccountId(string birthYear)
{
    var result = "{}";
    var year = DateTime.ParseExact(birthYear, "yyyy", CultureInfo.InvariantCulture);
    var select = new Select(UserConnection)
        .Column("Id")
        .Column("Name")
        .Column("BirthDate")
        .From("Contact")
        .Where()
        .OpenBlock("BirthDate").IsGreater(Column.Parameter(year))
            .Or("BirthDate").IsNull()
        .CloseBlock()
        .And("AccountId").Not().IsNull()
        .OrderByDesc("BirthDate")
            as Select;
    using (DBExecutor dbExecutor = UserConnection.EnsureDBConnection())
    {
        using (IDataReader dataReader = select.ExecuteReader(dbExecutor))
        {
            result = CreateJson(dataReader);
        }
    }
    return result;
}

```

Пример 4

Пример. Выбрать идентификатор и имя всех контактов, присоединив к ним идентификаторы и названия соответствующих контрагентов.

Метод SelectContactsJoinAccount

```
public string SelectContactsJoinAccount()
{
    var result = "{}";
    var select = new Select(UserConnection)
        .Column("Contact", "Id").As("ContactId")
        .Column("Contact", "Name").As("ContactName")
        .Column("Account", "Id").As("AccountId")
        .Column("Account", "Name").As("AccountName")
        .From("Contact")
        .Join(JoinType.Inner, "Account")
        .On("Contact", "Id").IsEqual("Account", "PrimaryContactId")
        as Select;
    using (DBExecutor dbExecutor = UserConnection.EnsureDBConnection())
    {
        using (IDataReader dataReader = select.ExecuteReader(dbExecutor))
        {
            result = CreateJson(dataReader);
        }
    }
    return result;
}
```

Пример 5

Пример. Выбрать идентификатор и имя контактов, являющихся основными для контрагентов.

Метод SelectAccountPrimaryContacts

```
public string SelectAccountPrimaryContacts()
{
    var result = "{}";
    var select = new Select(UserConnection)
        .Column("Id")
        .Column("Name")
```

```

.From("Contact").As("C")
.Where()
.Exists(new Select(UserConnection)
    .Column("A", "PrimaryContactId")
    .From("Account").As("A")
    .Where("A", "PrimaryContactId").IsEqual("C", "Id"))
        as Select;
using (DBExecutor dbExecutor = UserConnection.EnsureDBConnection())
{
    using (IDataReader dataReader = select.ExecuteReader(dbExecutor))
    {
        result = CreateJson(dataReader);
    }
}
return result;
}

```

Пример 6

Пример. Выбрать страны и количество городов в стране, если количество городов больше указанного.

Метод SelectCountriesWithCitiesCount

```

public string SelectCountriesWithCitiesCount(int count)
{
    var result = "{}";
    var select = new Select(UserConnection)
        .Column(Func.Count("City", "Id")).As("CitiesCount")
        .Column("Country", "Name").As("CountryName")
        .From("City")
        .Join(JoinType.Inner, "Country")
            .On("City", "CountryId").IsEqual("Country", "Id")
        .GroupBy("Country", "Name")
        .Having(Func.Count("City", "Id")).IsGreater(Column.Parameter(count))
        .OrderByDesc("CitiesCount")
            as Select;
    using (DBExecutor dbExecutor = UserConnection.EnsureDBConnection())
    {
        using (IDataReader dataReader = select.ExecuteReader(dbExecutor))
        {
            result = CreateJson(dataReader);
        }
    }
}

```

```
    return result;
}
```

Пример 7

Пример. Получить идентификатор контакта по его имени.

Метод SelectCountryIdByCityName

```
public string SelectCountryIdByCityName(string CityName)
{
    var result = "";
    var select = new Select(UserConnection)
        .Column("CountryId")
        .From("City")
        .Where("Name").IsEqual(Column.Parameter(CityName)) as Select;
    result = select.ExecuteScalar<Guid>().ToString();
    return result;
}
```

Добавить данные в базу данных



Сложный

На заметку. Примеры, приведенные в этой статье, реализованы в веб-сервисе. Пакет с реализацией веб-сервиса прикреплен в блоке "Ресурсы".

Пример 1

Пример. Добавить контакт с указанным именем.

Метод InsertContact

```
public string InsertContact(string contactName)
{
    contactName = contactName ?? "Unknown contact";
    var ins = new Insert(UserConnection)
        .Into("Contact")
```

```

    .Set("Name", Column.Parameter(contactName));
var affectedRows = ins.Execute();
var result = $"Inserted new contact with name '{contactName}'. {affectedRows} rows affected"
return result;
}

```

Пример 2

Пример. Добавить город с указанным названием, привязав его к указанной стране.

Метод InsertCity

```

public string InsertCity(string city, string country)
{
    city = city ?? "unknown city";
    country = country ?? "unknown country";

    var ins = new Insert(UserConnection)
        .Into("City")
        .Set("Name", Column.Parameter(city))
        .Set("CountryId",
            new Select(UserConnection)
                .Top(1)
                .Column("Id")
                .From("Country")
                .Where("Name")
                .IsEqual(Column.Parameter(country)));
    var affectedRows = ins.Execute();
    var result = $"Inserted new city with name '{city}' located in '{country}'. {affectedRows} r
    return result;
}

```

Добавить данные в базу данных с помощью подзапросов



Сложный

На заметку. Примеры, приведенные в этой статье, реализованы в веб-сервисе. Пакет с реализацией веб-сервиса прикреплен в блоке "Ресурсы".

Пример 1

Пример. Добавить контакт с указанными именем и названием контрагента.

Метод InsertContactWithAccount

```
public string InsertContactWithAccount(string contactName, string accountName)
{
    contactName = contactName ?? "Unknown contact";
    accountName = accountName ?? "Unknown account";

    var id = Guid.NewGuid();
    var selectQuery = new Select(UserConnection)
        .Column(Column.Parameter(contactName))
        .Column("Id")
        .From("Account")
        .Where("Name").IsEqual(Column.Parameter(accountName)) as Select;
    var insertSelectQuery = new InsertSelect(UserConnection)
        .Into("Contact")
        .Set("Name", "AccountId")
        .FromSelect(selectQuery);

    var affectedRows = insertSelectQuery.Execute();
    var result = $"Inserted new contact with name '{contactName}'" +
        $" and account '{accountName}'. " +
        $" Affected {affectedRows} rows.";
    return result;
}
```

Пример 2

Пример. Добавить контакт с указанным именем, связав его со всеми контрагентами.

Метод InsertAllAccountsContact

```
public string InsertAllAccountsContact(string contactName)
{
    contactName = contactName ?? "Unknown contact";

    var id = Guid.NewGuid();
    var insertSelectQuery = new InsertSelect(UserConnection)
```

```

    .Into("Contact")
    .Set("Name", "AccountId")
    .FromSelect(
        new Select(UserConnection)
            .Column(Column.Parameter(contactName))
            .Column("Id")
            .From("Account") as Select);

    var affectedRows = insertSelectQuery.Execute();
    var result = $"Inserted {affectedRows} new contacts with name '{contactName}'";
    return result;
}

```

Изменить данные в базе данных



Сложный

На заметку. Примеры, приведенные в этой статье, реализованы в веб-сервисе. Пакет с реализацией веб-сервиса прикреплен в блоке "Ресурсы".

В большинстве случаев запрос на изменение данных должен содержать условие `Where`, которое уточняет какие именно записи необходимо изменить. Если не указать условие `Where`, то будут изменены все записи.

Пример 1

Пример. Изменить имя контакта.

Метод `ChangeContactName`

```

public string ChangeContactName(string oldName, string newName)
{
    var update = new Update(UserConnection, "Contact")
        .Set("Name", Column.Parameter(newName))
        .Where ("Name").IsEqual(Column.Parameter(oldName));
    var cnt = update.Execute();
    return $"Contacts {oldName} changed to {newName}. {cnt} rows affected.";
}

```

Пример 2

Пример. Для всех существующих контактов поменять пользователя, изменившего запись, на указанного.

Метод ChangeAllContactModifiedBy

```
public string ChangeAllContactModifiedBy(string Name)
{
    var update = new Update(UserConnection, "Contact")
        .Set("ModifiedById",
            new Select(UserConnection).Top(1)
                .Column("Id")
                .From("Contact")
                .Where("Name").IsEqual(Column.Parameter(Name)));
    var cnt = update.Execute();
    return $"All contacts are changed by {Name} now. {cnt} rows affected.";
}
```

Условие `Where` относится к запросу `Select`. Запрос `Update` не содержит условия `Where`, поскольку необходимо изменить все записи.

Удалить данные из базы данных



Сложный

На заметку. Примеры, приведенные в этой статье, реализованы в веб-сервисе. Пакет с реализацией веб-сервиса прикреплен в блоке "Ресурсы".

В большинстве случаев запрос на удаление данных должен содержать условие `Where`, которое уточняет какие именно записи необходимо удалить. Если не указать условие `Where`, то будут удалены все записи.

Пример

Пример. Удалить контакт с указанным именем.

Метод DeleteContacts

```
public string DeleteContacts(string name)
{
    var delete = new Delete(UserConnection)
        .From("Contact")
```

```

    .Where("Name").IsEqual(Column.Parameter(name)));
var cnt = delete.Execute();
return $"Contacts with name {name} were deleted. {cnt} rows affected";
}

```

Класс Select C#



Сложный

Пространство имен `Terrasoft.Core.DB`.

Класс `Terrasoft.Core.DB.Select` предназначен для построения запросов выборки записей из таблиц базы данных. В результате создания и конфигурирования экземпляра этого класса будет построен запрос в базу данных приложения в виде SQL-выражения `SELECT`. В запрос можно добавить требуемые колонки, фильтры и условия ограничений. Результаты выполнения запроса возвращаются в виде экземпляра, реализующего интерфейс `System.Data.IDataReader`, либо скалярного значения требуемого типа.

Важно. При работе с классом `Select` не учитываются права доступа пользователя, использующего текущее соединение. Доступны абсолютно все записи из базы данных приложения. Также не учитываются данные, помещенные в [хранилище кэша](#). Если необходимы дополнительные возможности по управлению правами доступа и работе с хранилищем кэша Creatio, следует использовать класс `EntitySchemaQuery`.

На заметку. Полный перечень методов и свойств класса `Select`, его родительских классов, а также реализуемых им интерфейсов можно найти в [Библиотеке .NET классов](#).

Конструкторы

`Select(UserConnection userConnection)`

Создает экземпляр класса с указанным `UserConnection`.

`Select(UserConnection userConnection, CancellationToken cancellationToken)`

Создает экземпляр класса с указанным `UserConnection` и токеном отмены [выполнения управляемого потока](#).

`Select(Select source)`

Создает экземпляр класса, являющийся клоном экземпляра, переданного в качестве аргумента.

Свойства

UserConnection Terrasoft.Core.UserConnection

Пользовательское подключение, используемое при выполнении запроса.

RowCount int

Количество записей, которые вернет запрос после выполнения.

Parameters Terrasoft.Core.DB.QueryParameterCollection

Коллекция параметров запроса.

HasParameters bool

Определяет наличие параметров у запроса.

BuildParametersAsValue bool

Определяет, добавлять ли параметры запроса в текст запроса как значения.

Joins Terrasoft.Core.DB.JoinCollection

Коллекция выражений **Join** в запросе.

HasJoins bool

Определяет наличие выражений **Join** в запросе.

Condition Terrasoft.Core.DB.QueryCondition

Условие выражения **Where** запроса.

HasCondition bool

Определяет наличие выражения **Where** в запросе.

HavingCondition Terrasoft.Core.DB.QueryCondition

Условие выражения **Having** запроса.

`HasHavingCondition bool`

Определяет наличие выражения `Having` в запросе.

`OrderByItems Terrasoft.Core.DB.OrderByItemCollection`

Коллекция выражений, по которым выполняется сортировка результатов запроса.

`HasOrderByItems bool`

Определяет наличие условий сортировки результатов запроса.

`GroupByItems Terrasoft.Core.DB.QueryColumnExpressionCollection`

Коллекция выражений, по которым выполняется группировка результатов запроса.

`HasGroupByItems bool`

Определяет наличие условий группировки результатов запроса.

`IsDistinct bool`

Определяет, должен ли запрос возвращать только уникальные записи.

`Columns Terrasoft.Core.DB.QueryColumnExpressionCollection`

Коллекция выражений колонок запроса.

`OffsetFetchPaging bool`

Определяет возможность постраничного возврата результата запроса.

`RowsOffset int`

Количество строк, которые необходимо пропустить при возврате результата запроса.

`QueryKind Terrasoft.Common.QueryKind`

Тип запроса (см. статью [Настройка отдельного пула запросов](#)).

Методы

```
void ResetCachedSqlText()
```

Очищает закэшированный текст запроса.

```
QueryParameterCollection GetUsingParameters()
```

Возвращает коллекцию параметров, используемых запросом.

```
void ResetParameters()
```

Очищает коллекцию параметров запроса.

```
QueryParameterCollection InitializeParameters()
```

Инициализирует коллекцию параметров запроса.

```
IDataReader ExecuteReader(DBExecutor dbExecutor)
```

Выполняет запрос, используя экземпляр `DBExecutor`. Возвращает объект, реализующий интерфейс `IDataReader`.

Параметры

dbExecutor	Экземпляр <code>DBExecutor</code> , который используется для выполнения запроса.
------------	--

```
IDataReader ExecuteReader(DBExecutor dbExecutor, CommandBehavior behavior)
```

Выполняет запрос, используя экземпляр `DBExecutor`. Возвращает объект, реализующий интерфейс `IDataReader`.

Параметры

behavior	Предоставляет описание результатов запроса и их эффект на базу данных.
----------	--

| dbExecutor | Экземпляр `DBExecutor`, который используется для выполнения запроса. |

```
void ExecuteReader(ExecuteReaderReadMethod readMethod)
```

Выполняет запрос, вызывая переданный метод делегата `ExecuteReaderReadMethod` для каждой записи результирующего набора.

Параметры

readMethod	Метод делегата <code>ExecuteReaderReadMethod</code> .
------------	---

`TResult ExecuteScalar<TResult>()`

Выполняет запрос. Возвращает типизированный первый столбец первой записи результирующего набора.

`TResult ExecuteScalar<TResult>(DBExecutor dbExecutor)`

Выполняет запрос, используя экземпляр `DBExecutor`. Возвращает типизированный первый столбец первой записи результирующего набора.

Параметры

dbExecutor	Экземпляр <code>DBExecutor</code> , который используется для выполнения запроса.
------------	--

`Select Distinct()`

Добавляет к SQL-запросу ключевое слово `DISTINCT`. Исключает дублирование записей в результирующем наборе. Возвращает экземпляр запроса.

`Select Top(int rowCount)`

Устанавливает количество записей, возвращаемых в результирующем наборе. При этом меняется значение свойства `RowCount`. Возвращает экземпляр запроса.

Параметры

rowCount	Количество первых записей результирующего набора.
----------	---

`Select As(string alias)`

Добавляет указанный в аргументе псевдоним для последнего выражения запроса. Возвращает экземпляр запроса.

Параметры

alias	Псевдоним выражения запроса.
-------	------------------------------

```

Select Column(string sourceColumnAlias)
Select Column(string sourceAlias, string sourceColumnAlias)
Select Column(Select subSelect)
Select Column(Query subSelectQuery)
Select Column(QueryCase queryCase)
Select Column(QueryParameter queryParameter)
Select Column(QueryColumnExpression columnExpression)

```

Добавляет выражение, подзапрос или параметр в коллекцию выражений колонок запроса.
Возвращает экземпляр запроса.

Параметры

sourceColumnAlias	Псевдоним колонки, для которой добавляется выражение.
sourceAlias	Псевдоним источника, из которого добавляется выражение колонки.
subSelect	Добавляемый подзапрос выборки данных.
subSelectQuery	Добавляемый подзапрос.
queryCase	Добавляемое выражение для оператора <code>Case</code> .
queryParameter	Добавляемый параметр запроса.
columnExpression	Выражение, для результатов которого добавляется условие.

```

Select From(string schemaName)
Select From(Select subSelect)
Select From(Query subSelectQuery)
Select From(QuerySourceExpression sourceExpression)

```

Добавляет в запрос источник данных. Возвращает экземпляр запроса.

Параметры

schemaName	Название схемы.
subSelect	Подзапрос выборки, результаты которого становятся источником данных для текущего запроса.
subSelectQuery	Подзапрос, результаты которого становятся источником данных для текущего запроса.
sourceExpression	Выражение источника данных запроса.

```
Join Join(JoinType joinType, string schemaName)
Join Join(JoinType joinType, Select subSelect)
Join Join(JoinType joinType, Query subSelectQuery)
Join Join(JoinType joinType, QuerySourceExpression sourceExpression)
```

Присоединяет к текущему запросу схему, подзапрос или выражение.

Параметры

joinType	Тип присоединения.
schemaName	Название присоединяемой схемы.
subSelect	Присоединяемый подзапрос выборки данных.
subSelectQuery	Присоединяемый подзапрос.
sourceExpression	Присоединяемое выражение.

Возможные значения ([Terrasoft.Core.DB.JoinType](#))

Inner	Внутреннее соединение.
LeftOuter	Левое внешнее соединение.
RightOuter	Правое внешнее соединение.
FullOuter	Полное соединение.
Cross	Перекрестное соединение.

```
QueryCondition Where()
QueryCondition Where(string sourceColumnAlias)
```

```
QueryCondition Where(string sourceAlias, string sourceColumnAlias)
QueryCondition Where(Select subSelect)
QueryCondition Where(Query subSelectQuery)
QueryCondition Where(QueryColumnExpression columnExpression)
Query Where(QueryCondition condition)
```

Добавляет к текущему запросу начальное условие.

[Параметры](#)

sourceColumnAlias	Псевдоним колонки, для которой добавляется условие.
sourceAlias	Псевдоним источника.
subSelect	Подзапрос выборки данных, для результатов которого добавляется условие.
subSelectQuery	Подзапрос, для результатов которого добавляется условие.
columnExpression	Выражение, для результатов которого добавляется условие.
condition	Условие запроса.

```
QueryCondition And()
QueryCondition And(string sourceColumnAlias)
QueryCondition And(string sourceAlias, string sourceColumnAlias)
QueryCondition And(Select subSelect)
QueryCondition And(Query subSelectQuery)
QueryCondition And(QueryParameter parameter)
QueryCondition And(QueryColumnExpression columnExpression)
Query And(QueryCondition condition)
```

К текущему условию запроса добавляет условие (предикат), используя логическую операцию И.

[Параметры](#)

sourceColumnAlias	Псевдоним колонки, для которой добавляется предикат.
sourceAlias	Псевдоним источника.
subSelect	Подзапрос выборки данных, используемый в качестве предиката.
subSelectQuery	Подзапрос, используемый в качестве предиката.
parameter	Параметр, для которого добавляется предикат.
columnExpression	Выражение, используемое в качестве предиката.
condition	Условие запроса.

```

QueryCondition Or()
QueryCondition Or(string sourceColumnAlias)
QueryCondition Or(string sourceAlias, string sourceColumnAlias)
QueryCondition Or(Select subSelect)
QueryCondition Or(Query subSelectQuery)
QueryCondition Or(QueryParameter parameter)
QueryCondition Or(QueryColumnExpression columnExpression)
Query Or(QueryCondition condition)

```

К текущему условию запроса добавляет условие (предикат), используя логическую операцию ИЛИ.

Параметры

sourceColumnAlias	Псевдоним колонки, для которой добавляется предикат.
sourceAlias	Псевдоним источника.
subSelect	Подзапрос на выборку данных, используемый в качестве предиката.
subSelectQuery	Подзапрос, используемый в качестве предиката.
parameter	Параметр, для которого добавляется предикат.
columnExpression	Выражение, используемое в качестве предиката.
condition	Условие запроса.

Query OrderBy(OrderDirectionStrict direction, string sourceColumnAlias)

```

Query OrderBy(OrderDirectionStrict direction, string sourceColumnAlias)
Query OrderBy(OrderDirectionStrict direction, QueryFunction queryFunction)
Query OrderBy(OrderDirectionStrict direction, Select subSelect)
Query OrderBy(OrderDirectionStrict direction, Query subSelectQuery)
Query OrderBy(OrderDirectionStrict direction, QueryColumnExpression columnExpression)

```

Выполняет сортировку результатов запроса. Возвращает экземпляр запроса.

Параметры

direction	Порядок сортировки.
sourceColumnAlias	Псевдоним колонки, по которой выполняется сортировка.
sourceAlias	Псевдоним источника.
queryFunction	Функция, значение которой используется в качестве ключа сортировки.
subSelect	Подзапрос на выборку данных, результаты которого используются в качестве ключа сортировки.
subSelectQuery	Подзапрос, результаты которого используются в качестве ключа сортировки.
columnExpression	Выражение, результаты которого используются в качестве ключа сортировки.

```

Query OrderByAsc(string sourceColumnAlias)
Query OrderByAsc(string sourceAlias, string sourceColumnAlias)
Query OrderByAsc(Select subSelect)
Query OrderByAsc(Query subSelectQuery)
Query OrderByAsc(QueryColumnExpression columnExpression)

```

Выполняет сортировку результатов запроса в порядке возрастания. Возвращает экземпляр запроса.

Параметры

sourceColumnAlias	Псевдоним колонки, по которой выполняется сортировка.
sourceAlias	Псевдоним источника.
subSelect	Подзапрос на выборку данных, результаты которого используются в качестве ключа сортировки.
subSelectQuery	Подзапрос, результаты которого используются в качестве ключа сортировки.
columnExpression	Выражение, результаты которого используются в качестве ключа сортировки.

```
Query OrderByDesc(string sourceColumnAlias)
Query OrderByDesc(string sourceAlias, string sourceColumnAlias)
Query OrderByDesc(Select subSelect)
Query OrderByDesc(Query subSelectQuery)
Query OrderByDesc(QueryColumnExpression columnExpression)
```

Выполняет сортировку результатов запроса в порядке убывания. Возвращает экземпляр запроса.

Параметры

sourceColumnAlias	Псевдоним колонки, по которой выполняется сортировка.
sourceAlias	Псевдоним источника.
subSelect	Подзапрос выборки, результаты которого используются в качестве ключа сортировки.
subSelectQuery	Подзапрос, результаты которого используются в качестве ключа сортировки.
columnExpression	Выражение, результаты которого используются в качестве ключа сортировки.

```
Query GroupBy(string sourceColumnAlias)
Query GroupBy(string sourceAlias, string sourceColumnAlias)
Query GroupBy(QueryColumnExpression columnExpression)
```

Выполняет группировку результатов запроса. Возвращает экземпляр запроса.

Параметры

sourceColumnAlias	Псевдоним колонки, по которой выполняется группировка.
sourceAlias	Псевдоним источника.
columnExpression	Выражение, результаты которого используются в качестве ключа группировки.

```
QueryCondition Having()
QueryCondition Having(string sourceColumnAlias)
QueryCondition Having(string sourceAlias, string sourceColumnAlias)
QueryCondition Having(Select subSelect)
QueryCondition Having(Query subSelectQuery)
QueryCondition Having(QueryParameter parameter)
QueryCondition Having(QueryColumnExpression columnExpression)
```

Добавляет в текущий запрос групповое условие. Возвращает экземпляр

`Terrasoft.Core.DB.QueryCondition`, представляющий групповое условие для параметра запроса.

Параметры

sourceColumnAlias	Псевдоним колонки, по которой добавляется групповое условие.
sourceAlias	Псевдоним источника.
subSelect	Подзапрос выборки, для результатов которого добавляется групповое условие.
subSelectQuery	Подзапрос, для результатов которого добавляется групповое условие.
parameter	Параметр запроса, для которого добавляется групповое условие.
columnExpression	Выражение, используемое в качестве предиката.

```
Query Union(Select unionSelect)
Query Union(Query unionSelectQuery)
```

Объединяет результаты переданного запроса с результатами текущего запроса, исключая дубликаты из результирующего набора.

Параметры

unionSelect	Подзапрос выборки.
unionSelectQuery	Подзапрос.

```
Query UnionAll(Select unionSelect)
Query UnionAll(Query unionSelectQuery)
```

Объединяет результаты переданного запроса с результатами текущего запроса. Дубликаты из результирующего набора не исключаются.

Параметры

unionSelect	Подзапрос выборки.
unionSelectQuery	Подзапрос.

Класс Insert



Сложный

Пространство имен `Terrasoft.Core.DB`.

Класс `Terrasoft.Core.DB.Insert` предназначен для построения запросов на добавление записей в таблицы базы данных Creatio. В результате создания и конфигурирования экземпляра этого класса будет построен запрос в базу данных приложения в виде SQL-выражения `INSERT`. В результате выполнения запроса возвращается количество задействованных записей.

Важно. При работе с классом `Insert` на добавленные записи не применяются права доступа по умолчанию. Пользовательское соединение используется только для доступа к таблице базы данных.

На заметку. Полный перечень методов и свойств класса `Insert`, его родительских классов, а также реализуемых им интерфейсов, можно найти в [Библиотеке .NET классов](#).

Конструкторы

```
Entity(UserConnection userConnection)
```

Создает новый экземпляр класса `Entity` для заданного пользовательского подключения `UserConnection`.

`Insert(UserConnection userConnection)`

Создает экземпляр класса с указанным `UserConnection`.

`Insert(Insert source)`

Создает экземпляр класса, являющийся клоном экземпляра, переданного в качестве аргумента.

Свойства

`UserConnection Terrasoft.Core.UserConnection`

Пользовательское подключение, используемое при запросе.

`Source Terrasoft.Core.DB.ModifyQuerySource`

Источник данных

`Parameters Terrasoft.Core.DB.QueryParameterCollection`

Коллекция параметров запроса.

`HasParameters bool`

Определяет, имеет ли запрос параметры.

`BuildParametersAsValue bool`

Определяет, добавлять ли параметры запроса в текст запроса как значения.

`ColumnValues Terrasoft.Core.DB.ModifyQueryColumnValueCollection`

Коллекция значений колонок запроса.

`ColumnValuesCollection List<ModifyQueryColumnValueCollection>`

Коллекция значений колонок для множественного добавления записей.

Методы

`void ResetCachedSqlText()`

Очищает кэшированный текст запроса.

```
QueryParameterCollection GetUsingParameters()
```

Возвращает коллекцию параметров, используемых запросом.

```
void ResetParameters()
```

Очищает коллекцию параметров запроса.

```
void SetParameterValue(string name, object value)
```

Устанавливает значение для параметра запроса.

Параметры

<code>name</code>	Название параметра.
<code>value</code>	Значение.

```
void InitializeParameters()
```

Инициализирует коллекцию параметров запроса.

```
int Execute()
```

Выполняет запрос. Возвращает количество задействованных запросом записей.

```
int Execute(DBExecutor dbExecutor)
```

Выполняет запрос, используя экземпляр `DBExecutor`. Возвращает количество задействованных запросом записей.

```
Insert Into(string schemaName)
```

```
Insert Into(ModifyQuerySource source)
```

Добавляет в текущий запрос источник данных.

Параметры

<code>schemaName</code>	Название схемы.
<code>source</code>	Источник данных.

```
Insert Set(string sourceColumnAlias, Select subSelect)
Insert Set(string sourceColumnAlias, Query subSelectQuery)
Insert Set(string sourceColumnAlias, QueryColumnExpression columnExpression)
Insert Set(string sourceColumnAlias, QueryParameter parameter)
```

Добавляет в текущий запрос предложение `SET` для присвоения колонке переданного выражения или параметра. Возвращает текущий экземпляр `Insert`.

Параметры

<code>sourceColumnAlias</code>	Псевдоним колонки.
<code>subSelect</code>	Подзапрос на выборку.
<code>subSelectQuery</code>	Подзапрос.
<code>columnExpression</code>	Выражение колонки.
<code>parameter</code>	Параметр запроса.

Insert Values()

Инициализирует значения для множественного добавления записей.

Класс InsertSelect C#



Сложный

Пространство имен `Terrasoft.Core.DB`.

Класс `Terrasoft.Core.DB.InsertSelect` предназначен для построения запросов на добавление записей в таблицы базы данных Creatio. При этом в качестве источника добавляемых данных используется экземпляр класса `Terrasoft.Core.DB.Select`. В результате создания и конфигурирования экземпляра `Terrasoft.Core.DB.InsertSelect` будет построен запрос базы данных приложения в виде SQL-выражения `INSERT INTO SELECT`.

Важно. При работе с классом `InsertSelect` на добавленные записи не применяются права доступа по умолчанию. К таким записям не применены вообще никакие права приложения (по операциям на объект, по записям, по колонкам). Пользовательское соединение используется только для доступа к таблице базы данных.

На заметку. После выполнения запроса `InsertSelect` в базу данных будет добавлено столько записей, сколько вернется в его подзапросе `Select`.

На заметку. Полный перечень методов и свойств класса `InsertSelect`, его родительских классов, а также реализуемых им интерфейсов, можно найти в [Библиотеке .NET классов](#).

Конструкторы

`InsertSelect(UserConnection userConnection)`

Создает экземпляр класса с указанным `UserConnection`.

`InsertSelect(InsertSelect source)`

Создает экземпляр класса, являющийся клоном экземпляра, переданного в качестве аргумента.

Свойства

`UserConnection` `Terrasoft.Core.UserConnection`

Пользовательское подключение, используемое при запросе.

`Source` `Terrasoft.Core.DB.ModifyQuerySource`

Источник данных

`Parameters` `Terrasoft.Core.DB.QueryParameterCollection`

Коллекция параметров запроса.

`HasParameters` `bool`

Определяет, имеет ли запрос параметры.

`BuildParametersAsValue` `bool`

Определяет, добавлять ли параметры запроса в текст запроса как значения.

`Columns` `Terrasoft.Core.DB.ModifyQueryColumnValueCollection`

Коллекция значений колонок запроса.

`Select` `Terrasoft.Core.DB.Select`

Используемый в запросе экземпляр `Terrasoft.Core.DB.Select`.

Методы

`void ResetCachedSqlText()`

Очищает кэшированный текст запроса.

`QueryParameterCollection GetUsingParameters()`

Возвращает коллекцию параметров, используемых запросом.

`void ResetParameters()`

Очищает коллекцию параметров запроса.

`void SetParameterValue(string name, object value)`

Устанавливает значение для параметра запроса.

Параметры

<code>name</code>	Название параметра.
<code>value</code>	Значение.

`void InitializeParameters()`

Инициализирует коллекцию параметров запроса.

`int Execute()`

Выполняет запрос. Возвращает количество задействованных запросом записей.

`int Execute(DBExecutor dbExecutor)`

Выполняет запрос, используя экземпляр `DBExecutor`. Возвращает количество задействованных запросом записей.

`InsertSelect Into(string schemaName)`

`InsertSelect Into(ModifyQuerySource source)`

Добавляет в текущий запрос источник данных.

Параметры

schemaName	Название схемы.
source	Источник данных.

```
InsertSelect Set(IEnumerable<string> sourceColumnAliases)
InsertSelect Set(params string[] sourceColumnAliases)
InsertSelect Set(IEnumerable<ModifyQueryColumn> columns)
InsertSelect Set(params ModifyQueryColumn[] columns)
```

Добавляет в текущий запрос набор колонок, в которые будут добавлены значения с помощью подзапроса. Возвращает текущий экземпляр `InsertSelect`.

Параметры

sourceColumnAliases	Коллекция или массив параметров метода, содержащие псевдонимы колонок.
columns	Коллекция или массив параметров метода, содержащие экземпляры колонок.

```
InsertSelect FromSelect(Select subSelect)
InsertSelect FromSelect(Query subSelectQuery)
```

Добавляет в текущий запрос предложение `SELECT`.

Параметры

subSelect	Подзапрос на выборку.
subSelectQuery	Подзапрос.

Класс Update



Сложный

Пространство имен `Terrasoft.Core.DB`.

Класс `Terrasoft.Core.DB.Update` предназначен для построения запросов на изменение записей в таблице базы данных `Creatio`. В результате создания и конфигурирования экземпляра этого класса будет построен запрос базу данных приложения в виде SQL-выражения `UPDATE`.

На заметку. Полный перечень методов и свойств класса `Update`, его родительских классов, а также реализуемых им интерфейсов, можно найти в [Библиотеке .NET классов](#).

Конструкторы

`Update(UserConnection userConnection)`

Создает экземпляр класса, используя `UserConnection`.

`Update(UserConnection userConnection, string schemaName)`

Создает экземпляр класса для схемы с указанным названием, используя `UserConnection`.

`Update(UserConnection userConnection, ModifyQuerySource source)`

Создает экземпляр класса для указанного источника данных, используя `UserConnection`.

`Update(Insert source)`

Создает экземпляр класса, являющийся клоном экземпляра, переданного в качестве аргумента.

Свойства

`UserConnection` `Terrasoft.Core.UserConnection`

Пользовательское подключение, используемое при выполнении запроса.

`Condition` `Terrasoft.Core.DB.QueryCondition`

Условие выражения `Where` запроса.

`HasCondition` `bool`

Определяет наличие выражения `Where` в запросе.

`Source` `Terrasoft.Core.DB.ModifyQuerySource`

Источник данных запроса.

`ColumnValues` `Terrasoft.Core.DB.ModifyQueryColumnValueCollection`

Коллекция значений колонок запроса.

Методы

`void ResetCachedSqlText()`

Очищает закэшированный текст запроса.

`QueryParameterCollection GetUsingParameters()`

Возвращает коллекцию параметров, используемых запросом.

`int Execute()`

Выполняет запрос. Возвращает количество задействованных запросом записей.

`int Execute(DBExecutor dbExecutor)`

Выполняет запрос, используя экземпляр `DBExecutor`. Возвращает количество задействованных запросом записей.

`QueryCondition Where()`
`QueryCondition Where(string sourceColumnAlias)`
`QueryCondition Where(string sourceAlias, string sourceColumnAlias)`
`QueryCondition Where(Select subSelect)`
`QueryCondition Where(Query subSelectQuery)`
`QueryCondition Where(QueryColumnExpression columnExpression)`
`Query Where(QueryCondition condition)`

Добавляет к текущему запросу начальное условие.

Параметры

sourceColumnAlias	Псевдоним колонки, для которой добавляется условие.
sourceAlias	Псевдоним источника.
subSelect	Подзапрос выборки данных, для результатов которого добавляется условие.
subSelectQuery	Подзапрос, для результатов которого добавляется условие.
columnExpression	Выражение, для результатов которого добавляется условие.
condition	Условие запроса.

```
QueryCondition And()
QueryCondition And(string sourceColumnAlias)
QueryCondition And(string sourceAlias, string sourceColumnAlias)
QueryCondition And(Select subSelect)
QueryCondition And(Query subSelectQuery)
QueryCondition And(QueryParameter parameter)
QueryCondition And(QueryColumnExpression columnExpression)
Query And(QueryCondition condition)
```

К текущему условию запроса добавляет условие (предикат), используя логическую операцию И.

Параметры

sourceColumnAlias	Псевдоним колонки, для которой добавляется предикат.
sourceAlias	Псевдоним источника.
subSelect	Подзапрос выборки данных, используемый в качестве предиката.
subSelectQuery	Подзапрос, используемый в качестве предиката.
parameter	Параметр, для которого добавляется предикат.
columnExpression	Выражение, используемое в качестве предиката.
condition	Условие запроса.

```
QueryCondition Or()
QueryCondition Or(string sourceColumnAlias)
QueryCondition Or(string sourceAlias, string sourceColumnAlias)
```

```
QueryCondition Or(Select subSelect)
QueryCondition Or(Query subSelectQuery)
QueryCondition Or(QueryParameter parameter)
QueryCondition Or(QueryColumnExpression columnExpression)
Query Or(QueryCondition condition)
```

К текущему условию запроса добавляет условие (предикат), используя логическую операцию ИЛИ.

Параметры

sourceColumnAlias	Псевдоним колонки, для которой добавляется предикат.
sourceAlias	Псевдоним источника.
subSelect	Подзапрос на выборку данных, используемый в качестве предиката.
subSelectQuery	Подзапрос, используемый в качестве предиката.
parameter	Параметр, для которого добавляется предикат.
columnExpression	Выражение, используемое в качестве предиката.
condition	Условие запроса.

```
Update Set(string sourceColumnAlias, Select subSelect)
Update Set(string sourceColumnAlias, Query subSelectQuery)
Update Set(string sourceColumnAlias, QueryColumnExpression columnExpression)
Update Set(string sourceColumnAlias, QueryParameter parameter)
```

Добавляет в текущий запрос предложение `SET` для присвоения колонке переданного выражения или параметра. Возвращает текущий экземпляр `Update`.

Параметры

sourceColumnAlias	Псевдоним колонки.
subSelect	Подзапрос на выборку.
subSelectQuery	Подзапрос.
columnExpression	Выражение колонки.
parameter	Параметр запроса.

Класс UpdateSelect [c#](#)



Сложный

Пространство имен `Terrasoft.Core.DB`.

Класс `Terrasoft.Core.DB.UpdateSelect` предназначен для построения запросов на изменение записей в таблице базы данных Creatio. При этом в качестве источника добавляемых данных используется экземпляр класса `Terrasoft.Core.DB.Select`. В результате создания и конфигурирования экземпляра этого класса будет построен запрос в базу данных приложения в виде SQL-выражения `UPDATE FROM`.

На заметку. Полный перечень методов и свойств класса `UpdateSelect`, его родительских классов, а также реализуемых им интерфейсов, можно найти в [Библиотеке .NET классов](#).

Конструкторы

```
public UpdateSelect(UserConnection userConnection, string schemaName, string alias)
```

Создает экземпляр класса для схемы с указанным названием, используя `UserConnection`.

Параметры

<code>userConnection</code>	Пользовательское подключение, используемое при запросе.
<code>schemaName</code>	Название схемы.
<code>alias</code>	Псевдоним таблицы.

Свойства

`SourceAlias` `string`

Псевдоним таблицы данных, в которую вносятся изменения.

`SourceExpression` `Terrasoft.Core.DB.QuerySourceExpression`

Выражение для `SELECT`-запроса.

Методы

```
public UpdateSelect From(string schemaName, string alias)
```

Добавляет к запросу `FROM`-выражение.

Параметры

<code>schemaName</code>	Название таблицы, в которую вносятся изменения.
<code>alias</code>	Псевдоним таблицы, в которую вносятся изменения.

```
public UpdateSelect Set(string sourceColumnAlias, QueryColumnExpression columnExpression)
```

Добавляет к запросу `SET`-выражение для колонки.

Параметры

<code>sourceColumnAlias</code>	Псевдоним колонки.
<code>columnExpression</code>	Выражение, содержащее значение колонки.

Класс Delete C#



Сложный

Пространство имен `Terrasoft.Core.DB`.

Класс `Terrasoft.Core.DB.Delete` предназначен для построения запросов на удаление записей в таблице базы данных `Creatio`. В результате создания и конфигурирования экземпляра этого класса будет построен запрос базы данных приложения в виде SQL-выражения `DELETE`.

На заметку. Полный перечень методов и свойств класса `Delete`, его родительских классов, а также реализуемых им интерфейсов, можно найти в [Библиотеке .NET классов](#).

Конструкторы

```
Delete(UserConnection userConnection)
```

Создает экземпляр класса, используя `UserConnection`.

```
Delete(Delete source)
```

Создает экземпляр класса, являющийся клоном экземпляра, переданного в качестве аргумента.

Свойства

```
UserConnection Terrasoft.Core.UserConnection
```

Пользовательское подключение, используемое при выполнении запроса.

```
Condition Terrasoft.Core.DB.QueryCondition
```

Условие выражения `Where` запроса.

```
HasCondition bool
```

Определяет наличие выражения `Where` в запросе.

```
Source Terrasoft.Core.DB.ModifyQuerySource
```

Источник данных запроса.

Методы

```
void ResetCachedSqlText()
```

Очищает закэшированный текст запроса.

```
QueryParameterCollection GetUsingParameters()
```

Возвращает коллекцию параметров, используемых запросом.

```
int Execute()
```

Выполняет запрос. Возвращает количество задействованных запросом записей.

```
int Execute(DBExecutor dbExecutor)
```

Выполняет запрос, используя экземпляр `DBExecutor`. Возвращает количество задействованных запросом записей.

```
QueryCondition Where()
```

```
QueryCondition Where(string sourceColumnAlias)
```

```
QueryCondition Where(string sourceAlias, string sourceColumnAlias)
```

```
QueryCondition Where(Select subSelect)
```

```
QueryCondition Where(Query subSelectQuery)
```

```
QueryCondition Where(QueryColumnExpression columnExpression)
```

```
Query Where(QueryCondition condition)
```

Добавляет к текущему запросу начальное условие.

Параметры

sourceColumnAlias	Псевдоним колонки, для которой добавляется условие.
sourceAlias	Псевдоним источника.
subSelect	Подзапрос выборки данных, для результатов которого добавляется условие.
subSelectQuery	Подзапрос, для результатов которого добавляется условие.
columnExpression	Выражение, для результатов которого добавляется условие.
condition	Условие запроса.

```
QueryCondition And()
QueryCondition And(string sourceColumnAlias)
QueryCondition And(string sourceAlias, string sourceColumnAlias)
QueryCondition And(Select subSelect)
QueryCondition And(Query subSelectQuery)
QueryCondition And(QueryParameter parameter)
QueryCondition And(QueryColumnExpression columnExpression)
Query And(QueryCondition condition)
```

К текущему условию запроса добавляет условие (предикат), используя логическую операцию И.

Параметры

sourceColumnAlias	Псевдоним колонки, для которой добавляется предикат.
sourceAlias	Псевдоним источника.
subSelect	Подзапрос выборки данных, используемый в качестве предиката.
subSelectQuery	Подзапрос, используемый в качестве предиката.
parameter	Параметр, для которого добавляется предикат.
columnExpression	Выражение, используемое в качестве предиката.
condition	Условие запроса.

```

QueryCondition Or()
QueryCondition Or(string sourceColumnAlias)
QueryCondition Or(string sourceAlias, string sourceColumnAlias)
QueryCondition Or(Select subSelect)
QueryCondition Or(Query subSelectQuery)
QueryCondition Or(QueryParameter parameter)
QueryCondition Or(QueryColumnExpression columnExpression)
Query Or(QueryCondition condition)

```

К текущему условию запроса добавляет условие (предикат), используя логическую операцию ИЛИ.

Параметры

sourceColumnAlias	Псевдоним колонки, для которой добавляется предикат.
sourceAlias	Псевдоним источника.
subSelect	Подзапрос на выборку данных, используемый в качестве предиката.
subSelectQuery	Подзапрос, используемый в качестве предиката.
parameter	Параметр, для которого добавляется предикат.
columnExpression	Выражение, используемое в качестве предиката.
condition	Условие запроса.

```

Delete From(string schemaName)
Delete From((ModifyQuerySource source)

```

Добавляет в текущий запрос источник данных. Возвращает текущий экземпляр `Delete`.

Параметры

schemaName	Название схемы (таблицы, представления).
source	Источник данных.

Класс QueryFunction



Сложный

Класс `Terrasoft.Core.DB.QueryFunction` реализует функцию выражения.

Функция выражения реализована в следующих классах:

- `QueryFunction` — базовый класс функции выражения.
- `AggregationQueryFunction` — реализует агрегирующую функцию выражения.
- `IsNullQueryFunction` — заменяет значения `null` замещающим выражением.
- `CreateGuidQueryFunction` — реализует функцию выражения нового идентификатора.
- `CurrentDateTimeQueryFunction` — реализует функцию выражения текущей даты и времени.
- `CoalesceQueryFunction` — возвращает первое выражение из списка аргументов, не равное `null`.
- `DatePartQueryFunction` — реализует функцию выражения части значения типа `Дата/Время`.
- `DateAddQueryFunction` — реализует функцию выражения даты, полученной путем добавления указанного промежутка времени к заданной дате.
- `DateDiffQueryFunction` — реализует функцию выражения разницы дат, полученного путем вычитания заданных дат.
- `CastQueryFunction` — приводит выражение аргумента к заданному типу данных.
- `UpperQueryFunction` — преобразовывает символы выражения аргумента в верхний регистр.
- `CustomQueryFunction` — реализует пользовательскую функцию.
- `DataLengthQueryFunction` — определяет число байтов, использованных для представления выражения.
- `TrimQueryFunction` — удаляет начальные и конечные пробелы из выражения.
- `LengthQueryFunction` — возвращает длину выражения.
- `SubstringQueryFunction` — получает часть строки.
- `ConcatQueryFunction` — формирует строку, которая является результатом объединения строковых значений аргументов функции.
- `WindowQueryFunction` — реализует функцию SQL окна.

Класс QueryFunction c#

Пространство имен `Terrasoft.Core.DB`.

Базовый класс функции выражения.

На заметку. Полный перечень методов класса `QueryFunction`, его родительских классов, а также реализуемых им интерфейсов можно найти в [Библиотеке .NET классов](#).

Методы

```
static QueryColumnExpression Negate(QueryFunction operand)
```

Возвращает выражение отрицания значения переданной функции.

Параметры

operand	Функция выражения.
---------	--------------------

```
static QueryColumnExpression operator -(QueryFunction operand)
```

Перегрузка оператора отрицания переданной функции выражения.

Параметры

operand	Функция выражения.
---------	--------------------

```
static QueryColumnExpression Add(QueryFunction leftOperand, QueryFunction rightOperand)
```

Возвращает выражение арифметического сложения переданных функций выражения.

Параметры

leftOperand	Левый операнд в операции сложения.
rightOperand	Правый операнд в операции сложения.

```
static QueryColumnExpression operator +(QueryFunction leftOperand, QueryFunction rightOperand)
```

Перегрузка оператора сложения двух функций выражений.

Параметры

leftOperand	Левый операнд в операции сложения.
rightOperand	Правый операнд в операции сложения.

```
static QueryColumnExpression Subtract(QueryFunction leftOperand, QueryFunction rightOperand)
```

Возвращает выражение вычитания правой функции выражения из левой.

Параметры

leftOperand	Левый операнд в операции вычитания.
rightOperand	Правый операнд в операции вычитания.

```
static QueryColumnExpression operator -(QueryFunction leftOperand, QueryFunction rightOperand)
```

Перегрузка оператора вычитания правой функции выражения из левой.

Параметры

leftOperand	Левый операнд в операции вычитания.
rightOperand	Правый операнд в операции вычитания.

```
static QueryColumnExpression Subtract(QueryFunction leftOperand, QueryFunction rightOperand)
```

Возвращает выражение умножения переданных функций выражений.

Параметры

leftOperand	Левый операнд в операции умножения.
rightOperand	Правый операнд в операции умножения.

```
static QueryColumnExpression operator *(QueryFunction leftOperand, QueryFunction rightOperand)
```

Перегрузка оператора умножения двух функций выражений.

Параметры

leftOperand	Левый операнд в операции умножения.
rightOperand	Правый операнд в операции умножения.

```
static QueryColumnExpression Divide(QueryFunction leftOperand, QueryFunction rightOperand)
```

Возвращает выражение деления левой функции выражения на правую.

Параметры

leftOperand	Левый операнд в операции деления.
rightOperand	Правый операнд в операции деления.

```
static QueryColumnExpression operator /(QueryFunction leftOperand, QueryFunction rightOperand)
```

Перегрузка оператора деления функций выражений.

Параметры

leftOperand	Левый операнд в операции деления.
rightOperand	Правый операнд в операции деления.

```
abstract object Clone()
```

Создает копию текущего экземпляра `QueryFunction`.

```
abstract void BuildSqlText(StringBuilder sb, DBEngine dbEngine)
```

Формирует текст запроса с использованием переданных экземпляра `StringBuilder` и построителя запросов `DBEngine`.

Параметры

sb	Экземпляр <code>StringBuilder</code> , с помощью которого формируется текст запроса.
dbEngine	Экземпляр построителя запросов к базе данных.

```
virtual void AddUsingParameters(QueryParameterCollection resultParameters)
```

Добавляет переданную коллекцию параметров в аргументы функции.

Параметры

resultParameters	Коллекция параметров запроса, которые добавляются в аргументы функции.
------------------	--

```
QueryColumnExpressionCollection GetQueryColumnExpressions()
```

Возвращает коллекцию выражений колонки запроса для текущей функции запроса.

```
QueryColumnExpression GetQueryColumnExpression()
```

Возвращает выражение колонки запроса для текущей функции запроса.

Класс AggregationQueryFunction C#

Пространство имен `Terrasoft.Core.DB`.

Класс реализует агрегирующую функцию выражения.

На заметку. Полный перечень методов и свойств класса `AggregationQueryFunction`, его родительских классов, а также реализуемых им интерфейсов можно найти в [Библиотеке .NET классов](#).

Конструкторы

`AggregationQueryFunction()`

Инициализирует новый экземпляр `AggregationQueryFunction`.

`AggregationQueryFunction(AggregationTypeStrict aggregationType, QueryColumnExpression expression)`

Инициализирует новый экземпляр `AggregationQueryFunction` с заданным типом агрегирующей функции для указанного выражения колонки.

Параметры

<code>aggregationType</code>	Тип агрегирующей функции.
<code>expression</code>	Выражение колонки, к которому применяется агрегирующая функция.

`AggregationQueryFunction(AggregationTypeStrict aggregationType, IQueryColumnExpressionConvertible expression)`

Инициализирует новый экземпляр `AggregationQueryFunction` с заданным типом агрегирующей функции для указанного выражения колонки.

Параметры

<code>aggregationType</code>	Тип агрегирующей функции.
<code>expression</code>	Выражение колонки, к которому применяется агрегирующая функция.

`AggregationQueryFunction(AggregationQueryFunction source)`

Инициализирует новый экземпляр `AggregationQueryFunction`, являющийся клоном переданной агрегирующей функции выражения.

Параметры

source	Агрегирующая функция выражения <code>AggregationQueryFunction</code> , клон которой создается.
--------	--

Свойства

`AggregationType AggregationTypeStrict`

Тип агрегирующей функции.

`AggregationEvalType AggregationEvalType`

Область применения агрегирующей функции.

`Expression QueryColumnExpression`

Выражение аргумента функции.

Методы

`override void BuildSqlText(StringBuilder sb, DBEngine dbEngine)`

Формирует текст запроса с использованием заданных экземпляров `StringBuilder` и построителя запроса `DBEngine`.

Параметры

<code>sb</code>	Экземпляр <code>StringBuilder</code> , с помощью которого формируется текст запроса.
<code>dbEngine</code>	Экземпляр построителя запросов к базе данных.

`override void AddUsingParameters(QueryParameterCollection resultParameters)`

Добавляет переданную коллекцию параметров в аргументы функции.

Параметры

<code>resultParameters</code>	Коллекция параметров запроса, которые добавляются в аргументы функции.
-------------------------------	--

`override object Clone()`

Создает клон текущего экземпляра `AggregationQueryFunction`.

`AggregationQueryFunction All()`

Устанавливает для текущей агрегирующей функции область применения [*Ко всем значениям*].

`AggregationQueryFunction Distinct()`

Устанавливает для текущей агрегирующей функции область применения [*К уникальным значениям*].

Класс IsNullQueryFunction c#

Пространство имен `Terrasoft.Core.DB`.

Класс заменяет значения `null` замещающим выражением.

На заметку. Полный перечень методов и свойств класса `IsNullQueryFunction`, его родительских классов, а также реализуемых им интерфейсов можно найти в [Библиотеке .NET классов](#).

Конструкторы

`IsNullQueryFunction()`

Инициализирует новый экземпляр `IsNullQueryFunction`.

`IsNullQueryFunction(QueryColumnExpression checkExpression, QueryColumnExpression replacementExpr)`
`IsNullQueryFunction(IQueryColumnExpressionConvertible checkExpression, IQueryColumnExpressionConvertible replacementExpr)`

Инициализирует новый экземпляр `IsNullQueryFunction` для заданных проверяемого выражения и замещающего выражения.

Параметры

<code>checkExpression</code>	Выражение, которое проверяется на равенство <code>null</code> .
<code>replacementExpression</code>	Выражение, которое возвращается функцией, если <code>checkExpression</code> равно <code>null</code> .

`IsNullQueryFunction(IsNullQueryFunction source)`

Инициализирует новый экземпляр `IsNullQueryFunction`, являющийся клоном переданной функции выражения.

Параметры

source	Агрегирующая функция выражения <code>IIsNullQueryFunction</code> , КЛОН которой создается.
--------	--

Свойства

`CheckExpression` `QueryColumnExpression`

Выражение аргумента функции, которое проверяется на равенство значению `null`.

`ReplacementExpression` `QueryColumnExpression`

Выражение аргумента функции, которое возвращается, если проверяемое выражение равно `null`.

Методы

`override void BuildSqlText(StringBuilder sb, DBEngine dbEngine)`

Формирует текст запроса с использованием заданных экземпляров `StringBuilder` и построителя запросов `DBEngine`.

Параметры

sb	Экземпляр <code>StringBuilder</code> , с помощью которого формируется текст запроса.
dbEngine	Экземпляр построителя запросов к базе данных.

`override void AddUsingParameters(QueryParameterCollection resultParameters)`

Добавляет переданную коллекцию параметров в аргументы функции.

Параметры

resultParameters	Коллекция параметров запроса, которые добавляются в аргументы функции.
------------------	--

`override object Clone()`

Создает клон текущего экземпляра `IIsNullQueryFunction`.

Класс CreateGuidQueryFunction c#

Пространство имен `Terrasoft.Core.DB`.

Класс реализует функцию выражения нового идентификатора.

На заметку. Полный перечень методов класса `CreateGuidQueryFunction`, его родительских классов, а также реализуемых им интерфейсов можно найти в [Библиотеке .NET классов](#).

Конструкторы

`CreateGuidQueryFunction()`

Инициализирует новый экземпляр `CreateGuidQueryFunction`.

`CreateGuidQueryFunction(CreateGuidQueryFunction source)`

Инициализирует новый экземпляр `CreateGuidQueryFunction`, являющийся клоном переданной функции.

Параметры

<code>source</code>	ФУНКЦИЯ <code>CreateGuidQueryFunction</code> , КЛОН которой создается.
---------------------	--

Методы

`override void BuildSqlText(StringBuilder sb, DBEngine dbEngine)`

Формирует текст запроса с использованием заданных экземпляров `StringBuilder` и построителя запросов `DBEngine`.

Параметры

<code>sb</code>	Экземпляр <code>StringBuilder</code> , с помощью которого формируется текст запроса.
<code>dbEngine</code>	Экземпляр построителя запроса к базе данных.

`override object Clone()`

Создает клон текущего экземпляра `CreateGuidQueryFunction`.

Класс CurrentDateTimeQueryFunction c#

Пространство имен `Terrasoft.Core.DB`.

Класс реализует функцию выражения текущей даты и времени.

На заметку. Полный перечень методов класса `CurrentDateTimeQueryFunction`, его родительских классов, а также реализуемых им интерфейсов можно найти в [Библиотеке .NET классов](#).

Конструкторы

`CurrentDateTimeQueryFunction()`

Инициализирует новый экземпляр `CurrentDateTimeQueryFunction`.

`CurrentDateTimeQueryFunction(CurrentDateTimeQueryFunction source)`

Инициализирует новый экземпляр `CurrentDateTimeQueryFunction`, являющийся клоном переданной функции.

Параметры

source	ФУНКЦИЯ <code>CurrentDateTimeQueryFunction</code> , клон которой создается.
--------	---

Методы

`override void BuildSqlText(StringBuilder sb, DBEngine dbEngine)`

Формирует текст запроса с использованием заданных экземпляров `StringBuilder` и построителя запросов `DBEngine`.

Параметры

sb	Экземпляр <code>StringBuilder</code> , с помощью которого формируется текст запроса.
----	--

| dbEngine | Экземпляр построителя запроса к базе данных. |

`override object Clone()`

Создает клон текущего экземпляра `CurrentDateTimeQueryFunction`.

Класс CoalesceQueryFunction c#

Пространство имен `Terrasoft.Core.DB`.

Класс возвращает первое выражение из списка аргументов, не равное `null`.

На заметку. Полный перечень методов и свойств класса `CoalesceQueryFunction`, его родительских классов, а также реализуемых им интерфейсов можно найти в [Библиотеке .NET классов](#).

Конструкторы

`CoalesceQueryFunction()`

Инициализирует новый экземпляр `CoalesceQueryFunction`.

`CoalesceQueryFunction(CoalesceQueryFunction source)`

Инициализирует новый экземпляр `CoalesceQueryFunction`, являющийся клоном переданной функции.

Параметры

<code>source</code>	ФУНКЦИЯ <code>CoalesceQueryFunction</code> , клон которой создается.
---------------------	--

`CoalesceQueryFunction(QueryColumnExpressionCollection expressions)`

Инициализирует новый экземпляр `CoalesceQueryFunction` для переданной коллекции выражений колонок.

Параметры

<code>expressions</code>	Коллекция выражений колонок запроса.
--------------------------	--------------------------------------

`CoalesceQueryFunction(QueryColumnExpression[] expressions)`

`CoalesceQueryFunction(IQueryColumnExpressionConvertible[] expressions)`

Инициализирует новый экземпляр `CoalesceQueryFunction` для переданного массива выражений колонок.

Параметры

<code>expressions</code>	Массив выражений колонок запроса.
--------------------------	-----------------------------------

Свойства

`Expressions QueryColumnExpressionCollection`

Коллекция выражений аргументов функции.

Методы

`override void BuildSqlText(StringBuilder sb, DBEngine dbEngine)`

Формирует текст запроса с использованием заданных экземпляров `StringBuilder` и построителя запросов `DBEngine`.

Параметры

<code>sb</code>	Экземпляр <code>StringBuilder</code> , с помощью которого формируется текст запроса.
<code>dbEngine</code>	Экземпляр построителя запроса к базе данных.

`override object Clone()`

Создает клон текущего экземпляра `CoalesceQueryFunction`.

`override void AddUsingParameters(QueryParameterCollection resultParameters)`

Добавляет заданные параметры в коллекцию.

Параметры

<code>resultParameters</code>	Коллекция параметров запроса, которые добавляются в аргументы функции.
-------------------------------	--

Класс DatePartQueryFunction [c#](#)

Пространство имен `Terrasoft.Core.DB`.

Класс реализует функцию выражения части значения типа `Дата/Время`.

На заметку. Полный перечень методов и свойств класса `DatePartQueryFunction`, его родительских классов, а также реализуемых им интерфейсов можно найти в [Библиотеке .NET классов](#).

Конструкторы

`DatePartQueryFunction()`

Инициализирует новый экземпляр `DatePartQueryFunction`.

`DatePartQueryFunction(DatePartQueryFunctionInterval interval, QueryColumnExpression expression)`

`DatePartQueryFunction(DatePartQueryFunctionInterval interval, IQueryColumnExpressionConvertible`

Инициализирует новый экземпляр `DatePartQueryFunction` с заданным выражением колонки типа `дата/время` и указанной частью даты.

Параметры

<code>interval</code>	Часть даты.
<code>expression</code>	Выражение колонки типа <code>дата/время</code> .

`DatePartQueryFunction(DatePartQueryFunction source)`

Инициализирует новый экземпляр `DatePartQueryFunction`, являющийся клоном переданной функции.

Параметры

<code>source</code>	ФУНКЦИЯ <code>DatePartQueryFunction</code> , КЛОН которой создается.
---------------------	--

Свойства

`Expression` `QueryColumnExpression`

Выражение аргумента функции.

`Interval` `DatePartQueryFunctionInterval`

Часть даты, возвращаемая функцией.

`UseUtcOffset` `bool`

Использование смещения всеобщего скоординированного времени (UTC) относительно заданного местного времени.

`UtcOffset` `int?`

Смещение всеобщего скоординированного времени (UTC).

Методы

```
override void BuildSqlText(StringBuilder sb, DBEngine dbEngine)
```

Формирует текст запроса с использованием заданных экземпляров `StringBuilder` и построителя запросов `DBEngine`.

Параметры

<code>sb</code>	Экземпляр <code>StringBuilder</code> , с помощью которого формируется текст запроса.
<code>dbEngine</code>	Экземпляр построителя запроса к базе данных.

```
override void AddUsingParameters(QueryParameterCollection resultParameters)
```

Добавляет заданные параметры в коллекцию.

Параметры

<code>resultParameters</code>	Коллекция параметров запроса, которые добавляются в аргументы функции.
-------------------------------	--

```
override object Clone()
```

Создает клон текущего экземпляра `DatePartQueryFunction`.

Класс DateAddQueryFunction [C#](#)

Пространство имен `Terrasoft.Core.DB`.

Класс реализует функцию выражения даты, полученной путем добавления указанного промежутка времени к заданной дате.

На заметку. Полный перечень методов и свойств класса `DateAddQueryFunction`, его родительских классов, а также реализуемых им интерфейсов можно найти в [Библиотеке .NET классов](#).

Конструкторы

```
DateAddQueryFunction()
```

Инициализирует новый экземпляр `DateAddQueryFunction`.

```
DateAddQueryFunction(DatePartQueryFunctionInterval interval, int number, QueryColumnExpression expression)
DateAddQueryFunction(DatePartQueryFunctionInterval interval, IQueryColumnExpressionConvertible expression)
DateAddQueryFunction(DatePartQueryFunctionInterval interval, int number, IQueryColumnExpressionConvertible expression)
```

Инициализирует экземпляр `DateAddQueryFunction` с заданными параметрами.

Параметры

<code>interval</code>	Часть даты, к которой добавляется временной промежуток.
<code>number</code>	Значение, которое добавляется к <code>interval</code> .
<code>expression</code>	Выражение колонки, содержащей исходную дату.

`DateAddQueryFunction(DateAddQueryFunction source)`

Инициализирует экземпляр `DateAddQueryFunction`, являющийся клоном переданной функции.

Параметры

<code>source</code>	Экземпляр функции <code>DateAddQueryFunction</code> , клон которой создается.
---------------------	---

Свойства

Expression `QueryColumnExpression`

Выражение колонки, содержащей исходную дату.

Interval `DatePartQueryFunctionInterval`

Часть даты, к которой добавляется временной промежуток.

Number `int`

Добавляемый временной промежуток.

NumberExpression `QueryColumnExpression`

Выражение, содержащие добавляемый временной промежуток.

Методы

```
override void BuildSqlText(StringBuilder sb, DBEngine dbEngine)
```

Формирует текст запроса с использованием заданных экземпляров `StringBuilder` и построителя запросов `DBEngine`.

Параметры

<code>sb</code>	Экземпляр <code>StringBuilder</code> , с помощью которого формируется текст запроса.
<code>dbEngine</code>	Экземпляр построителя запроса к базе данных.

```
override void AddUsingParameters(QueryParameterCollection resultParameters)
```

Добавляет заданные параметры в коллекцию.

Параметры

<code>resultParameters</code>	Коллекция параметров запроса, которые добавляются в аргументы функции.
-------------------------------	--

```
override object Clone()
```

Создает клон текущего экземпляра `DateAddQueryFunction`.

Класс DateDiffQueryFunction [c#](#)

Пространство имен `Terrasoft.Core.DB`.

Класс реализует функцию выражения разницы дат, полученного путем вычитания заданных дат.

На заметку. Полный перечень методов и свойств класса `DateDiffQueryFunction`, его родительских классов, а также реализуемых им интерфейсов можно найти в [Библиотеке .NET классов](#).

Конструкторы

```
DateDiffQueryFunction(DateDiffQueryFunctionInterval interval, QueryColumnExpression startDateExp)
DateDiffQueryFunction(DateDiffQueryFunctionInterval interval, IQueryColumnExpressionConvertible
```

Инициализирует экземпляр `DateDiffQueryFunction` с заданными параметрами.

Параметры

interval	Единица измерения разницы дат.
startDateExpression	Выражение колонки, содержащей начальную дату.
endDateExpression	Выражение колонки, содержащей конечную дату.

`DateDiffQueryFunction(DateDiffQueryFunction source)`

Инициализирует экземпляр `DateDiffQueryFunction`, являющийся клоном переданной функции.

Параметры

source	Экземпляр функции <code>DateDiffQueryFunction</code> , клон которой создается.
--------	--

Свойства

`StartDateExpression QueryColumnExpression`

Выражение колонки, содержащей начальную дату.

`EndDateExpression QueryColumnExpression`

Выражение колонки, содержащей конечную дату.

`Interval DateDiffQueryFunctionInterval`

Единица измерения разницы дат, возвращаемая функцией.

Методы

`override void BuildSqlText(StringBuilder sb, DBEngine dbEngine)`

Формирует текст запроса с использованием заданных экземпляров `StringBuilder` и построителя запросов `DBEngine`.

Параметры

sb	Экземпляр <code>StringBuilder</code> , с помощью которого формируется текст запроса.
dbEngine	Экземпляр построителя запроса к базе данных.

```
override void AddUsingParameters(QueryParameterCollection resultParameters)
```

Добавляет заданные параметры в коллекцию.

Параметры

resultParameters	Коллекция параметров запроса, которые добавляются в аргументы функции.
------------------	--

```
override object Clone()
```

Создает клон текущего экземпляра `DateDiffQueryFunction`.

Класс CastQueryFunction C#

Пространство имен `Terrasoft.Core.DB`.

Класс приводит выражение аргумента к заданному типу данных.

На заметку. Полный перечень методов и свойств класса `CastQueryFunction`, его родительских классов, а также реализуемых им интерфейсов можно найти в [Библиотеке .NET классов](#).

Конструкторы

```
CastQueryFunction(QueryColumnExpression expression, DBValueType castType)
```

```
CastQueryFunction(IQueryColumnExpressionConvertible expression, DBValueType castType)
```

Инициализирует новый экземпляр `CastQueryFunction` с заданными выражением колонки и целевым типом данных.

Параметры

expression	Выражение колонки запроса.
castType	Целевой тип данных.

```
CastQueryFunction(CastQueryFunction source)
```

Инициализирует новый экземпляр `CastQueryFunction`, являющийся клоном переданной функции.

Параметры

source	ФУНКЦИЯ <code>CastQueryFunction</code> , Клон которой создается.
--------	--

Свойства

`Expression QueryColumnExpression`

Выражение аргумента функции.

`CastType DBDataType`

Целевой тип данных.

Методы

`override void BuildSqlText(StringBuilder sb, DBEngine dbEngine)`

Формирует текст запроса с использованием заданных экземпляров `StringBuilder` и построителя запросов `DBEngine`.

Параметры

<code>sb</code>	Экземпляр <code>StringBuilder</code> , с помощью которого формируется текст запроса.
<code>dbEngine</code>	Экземпляр построителя запроса к базе данных.

`override void AddUsingParameters(QueryParameterCollection resultParameters)`

Добавляет заданные параметры в коллекцию.

Параметры

<code>resultParameters</code>	Коллекция параметров запроса, которые добавляются в аргументы функции.
-------------------------------	--

`override object Clone()`

Создает клон текущего экземпляра `CastQueryFunction`.

Класс UpperQueryFunction

Пространство имен `Terrasoft.Core.DB`.

Класс преобразовывает символы выражения аргумента в верхний регистр.

На заметку. Полный перечень методов и свойств класса `UpperQueryFunction`, его родительских классов, а также реализуемых им интерфейсов можно найти в [Библиотеке .NET классов](#).

Конструкторы

`UpperQueryFunction()`

Инициализирует новый экземпляр `UpperQueryFunction`.

`UpperQueryFunction(QueryColumnExpression expression)`

`UpperQueryFunction(IQueryColumnExpressionConvertible expression)`

Инициализирует новый экземпляр `UpperQueryFunction` для заданного выражения колонки.

Параметры

<code>expression</code>	Выражение колонки запроса.
-------------------------	----------------------------

`UpperQueryFunction(UpperQueryFunction source)`

Инициализирует новый экземпляр `UpperQueryFunction`, являющийся клоном переданной функции.

Параметры

<code>source</code>	ФУНКЦИЯ <code>UpperQueryFunction</code> , клон которой создается.
---------------------	---

Свойства

Expression

Выражение аргумента функции.

Методы

`override void BuildSqlText(StringBuilder sb, DBEngine dbEngine)`

Формирует текст запроса с использованием заданных экземпляров `StringBuilder` и построителя запросов `DBEngine`.

Параметры

sb	Экземпляр <code>StringBuilder</code> , с помощью которого формируется текст запроса.
dbEngine	Экземпляр построителя запроса к базе данных.

```
override void AddUsingParameters(QueryParameterCollection resultParameters)
```

Добавляет заданные параметры в коллекцию.

Параметры

resultParameters	Коллекция параметров запроса, которые добавляются в аргументы функции.
------------------	--

```
override object Clone()
```

Создает клон текущего экземпляра `UpperQueryFunction`.

Класс CustomQueryFunction [c#](#)

Пространство имен `Terrasoft.Core.DB`.

Класс реализует пользовательскую функцию.

На заметку. Полный перечень методов и свойств класса `CustomQueryFunction`, его родительских классов, а также реализуемых им интерфейсов можно найти в [Библиотеке .NET классов](#).

Конструкторы

```
CustomQueryFunction()
```

Инициализирует новый экземпляр `CustomQueryFunction`.

```
CustomQueryFunction(string functionName, QueryColumnExpressionCollection expressions)
```

Инициализирует новый экземпляр `CustomQueryFunction` для заданной функции и переданной коллекции выражений колонок.

Параметры

functionName	Имя функции.
expressions	Коллекция выражений колонок запроса.

```
CustomQueryFunction(string functionName, QueryColumnExpression[] expressions)
CustomQueryFunction(string functionName, IQueryColumnExpressionConvertible[] expressions)
```

Инициализирует новый экземпляр `CustomQueryFunction` для заданной функции и переданного массива выражений колонок.

Параметры

functionName	Имя функции.
expressions	Массив выражений колонок запроса.

```
CustomQueryFunction(CustomQueryFunction source)
```

Инициализирует новый экземпляр `CustomQueryFunction`, являющийся клоном переданной функции.

Параметры

source	ФУНКЦИЯ <code>CustomQueryFunction</code> , клон которой создается.
--------	--

Свойства

`Expressions` `QueryColumnExpressionCollection`

Коллекция выражений аргументов функции.

`FunctionName` `string`

Имя функции.

Методы

```
override void BuildSqlText(StringBuilder sb, DBEngine dbEngine)
```

Формирует текст запроса с использованием заданных экземпляров `StringBuilder` и построителя запросов `DBEngine`.

Параметры

sb	Экземпляр <code>StringBuilder</code> , с помощью которого формируется текст запроса.
dbEngine	Экземпляр построителя запроса к базе данных.

```
override void AddUsingParameters(QueryParameterCollection resultParameters)
```

Добавляет заданные параметры в коллекцию.

Параметры

resultParameters	Коллекция параметров запроса, которые добавляются в аргументы функции.
------------------	--

```
override object Clone()
```

Создает клон текущего экземпляра `CustomQueryFunction`.

Класс DataLengthQueryFunction C#

Пространство имен `Terrasoft.Core.DB`.

Класс определяет число байтов, использованных для представления выражения.

На заметку. Полный перечень методов и свойств класса `DataLengthQueryFunction`, его родительских классов, а также реализуемых им интерфейсов можно найти в [Библиотеке .NET классов](#).

Конструкторы

```
DataLengthQueryFunction()
```

Инициализирует новый экземпляр `DataLengthQueryFunction`.

```
DataLengthQueryFunction(QueryColumnExpression expression)
```

Инициализирует новый экземпляр `DataLengthQueryFunction` для заданного выражения колонки.

Параметры

expression	Выражение колонки запроса.
------------	----------------------------

```
DataLengthQueryFunction(IQueryColumnExpressionConvertible columnNameExpression)
```

Инициализирует новый экземпляр `DataLengthQueryFunction` для заданного выражения колонки.

Параметры

columnNameExpression	Выражение колонки запроса.
----------------------	----------------------------

```
DataLengthQueryFunction(DataLengthQueryFunction source)
```

Инициализирует новый экземпляр `DataLengthQueryFunction`, являющийся клоном переданной функции.

Параметры

source	ФУНКЦИЯ <code>DataLengthQueryFunction</code> , КЛОН которой создается.
--------	--

Свойства

`Expression QueryColumnExpression`

Выражение аргумента функции.

Методы

```
override void BuildSqlText(StringBuilder sb, DBEngine dbEngine)
```

Формирует текст запроса с использованием заданных экземпляров `StringBuilder` и построителя запросов `DBEngine`.

Параметры

sb	Экземпляр <code>StringBuilder</code> , с помощью которого формируется текст запроса.
dbEngine	Экземпляр построителя запроса к базе данных.

```
override void AddUsingParameters(QueryParameterCollection resultParameters)
```

Добавляет в аргументы функции переданную коллекцию параметров.

Параметры

<code>resultParameters</code>	Коллекция параметров запроса, которые добавляются в аргументы функции.
-------------------------------	--

`override object Clone()`

Создает клон текущего экземпляра `DataLengthQueryFunction`.

Класс TrimQueryFunction [c#](#)

Пространство имен `Terrasoft.Core.DB`.

Класс удаляет начальные и конечные пробелы из выражения.

На заметку. Полный перечень методов и свойств класса `TrimQueryFunction`, его родительских классов, а также реализуемых им интерфейсов можно найти в [Библиотеке .NET классов](#).

Конструкторы

`TrimQueryFunction(QueryColumnExpression expression)`
`TrimQueryFunction(IQueryColumnExpressionConvertible expression)`

Инициализирует новый экземпляр `TrimQueryFunction` для заданного выражения колонки.

Параметры

<code>expression</code>	Выражение колонки запроса.
-------------------------	----------------------------

`TrimQueryFunction(TrimQueryFunction source)`

Инициализирует новый экземпляр `TrimQueryFunction`, являющийся клоном переданной функции.

Параметры

<code>source</code>	ФУНКЦИЯ <code>TrimQueryFunction</code> , Клон которой создается.
---------------------	--

Свойства

`Expression QueryColumnExpression`

Выражение аргумента функции.

Методы

`override void BuildSqlText(StringBuilder sb, DBEngine dbEngine)`

Формирует текст запроса с использованием заданных экземпляров `StringBuilder` и построителя запросов `DBEngine`.

Параметры

<code>sb</code>	Экземпляр <code>StringBuilder</code> , с помощью которого формируется текст запроса.
<code>dbEngine</code>	Экземпляр построителя запроса к базе данных.

`override void AddUsingParameters(QueryParameterCollection resultParameters)`

Добавляет в аргументы функции переданную коллекцию параметров.

Параметры

<code>resultParameters</code>	Коллекция параметров запроса, которые добавляются в аргументы функции.
-------------------------------	--

`override object Clone()`

Создает клон текущего экземпляра `TrimQueryFunction`.

Класс LengthQueryFunction [C#](#)

Пространство имен `Terrasoft.Core.DB`.

Класс возвращает длину выражения.

На заметку. Полный перечень методов и свойств класса `LengthQueryFunction`, его родительских классов, а также реализуемых им интерфейсов можно найти в [Библиотеке .NET классов](#).

Конструкторы

`LengthQueryFunction()`

Инициализирует новый экземпляр `LengthQueryFunction`.

```
LengthQueryFunction(QueryColumnExpression expression)
LengthQueryFunction(IQueryColumnExpressionConvertible expression)
```

Инициализирует новый экземпляр `LengthQueryFunction` для заданного выражения колонки.

Параметры

expression	Выражение колонки запроса.
------------	----------------------------

```
LengthQueryFunction(LengthQueryFunction source)
```

Инициализирует новый экземпляр `LengthQueryFunction`, являющийся клоном переданной функции.

Параметры

source	ФУНКЦИЯ <code>LengthQueryFunction</code> , клон которой создается.
--------	--

Свойства

`Expression` `QueryColumnExpression`

Выражение аргумента функции.

Методы

```
override void BuildSqlText(StringBuilder sb, DBEngine dbEngine)
```

Формирует текст запроса с использованием заданных экземпляров `StringBuilder` и построителя запросов `DBEngine`.

Параметры

sb	Экземпляр <code>StringBuilder</code> , с помощью которого формируется текст запроса.
dbEngine	Экземпляр построителя запроса к базе данных.

```
override void AddUsingParameters(QueryParameterCollection resultParameters)
```

Добавляет в аргументы функции переданную коллекцию параметров.

Параметры

<code>resultParameters</code>	Коллекция параметров запроса, которые добавляются в аргументы функции.
-------------------------------	--

`override object Clone()`

Создает клон текущего экземпляра `LengthQueryFunction`.

Класс SubstringQueryFunction c#

Пространство имен `Terrasoft.Core.DB`.

Класс получает часть строки.

На заметку. Полный перечень методов и свойств класса `SubstringQueryFunction`, его родительских классов, а также реализуемых им интерфейсов можно найти в [Библиотеке .NET классов](#).

Конструкторы

`SubstringQueryFunction(QueryColumnExpression expression, int start, int length)`
`SubstringQueryFunction(IQueryColumnExpressionConvertible expression, int start, int length)`

Инициализирует новый экземпляр `SubstringQueryFunction` для заданного выражения колонки, начальной позиции и длины подстроки.

Параметры

<code>expression</code>	Выражение колонки запроса.
<code>start</code>	Начальная позиция подстроки.
<code>length</code>	Длина подстроки.

`SubstringQueryFunction(SubstringQueryFunction source)`

Инициализирует новый экземпляр `SubstringQueryFunction`, являющийся клоном переданной функции.

Параметры

<code>source</code>	ФУНКЦИЯ <code>SubstringQueryFunction</code> , клон которой создается.
---------------------	---

Свойства

`Expression QueryColumnExpression`

Выражение аргумента функции.

`StartExpression QueryColumnExpression`

Начальная позиция подстроки.

`LengthExpression QueryColumnExpression`

Длина подстроки.

Методы

`override void BuildSqlText(StringBuilder sb, DBEngine dbEngine)`

Формирует текст запроса с использованием заданных экземпляров `StringBuilder` и построителя запросов `DBEngine`.

Параметры

<code>sb</code>	Экземпляр <code>StringBuilder</code> , с помощью которого формируется текст запроса.
<code>dbEngine</code>	Экземпляр построителя запроса к базе данных.

`override void AddUsingParameters(QueryParameterCollection resultParameters)`

Добавляет в аргументы функции переданную коллекцию параметров.

Параметры

<code>resultParameters</code>	Коллекция параметров запроса, которые добавляются в аргументы функции.
-------------------------------	--

`override object Clone()`

Создает клон текущего экземпляра `SubstringQueryFunction`.

Класс ConcatQueryFunction [C#](#)

Пространство имен `Terrasoft.Core.DB`.

Класс формирует строку, которая является результатом объединения строковых значений аргументов

функции.

На заметку. Полный перечень методов и свойств класса `ConcatQueryFunction`, его родительских классов, а также реализуемых им интерфейсов можно найти в [Библиотеке .NET классов](#).

Конструкторы

`ConcatQueryFunction(QueryColumnExpressionCollection expressions)`

Инициализирует новый экземпляр `ConcatQueryFunction` для переданной коллекции выражений.

Параметры

<code>expressions</code>	Коллекция выражений колонок запроса.
--------------------------	--------------------------------------

`ConcatQueryFunction(ConcatQueryFunction source)`

Инициализирует новый экземпляр `ConcatQueryFunction`, являющийся клоном переданной функции.

Параметры

<code>source</code>	ФУНКЦИЯ <code>ConcatQueryFunction</code> , клон которой создается.
---------------------	--

Свойства

`Expressions` `QueryColumnExpressionCollection`

Коллекция выражений аргументов функции.

Методы

`override void BuildSqlText(StringBuilder sb, DBEngine dbEngine)`

Формирует текст запроса с использованием заданных экземпляров `StringBuilder` и построителя запросов `DBEngine`.

Параметры

sb	Экземпляр <code>StringBuilder</code> , с помощью которого формируется текст запроса.
dbEngine	Экземпляр построителя запроса к базе данных.

```
override void AddUsingParameters(QueryParameterCollection resultParameters)
```

Добавляет в аргументы функции переданную коллекцию параметров.

Параметры

resultParameters	Коллекция параметров запроса, которые добавляются в аргументы функции.
------------------	--

```
override object Clone()
```

Создает клон текущего экземпляра `ConcatQueryFunction`.

Класс WindowQueryFunction c#

Пространство имен `Terrasoft.Core.DB`.

Класс реализует функцию SQL окна.

На заметку. Полный перечень методов и свойств класса `WindowQueryFunction`, его родительских классов, а также реализуемых им интерфейсов можно найти в [Библиотеке .NET классов](#).

Конструкторы

```
WindowQueryFunction(QueryFunction innerFunction)
```

Реализует функцию SQL окна.

Параметры

innerFunction	Вложенная функция.
---------------	--------------------

```
WindowQueryFunction(QueryFunction innerFunction, QueryColumnExpression partitionByExpression = r
```

Реализует функцию SQL окна.

Параметры

innerFunction	Вложенная функция.
partitionByExpression	Выражение для разделения запроса.
orderByExpression	Выражение для сортировки запроса.

`WindowQueryFunction(WindowQueryFunction source) : this(source.InnerFunction, source.PartitionBy)`
Инициализирует новый экземпляр `WindowQueryFunction`, являющийся клоном переданной функции.

Параметры

source	Функция <code>WindowQueryFunction</code> , клон которой создается.
--------	--

Свойства

`InnerFunction QueryFunction`

Функция для применения.

`PartitionByExpression QueryColumnExpression`

Разделение по пунктам.

`OrderByExpression QueryColumnExpression`

Сортировать по пункту.

Методы

`override void BuildSqlText(StringBuilder sb, DBEngine dbEngine)`

Формирует текст запроса с использованием заданных экземпляров `StringBuilder` и построителя запросов `DBEngine`.

Параметры

sb	Экземпляр <code>StringBuilder</code> , с помощью которого формируется текст запроса.
dbEngine	Экземпляр построителя запроса к базе данных.

```
override void AddUsingParameters(QueryParameterCollection resultParameters)
```

Добавляет в аргументы функции переданную коллекцию параметров.

Параметры

resultParameters	Коллекция параметров запроса, которые добавляются в аргументы функции.
------------------	--

```
override object Clone()
```

Создает клон текущего экземпляра `WindowQueryFunction`.

Операции с локализуемыми ресурсами



Сложный

Инструменты, которые позволяют выполнять операции с локализуемыми ресурсами:

- Creatio IDE.
- База данных.
- Система контроля версий SVN.
- Файловая система.

Использовать Creatio IDE для выполнения операций с локализуемыми ресурсами

Использование Creatio IDE позволяет выполнять следующие **операции с локализуемыми ресурсами**:

- Добавить локализуемую колонку.
- Добавить локализуемую строку.

Добавить локализуемую колонку

Назначение локализуемой колонки — возможность отображения в интерфейсе приложения данных объекта на нескольких языках (т. н. мультиязычие). Значение локализуемой колонки зависит от выбранного в профиле пользователя языка. Локализуемая колонка настраивается в дизайнере объекта, а ее значение хранится в базе данных.

Чтобы **добавить локализуемую колонку**:

1. Добавьте колонку, которую необходимо локализовать.
 - a. [Перейдите в раздел \[Конфигурация \]](#) ([Configuration]) и выберите пользовательский [пакет](#), в который будет добавлена схема.
 - b. На панели инструментов реестра раздела нажмите [Добавить] ([Add]) и выберите вид схемы

([Объект] ([Object]) или [Замещающий объект] ([Replacing object])).



- c. В дизайнере объектов заполните свойства схемы.
- d. При необходимости добавьте колонки, которые требуется локализовать, или выберите существующую колонку объекта.
- e. В блоке свойств [Общие] ([General]) соответствующей колонки установите признак [Локализуемый текст] ([Localizable text]).

General

Code
Address

Data type
Unlimited length text

Required
No

Copy this value when copying records

Localizable text

На заметку. Типы колонок, которые можно локализовать:

- [Стока (50 символов)] ([Text (50 characters)]).
- [Стока (250 символов)] ([Text (250 characters)]).
- [Стока (500 символов)] ([Text (500 characters)]).
- [Стока неограниченной длины] ([Unlimited length text]).

- j. На панели инструментов дизайнера объектов нажмите [Сохранить] ([Save]), а затем [Опубликовать] ([Publish]).
2. Выполните перевод значения локализуемой колонки. Для этого воспользуйтесь инструкцией статьи [Перевести элементы интерфейса в разделе \[Переводы \]](#).

В результате колонка будет содержать значения на разных языках. В интерфейсе приложения будет

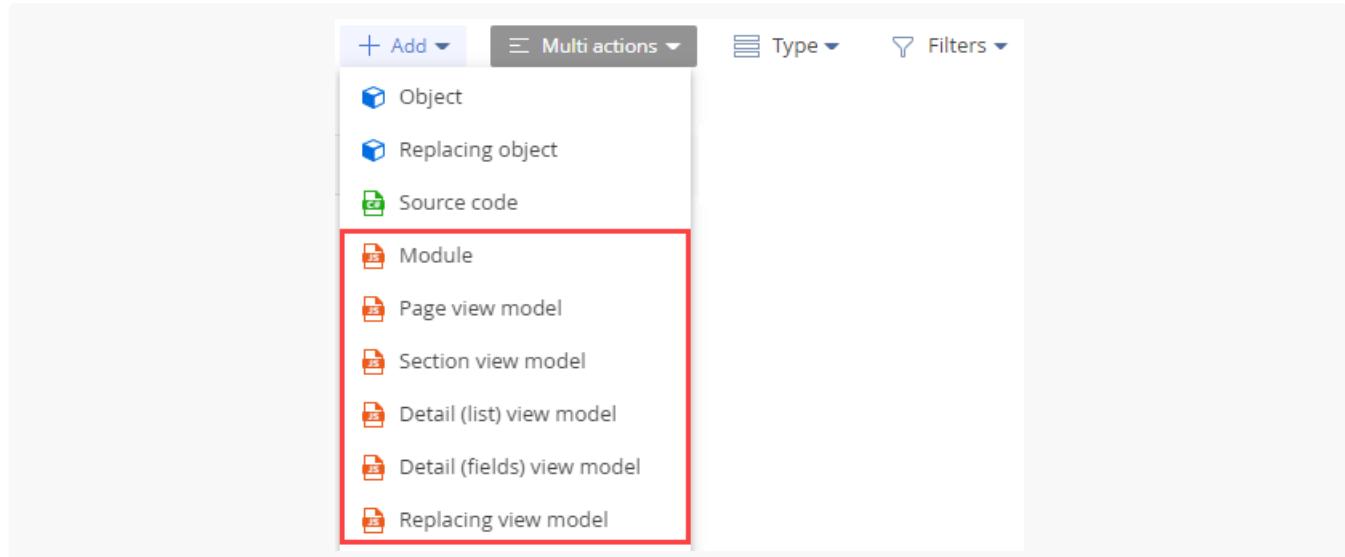
использоваться значение, которое зависит от выбранного в профиле пользователя языка.

Добавить локализуемую строку

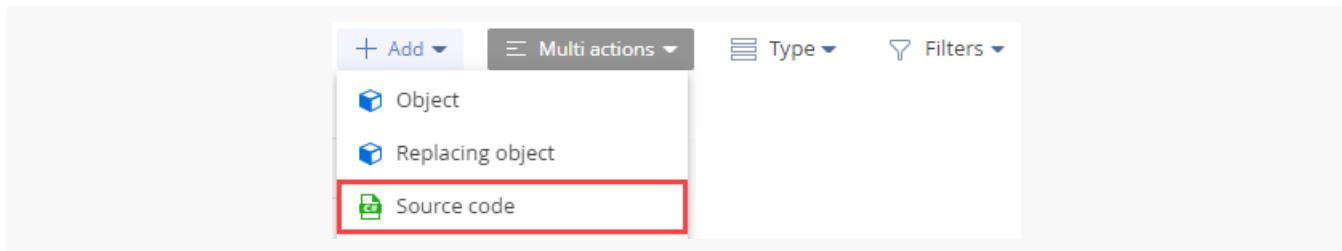
Назначение локализуемой строки — возможность отображения в интерфейсе приложения данных модуля на нескольких языках (т. н. мультиязычие). Значение локализуемой строки зависит от выбранного в профиле пользователя языка. Локализуемая строка настраивается в дизайнере модуля и в дизайнере исходного кода, а ее значение хранится в базе данных.

Чтобы **добавить локализуемую строку**:

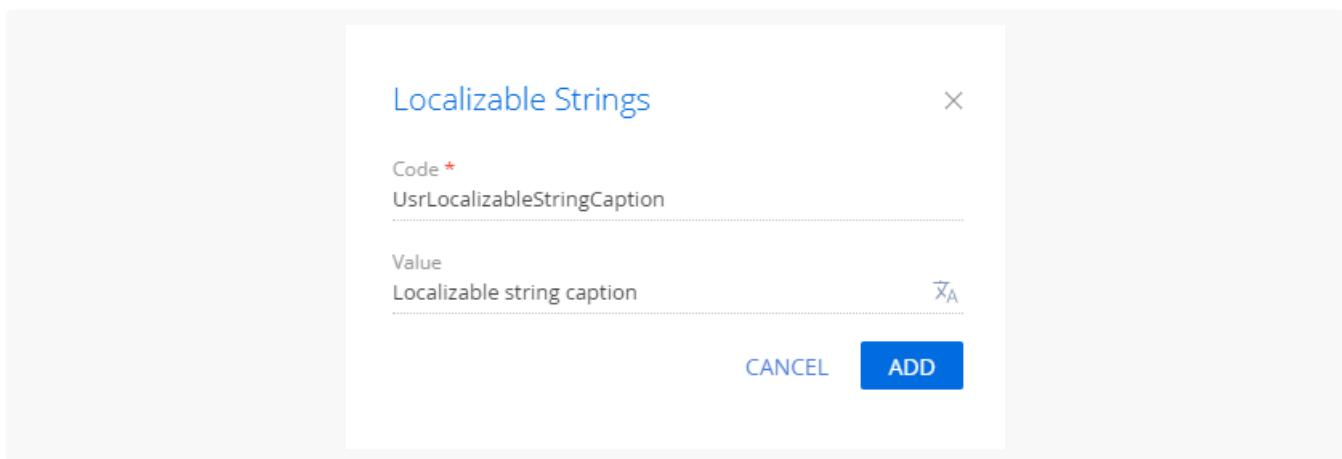
1. [Перейдите в раздел \[Конфигурация \]](#) ([Configuration]) и выберите пользовательский [пакет](#), в который будет добавлена схема.
2. На панели инструментов реестра раздела нажмите [Добавить] ([Add]) и выберите вид схемы.
 - Если необходимо **локализовать строку клиентского модуля**, то выберите один из вариантов:
 - [Модуль] ([Module]).
 - [Модель представления страницы] ([Page view model]).
 - [Модель представления раздела] ([Section view model]).
 - [Модель представления детали с реестром] ([Detail (list) view model]).
 - [Модель представления детали с полями] ([Detail (fields) view model]).
 - [Замещающая модель представления] ([Replacing view model]).



- Если необходимо **локализовать строку исходного кода**, то выберите [Исходный код] ([Source code]).



3. В дизайнере заполните свойства схемы.
4. Добавьте строку, которую требуется локализовать.
 - a. В контекстном меню узла [Локализуемые строки] ([Localizable strings]) нажмите кнопку .
 - b. Заполните свойства локализуемой строки:
 - [Код] ([Code]) — название локализуемой строки (обязательное свойство). Должно содержать префикс (по умолчанию `Usr`), указанный в системной настройке [Префикс названия объекта] (код [SchemaNamePrefix]).
 - [Значение] ([Value]) — значение локализуемой строки. По умолчанию введенное значение сохраняется для выбранного в профиле пользователя языка. Чтобы задать значения локализуемой строки на других языках, нажмите кнопку .



- e. Для добавления локализуемой строки нажмите [Добавить] ([Add]).
5. На панели инструментов дизайнера нажмите [Сохранить] ([Save]), а затем [Опубликовать] ([Publish]).

В результате строка будет содержать значения на разных языках. В интерфейсе приложения будет использоваться значение, которое зависит от выбранного в профиле пользователя языка.

Использовать базу данных для выполнения операций с локализуемыми ресурсами

Использование базы данных позволяет выполнять следующие **операции с локализуемыми ресурсами**:

- Получить локализуемые ресурсы из базы данных.

- Обновить локализуемые ресурсы в базе данных.
- Сохранить локализуемые ресурсы в базе данных.
- Отключить локализуемые ресурсы в базе данных.

Получить локализуемые ресурсы из базы данных

Классы, которые реализуют логику получения локализуемых ресурсов из базы данных:

- `Terrasoft.Core.Entities.EntitySchemaQuery` — получение данных из базы данных через ORM-модель. Класс по умолчанию поддерживает работу с мультиязычными данными.
- `Terrasoft.Core.Entities.Entity` — работа с сущностью базы данных.

Подробнее о получении данных из базы данных с помощью классов

`Terrasoft.Core.Entities.EntitySchemaQuery` И `Terrasoft.Core.Entities.Entity` читайте в статье [Доступ к данным через ORM](#).

Правила формирования выборки мультиязычных данных:

- Если в профиле пользователя **выбран основной язык**, то выборка данных формируется из основной таблицы базы данных.
- Если в профиле пользователя **выбран дополнительный язык** и в таблице `[SysLocalizableValue]` **присутствует значение локализуемого ресурса**, то выборка данных формируется из таблицы `[SysLocalizableValue]` базы данных.
- Если в профиле пользователя **выбран дополнительный язык** и в таблице `[SysLocalizableValue]` **отсутствует значение локализуемого ресурса**, то выборка данных формируется из основной таблицы базы данных.

Для выборки данных можно использовать **представления** (`View`), которые позволяют получить данные из локализуемых колонок. Для формирования выборки локализуемых данных через представление необходимо настроить локализуемые представления.

Чтобы **настроить локализуемые представления**:

1. Создайте схему объекта для представления. Шаблон названия схемы: `UsrVwИмяСхемыОбъекта`.

Обязательные составляющие названия схемы:

- Префикс (по умолчанию `Usr`), указанный в системной настройке [*Префикс названия объекта*] (код [*SchemaNamePrefix*]).
 - Префикс `Vw` (сокращение от `View`) который указывает, что схема является представлением в базе данных.
2. Настройте локализуемую колонку. Настройка локализуемых колонок описана в пункте [Добавить локализуемую колонку](#).
 3. Добавьте новое представление локализации в базе данных.

Обновить локализуемые ресурсы в базе данных

При изменении значения локализуемого ресурса в таблице `[SysLocalizableValue]` необходимо выполнить

обновление локализуемых ресурсов в базе данных.

Чтобы для соответствующей схемы **обновить локализуемые ресурсы в базе данных**, необходимо с помощью SQL-запроса изменить значение колонки `[IsChanged]` таблицы `[SysPackageResourceChecksum]`. В другом случае при обновлении пакета в приложении возникнет конфликт локализуемых ресурсов. Он не будет обнаружен, что приведен к потере значения локализуемого ресурса.

Важно. Запрещено добавлять данные в таблицу `[SysLocalizableValue]` с использованием прямого SQL-запроса, поскольку в метаданных соответствующей схемы будет отсутствовать информация о добавленных локализуемых ресурсах. Чтобы **добавить локализуемые ресурсы**, необходимо использовать Creatio IDE, систему контроля версий SVN или файловую систему.

Сохранить локализуемые ресурсы в базе данных

Чтобы **сохранить локализуемые ресурсы**, необходимо использовать метод `Entity.SetColumnValue()`, который может принимать параметры типа `string` и параметры типа `LocalizableString`.

Сохранить локализуемые ресурсы с использованием параметров типа `string`

Особенности сохранения локализуемых ресурсов при использовании параметров типа `string`:

- Если **запись добавляется** пользователем с дополнительным языком, то данные сохраняются и в основную таблицу объекта `Entity`, и в таблицу локализации объекта `Entity`.
- Если **запись изменяется** пользователем с дополнительным языком, то данные сохраняются в таблицу локализации объекта `Entity`.
- Если **запись добавляется или изменяется** пользователем с основным языком, то данные сохраняются в основную таблицу объекта `Entity`.

Шаблон формирования имени таблицы локализации: `[SysИмяОсновнойТаблицыLcz]`.

Ниже приведен пример сохранения локализуемых ресурсов с использованием параметра типа `string` для пользователя с основным языком (русский).

Пример сохранения локализуемых ресурсов с использованием параметра типа `string`

```
var userConnection = (UserConnection)HttpContext.Current.Session["UserConnection"];
EntitySchema schema = userConnection.EntitySchemaManager.FindInstanceByName("AccountType");
Entity entity = schema.CreateEntity(userConnection);

/* Установка для колонок значений по умолчанию. */
entity.SetDefColumnValues();

/* Установка значения для колонки [Название]. */
entity.SetColumnValue("Name", "Новый клиент");

/* Сохранение. */
entity.Save();
```

```
var name = String.Format("Name: {0}", entity.GetTypedColumnValue<string>("Name"));
return name;
```

При выполнении этого кода пользователем с основным языком (русский) будет выполнен SQL-запрос к основной таблице `[AccountType]` базы данных. В параметре `@P2` указано значение "Новый клиент".

SQL-запрос

```
exec sp_executesql N'
INSERT INTO [dbo].[AccountType]([Id], [Name], [CreatedOn], [CreatedBy], [ModifiedOn], [ModifiedBy], [IsDeleted])
VALUES(@P1, @P2, @P3, @P4, @P5, @P6, @P7, @P8)',N'@P1 uniqueidentifier,@P2 nvarchar(12),@P3 date,@P4 time,@P5 uniqueidentifier,@P6 date,@P7 time,@P8 uniqueidentifier'
```

Ниже приведен пример сохранения локализуемых ресурсов с использованием параметра типа `string` для пользователя с дополнительным языком (например, английским).

Пример сохранения локализуемых ресурсов с использованием параметра типа `string`

```
var userConnection = (UserConnection)HttpContext.Current.Session["UserConnection"];
EntitySchema schema = userConnection.EntitySchemaManager.FindInstanceByName("AccountType");
Entity entity = schema.CreateEntity(userConnection);
entity.SetDefColumnValues();
entity.SetColumnValue("Name", "New Customer");
entity.Save();
var name = String.Format("Name: {0}", entity.GetTypedColumnValue<string>("Name"));
return name;
```

При выполнении этого кода пользователем с дополнительным языком (например, английским) будут выполнены:

1. SQL-запрос к основной таблице `[AccountType]` базы данных. SQL-запрос такой же, как и для основной локализации, но в параметре `@P2` указано значение "New Customer".

SQL-запрос

```
exec sp_executesql N'
INSERT INTO [dbo].[AccountType]([Id], [Name], [CreatedOn], [CreatedBy], [ModifiedOn], [ModifiedBy], [IsDeleted])
VALUES(@P1, @P2, @P3, @P4, @P5, @P6, @P7, @P8)',N'@P1 uniqueidentifier,@P2 nvarchar(12),@P3 date,@P4 time,@P5 uniqueidentifier,@P6 date,@P7 time,@P8 uniqueidentifier'
```

2. SQL-запрос в таблицу локализации `[SysAccountTypeLcz]`. В параметре `@P5` указано значение "New Customer".

SQL-запрос

```
exec sp_executesql N'
INSERT INTO [dbo].[SysAccountTypeLcz]([Id], [ModifiedOn], [RecordId], [SysCultureId], [Name])
VALUES(@P1, @P2, @P3, @P4, @P5)',N'@P1 uniqueidentifier,@P2 datetime2(7),@P3 uniqueidentifier
```

Таким образом, в основную таблицу `[AccountType]` будет помещено значение, которое не соответствует основной локализации. Чтобы этого избежать, необходимо выполнять сохранение локализуемых ресурсов с использованием параметров типа `LocalizableString`.

Сохранить локализуемые ресурсы с использованием параметров типа `LocalizableString`

Ниже приведен пример сохранения данных с использованием параметра типа `LocalizableString`.

Пример сохранения локализуемых ресурсов с использованием параметра типа `LocalizableString`

```
var userConnection = (UserConnection)HttpContext.Current.Session["UserConnection"];
EntitySchema schema = userConnection.EntitySchemaManager.FindInstanceByName("AccountType");
Entity entity = schema.CreateEntity(userConnection);
entity.SetDefColumnValues();

/* Создание локализуемой строки с локализованными значениями для разных языковых культур. */
var localizableString = new LocalizableString();
localizableString.SetValue(new CultureInfo("ru-RU"), "Новый клиент ru-RU");
localizableString.SetValue(new CultureInfo("en-US"), "New Customer en-US");

/* Установка значения колонки с помощью локализуемой строки. */
entity.SetValue("Name", localizableString);
entity.Save();

/* Результат будет выведен в текущей языковой культуре пользователя. */
var name = String.Format("Name: {0}", entity.GetTypedColumnValue<string>("Name"));
return name;
```

При выполнении этого кода, независимо от выбранного в профиле пользователя языка, будут выполнены:

1. SQL-запрос в основную таблицу `[AccountType]`. В параметре `@P2` указано значение "Новый клиент ru-RU".

SQL-запрос

```
exec sp_executesql N'
INSERT INTO [dbo].[AccountType]([Id], [Name], [CreatedOn], [CreatedBy], [ModifiedOn], [ModifiedBy])
VALUES(@P1, @P2, @P3, @P4, @P5, @P6, @P7, @P8)',N'@P1 uniqueidentifier,@P2 nvarchar(18),@P3 d
```

2. SQL-запрос в таблицу локализации `[SysAccountTypeLcz]`. В параметре `@P5` указано значение "New Customer en-US".

SQL-запрос

```
exec sp_executesql N'
INSERT INTO [dbo].[SysAccountTypeLcz]([Id], [ModifiedOn], [RecordId], [SysCultureId], [Name])
VALUES(@P1, @P2, @P3, @P4, @P5)',N'@P1 uniqueidentifier,@P2 datetime2(7),@P3 uniqueidentifier
```

При выполнении этого кода пользователем с дополнительным языком и отсутствии значения локализуемой строки на основном языке, в основную таблицу `[AccountType]` будет добавлена запись со значением для дополнительного языка.

Отключить локализуемые ресурсы в базе данных

Чтобы **отключить локализуемые ресурсы**, необходимо для свойства `UseLocalization` экземпляра класса `EntitySchemaQuery` установить значение `false`. Отключение локализуемых ресурсов не приводит к удалению локализуемых ресурсов из таблиц базы данных и не зависит от выбранного в профиле пользователя языка.

Пример отключения получения локализуемых ресурсов

```
/* Пользовательское подключение. */
var userConnection = (UserConnection)HttpContext.Current.Session["UserConnection"];

/* Формирование запроса. */
var esqResult = new EntitySchemaQuery(userConnection.EntitySchemaManager, "City");

/* Добавление колонки в запрос. */
esqResult.AddColumn("Name");

/* Отключение механизма выборки локализованных данных. */
esqResult.UseLocalization = false;

/* Выполнение запроса к базе данных и получение всей результирующей коллекции объектов. */
var entities = esqResult.GetEntityCollection(userConnection);

/* Получение текста запроса. */
var s = esqResult.GetSelectQuery(userConnection).GetSqlText();

/* Возврат результата. */
return s;
```

При выполнении этого кода, независимо от выбранного в профиле пользователя языка, будет выполнен SQL-запрос к основной таблице `[city]` базы данных.

SQL-запрос

```
SELECT
    [City].[Name]  [Name]
FROM
    [dbo].[City] [City] WITH(NOLOCK)
```

Использовать систему контроля версий SVN для выполнения операций с локализуемыми ресурсами

Использование системы контроля версий SVN позволяет выполнять следующие **операции с локализуемыми ресурсами**:

- Обновить локализуемые ресурсы из хранилища SVN.
- Фиксировать локализуемые ресурсы в хранилище SVN.

Описание работы с SVN содержится в статье [Контроль версий в Subversion](#).

Обновить локализуемые ресурсы из хранилища SVN

Чтобы **обновить локализуемые ресурсы из хранилища SVN**, необходимо выполнить обновление пакета из хранилища SVN. Обновление пакета подробно описано в статье [Контроль версий в Creatio IDE](#).

Changes

Name	Type	Status
✓ sdkCustomPackage	Package	Modified
UsrClientUnitSchema	Schema Resource	Modified
UsrEntitySchema	Schema Resource	Modified
UsrEntitySchema	Schema Resource	Modified

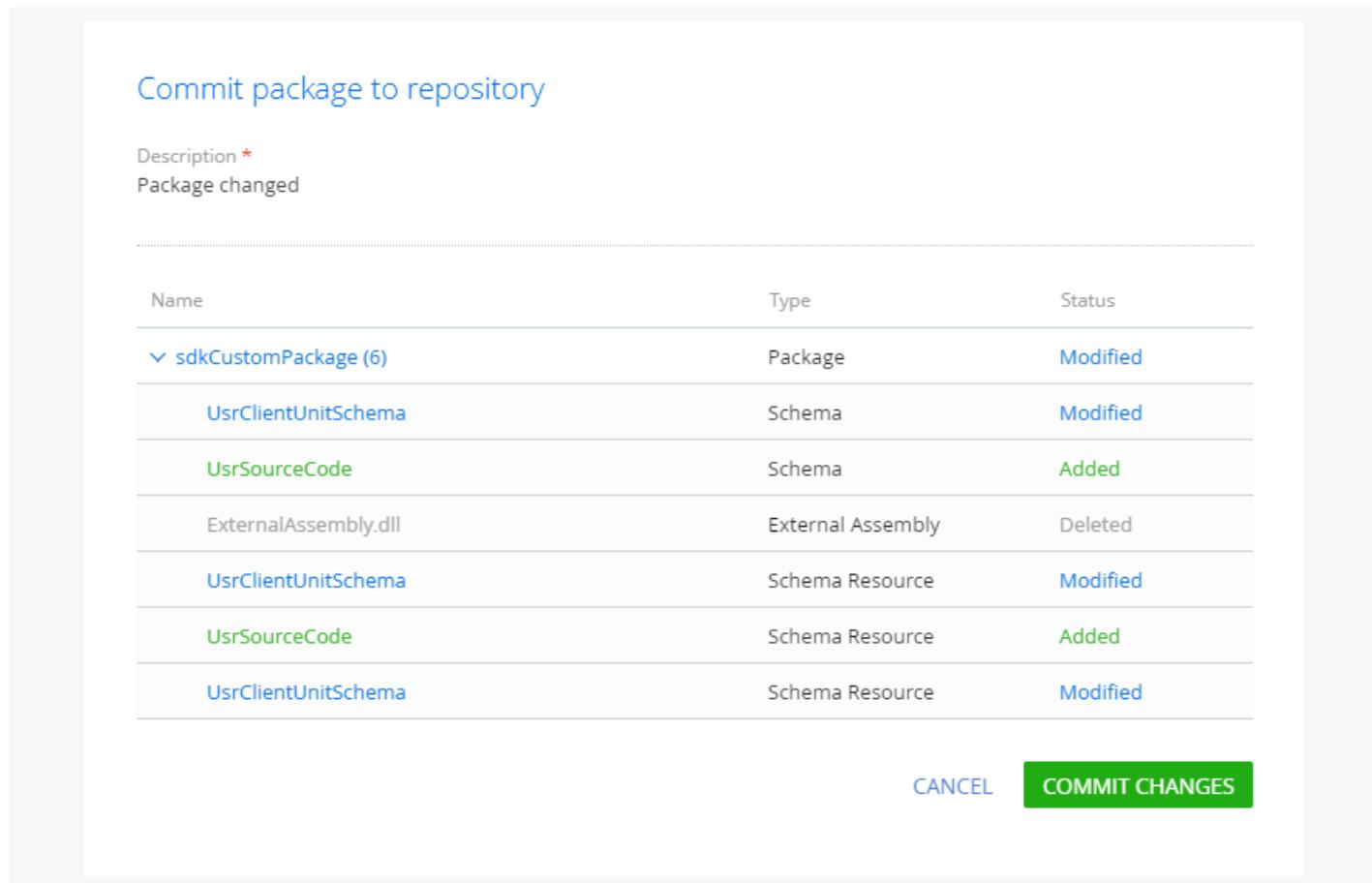
CLOSE

Возможные состояния локализуемых ресурсов:

- [Изменено] ([*Modified*]) — локализуемый ресурс изменен.
- [Добавлено] ([*Added*]) — локализуемый ресурс добавлен.
- [Удалено] ([*Deleted*]) — локализуемый ресурс удален.
- [Конфликт] ([*Conflicted*]) — работа с локализуемым ресурсом выполнялась одновременно двумя разработчиками, один из которых зафиксировал изменения локализуемого ресурса в хранилище SVN.

Фиксировать локализуемые ресурсы в хранилище SVN

Чтобы **фиксировать локализуемые ресурсы в хранилище SVN**, необходимо выполнить фиксацию пакета в хранилище SVN. Фиксация пакета подробно описана в статье [Контроль версий в Creatio IDE](#).



Используя Creatio IDE, можно заблокировать пакет в хранилище SVN. Блокирование пакета позволяет избежать возникновения состояния [Конфликт] ([*Conflicted*]) локализуемого ресурса, поскольку отключена опция одновременной работы с пакетом. Например, один разработчик вносит изменения в локализуемые ресурсы пакета без его предварительной блокировки. При фиксации пакета в хранилище SVN может возникнуть ситуация, когда в текущем пакете другим разработчиком были изменены и зафиксированы в хранилище SVN эти же локализуемые ресурсы. В этом случае при обновлении пакета в Creatio IDE будет отображен перечень локализуемых ресурсов в состоянии [Конфликт] ([*Conflicted*]). Это означает, что версия и содержимое измененных локализуемых ресурсов не совпадают с версией и содержимым локализуемых ресурсов, зафиксированных в хранилище SVN. При выполнении фиксации

изменений локализуемых ресурсов будут утеряны изменения, зафиксированные в хранилище SVN. Такие конфликтные ситуации необходимо решать вручную, используя SVN-клиенты, например, [TortoiseSVN](#).

The screenshot shows a 'Changes' dialog box with the following content:

Name	Type	Status
✓ sdkCustomPackage (1)	Package	Modified
UsrClientUnitSchema	Schema Resource	Conflicted

CLOSE

Использовать файловую систему для выполнения операций с локализуемыми ресурсами

Использование файловой системы позволяет редактировать локализуемые ресурсы.

Чтобы **редактировать локализуемые ресурсы из файловой системы**:

1. Настройте Creatio для работы в файловой системе. Подробнее читайте в статье [Внешние IDE](#).
2. Используя SVN-клиент, экспортируйте локализуемые ресурсы в файловую систему.
3. Измените локализуемые ресурсы.
4. Зафиксируйте изменения в хранилище SVN.

Важно. Значению локализуемого ресурса соответствует одна запись в таблице [SysLocalizableValue] базы данных. Эта запись содержит ссылки на соответствующие идентификаторы ([SysPackageId], [SysSchemaId], [SysCultureId]) и ключ ([Key]). Поэтому при фиксации ресурса с состоянием [Конфликт] ([Conflicted]) в таблицу будет записано последнее значение.

Получить локализуемые ресурсы из базы данных



Сложный

Пример 1

Пример. С помощью класса `Terrasoft.Core.Entities.EntitySchemaQuery` получить локализуемые ресурсы для произвольной языковой культуры.

Класс `Terrasoft.Core.Entities.EntitySchemaQuery` позволяет выполнить выборку данных для произвольной языковой культуры (языковой культуры, которая отличается от дополнительной языковой культуры пользователя и основной языковой культуры приложения).

Чтобы **получить локализуемые ресурсы для произвольной языковой культуры**:

- Перед получением данных в экземпляре класса `Terrasoft.Core.Entities.EntitySchemaQuery` вызовите метод `SetLocalizationCultureId(Guid cultureId)`.
- В метод `SetLocalizationCultureId(Guid cultureId)` передайте идентификатор языковой культуры, данные которой необходимо получить.

Выполнение выборки данных для произвольной языковой культуры

```
/* Пользовательское подключение. */
var userConnection = (UserConnection)HttpContext.Current.Session["UserConnection"];

/* Получение Id необходимой языковой культуры, например, итальянской. */
var sysCulture = new SysCulture(userConnection);
if (!sysCulture.FetchPrimaryInfoFromDB("Name", "it-IT"))
{
    /* Ошибка, запись не найдена. */
    return "Культура не найдена";
}
Guid italianCultureId = sysCulture.Id;

/* Формирование запроса. */
var esqResult = new EntitySchemaQuery(userConnection.EntitySchemaManager, "City");

/* Добавление колонки в запрос. */
esqResult.AddColumn("Name");

/* Установка необходимой локализации. */
esqResult.SetLocalizationCultureId(italianCultureId);
```

```
/* Выполнение запроса к базе данных и получение всей результирующей коллекции объектов. */
var entities = esqResult.GetEntityCollection(userConnection);

/* Получение текста запроса. */
var s = esqResult.GetSelectQuery(userConnection).GetSqlText();

/* Возврат результата. */
return s;
```

При выполнении этого кода будет выполнен SQL-запрос. Параметр `@P1` принимает значение идентификатора записи (`[Id]`), которое хранится в переменной `italianCultureId`.

SQL-запрос

```
SELECT
    ISNULL([SysCityLcz].[Name], [City].[Name])[Name]
FROM
    [dbo].[City] [City] WITH(NOLOCK)
    LEFT OUTER JOIN [dbo].[SysCityLcz] [SysCityLcz] WITH(NOLOCK) ON ([SysCityLcz].[RecordId] = [
    AND [SysCityLcz].[SysCultureId] = @P1)
```

Пример 2

Пример. С помощью метода `FetchFromDB()` класса `Terrasoft.Core.Entities.Entity` получить значение локализуемой колонки `[Название]` (`[Name]`) объекта схемы `[Тип контрагента]` (`[AccountType]`).

Методы `FetchFromDB()` Класса `Terrasoft.Core.Entities.Entity` позволяют получить мультиязычные данные. Ниже приведен пример использования одной из перегрузок методов `FetchFromDB()` для пользователя с основной (русской) и дополнительной (например, английской) языковыми культурами. Эти методы можно использовать, например, в методах [пользовательского веб-сервиса](#).

Пример использования метода `FetchFromDB()`

```
/* Пользовательское подключение. */
var userConnection = (UserConnection)HttpContext.Current.Session["UserConnection"];

/* Получение схемы [Тип контрагента]. */
EntitySchema schema = userConnection.EntitySchemaManager.FindInstanceByName("AccountType");

/* Создание экземпляра Entity (объекта). */
Entity entity = schema.CreateEntity(userConnection);

/* Коллекция имен колонок для выборки. */
```

```

List<string> columnNamesToFetch = new List<string> {
    "Name",
    "Description"
};

/* Получение данных для объекта, у которого значение колонки [Название] равно "Клиент". */
entity.FetchFromDB("Name", "Клиент", columnNamesToFetch);

/* Формирование и отправка ответа.*/
var name = String.Format("Name: {0}", entity.GetTypedColumnValue<string>("Name"));
return name;

```

При выполнении этого кода пользователем с основной языковой культурой (русской) будет выполнен SQL-запрос к основной таблице `[AccountType]` базы данных. В параметре `@P1` указано значение "Клиент", которое определяет соответствующую запись основной таблицы `[AccountType]` базы данных.

SQL-запрос

```

exec sp_executesql N'
SELECT
    [AccountType].[Name] [Name],
    [AccountType].[Description] [Description]
FROM
    [dbo].[AccountType] [AccountType] WITH(NOLOCK)
WHERE
    [AccountType].[Name] = @P1',N'@P1 nvarchar(6)',@P1=N'Клиент'

```

При выполнении этого кода пользователем с дополнительной языковой культурой (например, английской) будет выполнен SQL-запрос к таблице локализации `[SysAccountTypeLcz]` базы данных. В параметре `@P1` указано значение "Клиент", которое определяет соответствующую запись основной таблицы `[AccountType]` базы данных. В параметре `@P2` указано значение идентификатора (`[SysCultureId]`) дополнительной языковой культуры из таблицы `[SysCulture]`, которое определяет соответствующую запись таблицы локализации `[SysAccountTypeLcz]` базы данных.

SQL-запрос

```

exec sp_executesql N'
SELECT
    ISNULL([SysAccountTypeLcz].[Name], [AccountType].[Name]) [Name],
    ISNULL([SysAccountTypeLcz].[Description], [AccountType].[Description]) [Description]
FROM
    [dbo].[AccountType] [AccountType] WITH(NOLOCK)
    LEFT OUTER JOIN [dbo].[SysAccountTypeLcz] [SysAccountTypeLcz] WITH(NOLOCK) ON ([SysAccountTypeLcz].[SysCultureId] = @P2)
    AND [SysAccountTypeLcz].[SysCultureId] = @P2
WHERE
    [AccountType].[Name] = @P1',N'@P1 nvarchar(6),@P2 uniqueidentifier',@P1=N'Клиент',@P2='A542E

```

В результате значению переменной `Name` для пользователя с русской языковой культурой соответствует значение "Клиент", а для пользователя с английской языковой культурой — "Customer".

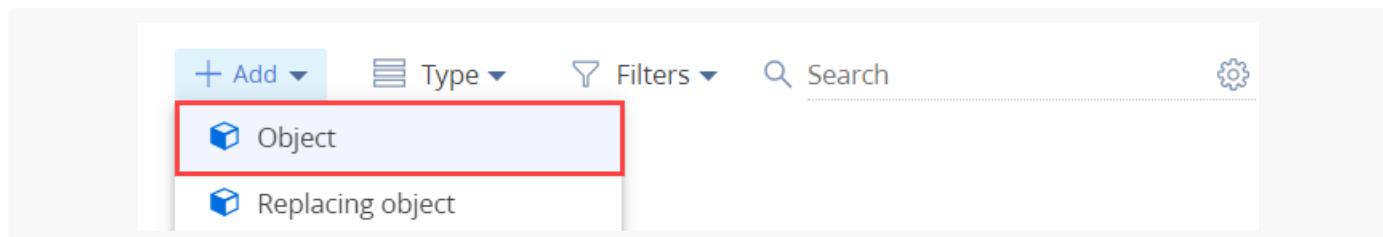
Добавить локализуемую колонку

 Средний

Пример. Добавить локализуемую колонку.

1. Создать объект

- Перейдите в раздел [Конфигурация] ([Configuration]) и выберите пользовательский [пакет](#), в который будет добавлена схема.
- На панели инструментов реестра раздела нажмите [Добавить] —> [Объект] ([Add] —> [Object]).



- В дизайнере объекта заполните свойства схемы:

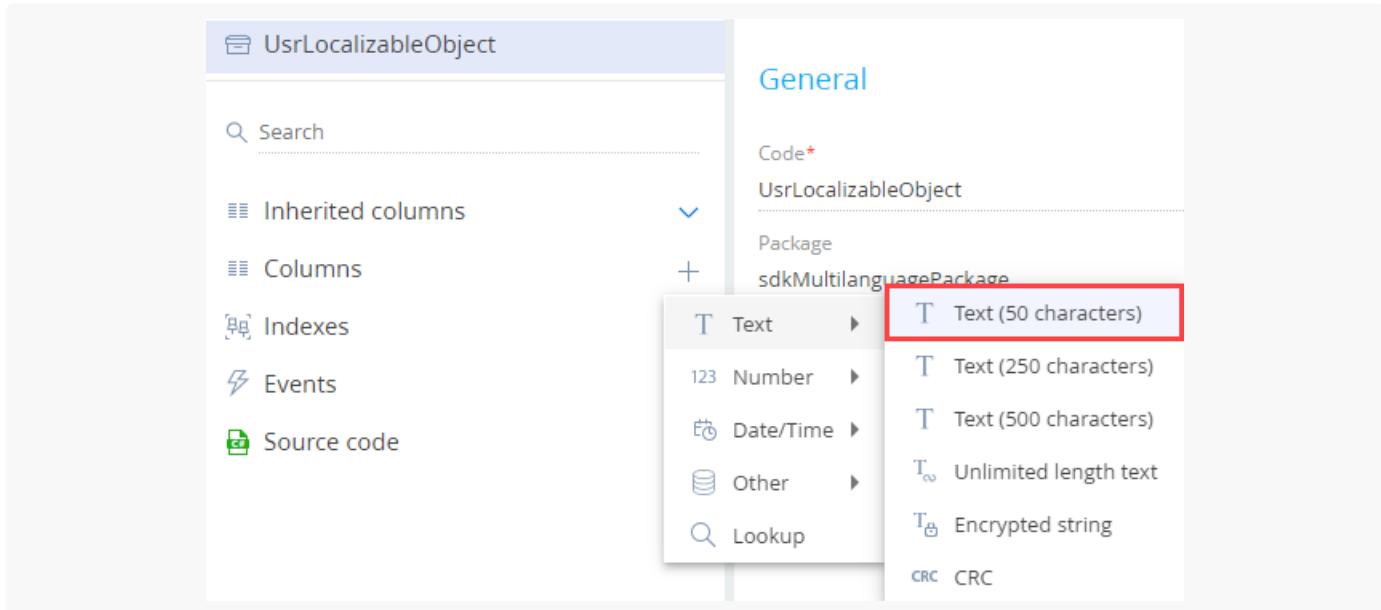
- [Код] ([Code]) — "UsrLocalizableObject".
- [Заголовок] ([Title]) — "LocalizableObject".
- [Родительский объект] ([Parent object]) — выберите "BaseEntity".

Code*	Title*
UsrLocalizableObject	LocalizableObject
Package	Description
sdkMultilanguagePackage	X

Parent object	<input type="checkbox"/> Replace parent
BaseEntity	

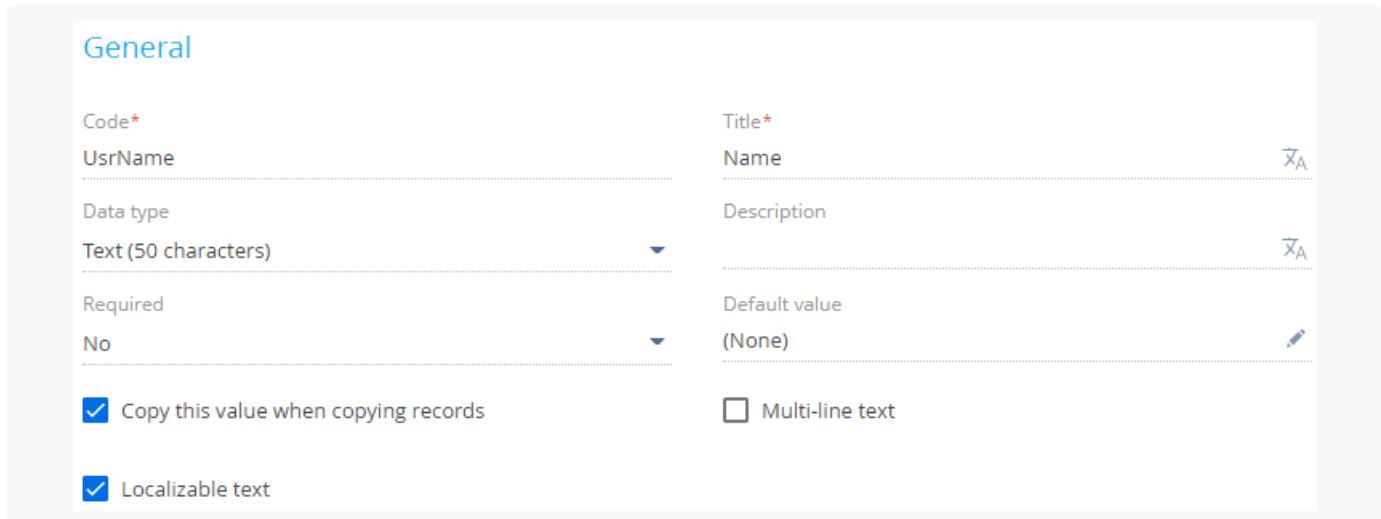
2. Добавить локализуемую колонку

1. В контекстном меню узла [Колонки] ([Columns]) структуры объекта нажмите **+**.
2. В выпадающем меню нажмите [Стока] —> [Стока (50 символов)] ([Text] —> [Text (50 characters)]).



3. В дизайнере объекта заполните свойства добавляемой колонки:

- [Код] ([Code]) — "UsrName".
- [Заголовок] ([Title]) — "Name".
- Установите признак [Локализуемый текст] ([Localizable text]).



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

Результат выполнения примера

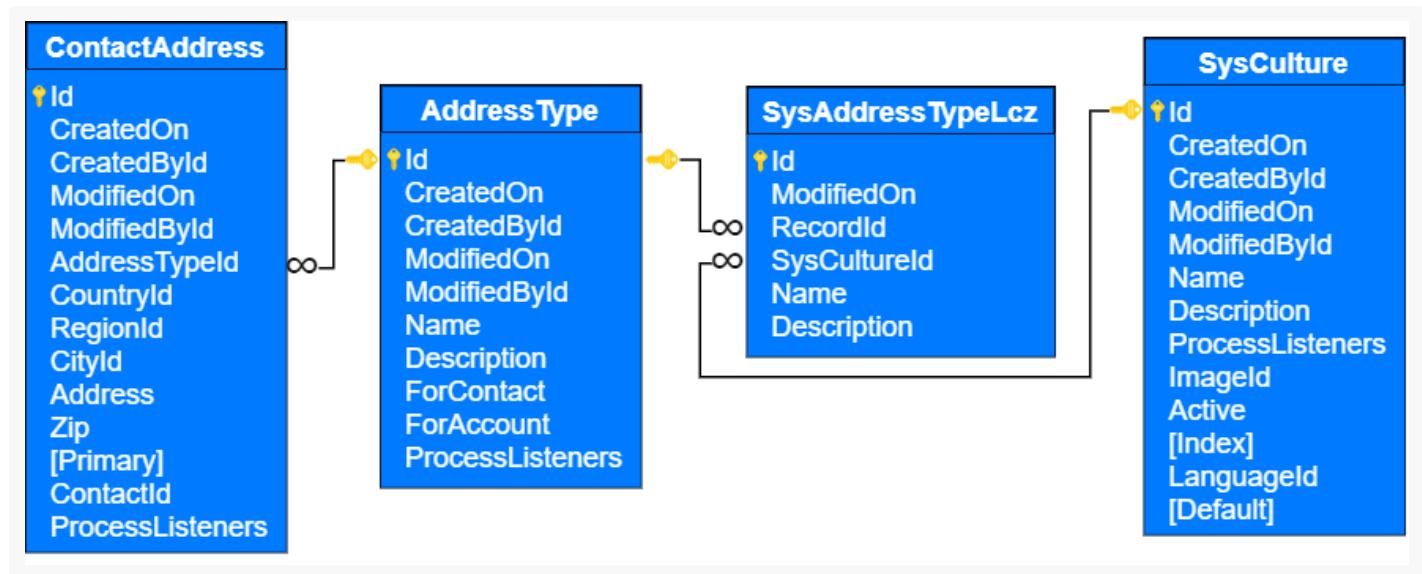
В результате выполнения примера в базе данных создана таблица локализации `[SysUsrLocalizableObjectLcz]`, в которой будут храниться локализованные данные для всех локализуемых колонок.

Локализовать представление в базе данных

 Сложный

Пример. Создать представление, которое формирует выборку, состоящую из локализованных значений адресов контактов (колонка `[Name]` таблицы `[AddressType]` базы данных) и значений адресов контактов (колонка `[Address]` таблицы `[ContactAddress]` базы данных).

В Creatio реализована схема объекта `[Адрес контакта]` (`[ContactAddress]`). Колонка схемы ссылается на справочник `[Тип адреса]` (`[AddressType]`), который содержит локализуемую колонку `[Название]` (`[Name]`). Структура и связи таблиц представлена на рисунке ниже.

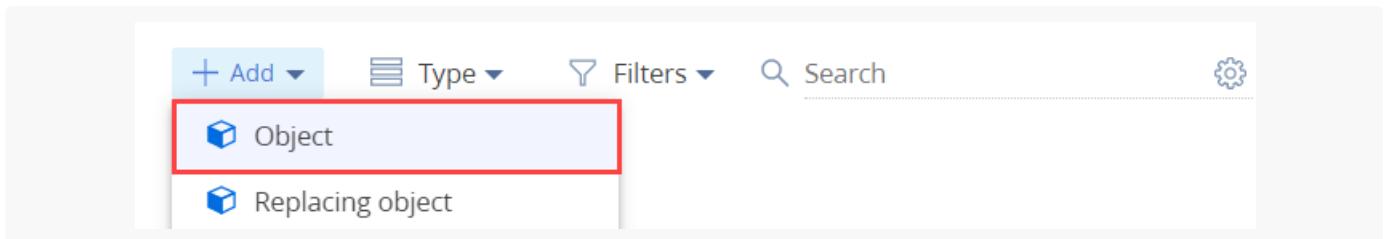


- `[ContactAddress]` — таблица базы данных, которая содержит перечень значений адресов контактов. Связана с таблицей `[AddressType]` по колонке `[AddressTypeId]`.
- `[AddressType]` — таблица базы данных, которая содержит перечень типов адресов контактов. Колонка `[Name]` таблицы содержит перечень значений типов адресов на основном языке. Значения на дополнительных языках содержатся в таблице `[SysAddressTypeLcz]`.
- `[SysAddressTypeLcz]` — автоматически генерируемая системная таблица базы данных, которая содержит перечень локализованных значений типов адресов контактов. Связана с таблицей `[AddressType]` по колонке `[RecordId]` и с таблицей `[SysCulture]` по колонке `[SysCultureId]`. Колонка `[Name]` таблицы содержит перечень локализованных значений типов адресов контактов для языковой культуры, которая указана в колонке `[SysCultureId]` текущей таблицы.

- [SysCulture] — системная таблица базы данных, которая содержит перечень языковых культур.

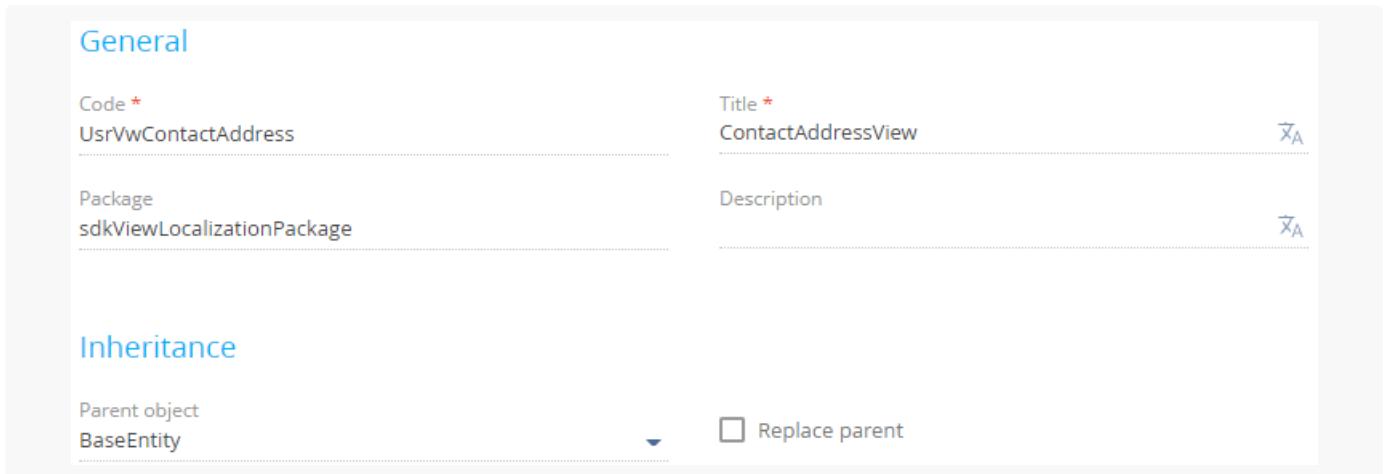
1. Создать схему объекта для представления

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



3. В дизайнере объекта заполните свойства схемы:

- [Код] ([Code]) — "UsrVwContactAddress".
- [Заголовок] ([Title]) — "ContactAddressView".
- [Родительский объект] ([Parent object]) — выберите "BaseEntity".



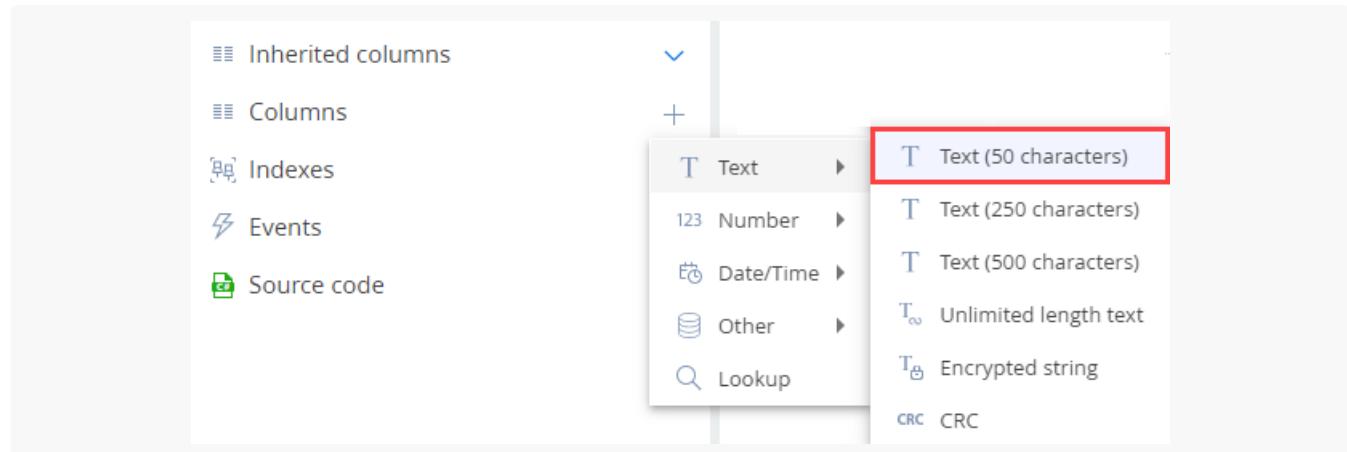
4. В блоке свойств [Поведение] ([Behavior]) установите признак [Представление в базе данных] ([Represent Structure of Database View]).



2. Добавить колонки

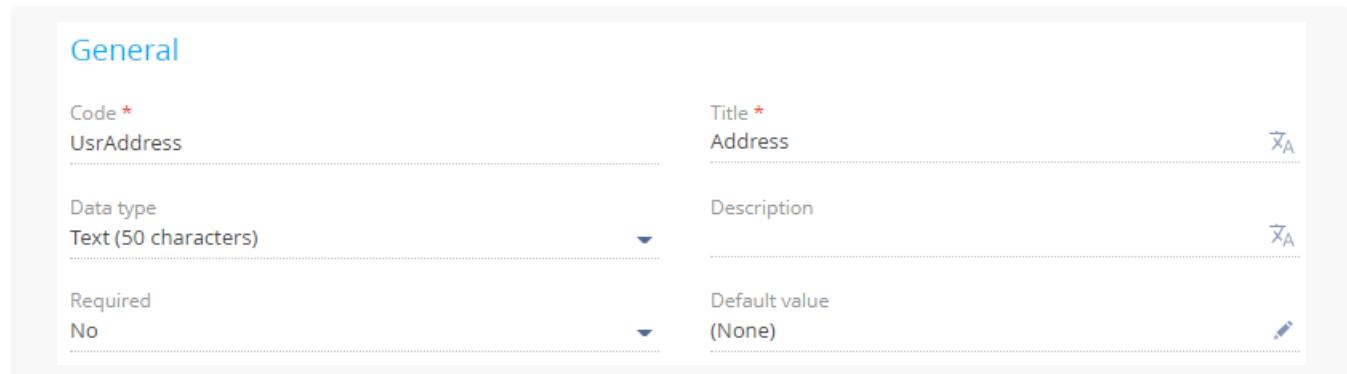
1. Добавьте колонку, которая будет содержать перечень значений адресов контактов на основном языке.

- В контекстном меню узла [Колонки] ([Columns]) структуры объекта нажмите .
- В выпадающем меню нажмите [Стока] —> [Стока (50 символов)] ([Text] —> [Text (50 characters)]).



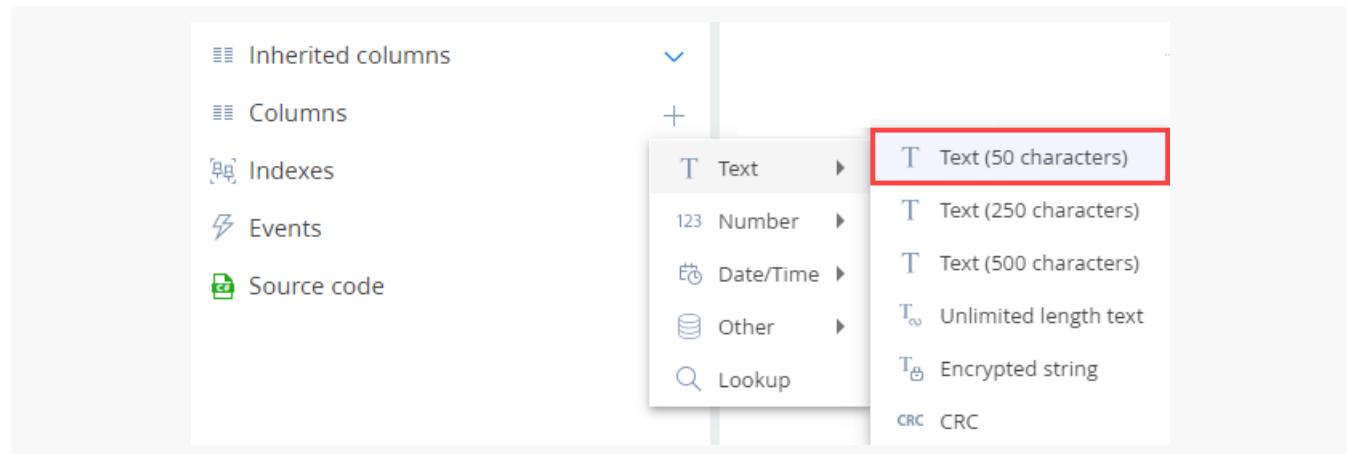
c. В дизайнере объекта заполните свойства добавляемой колонки:

- [Код] ([Code]) — "UsrAddress".
- [Заголовок] ([Title]) — "Address".



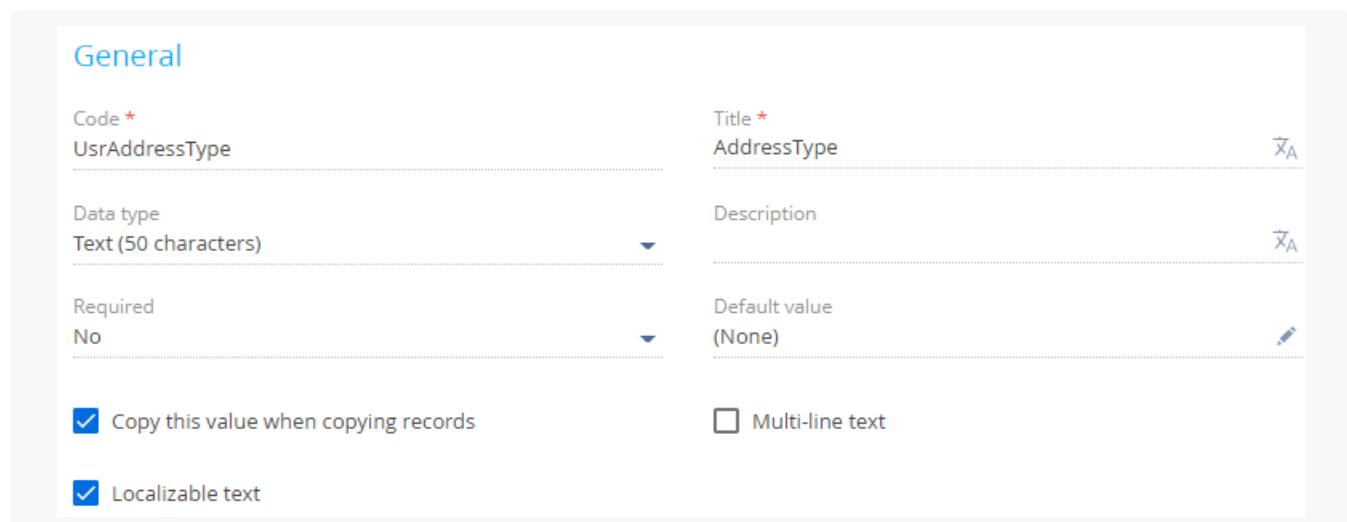
2. Добавьте колонку, которая будет содержать перечень типов адресов контактов на дополнительном языке.

- В контекстном меню узла [Колонки] ([Columns]) структуры объекта нажмите .
- В выпадающем меню нажмите [Стока] —> [Стока (50 символов)] ([Text] —> [Text (50 characters)]).



с. В дизайнере объекта заполните свойства добавляемой колонки:

- [Код] ([Code]) — "UsrAddressType".
- [Заголовок] ([Title]) — "AddressType".
- Установите признак [Локализуемый текст] ([Localizable text]).



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

3. Создать представления в базе данных

1. Создайте представление [UsrVwContactAddress] в базе данных. Для этого выполните SQL-запрос.

SQL-запрос

```
-- Название представления должно соответствовать названию таблицы.
CREATE VIEW dbo.UsrVwContactAddress
AS
SELECT
    ContactAddress.Id,
```

```
-- Колонки представления должны соответствовать названиям колонок схемы.
ContactAddress.Address AS UsrAddress,
AddressType.Name AS UsrAddressType
FROM ContactAddress
INNER JOIN AddressType ON ContactAddress.AddressTypeId = AddressType.Id;
```

2. Создайте локализуемое представление `[SysUsrVwContactAddressLcz]` в базе данных. Для этого выполните SQL-запрос.

SQL-запрос

```
-- Название представления должно соответствовать названию таблицы локализации.
CREATE VIEW dbo.SysUsrVwContactAddressLcz
AS
SELECT
    SysAddressTypeLcz.Id,
    ContactAddress.id AS RecordId,
    SysAddressTypeLcz.SysCultureId,
    -- Колонки представления должны соответствовать названиям колонок схемы.
    SysAddressTypeLcz.Name AS UsrAddressType
FROM ContactAddress
INNER JOIN AddressType ON ContactAddress.AddressTypeId = AddressType.Id
INNER JOIN SysAddressTypeLcz ON AddressType.Id = SysAddressTypeLcz.RecordId;
```

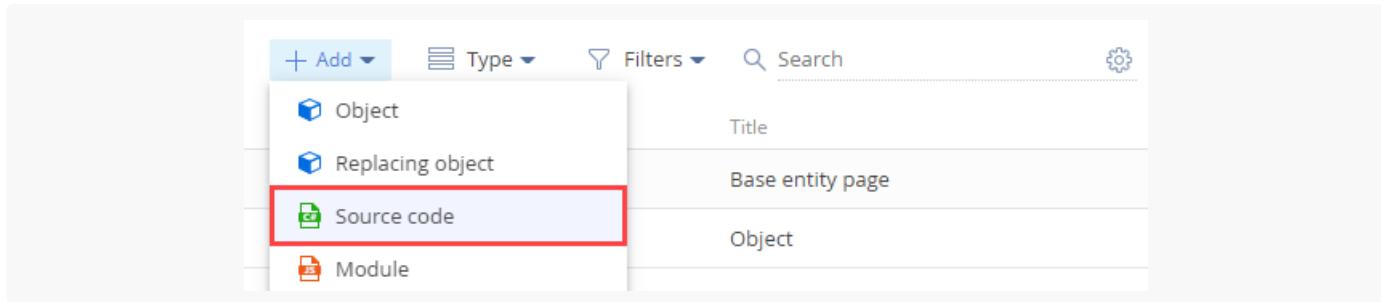
В результате при чтении данных с помощью `EntitySchemaQuery` из колонки `[UsrAddressType]` представления `[UsrVwContactAddress]` будут отображены корректные значения для разных языков.

Результат выполнения примера

Для проверки результата выполнения примера создайте пользовательский веб-сервис с аутентификацией на основе cookies.

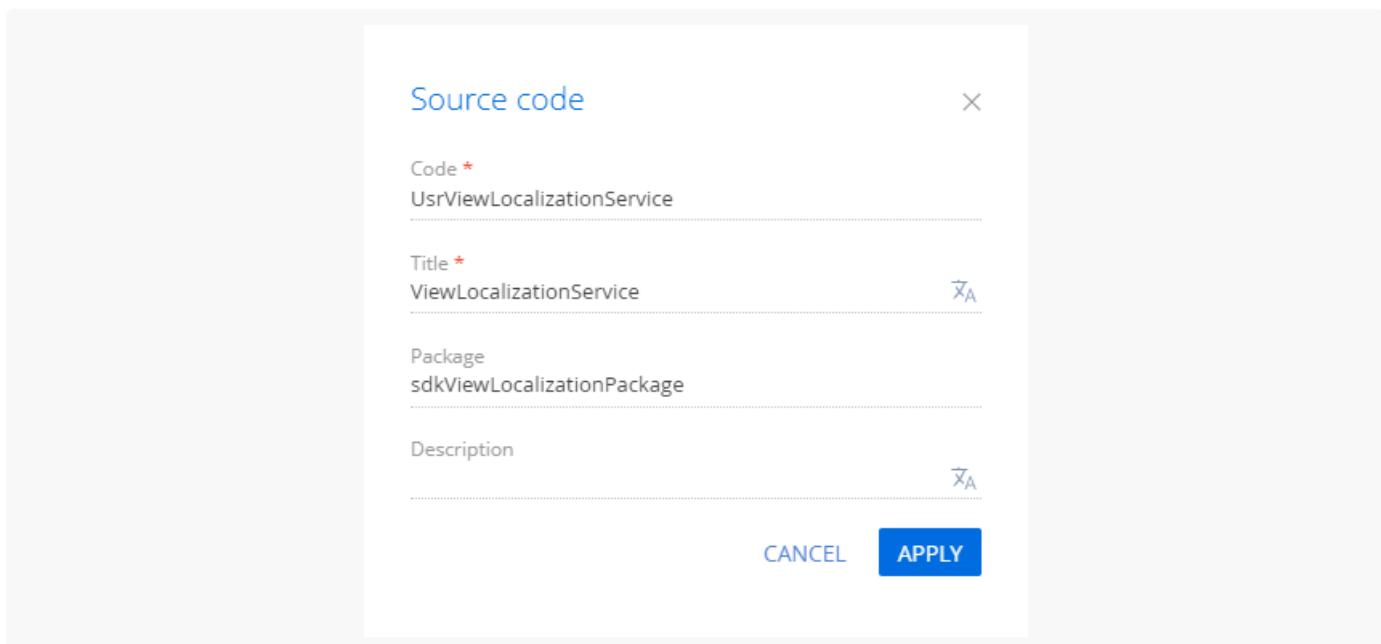
1. Создать схему [Исходный код]

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



3. В дизайнере схем заполните свойства схемы:

- [Код] ([Code]) — "UsrViewLocalizationService".
- [Заголовок] ([Title]) — "ViewLocalizationService".



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

2. Создать класс сервиса

1. В дизайнере схем добавьте пространство имен `Terrasoft.Configuration`.
2. С помощью директивы `using` добавьте пространства имен, типы данных которых будут задействованы в классе.
3. Добавьте название класса, которое соответствует названию схемы (свойство [Код] ([Code])).
4. В качестве родительского класса укажите `System.Web.SessionState.IReadOnlySessionState`.
5. Для класса добавьте атрибуты `[ServiceContract]` и `[AspNetCompatibilityRequirements(RequirementsMode = AspNetCompatibilityRequirementsMode.Required)]`.

3. Реализовать методы класса

1. Реализуйте метод, который вернет перечень типов адресов контактов и значений адресов контактов из созданного нелокализуемого представления [UsrVwContactAddress]. В дизайнере схем добавьте в класс метод `public string GetNonLocalizableView()`, который реализует конечную точку пользовательского веб-сервиса. С помощью `EntitySchemaQuery` метод отправит запрос к базе данных.
2. Реализуйте метод, который вернет перечень типов адресов контактов и значений адресов контактов из созданного локализуемого представления [UsrVwContactAddress]. В дизайнере схем добавьте в класс метод `public string GetLocalizableView()`, который реализует конечную точку пользовательского веб-сервиса. С помощью `EntitySchemaQuery` метод отправит запрос к базе данных.

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

```
UsrViewLocalizationService

namespace Terrasoft.Configuration
{
    using System.ServiceModel;
    using System.ServiceModel.Web;
    using System.ServiceModel.Activation;
    using System.Web;
    using Terrasoft.Core;
    using Terrasoft.Core.Entities;
    using System;
    using System.Collections.Generic;

    [ServiceContract]
    [AspNetCompatibilityRequirements(RequirementsMode = AspNetCompatibilityRequirementsMode.Req
    public class UsrViewLocalizationService : System.Web.SessionState.IReadOnlySessionState
    {

        [OperationContract]
        [WebInvoke(Method = "GET", UriTemplate = "GetNonLocalizableView")]
        public string GetNonLocalizableView()
        {
            var userConnection = (UserConnection)HttpContext.Current.Session["UserConnection"];
            var esqResult = new EntitySchemaQuery(userConnection.EntitySchemaManager, "UsrVwCont
            esqResult.AddColumn("UsrAddress");
            esqResult.AddColumn("UsrAddressType");
            var entities = esqResult.GetEntityCollection(userConnection);
            var s = "";
            foreach (var item in entities)
            {
                s += item.GetTypedColumnValue<string>("UsrAddressType") + ": ";
                s += item.GetTypedColumnValue<string>("UsrAddress") + "; ";
            }
            return s;
        }
    }
}
```

```
[OperationContract]
[WebInvoke(Method = "GET", UriTemplate = "GetLocalizableView")]
public string GetLocalizableView()
{
    var userConnection = (UserConnection)HttpContext.Current.Session["UserConnection"];
    var sysCulture = new SysCulture(userConnection);
    if (!sysCulture.FetchPrimaryInfoFromDB("Name", "ru-ru"))
    {
        return "Культура не найдена";
    }
    Guid CultureId = sysCulture.Id;

    var esqResult = new EntitySchemaQuery(userConnection.EntitySchemaManager, "UsrVwCont");
    esqResult.AddColumn("UsrAddress");
    esqResult.AddColumn("UsrAddressType");
    esqResult.SetLocalizationCultureId(CultureId);

    var entities = esqResult.GetEntityCollection(userConnection);
    var s = "";
    foreach (var item in entities)
    {
        s += item.GetTypedColumnValue<string>("UsrAddressType") + ": ";
        s += item.GetTypedColumnValue<string>("UsrAddress") + "; ";
    }
    return s;
}
}
```

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

Результат работы ПОЛЬЗОВАТЕЛЬСКОГО веб-сервиса

В результате выполнения примера в Creatio появится пользовательский веб-сервис

`UsrViewLocalizationService` с конечными точками `GetLocalizableView` и `GetNonLocalizableView`.

Авторизуйтесь в приложении и из браузера обратитесь к конечной точке `GetLocalizableView` веб-сервиса.

Строка запроса

`http://mycreatio.com/0/rest/UsrViewLocalizationService/GetLocalizableView`

В результате будет получена выборка, которая состоит из локализуемых значений типов адресов контактов и значений адресов контактов.

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
<string xmlns="http://schemas.microsoft.com/2003/10/Serialization/">Рабочий: 186 Tremont Street; Домашний: 82 Chestnut Street; Домашний: 39 Columbia Street; </string>
```

Из браузера обратитесь к конечной точке `GetNonLocalizableView` веб-сервиса.

Строка запроса

`http://mycreatio.com/0/rest/UsrViewLocalizationService/GetNonLocalizableView`

В результате будет получена выборка, которая состоит из нелокализуемых значений типов адресов контактов и значений адресов контактов.

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
<string xmlns="http://schemas.microsoft.com/2003/10/Serialization/">Business: 186 Tremont Street; Places of residence: 82 Chestnut Street; Places of residence: 39 Columbia Street; </string>
```

Класс LocalizableValue<T>



Сложный

Пространство имен `Terrasoft.Common`.

Класс `Terrasoft.Common.LocalizableValue<T>` является базовым классом для классов `Terrasoft.Common.LocalizableString` (отвечает за отображение локализуемых строк) и `Terrasoft.Common.LocalizableImage` (отвечает за отображение локализуемых изображений), которые используются для работы с локализуемыми ресурсами.

Класс `Terrasoft.Common.LocalizableValue<T>` является шаблоном для локализуемых значений различных типов и предоставляет методы для работы с ними.

Note. Полный перечень свойств и методов класса `LocalizableValue<T>`, его родительских классов, а также реализуемых им интерфейсов, можно найти в документации [Библиотеки .NET классов](#).

Свойства

`Value T`

Возвращает и устанавливает локализуемое значение с учетом текущей языковой культуры.

`HasValue bool`

Возвращает признак, который определяет наличие локализуемого значения данного типа для текущей языковой культуры.

`CultureValues IDictionary<CultureInfo, T>`

Возвращает справочник локализуемых значений текущего экземпляра для поддерживаемых языковых культур.

Методы

`void ClearCultureValue(CultureInfo culture)`

Удаляет локализуемое значение для поддерживаемых языковых культур.

Параметры

<code>System.Globalization.CultureInfo culture</code>	Языковая культура.
---	--------------------

`T GetCultureValue(CultureInfo culture, bool throwIfNoManager)`

Получает локализуемое значение указанного типа для поддерживаемых языковых культур. В зависимости от значения параметра `throwIfNoManager`, метод может сгенерировать исключение типа `ItemNotFoundException`, если для этого локализуемого значения не задан диспетчер ресурсов.

Параметры

<code>System.Globalization.CultureInfo culture</code>	Языковая культура.
<code>System.Boolean throwIfNoManager</code>	Флаг, который указывает необходимость вызова исключения <code>ItemNotFoundException</code> .

`T GetCultureValueWithFallback(CultureInfo culture, bool throwIfNoManager)`

Получает локализуемое значение указанного типа для поддерживаемых языковых культур. Если значение локализуемого ресурса для указанной языковой культуры не найдено, то возвращается

значение на основной языковой культуре. В зависимости от значения параметра `throwIfNoManager`, метод может сгенерировать исключение типа `ItemNotFoundException`, если для этого локализуемого значения не задан диспетчер ресурсов.

Параметры

<code>System.Globalization.CultureInfo culture</code>	Языковая культура.
<code>System.Boolean throwIfNoManager</code>	Флаг, который указывает необходимость вызова исключения <code>ItemNotFoundException</code> .

```
bool HasCultureValue(CultureInfo culture)
```

Определяет, существует ли локализуемое значение для заданной языковой культуры.

Параметры

<code>System.Globalization.CultureInfo culture</code>	Языковая культура.
---	--------------------

```
void LoadCultureValues()
```

Загружает перечень локализуемых значений данного типа для всех языковых культур, которые определены в глобальном хранилище ресурсов.

```
void SetCultureValue(CultureInfo culture, T value)
```

Устанавливает заданное локализуемое значение для заданной языковой культуры.

Параметры

<code>System.Globalization.CultureInfo culture</code>	Языковая культура.
<code>T value</code>	Локализуемое значение.

Доступ к данным через ORM



Средний

Способы доступа к базе данных, которые предоставляют back-end компоненты ядра:

- Доступ через ORM-модель.
- Прямой доступ.

В этой статье будет рассмотрено выполнение запросов к базе данных через ORM-модель.

ORM (Object-relational mapping) — технология, которая позволяет работать с данными, полученными из базы данных, путем использования объектно-ориентированных языков программирования. **Назначение** ORM — связывание объектов, реализованных в коде, с записями в таблицах базы данных.

Классы, которые реализуют работу с данными через ORM-модель данных:

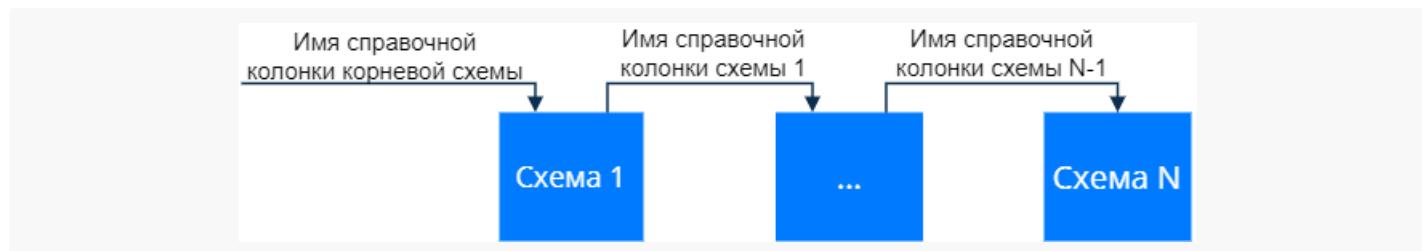
- `Terrasoft.Core.Entities.EntitySchemaQuery` — построение запросов на получение записей из таблиц базы данных с учетом прав доступа текущего пользователя.
- `Terrasoft.Core.Entities.Entity` — доступ к сущности, которая представляет собой запись в таблице базы данных.

Для доступа к данным рекомендуется использовать ORM-модель, хотя прямой доступ к базе данных также реализован в back-end компонентах ядра. Выполнение запросов к базе данных через прямой доступ подробно описано в статье [Прямой доступ к данным](#).

Сформировать путь к колонке относительно корневой схемы

Основой механизма построения запроса на выборку с применением `EntitySchemaQuery` является корневая схема. **Корневая схема** — это таблица в базе данных, относительно которой строятся пути к колонкам в запросе, в том числе к колонкам присоединяемых таблиц. Для использования в запросе колонки из таблицы базы данных необходимо корректно задать путь к этой колонке.

При построении путей к колонкам применяется **принцип связей через справочные поля**. Имя колонки, добавляемой в запрос, строится в виде цепочки взаимосвязанных звеньев. Каждое звено представляет собой "контекст" конкретной схемы, которая связывается с предыдущей по справочной колонке.



Шаблон формирования пути к колонке из схемы N:

`КонтекстСхемы1.[...].КонтекстСхемыN.ИмяКолонкиИзСправочнойСхемы`.

Сформировать путь к колонке по прямым связям

Шаблон формирования пути к колонке по **прямым связям**:

`ИмяСправочнойКолонки.ИмяКолонкиСхемыИзСправочнойСхемы`.

Прямые связи используются, когда справочная колонка для связи присутствует в основной схеме и ссылается на присоединяемую схему. Например, есть корневая схема `[city]` со справочной колонкой `[Country]`, которая через колонку `[Id]` связана со справочной схемой `[Country]`.



Путь к колонке с наименованием страны, которой принадлежит город по прямым связям: `Country.Name`.

Здесь:

- `Country` — имя справочной колонки корневой схемы `[city]` (ссылается на схему `[Country]`).
- `Name` — имя колонки из справочной схемы `[Country]`.

Сформировать путь к колонке по обратным связям

Отличие присоединения по обратным связям от присоединения по прямым связям — справочное поле для присоединения должно быть у присоединяемой сущности, а не у основной.

Шаблон формирования пути к колонке по **обратным связям**:

`[ИмяПрисоединяемойСхемы:ИмяКолонкиДляСвязиПрисоединяемойСхемы:ИмяКолонкиДляСвязиТекущейСхемы].`

`ИмяКолонкиПрисоединяемойСхемы`

Путь к колонке с названием контрагента, у которого в поле `[Город]` указана выбранная в запросе запись `[city]` по обратным связям: `[Account:City:Id].Name`. Здесь:

- `Account` — имя присоединяемой схемы.
- `City` — имя колонки схемы `[Account]` для установки связи присоединяемой схемы.
- `Id` — имя справочной колонки схемы `[City]` для установки связи текущей схемы.
- `Name` — значение справочной колонки схемы `[Account]`.

Если в качестве колонки для связи у текущей схемы выступает колонка `[Id]`, то ее можно не указывать: `[ИмяПрисоединяемойСхемы:ИмяКолонкиДляСвязиПрисоединяемойСхемы].ИмяКолонкиПрисоединяемойСхемы`. Например, `[Account:City].Name`.

Получить данные из базы данных

Классы, которые реализуют получение данных из базы данных:

- `Terrasoft.Core.DB.Select` — получение данных из базы данных через прямой доступ. Подробнее читайте в статье [Прямой доступ к данным](#).
- `Terrasoft.Core.Entities.EntitySchemaQuery` — получение данных из базы данных через ORM-модель.

Назначение класса `Terrasoft.Core.Entities.EntitySchemaQuery` — построение запросов на выборку записей из таблиц базы данных. Максимальное количество записей, которые можно получить по запросу, задается настройкой `MaxEntityRowCount` (по умолчанию — 20 000). Изменить значение настройки можно в файле `...\\Terrasoft.WebApp\\Web.config`.

Важно. Не рекомендуется изменять настройку `MaxEntityRowCount`. Изменение настройки может

привести к проблемам производительности.

После создания и конфигурирования экземпляра класса будет построен `SELECT`-запрос к базе данных приложения. В запрос можно добавить колонки, фильтры и условия ограничений. Также можно задать параметры для постраничного вывода результатов выполнения запроса. Используя класс `Terrasoft.Core.Entities.EntitySchemaQueryOptions`, можно задать параметры построения иерархического запроса. Передача одного и того же экземпляра `EntitySchemaQueryOptions` в качестве параметра метода `GetEntityCollection()` соответствующего запроса позволяет получить результат выполнения различных запросов.

Особенности класса `Terrasoft.Core.Entities.EntitySchemaQuery`:

- В результирующем запросе учитываются права доступа текущего пользователя.
- Класс позволяет управлять правами доступа текущего пользователя на таблицы, присоединенные в запрос с помощью SQL-оператора `JOIN`.
- Класс позволяет работать с данными хранилища кэша или произвольного хранилища, которое определено пользователем.

При выполнении запроса данные, полученные из базы данных, помещаются в кэш. В качестве кэша запроса может выступать произвольное хранилище, которое реализует интерфейс `Terrasoft.Core.Store.ICacheStore`. По умолчанию используется кэш Creatio уровня сессии с локальным хранением данных. Кэш запроса определяется свойством `Cache` экземпляра класса `EntitySchemaQuery`. С помощью свойства `CacheItemName` задается ключ доступа к кэшу запроса. Уровни хранилищ Creatio подробно описаны в статье [Хранилища данных и кэш](#).

Результат выполнения запроса — экземпляр `Terrasoft.Nui.ServiceModel.DataContract.EntityCollection` или коллекция экземпляров класса `Terrasoft.Core.Entities.Entity`. Каждый экземпляр `Entity` в коллекции представляет собой строку набора данных, возвращаемого запросом.

Управление присоединенными таблицами

Класс `EntitySchemaQuery` позволяет указать тип присоединения схемы к запросу. Для добавления в запрос колонки из присоединяемой схемы используется оператор `JOIN`.

Шаблон формирования типа соединения к колонке присоединяемой схемы:

`СпецсимволТипаСоединенияИмяКолонкиДляСвязиПрисоединяемойСхемы`.

Описание типов соединения

Тип соединения	Спецсимвол типа соединения	Пример использования
INNER JOIN	=	=Country.Name
LEFT OUTER JOIN	>	>Country.Name
RIGHT OUTER JOIN	<	<Country.Name
FULL OUTER JOIN	<>	<>Country.Name
CROSS JOIN	*	*Country.Name

По умолчанию используется тип присоединения LEFT OUTER JOIN.

Ниже представлен пример добавления колонок в запрос с использованием разных типов соединения.

Пример добавления колонок в запрос

```
/* Создание экземпляра запроса с корневой схемой City. */
var esqResult = new EntitySchemaQuery(UserConnection.EntitySchemaManager, "City");

/* К запросу будет добавлена схема Country с типом присоединения LEFT OUTER JOIN. */
esqResult.AddColumn("Country.Name");

/* К запросу будет добавлена схема Country с типом присоединения INNER JOIN. */
esqResult.AddColumn("=Country.Name");

/* К запросу будут присоединены схема Country с типом присоединения LEFT OUTER JOIN и схема Cont
esqResult.AddColumn(">Country.<CreatedBy.Name");
```

В результате будет сформирован SQL-запрос.

SQL-запрос

```
SELECT
    [Country].[Name] [Country.Name],
    [Country1].[Name] [Country1.Name],
    [CreatedBy].[Name] [CreatedBy.Name]
FROM
    [dbo].[City] [City]
    LEFT OUTER JOIN [dbo].[Country] [Country] ON ([Country].[Id] = [City].[CountryId])
    INNER JOIN [dbo].[Country] [Country1] ON ([Country1].[Id] = [City].[CountryId])
    LEFT OUTER JOIN [dbo].[Country] [Country2] ON ([Country2].[Id] = [City].[CountryId])
    RIGHT OUTER JOIN [dbo].[Contact] [CreatedBy] ON ([CreatedBy].[Id] = [Country2].[CreatedBy])
```

Если запрос содержит корневую схему и присоединяемые схемы, которые администрируются по записям, то можно применить права доступа текущего пользователя. Варианты применения прав доступа по записям к присоединенным схемам заданы перечислением

```
Terrasoft.Core.DB.QueryJoinRightLevel .
```

Варианты применения прав доступа к присоединяемым схемам запроса:

- Всегда применяются.
- Применяются, если в присоединяемой схеме запроса используются колонки, отличные от первичной и первичной для отображения. Чаще всего это колонки `[Id]` и `[Name]` .
- Не применяются.

Порядок применения прав доступа определяется значением свойства `JoinRightState` запроса. Значение этого свойства по умолчанию определяется системной настройкой [Способ администрирования связанных объектов] (код `QueryJoinRightLevel`). Если значение этой системной настройки не задано, то права доступа применяются, если в присоединяемой схеме запроса используются колонки, отличные от первичной и первичной для отображения.

Управлять фильтрами в запросе

Фильтр — набор условий, применяемых при отображении данных запроса. В терминах SQL фильтр представляет собой отдельный предикат (условие) оператора `WHERE` .

Структура фильтра

```
Filter = {[AggregationType] {<LeftExpression> | <LeftExpressionColumnPath>}  
    <ComparisonType>  
    {{<RightExpression> | {<RightExpressionColumnPath>, ...}} | {<Macros>, [MacrosValue]}}  
}
```

Для создания простого фильтра в `EntitySchemaQuery` используется метод `CreateFilter()` , который возвращает экземпляр типа `EntitySchemaQueryFilter` . Для этого метода в `EntitySchemaQuery` реализован ряд перегрузок. Это позволяет создавать фильтры с разными исходными параметрами. В `EntitySchemaQuery` реализованы методы создания фильтров специального вида.

Экземпляр `EntitySchemaQuery` имеет свойство `Filters` , которое представляет собой коллекцию фильтров данного запроса (экземпляр класса `EntitySchemaQueryFilterCollection`). Экземпляр класса `EntitySchemaQueryFilterCollection` представляет собой типизированную коллекцию элементов `IEntitySchemaQueryFilterItem` .

Алгоритм добавления фильтра в запрос:

- Создайте экземпляр фильтра для данного запроса (метод `CreateFilter()` , методы создания фильтров специального вида).
- Добавьте созданный экземпляр фильтра в коллекцию фильтров запроса (метод `Add()` коллекции).

По умолчанию фильтры, добавляемые в коллекцию `Filters`, объединяются между собой логической операцией `AND`. При реализации логической операции `OR` необходимо использовать свойство `LogicalOperation` коллекции `Filters`. Это свойство принимает значения перечисления `LogicalOperationStrict` и позволяет указать логическую операцию, которой необходимо объединять фильтры.

В запросах `EntitySchemaQuery` реализована возможность управления фильтрами, участвующими в построении результирующего набора данных. Каждый элемент коллекции `Filters` имеет свойство `IsEnabled`, которое определяет, участвует ли данный элемент в построении результирующего запроса (`true` — участвует, `false` — не участвует). Аналогичное свойство `IsEnabled` также определено для коллекции `Filters`. Установив это свойство в `false`, можно отключить фильтрацию для запроса, при этом коллекция фильтров запроса останется неизменной. Таким образом, изначально сформировав коллекцию фильтров запроса, в дальнейшем можно использовать различные комбинации, не внося изменений в саму коллекцию.

Управлять сущностью базы данных

`Terrasoft.Core.Entities.Entity` — класс, который реализует работу с сущностью базы данных.

Назначение класса `Terrasoft.Core.Entities.Entity` — доступ к объекту, который является записью в таблице базы данных. Класс также можно использовать для CRUD-операций над соответствующими записями.

Управлять сущностями базы данных



Сложный

На заметку. Примеры 1-5, приведенные в этой статье, реализованы в веб-сервисе. Пакет с реализацией веб-сервиса прикреплен в блоке "Ресурсы".

Пример 1

Пример. Получить значение колонки схемы [`City`] с именем `Name`.

Метод `GetEntityColumnData`

```
public string GetEntityColumnData()
{
    var result = "";
    /* Создание запроса к схеме City, добавление в запрос колонки Name. */
    var esqResult = new EntitySchemaQuery(UserConnection.EntitySchemaManager, "City");
    var colName = esqResult.AddColumn("Name");
    /* Выполнение запроса к базе данных и получение объекта с заданным идентификатором. UIId объе
```

```

var entity = esqResult.GetEntity(UserConnection, new Guid("100B6B13-E8BB-DF11-B00F-001D60E93
/* Получение значения колонки объекта. */
result += entity.GetColumnValue(colName.Name).ToString();
return result;
}

```

Пример 2

Пример. Получить коллекцию имен колонок схемы [*City*].

Метод `GetEntityColumns`

```

public IEnumerable<string> GetEntityColumns()
{
    /* Создание объекта строки данных схемы City (по идентификатору схемы, полученному из базы д
    var entity = new Entity(UserConnection, new Guid("5CA90B6A-93E7-4448-BEFE-AB5166EC2CFE"));
    /* Получение из базы данных объекта с заданным идентификатором. UId объекта можно получить и
    entity.FetchFromDB(new Guid("100B6B13-E8BB-DF11-B00F-001D60E938C6"),true);
    /* Получение коллекции имен колонок объекта. */
    var result = entity.GetColumnValueNames();
    return result;
}

```

Пример 3

Пример. Удалить из базы данных записи схемы [*Order*].

Метод `DeleteEntity`

```

public bool DeleteEntity()
{
    /* Создание запроса к схеме Order, добавление в запрос всех колонок схемы. */
    var esqResult = new EntitySchemaQuery(UserConnection.EntitySchemaManager, "Order");
    esqResult.AddAllSchemaColumns();
    /* Выполнение запроса к базе данных и получение объекта с заданным идентификатором. UId объе
    var entity = esqResult.GetEntity(UserConnection, new Guid("e3bfa32f-3fe9-4bae-9332-16c162c51
    /* Удаление объекта из базы данных. */
    entity.Delete();
    /* Проверка, существует ли в базе данных объект с заданным идентификатором. */
    var result = entity.ExistInDB(new Guid("e3bfa32f-3fe9-4bae-9332-16c162c51e0d"));
}

```

```

    return result;
}

```

Пример 4

Пример. Изменить статус заказа.

Метод UpdateEntity

```

public bool UpdateEntity()
{
    /* Создание запроса к схеме Order, добавление в запрос всех колонок схемы. */
    var esqResult = new EntitySchemaQuery(UserConnection.EntitySchemaManager, "Order");
    esqResult.AddAllSchemaColumns();
    /* Выполнение запроса к базе данных и получение объекта с заданным идентификатором. UInt обьес*/
    var entity = esqResult.GetEntity(UserConnection, new Guid("58be5223-715d-4b16-a5c4-e3d4ec041
    /* Создание объекта строки данных схемы OrderStatus. */
    var statusSchema = UserConnection.EntitySchemaManager.GetInstanceByName("OrderStatus");
    var newStatus = statusSchema.CreateEntity(UserConnection);
    /* Получение из базы данных объекта с заданным названием. */
    newStatus.FetchFromDB("Name", "4. Completed");
    /* Присваивает колонке StatusId новое значение. */
    entity.SetColumnValue("StatusId", newStatus.GetTypedColumnValue<Guid>("Id"));
    /* Сохранение измененного объекта в базе данных. */
    var result = entity.Save();
    return result;
}

```

Пример 5

Пример. Добавить город с указанным названием, привязав его к указанной стране.

Метод InsertEntity

```

public bool InsertEntity(string city, string country)
{
    city = city ?? "unknown city";
    country = country ?? "unknown country";
    var citySchema = UserConnection.EntitySchemaManager.GetInstanceByName("City");
    var entity = citySchema.CreateEntity(UserConnection);

```

```

entity.FetchFromDB("Name", city);
/* Устанавливает для колонок объекта значения по умолчанию. */
entity.SetDefColumnValues();
var contryEntity = new Entity(UserConnection, new Guid("09FCE1F8-515C-4296-95CD-8CD93F79A6CF");
contryEntity.FetchFromDB("Name", country);
/* Присваивает колонке Name переданное название города. */
entity.SetColumnValue("Name", city);
/* Присваивает колонке CountryId UId переданной страны. */
entity.SetColumnValue("CountryId", contryEntity.GetTypedColumnValue<Guid>("Id"));
var result = entity.Save();
return result;
}

```

Пример 6

Пример. Создать контакт с именем "User01".

```

EntitySchema contactSchema = UserConnection.EntitySchemaManager.GetInstanceByName("Contact");
Entity contactEntity = contactSchema.CreateEntity(UserConnection);
contactEntity.SetDefColumnValues();
contactEntity.SetColumnValue("Name", "User01");
contactEntity.Save();

```

Пример 7

Пример. Изменить имя контакта на "User02".

```

EntitySchema entitySchema = UserConnection.EntitySchemaManager.GetInstanceByName("Contact");
Entity entity = entitySchema.CreateEntity(UserConnection);
if (!entity.FetchFromDB(some_id)) {
    return false;
}
entity.SetColumnValue("Name", "User02");
return entity.Save();

```

Пример 8

Пример. Удалить контакт с именем "User02".

```
EntitySchema entitySchema = UserConnection.EntitySchemaManager.GetInstanceByName("Contact");
Entity entity = entitySchema.CreateEntity(UserConnection);
var fetchConditions = new Dictionary<string, object> {
    {"Name", "User02"}
};
if (entity.FetchFromDB(fetchConditions)) {
    entity.Delete();
}
```

Получить данные из базы данных с учетом прав пользователя

 Сложный

На заметку. Примеры, приведенные в этой статье, реализованы в веб-сервисе. Пакет с реализацией веб-сервиса прикреплен в блоке "Ресурсы".

Пример 1

Пример. Создать экземпляр `EntitySchemaQuery`.

Метод `CreateESQ`

```
public string CreateESQ()
{
    var result = "";
    /* Получение экземпляра менеджера схем объектов. */
    EntitySchemaManager esqManager = SystemUserConnection.EntitySchemaManager;
    /* Получение экземпляра схемы, которая будет установлена в качестве корневой для создаваемого запроса. */
    var rootEntitySchema = esqManager.GetInstanceByName("City") as EntitySchema;
    /* Создание экземпляра EntitySchemaQuery, у которого в качестве корневой схемы установлена схема City. */
    var esqResult = new EntitySchemaQuery(rootEntitySchema);
    /* Добавление колонок, которые будут выбираться в результирующем запросе. */
    esqResult.AddColumn("Id");
    esqResult.AddColumn("Name");
    /* Получение экземпляра Select, ассоциированного с созданным запросом EntitySchemaQuery. */
    Select selectEsq = esqResult.GetSelectQuery(SystemUserConnection);
```

```
/* Получение текста результирующего запроса созданного экземпляра EntitySchemaQuery. */
result = selectEsq.GetSqlText();
return result;
}
```

Пример 2

Пример. Создать клон экземпляра `EntitySchemaQuery`.

Метод `CreateESQClone`

```
public string CreateESQClone()
{
    var result = "";
    EntitySchemaManager esqManager = SystemUserConnection.EntitySchemaManager;
    var esqSource = new EntitySchemaQuery(esqManager, "Contact");
    esqSource.AddColumn("Id");
    esqSource.AddColumn("Name");
    /* Создание экземпляра EntitySchemaQuery, являющегося клоном экземпляра esqSource. */
    var esqClone = new EntitySchemaQuery(esqSource);
    result = esqClone.GetSelectQuery(SystemUserConnection).GetSqlText();
    return result;
}
```

Пример 3

Пример. Получить результат выполнения запроса.

Метод `GetEntitiesExample`

```
public string GetEntitiesExample()
{
    var result = "";
    /* Создание запроса к схеме City, добавление в запрос колонки Name. */
    var esqResult = new EntitySchemaQuery(UserConnection.EntitySchemaManager, "City");
    var colName = esqResult.AddColumn("Name");

    /* Выполнение запроса к базе данных и получение всей результирующей коллекции объектов. */
    var entities = esqResult.GetEntityCollection(UserConnection);
    for (int i=0; i < entities.Count; i++) {
```

```

        result += entities[i].GetColumnValue(colName.Name).ToString();
        result += "\n";
    }

/* Выполнение запроса к базе данных и получение объекта с заданным идентификатором. */
var entity = esqResult.GetEntity(UserConnection, new Guid("100B6B13-E8BB-DF11-B00F-001D60E93
result += "\n";
result += entity.GetColumnValue(colName.Name).ToString();
return result;
}

```

Пример 4

Пример. Использовать кэш запроса `EntitySchemaQuery`.

Метод `UsingCacheExample`

```

public Collection<string> UsingCacheExample()
{
    /* Создание запроса к схеме City, добавление в запрос колонки Name. */
    var esqResult = new EntitySchemaQuery(UserConnection.EntitySchemaManager, "City");
    esqResult.AddColumn("Name");

    /* Определение ключа, под которым в кэше будут храниться результаты выполнения запроса. В ка
    esqResult.CacheItemName = "EsqResultItem";

    /* Коллекция, в которую будут помещены результаты выполнения запроса. */
    var esqCityNames = new Collection<string>();

    /* Коллекция, в которую будут помещаться закэшированные результаты выполнения запроса. */
    var cachedEsqCityNames = new Collection<string>();

    /* Выполнение запроса к базе данных и получение результирующей коллекции объектов.
    После выполнения этой операции результаты запроса будут помещены в кэш. */
    var entities = esqResult.GetEntityCollection(UserConnection);

    /* Обработка результатов выполнения запроса и заполнение коллекции esqCityNames. */
    foreach (var entity in entities)
    {
        esqCityNames.Add(entity.GetTypedColumnValue<string>("Name"));
    }

    /* Получение ссылки на кэш запроса esqResult по ключу CacheItemName в виде таблицы данных в
    var esqCacheStore = esqResult.Cache[esqResult.CacheItemName] as DataTable;
}

```

```

/* Заполнение коллекции cachedEsqCityNames значениями из кэша запроса. */
if (esqCacheStore != null)
{
    foreach (DataRow row in esqCacheStore.Rows)
    {
        cachedEsqCityNames.Add(row[0].ToString());
    }
}
return cachedEsqCityNames;
}

```

Пример 5

Пример. Использовать дополнительные настройки запроса `EntitySchemaQuery`.

Метод `UsingCacheExample`

```

public Collection<string> ESQOptionsExample()
{
    /* Создание экземпляра запроса с корневой схемой City. */
    var esqCities = new EntitySchemaQuery(UserConnection.EntitySchemaManager, "City");
    esqCities.AddColumn("Name");

    /* Создание запроса с корневой схемой Country. */
    var esqCountries = new EntitySchemaQuery(UserConnection.EntitySchemaManager, "Country");
    esqCountries.AddColumn("Name");

    /* Создание экземпляра настроек для возврата запросом первых 5 строк. */
    var esqOptions = new EntitySchemaQueryOptions()
    {
        PageableDirection = PageableSelectDirection.First,
        PageableRowCount = 5,
        PageableConditionValues = new Dictionary<string, object>()
    };

    /* Получение коллекции городов, которая будет содержать первые 5 городов результирующего набора */
    var cities = esqCities.GetEntityCollection(UserConnection, esqOptions);

    /* Получение коллекции стран, которая будет содержать первые 5 стран результирующего набора */
    var countries = esqCountries.GetEntityCollection(UserConnection, esqOptions);
    var esqStringCollection = new Collection<string>();
    foreach (var entity in cities)
    {

```

```

        esqStringCollection.Add(entity.GetTypedColumnValue<string>("Name"));
    }
    foreach (var entity in countries)
    {
        esqStringCollection.Add(entity.GetTypedColumnValue<string>("Name"));
    }
    return esqStringCollection;
}

```

Класс EntitySchemaQuery

 Сложный

Пространство имен `Terrasoft.Core.Entities`.

Класс `Terrasoft.Core.Entities.EntitySchemaQuery` предназначен для построения запросов выборки записей из таблиц базы данных с учетом прав доступа текущего пользователя. В результате создания и конфигурирования экземпляра этого класса будет построен запрос в базу данных приложения в виде SQL-выражения `SELECT`. В запрос можно добавить требуемые колонки, фильтры и условия ограничений.

На заметку. Полный перечень методов и свойств класса `EntitySchemaQuery`, его родительских классов, а также реализуемых им интерфейсов, можно найти в документации "[.NET библиотеки классов ядра платформы](#)".

Конструкторы

`EntitySchemaQuery(EntitySchema rootSchema)`

Создает экземпляр класса, в котором в качестве корневой схемы устанавливается переданный экземпляр `EntitySchema`. В качестве менеджера схем устанавливается менеджер переданного экземпляра корневой схемы.

`EntitySchemaQuery(EntitySchemaManager entitySchemaManager, string sourceSchemaName)`

Создает экземпляр класса с указанным `EntitySchemaManager` и корневой схемы с именем, переданным в качестве аргумента.

`EntitySchemaQuery(EntitySchemaQuery source)`

Создает экземпляр класса, являющийся клоном экземпляра, переданного в качестве аргумента.

Свойства

Cache `Terrasoft.Core.Store.ICacheStore`

Кэш запроса.

CacheItemName `string`

Имя элемента кэша.

CanReadUncommittedData `bool`

Определяет, попадут ли в результаты запроса данные, для которых не завершена транзакция.

Caption `Terrasoft.Common.LocalizableString`

Заголовок.

ChunkSize `int`

Количество строк запроса в одном чанке.

Columns `Terrasoft.Core.Entities.EntitySchemaQueryColumnCollection`

Коллекция колонок текущего запроса к схеме объекта.

DataValueTypeManager `DataValueTypeManager`

Менеджер значений типов данных.

EntitySchemaManager `Terrasoft.Core.Entities.EntitySchemaManager`

Менеджер схем объектов.

Filters `Terrasoft.Core.Entities.EntitySchemaQueryFilterCollection`

Коллекция фильтров текущего запроса к схеме объекта.

HideSecurityValue `bool`

Определяет, будут ли скрыты значения зашифрованных колонок.

IgnoreDisplayValues `bool`

Определяет, будут ли в запросе использоваться отображаемые значения колонок.

IsDistinct bool

Определяет, убирать ли дубли в результирующем наборе данных.

IsInherited bool

Определяет, является ли запрос унаследованным.

JoinRightState QueryJoinRightLevel

Определяет условие наложения прав при использовании связанных таблиц, если схема администрируется по записям.

Manager Terrasoft.Core.IManager

Менеджер схем.

ManagerItem Terrasoft.Core.IManagerItem

Элемент менеджера.

Name string

Имя.

ParentCollection Terrasoft.Core.Entities.EntitySchemaQueryCollection

Коллекция запросов, которой принадлежит текущий запрос к схеме объекта.

ParentEntitySchema Terrasoft.Core.Entities.EntitySchema

Родительская схема запроса.

PrimaryQueryColumn Terrasoft.Core.Entities.EntitySchemaQueryColumn

Колонка, созданная по первичной колонке корневой схемы. Заполняется при первом обращении.

QueryOptimize bool

Разрешает использование оптимизации запроса.

RootSchema Terrasoft.Core.Entities.EntitySchema

Корневая схема.

`RowCount int`

Количество строк, возвращаемых запросом.

`SchemaAliasPrefix string`

Префикс, используемый для создания псевдонимов схем.

`SkipRowCount int`

Количество строк, которые необходимо пропустить при возврате результата запроса.

`UseAdminRights bool`

Определяет будут ли учитываться права при построении запроса получения данных.

`UseLocalization bool`

Определяет, будут ли использоваться локализованные данные.

`UseOffsetFetchPaging bool`

Определяет возможность постраничного возврата результата запроса.

`UseRecordDeactivation bool`

Определяет, будут ли данные исключены из фильтрации.

`AdminUnitRoleSources int`

Целочисленное свойство, которое соответствует критериям фильтрации записей по источнику вхождения пользователя в роли. Значение по умолчанию: `0`.

Чтобы сформировать `AdminUnitRoleSources`, необходимо с помощью побитового или " | " перечислить следующие константы из серверного класса `AdminUnitRoleSources`:

- `ExplicitEntry`.
- `Delegated`.
- `FuncRoleFromOrgRole`.
- `UpHierarchy`.
- `AsManager`.

- `All`.
- `None`.

В результате, отработает следующее правило: возвращать запись только, если у пользователя есть хоть одна роль, которой доступна запись, и пользователь входит в эту роль в соответствии с источниками, указанными в условиях фильтрации.

Методы

```
void AddAllSchemaColumns(bool skipSystemColumns)
```

В коллекцию колонок текущего запроса к схеме объекта добавляет все колонки корневой схемы.

```
EntitySchemaQueryColumn AddColumn(string columnPath, AggregationTypeStrict aggregationType, out void AddColumn(EntitySchemaQueryColumn queryColumn)
EntitySchemaQueryColumn AddColumn(string columnPath)
EntitySchemaQueryColumn AddColumn(EntitySchemaQueryFunction function)
EntitySchemaQueryColumn AddColumn(object parameterValue, DataValueType parameterDataValueType)
EntitySchemaQueryColumn AddColumn(EntitySchemaQuery subQuery)
```

Создает и добавляет колонку в текущий запрос к схеме объекта.

Параметры

<code>columnPath</code>	Путь к колонке схемы относительно корневой схемы.
<code>aggregationType</code>	Тип агрегирующей функции. В качестве параметра передаются значения перечисления типов агрегирующей функции <code>Terrasoft.Common.AggregationTypeStrict</code> .
<code>subQuery</code>	Ссылка на созданный подзапрос, помещенный в колонку.
<code>queryColumn</code>	Экземпляр <code>EntitySchemaQueryColumn</code> , добавляемый в коллекцию колонок текущего запроса.
<code>function</code>	Экземпляр функции <code>EntitySchemaQueryFunction</code> .
<code>parameterValue</code>	Значение параметра, добавляемого в запрос в качестве колонки.
<code>parameterDataValueType</code>	Тип значения параметра, добавляемого в запрос в качестве колонки.

```
void ClearCache()
```

Очищает кэш текущего запроса.

```
static void ClearDefCache(string cacheItemName)
```

Удаляет из кэша запроса элемент с заданным именем `cacheItemName`.

```
object Clone()
```

Создает клон текущего экземпляра `EntitySchemaQuery`.

```
EntitySchemaQueryExpression CreateAggregationEntitySchemaExpression(string leftExprColumnPath, A
```

Возвращает выражение агрегирующей функции с заданным типом агрегации из перечисления `Terrasoft.Common.AggregationTypeStrict` для колонки, расположенной по заданному пути `leftExprColumnPath`.

```
static EntitySchemaQueryExpression CreateParameterExpression(object parameterValue)
static EntitySchemaQueryExpression CreateParameterExpression(object parameterValue, DataValueType
static EntitySchemaQueryExpression CreateParameterExpression(object parameterValue, string displ
```

Создает выражение для параметра запроса.

Параметры

<code>parameterValue</code>	Значение параметра.
<code>valueType</code>	Тип значения параметра.
<code>displayValue</code>	Значение для отображения параметра.

```
static IEnumerable<EntitySchemaQueryExpression> CreateParameterExpressions(DataValueType valueType, params object[] parameter
static IEnumerable<EntitySchemaQueryExpression> CreateParameterExpressions(DataValueType valueType, IEnumerable<object> parameter
```

Создает коллекцию выражений для параметров запроса с определенным типом данных `DataValueType`.

```
static EntitySchemaQueryExpression CreateSchemaColumnExpression(EntitySchemaQuery parentQuery, E
static EntitySchemaQueryExpression CreateSchemaColumnExpression(EntitySchema rootSchema, string
EntitySchemaQueryExpression CreateSchemaColumnExpression(string columnPath, bool useCoalesceFunc
```

Возвращает выражение колонки схемы объекта.

Параметры

parentQuery	Запрос к схеме объекта, для которого создается выражение колонки.
rootSchema	Корневая схема.
columnPath	Путь к колонке относительно корневой схемы.
useCoalesceFunctionForMultiLookup	Признак, использовать ли для колонки типа справочник функцию <code>COALESCE</code> . Необязательный параметр, по умолчанию равен <code>true</code> .
useDisplayValue	Признак, использовать ли для колонки значение для отображения. Необязательный параметр, по умолчанию равен <code>false</code> .

```
Enumerable CreateSchemaColumnExpressions(params string[] columnPaths)
IEnumerable CreateSchemaColumnExpressions(IEnumerable columnPaths, bool useCoalesceFunctionForMultiLookup)
```

Возвращает коллекцию выражений колонок запроса к схеме объекта по заданной коллекции путей к колонкам `columnPaths`.

```
IEnumerable CreateSchemaColumnExpressionsWithoutCoalesce(params string[] columnPaths)
```

Возвращает коллекцию выражений колонок запроса к схеме объекта по заданному массиву путей к колонкам. При этом, если колонка имеет тип множественный справочник, к ее значениям не применяется функция `COALESCE`.

```
static EntitySchemaQueryExpression CreateSchemaColumnQueryExpression(string columnPath, EntitySchemaQueryExpression)
static EntitySchemaQueryExpression CreateSchemaColumnQueryExpression(string columnPath, EntitySchemaQueryExpression)
```

Возвращает выражение запроса к схеме объекта по заданным путем к колонке, корневой схеме и экземпляру колонки схемы. При этом для колонки можно определить, какой тип ее значения использовать в выражении — хранимое значение или значение для отображения.

```
EntitySchemaQueryExpression CreateSubEntitySchemaExpression(string leftExprColumnPath)
```

Возвращает выражение подзапроса к схеме объекта для колонки, расположенной по заданному пути `leftExprColumnPath`.

```
EntitySchemaAggregationQueryFunction CreateAggregationFunction(AggregationTypeStrict aggregationType)
```

Возвращает экземпляр агрегирующей функции `EntitySchemaAggregationQueryFunction` с заданным типом агрегации из перечисления `Terrasoft.Common.AggregationTypeStrict` для колонки по указанному пути относительно корневой схемы `columnPath`.

`EntitySchemaCaseNotNullQueryFunction CreateCaseNotNullFunction(params EntitySchemaCaseNotNullQueryFunction[] expressions)`

Возвращает экземпляр `CASE`-функции `EntitySchemaCaseNotNullQueryFunction` для заданного массива выражений условий `EntitySchemaCaseNotNullQueryFunctionWhenItem`.

`EntitySchemaCaseNotNullQueryFunctionWhenItem CreateCaseNotNullQueryFunctionWhenItem(string whenCondition)`

Возвращает экземпляр выражения для SQL-конструкции вида

$$\text{WHEN } <\!\text{Выражение_1}\!> \text{ IS NOT NULL THEN } <\!\text{Выражение_2}\!>$$

Параметры

<code>whenColumnPath</code>	Путь к колонке, содержащей выражение предложения <code>WHEN</code> .
<code>thenParameterValue</code>	Путь к колонке, содержащей выражение предложения <code>THEN</code> .

`EntitySchemaCastQueryFunction CreateCastFunction(string columnPath, DBDataValueType castType)`

Возвращает экземпляр `CAST`-функции `EntitySchemaCastQueryFunction` для выражения колонки, расположенной по заданному пути относительно корневой схемы `columnPath`, и указанным целевым типом данных `DBDataValueType`.

`EntitySchemaCoalesceQueryFunction CreateCoalesceFunction(params string[] columnPaths)`

`static EntitySchemaCoalesceQueryFunction CreateCoalesceFunction(EntitySchemaQuery parentQuery, EntitySchemaQuery childQuery)`

`static EntitySchemaCoalesceQueryFunction CreateCoalesceFunction(EntitySchema rootSchema, params EntitySchemaQuery[] children)`

Возвращает экземпляр функции, возвращающей первое не равное `null` выражение из списка аргументов, для заданных колонок.

Параметры

<code>columnPaths</code>	Массив путей к колонкам относительно корневой схемы.
<code>parentQuery</code>	Запрос к схеме объекта, для которого создается экземпляр функции.
<code>rootSchema</code>	Корневая схема.

`EntitySchemaConcatQueryFunction CreateConcatFunction(params EntitySchemaQueryExpression[] expressions)`

Возвращает экземпляр функции для формирования строки, являющейся результатом объединения строковых значений аргументов функции для заданного массива выражений `EntitySchemaQueryExpression`.

```
EntitySchemaDatePartQueryFunction CreateDatePartFunction(EntitySchemaDatePartQueryFunctionInterval)
```

Возвращает экземпляр `DATEPART`-функции `EntitySchemaDatePartQueryFunction`, определяющей интервал даты, заданный перечислением `EntitySchemaDatePartQueryFunctionInterval` (месяц, день, час, год, день недели...), для значения колонки, расположенной по указанному пути относительно корневой схемы.

```
EntitySchemaDatePartQueryFunction CreateDayFunction(string columnPath)
```

Возвращает экземпляр функции `EntitySchemaDatePartQueryFunction`, определяющей интервал даты [День] для значения колонки, расположенной по указанному пути относительно корневой схемы.

```
EntitySchemaDatePartQueryFunction CreateHourFunction(string columnPath)
```

Возвращает экземпляр функции `EntitySchemaDatePartQueryFunction`, возвращающей часть даты [Час] для значения колонки, расположенной по указанному пути относительно корневой схемы.

```
EntitySchemaDatePartQueryFunction CreateHourMinuteFunction(string columnPath)
```

Возвращает экземпляр функции `EntitySchemaDatePartQueryFunction`, возвращающей часть даты [Минута] для значения колонки, расположенной по указанному пути относительно корневой схемы.

```
EntitySchemaDatePartQueryFunction CreateMonthFunction(string columnPath)
```

Возвращает экземпляр функции `EntitySchemaDatePartQueryFunction`, возвращающей часть даты [Месяц] для значения колонки, расположенной по указанному пути относительно корневой схемы.

```
EntitySchemaDatePartQueryFunction CreateWeekdayFunction(string columnPath)
```

Возвращает экземпляр функции `EntitySchemaDatePartQueryFunction`, возвращающей часть даты [День недели] для значения колонки, расположенной по указанному пути относительно корневой схемы.

```
EntitySchemaDatePartQueryFunction CreateWeekFunction(string columnPath)
```

Возвращает экземпляр функции `EntitySchemaDatePartQueryFunction`, возвращающей часть даты [Неделя] для значения колонки, расположенной по указанному пути относительно корневой схемы.

```
EntitySchemaDatePartQueryFunction CreateYearFunction(string columnPath)
```

Возвращает экземпляр функции `EntitySchemaDatePartQueryFunction`, возвращающей часть даты [Год] для значения колонки, расположенной по указанному пути относительно корневой схемы.

```
EntitySchemaIsNullQueryFunction CreateIsNullFunction(string checkColumnPath, string replacementC
```

Возвращает экземпляр функции `EntitySchemaIsNullQueryFunction` для колонок с проверяемым и замещающим значениями, которые расположены по заданным путем относительно корневой схемы.

```
EntitySchemaLengthQueryFunction CreateLengthFunction(string columnPath)
EntitySchemaLengthQueryFunction CreateLengthFunction(params EntitySchemaQueryExpression[] express
```

Создание экземпляра функции `LEN` (функция для возврата длины выражения) для выражения колонки по заданному пути относительно корневой схемы или для заданного массива выражений.

```
EntitySchemaTrimQueryFunction CreateTrimFunction(string columnPath)
EntitySchemaTrimQueryFunction CreateTrimFunction(params EntitySchemaQueryExpression[] expressio
```

Возвращает экземпляр функции `TRIM` (функция для удаления начальных и конечных пробелов из выражения) для выражения колонки по заданному пути относительно корневой схемы или для заданного массива выражений.

```
EntitySchemaUpperQueryFunction CreateUpperFunction(string columnPath)
```

Возвращает экземпляр функции `EntitySchemaUpperQueryFunction`, для преобразования символов выражения аргумента к верхнему регистру, для выражения колонки по заданному пути относительно корневой схемы.

```
EntitySchemaCurrenddateQueryFunction CreateCurrenddateFunction()
```

Возвращает экземпляр функции `EntitySchemaCurrenddateQueryFunction`, определяющей текущую дату.

```
EntitySchemaCurrenddateTimeQueryFunction CreateCurrenddateTimeFunction()
```

Возвращает экземпляр функции `EntitySchemaCurrenddateTimeQueryFunction`, определяющей текущие дату и время.

```
EntitySchemaCurrentTimeQueryFunction CreateCurrentTimeFunction()
```

Возвращает экземпляр функции `EntitySchemaCurrentTimeQueryFunction`, определяющей текущее время.

```
EntitySchemaCurrentUserAccountQueryFunction CreateCurrentUserAccountFunction()
```

Возвращает экземпляр функции `EntitySchemaCurrentUserAccountQueryFunction`, определяющей идентификатор контрагента текущего пользователя.

```
EntitySchemaCurrentUserContactQueryFunction CreateCurrentUserContactFunction()
```

Возвращает экземпляр функции `EntitySchemaCurrentUserContactQueryFunction`, определяющей

идентификатор контакта текущего пользователя.

```
EntitySchemaCurrentUserQueryFunction CreateCurrentUserFunction()
```

Возвращает экземпляр функции `EntitySchemaCurrentUserQueryFunction`, определяющей текущего пользователя.

```
EntitySchemaQueryFilter CreateExistsFilter(string rightExpressionColumnPath)
```

Создает фильтр сравнения типа [Существует по заданному условию] и устанавливает в качестве проверяемого значения выражение колонки, расположенной по пути `rightExpressionColumnPath`.

```
IEntitySchemaQueryFilterItem CreateFilter(FilterComparisonType comparisonType, string leftExpres
IEntitySchemaQueryFilterItem CreateFilter(FilterComparisonType comparisonType, string leftExpres
IEntitySchemaQueryFilterItem CreateFilter(FilterComparisonType comparisonType, string leftExpres
IEntitySchemaQueryFilterItem CreateFilter(FilterComparisonType comparisonType, EntitySchemaQuery
IEntitySchemaQueryFilterItem CreateFilter(FilterComparisonType comparisonType, EntitySchemaQuery
IEntitySchemaQueryFilterItem CreateFilter(FilterComparisonType comparisonType, EntitySchemaQuery
IEntitySchemaQueryFilterItem CreateFilter(FilterComparisonType comparisonType, string leftExpres
EntitySchemaQueryFilter CreateFilter(FilterComparisonType comparisonType, string leftExprColumnF
IEntitySchemaQueryFilterItem CreateFilter(FilterComparisonType comparisonType, string leftExprCc
IEntitySchemaQueryFilterItem CreateFilter(FilterComparisonType comparisonType, string leftExprCc
```

Создает фильтр запроса для выборки записей по определенным условиям.

Параметры

<code>comparisonType</code>	Тип сравнения из перечисления <code>Terrasoft.Core.Entities.FilterComparisonType</code> .
<code>leftExpressionColumnPath</code>	Путь к колонке, содержащей выражение левой части фильтра.
<code>leftExpression</code>	Выражение в левой части фильтра.
<code>leftExprAggregationType</code>	Тип агрегирующей функции.
<code>leftExprSubQuery</code>	Параметр, в котором возвращается подзапрос для выражения в левой части фильтра (если он не равен <code>null</code>) либо подзапрос для первого выражения в правой части фильтра (если выражение левой части фильтра равно <code>null</code>).
<code>rightExpressionColumnPaths</code>	Массив путей к колонкам, содержащим выражения правой части фильтра.
<code>rightExpression</code>	Выражение в правой части фильтра.
<code>rightExpressionValue</code>	Экземпляр функции выражения в правой части фильтра (тип параметра <code>EntitySchemaQueryFunction</code>) или выражение подзапроса в правой части фильтра (тип параметра <code>EntitySchemaQuery</code>).
<code>rightValue</code>	Значение, которое обрабатывается макросом в правой части фильтра.
<code>rightExprParameterValue</code>	Значение параметра, к которому применяется агрегирующая функция в правой части фильтра.
<code>macroType</code>	Тип макроса из перечисления <code>Terrasoft.Core.Entities.EntitySchemaQueryMacroType</code> .
<code>daysCount</code>	Значение, к которому применяется макрос в правой части фильтра. Необязательный параметр, по умолчанию равен 0.

```
IEntitySchemaQueryFilterItem CreateFilterWithParameters(FilterComparisonType comparisonType, boc
IEntitySchemaQueryFilterItem CreateFilterWithParameters(FilterComparisonType comparisonType, str
IEntitySchemaQueryFilterItem CreateFilterWithParameters(FilterComparisonType comparisonType, str
static IEntitySchemaQueryFilterItem CreateFilterWithParameters(EntitySchemaQuery parentQuery, Er
static IEntitySchemaQueryFilterItem CreateFilterWithParameters(EntitySchema rootSchema, FilterCc
```

Создает параметризованный фильтр для выборки записей по определенным условиям.

Параметры

parentQuery	Родительский запрос, для которого создается фильтр.
rootSchema	Корневая схема.
comparisonType	Тип сравнения из перечисления <code>Terrasoft.Core.Entities.FilterComparisonType</code> .
useDisplayValue	Признак типа значения колонки, которое используется в фильтре: <code>true</code> - значение для отображения; <code>false</code> - хранимое значение.
leftExpressionColumnPath	Путь к колонке, содержащей выражение левой части фильтра.
rightExpressionParameterValues	Коллекция выражений параметров в правой части фильтра.

`IEntitySchemaQueryFilterItem CreateIsNotNullFilter(string leftExpressionColumnPath)`

Создает фильтр сравнения типа [Не является null в базе данных], устанавливая в качестве проверяемого значения выражение колонки, расположенной по указанному в параметре `leftExpressionColumnPath` пути.

`IEntitySchemaQueryFilterItem CreateIsNullFilter(string leftExpressionColumnPath)`

Создает фильтр сравнения типа [Является null в базе данных], устанавливая в качестве условия проверки выражение колонки, расположенной по указанному в параметре `leftExpressionColumnPath` пути.

`EntitySchemaQueryFilter CreateNotExistsFilter(string rightExpressionColumnPath)`

Создает фильтр сравнения типа [Не существует по заданному условию] и устанавливает в качестве проверяемого значения выражение колонки, расположенной по пути `rightExpressionColumnPath`.

`DataTable GetDataTable(UserConnection userConnection)`

Возвращает результат выполнения текущего запроса к схеме объекта в виде таблицы данных в памяти, используя пользовательское подключение `UserConnection`.

`static int GetDayOfWeekNumber(UserConnection userConnection, DayOfWeek dayOfWeek)`

Возвращает порядковый номер дня недели для объекта `System.DayOfWeek` с учетом региональных

настроек.

```
Entity GetEntity(UserConnection userConnection, object primaryColumnName)
```

Возвращает экземпляр Entity по первичному ключу `primaryColumnName`, используя пользовательское подключение `UserConnection`.

```
EntityCollection GetEntityCollection(UserConnection userConnection, EntitySchemaQueryOptions opt)
EntityCollection GetEntityCollection(UserConnection userConnection)
```

Возвращает коллекцию экземпляров `Entity`, представляющих результаты выполнения текущего запроса, используя пользовательское подключение `UserConnection` и заданные дополнительные настройки запроса `EntitySchemaQueryOptions`.

```
EntitySchema GetSchema()
```

Возвращает экземпляр схемы объекта `EntitySchema` текущего экземпляра `EntitySchemaQuery`.

```
Select GetSelectQuery(UserConnection userConnection)
```

```
Select GetSelectQuery(UserConnection userConnection, EntitySchemaQueryOptions options)
```

Возвращает экземпляр запроса на выборку данных, используя пользовательское подключение `UserConnection` и заданные дополнительные настройки запроса `EntitySchemaQueryOptions`.

```
EntitySchemaQueryColumnCollection GetSummaryColumns()
```

```
EntitySchemaQueryColumnCollection GetSummaryColumns(IEnumerable<string> columnNames)
```

Возвращает коллекцию выражений колонок запроса, для которых вычисляются итоговые значения.

```
Entity GetSummaryEntity(UserConnection userConnection, EntitySchemaQueryColumnCollection summary)
```

```
Entity GetSummaryEntity(UserConnection userConnection)
```

```
Entity GetSummaryEntity(UserConnection userConnection, IEnumerable<string> columnNames)
```

```
Entity GetSummaryEntity(UserConnection userConnection, params string[] columnNames)
```

Возвращает экземпляр `Entity` для результата, возвращаемого запросом на выборку итоговых значений.

Параметры

userConnection	Пользовательское подключение.
summaryColumns	Коллекция колонок запроса, для которых выбираются итоговые значения.
columnNames	Коллекция имен колонок.

```
Select GetSummarySelectQuery(UserConnection userConnection, EntitySchemaQueryColumnCollection summaryColumns)
Select GetSummarySelectQuery(UserConnection userConnection)
Select GetSummarySelectQuery(UserConnection userConnection, IEnumerable<string> columnNames)
Select GetSummarySelectQuery(UserConnection userConnection, params string[] columnNames)
```

Строит запрос на выборку итоговых значений для заданной коллекции колонок текущего экземпляра `EntitySchemaQuery`.

Параметры

userConnection	Пользовательское подключение.
summaryColumns	Коллекция колонок запроса, для которых выбираются итоговые значения.
columnNames	Коллекция имен колонок.

```
T GetTypedColumnValue(Entity entity, string columnName)
```

Возвращает типизированное значение колонки с именем `columnName` из переданного экземпляра `Entity`.

```
void LoadDataTableData(UserConnection userConnection, DataTable dataTable)
```

```
void LoadDataTableData(UserConnection userConnection, DataTable dataTable, EntitySchemaQueryOptions options)
```

Загружает результат выполнения текущего запроса к схеме объекта в объект `System.Data.DataTable`, используя пользовательское подключение `UserConnection` и заданные дополнительные настройки запроса `EntitySchemaQueryOptions`.

```
void RemoveColumn(string columnName)
```

Удаляет колонку с именем `columnName` из коллекции колонок текущего запроса.

```
void ResetSchema()
```

Очищает схему текущего экземпляра `EntitySchemaQuery`.

```
void ResetSelectQuery()
```

Очищает запрос на выборку для текущего запроса к схеме объекта.

```
void SetLocalizationCultureId(System.Guid cultureId)
```

Устанавливает идентификатор локальной культуры.

Класс Entity



Сложный

Пространство имен `Terrasoft.Core.Entities`.

Класс `Terrasoft.Core.Entities.Entity` предназначен для доступа к объекту, который представляет собой запись в таблице базы данных.

На заметку. Полный перечень методов и свойств класса `Entity`, его родительских классов, а также реализуемых им интерфейсов, можно найти в [Библиотеке .NET классов](#).

Конструкторы

```
Entity(UserConnection userConnection)
```

Создает новый экземпляр класса `Entity` для заданного пользовательского подключения `UserConnection`.

```
Entity(UserConnection userConnection, Guid schemaUUID)
```

Создает новый экземпляр класса `Entity` для заданного пользовательского подключения `UserConnection` и схемы заданной идентификатором `schemaUUID`.

```
Entity(Entity source)
```

Создает экземпляр класса, являющийся клоном экземпляра, переданного в качестве аргумента.

Свойства

```
ChangeType EntityChangeType
```

Тип изменения состояния объекта (добавлен, изменен, удален, без изменений).

`EntitySchemaManager EntitySchemaManager`

Экземпляр менеджера схемы объекта.

`EntitySchemaManagerName string`

Имя менеджера схемы объекта.

`HasColumnValues bool`

Определяет, имеет ли объект хотя бы одну колонку.

`HierarchyColumnName Guid`

Значение колонки связи с родительской записью для иерархических объектов.

`InstanceId Guid`

Идентификатор экземпляра объекта.

`IsDeletedFromDB bool`

Определяет, удален ли объект из базы данных.

`IsInColumnValueChanged bool`

Определяет, выполняется ли обработка события `ColumnValueChanged`.

`IsInColumnValueChanging bool`

Определяет, выполняется ли обработка события `ColumnValueChanging`.

`IsInDefColumnValuesSet bool`

Определяет, выполняется ли обработка события `DefColumnValuesSet`.

`IsInDeleted bool`

Определяет, выполняется ли обработка события `Deleted`.

`IsInDeleting bool`

Определяет, выполняется ли обработка события `Deleting`.

```
IsInInserted bool
```

Определяет, выполняется ли обработка события `Inserted`.

```
IsInInserting bool
```

Определяет, выполняется ли обработка события `Inserting`.

```
IsInLoaded bool
```

Определяет, выполняется ли обработка события `Loaded`.

```
IsInLoading bool
```

Определяет, выполняется ли обработка события `Loading`.

```
IsInSaved bool
```

Определяет, выполняется ли обработка события `Saved`.

```
IsInSaveError bool
```

Определяет, выполняется ли обработка события `SaveError`.

```
IsInSaving bool
```

Определяет, выполняется ли обработка события `Saving`.

```
IsInUpdated bool
```

Определяет, выполняется ли обработка события `Updated`.

```
IsInUpdating bool
```

Определяет, выполняется ли обработка события `Updating`.

```
IsInValidating bool
```

Определяет, выполняется ли обработка события `Validating`.

```
IsSchemaInitialized bool
```

Определяет, является ли схема объекта проинициализированной.

LicOperationPrefix string

Префикс лицензируемой операции.

LoadState EntityLoadState

Состояние загрузки объекта.

PrimaryColumnName Guid

Идентификатор первичной колонки.

PrimaryDisplayColumnName string

Значение для отображения первичной колонки.

Process Process

Встроенный процесс объекта.

Schema EntitySchema

Экземпляр схемы объекта.

SchemaName string

Имя схемы объекта.

StoringState StoringObjectState

Состояние объекта (изменен, добавлен, удален, без изменений).

UseAdminRights bool

Определяет, будут ли учитываться права при вставке, обновлении, удалении и получении данных.

UseDefRights bool

Определяет, использовать ли права по умолчанию на объект.

UseLazyLoad bool

Определяет, использовать ли ленивую первоначальную загрузку данных объекта.

UserConnection UserConnection

Пользовательское подключение.

ValidationMessages EntityValidationMessageCollection

Коллекция сообщений, выводимых при проверке объекта.

ValueListSchemaManager ValueListSchemaManager

Экземпляр менеджера перечислений объекта.

ValueListSchemaManagerName string

Имя менеджера перечислений объекта.

Методы

void AddDefRights()**void AddDefRights(Guid primaryColumnName)****void AddDefRights(IEnumerable<Guid> primaryColumnValues)**

Для данного объекта устанавливает права по умолчанию.

Параметры

primaryColumnName	Идентификатор значения права доступа.
primaryColumnValues	Массив идентификаторов значений прав доступа.

virtual object Clone()

Создает клон текущего экземпляра **Entity**.

Insert CreateInsert(bool skipLookupColumnValues)

Создает запрос на добавление данных в базу.

Параметры

skipLookupColumnValues	Признак добавления данных с учетом справочных колонок. По умолчанию установлено значение <code>false</code> .
------------------------	---

```
Update CreateUpdate(bool skipLookupColumnValues)
```

Создает запрос на обновление данных в базе.

Параметры

skipLookupColumnValues	Параметр, определяющий необходимость добавления в базу
------------------------	--

данных колонок типа справочник. Если параметр равен `true`, то колонки типа справочник не будут добавлены в базу.

Значение по умолчанию — `false`.

```
virtual bool Delete()
```

```
virtual bool Delete(object keyValue)
```

Удаляет из базы данных запись объекта.

Параметры

keyValue	Значение ключевого поля.
----------	--------------------------

```
bool DeleteWithCancelProcess()
```

Удаляет из базы данных запись объекта и отменяет запущенный процесс.

```
static Entity DeserializeFromJson(UserConnection userConnection, string jsonValue)
```

Создает объект типа `Entity`, используя пользовательское подключение `userConnection`, и заполняет значения его полей из указанной строки формата JSON `jsonValue`.

Параметры

jsonValue	Строка формата JSON.
-----------	----------------------

| userConnection | Пользовательское подключение. |

```
bool ExistInDB(EntitySchemaColumn conditionColumn, object conditionValue)
```

```
bool ExistInDB(string conditionColumnName, object conditionValue)
```

```
bool ExistInDB(object keyValue)
```

```
bool ExistInDB(Dictionary<string,object> conditions)
```

Определяет, существует ли в базе данных запись, отвечающая заданному условию запроса `conditionValue` к данной колонке схемы объекта `conditionColumn` либо с заданным первичным ключом `keyValue`.

Параметры

<code>conditionColumn</code>	Колонка, для которой задается условие выборки.
<code>conditionColumnName</code>	Название колонки, для которой задается условие выборки.
<code>conditionValue</code>	Значение колонки условия для выбираемых данных.
<code>conditions</code>	Набор условий фильтрации выборки записей объекта.
<code>keyValue</code>	Значение ключевого поля.

```
bool FetchFromDB(EntitySchemaColumn conditionColumn, object conditionValue, bool useDisplayValues)
bool FetchFromDB(string conditionColumnName, object conditionValue, bool useDisplayValues)
bool FetchFromDB(object keyValue, bool useDisplayValues)
bool FetchFromDB(Dictionary<string,object> conditions, bool useDisplayValues)
bool FetchFromDB(EntitySchemaColumn conditionColumn, object conditionValue, IEnumerable<EntitySchemaColumn> columnsToFetch, bool useDisplayValues)
bool FetchFromDB(string conditionColumnName, object conditionValue, IEnumerable<string> columnNamesToFetch, bool useDisplayValues)
```

По заданному условию загружает объект из базы данных.

Параметры

<code>conditionColumn</code>	Колонка, для которой задается условие выборки.
<code>conditionColumnName</code>	Название колонки, для которой задается условие выборки.
<code>conditionValue</code>	Значение колонки условия для выбираемых данных.
<code>columnsToFetch</code>	Список колонок, которые будут выбраны.
<code>columnNamesToFetch</code>	Список названий колонок, которые будут выбраны.
<code>conditions</code>	Набор условий фильтрации выборки записей объекта.
<code>keyValue</code>	Значение ключевого поля.
<code>useDisplayValues</code>	Признак получения в запросе первичных отображаемых значений. Если параметр равен <code>true</code> , в запросе будут возвращены первичные отображаемые значения.

```
bool FetchPrimaryColumnFromDB(object keyValue)
```

По заданному условию `keyValue` загружает из базы данных объект с первичной колонкой.

Параметры

<code>keyValue</code>	Значение ключевого поля.
-----------------------	--------------------------

```
bool FetchPrimaryInfoFromDB(EntitySchemaColumn conditionColumn, object conditionValue)
bool FetchPrimaryInfoFromDB(string conditionColumnName, object conditionValue)
```

По заданному условию загружает из базы данных объект с первичными колонками, включая колонку, первичную для отображения.

Параметры

<code>conditionColumn</code>	Колонка, для которой задается условие выборки.
<code>conditionColumnName</code>	Название колонки, для которой задается условие выборки.
<code>conditionValue</code>	Значение колонки условия для выбираемых данных.

```
byte[] GetBytesValue(string valueName)
```

Возвращает значение заданной колонки объекта в виде массива байт.

Параметры

<code>valueName</code>	Имя колонки объекта.
------------------------	----------------------

```
IEnumerable<EntityColumnValue> GetChangedColumnValues()
```

Возвращает коллекцию имен колонок объекта, которые были изменены.

```
string GetColumnDisplayValue(EntitySchemaColumn column)
```

Возвращает значение для отображения свойства объекта, соответствующее заданной колонке схемы объекта.

Параметры

<code>column</code>	Определенная колонка схемы объекта.
---------------------	-------------------------------------

```
object GetColumnOldValue(string valueName)
object GetColumnOldValue(EntitySchemaColumn column)
```

Возвращает предыдущее значение заданного свойства объекта.

Параметры

column	Определенная колонка схемы объекта.
valueName	Имя колонки объекта.

```
virtual object GetColumnValue(string valueName)
virtual object GetColumnValue(EntitySchemaColumn column)
```

Возвращает значение колонки объекта с заданным именем, соответствующее переданной колонке схемы объекта.

Параметры

column	Определенная колонка схемы объекта.
valueName	Имя колонки объекта.

```
IEnumerable<string> GetColumnValueNames()
```

Возвращает коллекцию имен колонок объекта.

```
virtual bool GetIsColumnValueLoaded(string valueName)
bool GetIsColumnValueLoaded(EntitySchemaColumn column)
```

Возвращает признак, определяющий, загружено ли заданное свойство объекта.

Параметры

column	Определенная колонка схемы объекта.
valueName	Имя колонки объекта.

```
virtual MemoryStream GetStreamValue(string valueName)
```

Возвращает преобразованное в экземпляр типа `System.IO.MemoryStream` значение переданной колонки схемы объекта.

Параметры

valueName	Имя колонки объекта.
-----------	----------------------

```
 TResult GetTypedColumnValue<TResult>(EntitySchemaColumn column)
```

Возвращает типизированное значение свойства объекта, соответствующее заданной колонке схемы объекта.

Параметры

column	Определенная колонка схемы объекта.
--------	-------------------------------------

```
 TResult GetTypedOldColumnValue<TResult>(string valueName)
```

```
 TResult GetTypedOldColumnValue<TResult>(EntitySchemaColumn column)
```

Возвращает типизированное предыдущее значение свойства объекта, соответствующее заданной колонке схемы объекта.

Параметры

column	Определенная колонка схемы объекта.
valueName	Имя колонки объекта.

```
virtual bool InsertToDB(bool skipLookupColumnValues, bool validateRequired)
```

Добавляет запись текущего объекта в базу данных.

Параметры

skipLookupColumnValues	Параметр, определяющий необходимость добавления в базу данных колонок типа справочник. Если параметр равен <code>true</code> , то колонки типа справочник не будут добавлены в базу. Значение по умолчанию — <code>false</code> .
validateRequired	Параметр, определяющий необходимость проверки заполнения обязательных значений. Значение по умолчанию — <code>true</code> .

```
bool IsColumnValueLoaded(string valueName)
```

```
bool IsColumnValueLoaded(EntitySchemaColumn column)
```

Определяет, загружено ли значение свойства объекта с заданным именем.

Параметры

column	Определенная колонка схемы объекта.
valueName	Имя колонки объекта.

```

virtual bool Load(DataRow dataRow)
virtual bool Load(DataRow dataRow, Dictionary<string,string> columnMap)
virtual bool Load(IDataReader dataReader)
virtual bool Load(IDataReader dataReader, IDictionary<string,string> columnMap)
virtual bool Load(object dataSource)
virtual bool Load(object dataSource, IDictionary<string,string> columnMap)

```

Заполняет объект переданными данными.

Параметры

columnMap	Свойства объекта, заполняемые данными.
dataRow	Экземпляр <code>System.Data.DataRow</code> , из которого загружаются данные в объект.
dataReader	Экземпляр <code>System.Data.IDataReader</code> , из которого загружаются данные.
dataSource	Экземпляр <code>System.Object</code> , из которого загружаются данные.

```

void LoadColumnValue(string columnName, IDataReader dataReader, int fieldIndex, int binaryF
void LoadColumnValue(string columnName, IDataReader dataReader, int fieldIndex)
void LoadColumnValue(string columnName, object value)
void LoadColumnValue(EntitySchemaColumn column, object value)

```

Для свойства с заданным именем загружает его значение из переданного экземпляра.

Параметры

binaryPackageSize	Размер загружаемого значения.
column	Колонка схемы объекта.
columnName	Имя свойства объекта.
dataReader	Экземпляр <code>System.Data.IDataReader</code> , из которого загружается значение свойства.
fieldIndex	Индекс загружаемого из <code>System.Data.IDataReader</code> поля.
value	Загружаемое значение свойства.

```
static Entity Read(UserConnection userConnection, DataReader dataReader)
```

Возвращает значение текущего свойства типа `Entity` из потока ввода.

Параметры

dataReader	Экземпляр <code>System.Data.IDataReader</code> , из которого загружается значение свойства.
userConnection	Пользовательское подключение.

```
void ReadData(DataReader reader)
void ReadData(DataReader reader, EntitySchema schema)
```

Считывает данные из схемы объекта в заданный объект типа `System.Data.IDataReader`.

Параметры

reader	Экземпляр <code>System.Data.IDataReader</code> , в который загружаются данные схемы объекта.
schema	Схема объекта.

```
void ResetColumnValues()
```

Для всех свойств объекта отменяет изменения.

```
void ResetOldColumnValues()
```

Для всех свойств объекта отменяет изменения, устанавливая предыдущее значение.

```
bool Save(bool validateRequired)
```

Сохраняет объект в базе данных.

Параметры

validateRequired	Определяет необходимость проверки заполнения обязательных значений. Значение по умолчанию — <code>true</code> .
------------------	---

```
static string SerializeToJson(Entity entity)
```

Преобразует объект `entity` в строку формата `JSON`.

Параметры

entity	Экземпляр <code>Entity</code> .
--------	---------------------------------

```
virtual void SetBytesValue(string valueName, byte[] streamBytes)
```

Устанавливает для заданного свойства объекта переданное значение типа `System.Byte`.

Параметры

streamBytes	Значение типа <code>System.Byte</code> , которое устанавливается в заданную колонку объекта.
valueName	Имя колонки объекта.

```
bool SetColumnBothValues(EntitySchemaColumn column, object value, string displayValue)
```

```
bool SetColumnBothValues(string columnName, object value, string displayColumnName, st
```

Устанавливает свойству объекта, соответствующему заданной колонке схемы, переданные значение `value` и значение для отображения `displayValue`.

Параметры

column	Колонка схемы объекта.
displayValue	Загружаемое значение для отображения.
displayColumnName	Имя колонки, содержащей значение для отображения.
value	Загружаемое значение колонки.

```
bool SetColumnValue(string valueName, object value)
bool SetColumnValue(EntitySchemaColumn column, object value)
```

Устанавливает заданной колонке схемы переданное значение `value`.

Параметры

column	Колонка схемы объекта.
value	Загружаемое значение колонки.
valueName	Имя колонки объекта.

```
void SeddefColumnValue(string columnValueName, object defValue)
void SeddefColumnValue(string columnValueName)
```

Устанавливает значение по умолчанию свойству с заданным именем.

Параметры

columnValueName	Имя колонки объекта.
defValue	Значение по умолчанию.

```
void SeddefColumnValues()
```

Для всех свойств объекта устанавливает значения по умолчанию.

```
bool SetStreamValue(string valueName, Stream value)
```

Устанавливает для заданного свойства объекта переданное значение типа `System.IO.Stream`.

Параметры

<code>value</code>	Загружаемое значение колонки.
<code>valueName</code>	Имя колонки объекта.

```
virtual bool UpdateInDB(bool validateRequired)
```

Обновляет запись объекта в базе данных.

Параметры

<code>validateRequired</code>	Определяет необходимость проверки заполнения обязательных значений. Значение по умолчанию — <code>true</code> .
-------------------------------	---

```
bool Validate()
```

Проверяет заполнение обязательных полей.

```
static void Write(DataWriter dataWriter, Entity entity, string propertyName)
static void Write(DataWriter dataWriter, Entity entity, string propertyName, bool couldConvertFc
```

Осуществляет запись значения типа `Entity` в поток вывода с заданными именем.

Параметры

<code>couldConvertForXml</code>	Разрешить преобразование для xml-сериализации.
<code>dataWriter</code>	Экземпляр класса <code>Terrasoft.Common.DataWriter</code> , предоставляющий методы последовательной записи значений в поток вывода.
<code>entity</code>	Значение для записи типа <code>Entity</code> .
<code>propertyName</code>	Имя объекта.

```
void Write(DataWriter dataWriter, string propertyName)
```

Осуществляет запись данных в поток вывода с заданным именем.

Параметры

<code>dataWriter</code>	Экземпляр класса <code>Terrasoft.Common.DataWriter</code> , предоставляющий методы последовательной записи значений в поток вывода.
<code>propertyName</code>	Имя свойства.

```
void WriteData(DataWriter writer)
void WriteData(DataWriter writer, EntitySchema schema)
```

Осуществляет запись в поток вывода для указанной либо текущей схемы объекта.

Параметры

<code>schema</code>	Схема объекта.
<code>writer</code>	Экземпляр класса <code>Terrasoft.Common.DataWriter</code> , предоставляющий методы последовательной записи значений в поток вывода.

События

```
event EventHandler<EntityColumnEventArgs> ColumnValueChanged
```

Обработчик события, возникающего после изменения значения колонки объекта.

Обработчик события получает аргумент типа `EntityColumnEventArgs`.

Свойства `EntityColumnEventArgs` предоставляющие сведения, относящиеся к событию:

- `ColumnName` ;
- `DisplayColumnName` .

```
event EventHandler<EntityColumnEventArgs> ColumnValueChanging
```

Обработчик события, возникающего перед изменением значения колонки объекта.

Обработчик события получает аргумент типа `EntityColumnEventArgs`.

Свойства `EntityColumnEventArgs` предоставляющие сведения, относящиеся к событию:

- `ColumnStreamValue` .
- `ColumnValue` .
- `ColumnName` .
- `DisplayColumnValue` .
- `DisplayColumnName` .

```
event EventHandler<EventArgs> DefColumnValuesSet
```

Обработчик события, возникающего после установки значений по умолчанию полей объекта.

```
event EventHandler<EntityAfterEventArgs> Deleted
```

Обработчик события, возникающего после удаления объекта.

Обработчик события получает аргумент типа `EntityAfterEventArgs`.

Свойства `EntityAfterEventArgs` предоставляющие сведения, относящиеся к событию:

- `ModifiedColumnValues`.
- `PrimaryColumnName`.

```
event EventHandler<EntityBeforeEventArgs> Deleting
```

Обработчик события, возникающего перед удалением объекта.

Обработчик события получает аргумент типа `EntityBeforeEventArgs`.

Свойства `EntityBeforeEventArgs` предоставляющие сведения, относящиеся к событию:

- `AdditionalCondition`.
- `IsCanceled`.
- `KeyValue`.

```
event EventHandler<EntityAfterEventArgs> Inserted
```

Обработчик события, возникающего после вставки объекта.

Обработчик события получает аргумент типа `EntityAfterEventArgs`.

Свойства `EntityAfterEventArgs` предоставляющие сведения, относящиеся к событию:

- `ModifiedColumnValues`.
- `PrimaryColumnName`.

```
event EventHandler<EntityBeforeEventArgs> Inserting
```

Обработчик события, возникающего перед вставкой объекта.

Обработчик события получает аргумент типа `EntityBeforeEventArgs`.

Свойства `EntityBeforeEventArgs` предоставляющие сведения, относящиеся к событию:

- `AdditionalCondition`.
- `IsCanceled`.
- `KeyValue`.

```
event EventHandler<EntityAfterLoadEventArgs> Loaded
```

Обработчик события, возникающего после загрузки объекта.

Обработчик события получает аргумент типа `EntityAfterLoadEventArgs`.

Свойства `EntityAfterLoadEventArgs` предоставляющие сведения, относящиеся к событию:

- `ColumnMap`.
 - `DataSource`.
-

```
event EventHandler<EntityBeforeLoadEventArgs> Loading
```

Обработчик события, возникающего перед загрузкой объекта.

Обработчик события получает аргумент типа `EntityBeforeLoadEventArgs`.

Свойства `EntityBeforeLoadEventArgs` предоставляющие сведения, относящиеся к событию:

- `ColumnMap`.
 - `DataSource`.
 - `IsCanceled`.
-

```
event EventHandler<EntityAfterEventArgs> Saved
```

Обработчик события, возникающего после сохранения объекта.

Обработчик события получает аргумент типа `EntityAfterEventArgs`.

Свойства `EntityAfterEventArgs` предоставляющие сведения, относящиеся к событию:

- `ModifiedColumnValues`.
 - `PrimaryColumnName`.
-

```
event EventHandler<EntitySaveErrorEventArgs> SaveError
```

Обработчик события, возникающего при ошибке сохранения объекта.

Обработчик события получает аргумент типа `EntitySaveErrorEventArgs`.

Свойства `EntitySaveErrorEventArgs` предоставляющие сведения, относящиеся к событию:

- `Exception`.
 - `IsHandled`.
-

```
event EventHandler<EntityBeforeEventArgs> Saving
```

Обработчик события, возникающего перед сохранением объекта.

Обработчик события получает аргумент типа `EntityBeforeEventArgs`.

Свойства `EntityBeforeEventArgs` предоставляющие сведения, относящиеся к событию:

- `AdditionalCondition`.
- `IsCanceled`.
- `KeyValue`.

`event EventHandler<EntityAfterEventArgs> Updated`

Обработчик события, возникающего после обновления объекта.

Обработчик события получает аргумент типа `EntityAfterEventArgs`.

Свойства `EntityAfterEventArgs` предоставляющие сведения, относящиеся к событию:

- `ModifiedColumnValues`.
- `PrimaryColumnName`.

`event EventHandler<EntityBeforeEventArgs> Updating`

Обработчик события, возникающего перед обновлением объекта.

Обработчик события получает аргумент типа `EntityBeforeEventArgs`.

Свойства `EntityBeforeEventArgs` предоставляющие сведения, относящиеся к событию:

- `AdditionalCondition`.
- `IsCanceled`.
- `KeyValue`.

`event EventHandler<EntityValidationEventArgs> Validating`

Обработчик события, возникающего при проверке объекта.

Обработчик события получает аргумент типа `EntityValidationEventArgs`.

Свойства `EntityValidationEventArgs` предоставляющие сведения, относящиеся к событию:

- `Messages`.

Класс EntityMapper c#



Сложный

Класс `Terrasoft.Configuration.EntityMapper` — это утилитный класс конфигурации, который находится в пакете [*FinAppLending*] продукта Lending. `EntityMapper` позволяет сопоставлять данные одной сущности (`Entity`) с другой по правилам, определенным в конфигурационном файле. Использование подхода сопоставления данных разных сущностей позволяет избежать появления однообразного кода.

В продукте Lending существует два объекта, содержащих одинаковые колонки. Это объекты [*Физ. лицо*

][Contact]) и [Анкета] ([AppForm]). Также существует несколько деталей, относящихся к объекту [Физ. лицо] ([Contact]) и имеющих похожие детали, относящиеся к [Анкета] ([AppForm]). Очевидно, что при заполнении анкеты должна быть возможность по колонке [Id] объекта [Физ. лицо] ([Contact]) получить список всех его колонок и значений, а также список нужных деталей с их колонками и значениями, и сопоставить эти данные с данными, связанными с анкетой. После этого можно автоматически заполнить поля анкеты сопоставленными данными. Таким образом можно существенно уменьшить затраты на ручной ввод одинаковых данных.

Идея сопоставления данных разных сущностей реализована в следующих классах:

- EntityMapper — реализует логику сопоставления.
- EntityResult — определяет в каком виде вернется сопоставленная сущность.
- MapConfig — представляет набор правил для сопоставления.
- DetailMapConfig — используется для установки списка правил сопоставления деталей и связанных с ними сущностей.
- RelationEntityMapConfig — содержит правила для сопоставления связанных сущностей.
- EntityFilterMap — представляет из себя фильтр для запроса в базу данных.

Класс EntityMapper c#

Пространство имен `Terrasoft.Configuration`.

Класс реализует логику сопоставления.

Методы

`virtual EntityResult GetMappedEntity(Guid recId, MapConfig config)`

Возвращает сопоставленные данные для двух объектов `Entity`.

Параметры

<code>recId</code>	GUID записи в базе данных.
<code>config</code>	Экземпляр класса <code>MapConfig</code> , представляющий из себя набор правил сопоставления.

`virtual Dictionary<string, object> GetColumnsValues(Guid recordId, MapConfig config, Dictionary<`

Получает из базы данных главную сущность и сопоставляет ее колонки и значения по правилам, указанным в объекте `config`.

Параметры

recordId	GUID записи в базе данных.
config	Экземпляр класса <code>MapConfig</code> , представляющий из себя набор правил сопоставления.
result	Словарь колонок и их значений уже сопоставленной сущности.

`virtual Dictionary<string, object> GetRelationEntityColumnsValues(List<RelationEntityMapConfig>`
Получает из базы данных связанные сущности и сопоставляет их с основными сущностями.

Параметры

relations	Список правил для получения связанных записей.
dictionaryToMerge	Словарь с колонками и их значениями.
columnName	Название родительской колонки.
entitylookup	Объект, содержащий название и Id записи в базе.

Класс EntityResult [c#](#)

Пространство имен `Terrasoft.Configuration`.

Класс определяет в каком виде вернется сопоставленная сущность.

Свойства

`Columns Dictionary<string, object>`

Словарь с названиями колонок основной сущности и их значениями.

`Details Dictionary<string, List<Dictionary<string, object>>>`

Словарь названий деталей со списком их колонок и значений.

Класс MapConfig [c#](#)

Пространство имен `Terrasoft.Configuration`.

Класс представляет набор правил для сопоставления.

Свойства

```
SourceEntityName string
```

Название сущности в базе данных.

```
Columns Dictionary<string, object>
```

Словарь с названиями колонок одной сущности и сопоставляемыми колонками другой сущности.

```
DetailsConfig List<DetailMapConfig>
```

Список конфигурационных объектов с правилами для деталей.

```
CleanDetails List<string>
```

Список названий деталей для очистки их значений.

```
RelationEntities List<RelationEntityMapConfig>
```

Список конфигурационных объектов с правилами сопоставления связанных записей с главной сущностью.

Класс DetailMapConfig [C#](#)

Пространство имен `Terrasoft.Configuration`.

Класс используется для установки списка правил сопоставления деталей и связанных с ними сущностей.

Свойства

```
DetailName string
```

Название детали (для обеспечения уникальности экземпляра детали).

```
SourceEntityName string
```

Название сущности в базе данных.

```
Columns Dictionary<string, object>
```

Словарь с названиями колонок одной сущности и сопоставляемыми колонками другой сущности.

```
Filters List<EntityFilterMap>
```

Список конфигурационных объектов с правилами фильтрации для более точных выборок из базы данных.

`RelationEntities List<RelationEntityMapConfig>`

Список конфигурационных объектов с правилами сопоставления связанных записей с главной сущностью.

Класс RelationEntityMapConfig [c#](#)

Пространство имен `Terrasoft.Configuration`.

Класс содержит правила для сопоставления связанных сущностей.

Свойства

`ParentColumnName string`

Название родительской колонки, при нахождении которой будет срабатывать логика получения и сопоставления данных по сущности.

`SourceEntityName string`

Название сущности в базе данных.

`Columns Dictionary<string, object>`

Словарь с названиями колонок одной сущности и сопоставляемыми колонками другой сущности.

`Filters List<EntityFilterMap>`

Список конфигурационных объектов с правилами фильтрации для более точных выборок из базы данных.

`RelationEntities List<RelationEntityMapConfig>`

Список конфигурационных объектов с правилами сопоставления связанных записей с главной сущностью.

Класс EntityFilterMap [c#](#)

Пространство имен `Terrasoft.Configuration`.

Класс представляет из себя фильтр для запроса в базу данных.

Свойства

`ColumnName string`

Название колонки, при нахождении которой будет срабатывать логика фильтрации.

`Value object`

Значение, с которым необходимо сравнение.

Класс EntitySchemaQueryFunction c#



Сложный

Класс `Terrasoft.Core.Entities.EntitySchemaQueryFunction` реализует функцию выражения.

Идея функции выражения реализована в следующих классах:

- `EntitySchemaQueryFunction` — базовый класс функции выражения запроса к схеме объекта.
- `EntitySchemaAggregationQueryFunction` — реализует агрегирующую функцию выражения.
- `EntitySchemaIsNullQueryFunction` — заменяет значения `null` замещающим выражением.
- `EntitySchemaCoalesceQueryFunction` — возвращает первое выражение из списка аргументов, не равное `null`.
- `EntitySchemaCaseNotNullQueryFunctionWhenItem` — класс, описывающий выражение условия sql-оператора `CASE`.
- `EntitySchemaCaseNotNullQueryFunctionWhenItems` — коллекция выражений условий sql-оператора `CASE`.
- `EntitySchemaStartOfCurrentHourQueryFunction(EntitySchemaQuery parentQuery, EntitySchemaQueryExpression expression, int offset = 0) : this(parentQuery, offset)`
- `EntitySchemaCaseNotNullQueryFunction` — возвращает одно из множества возможных значений в зависимости от указанных условий.
- `EntitySchemaSystemValueQueryFunction` — возвращает выражение системного значения.
- `EntitySchemaCurrentDateTimeQueryFunction` — реализует функцию выражения текущей даты и времени.
- `EntitySchemaBaseCurrentDateQueryFunction` — базовый класс функции выражения для базовой даты.
- `EntitySchemaCurrentDateQueryFunction` — реализует функцию выражения текущей даты.
- `EntitySchemaDateToCurrentYearQueryFunction` — реализует функцию выражения даты начала текущей недели.
- `EntitySchemaStartOfCurrentWeekQueryFunction` — реализует функцию, которая конвертирует выражение даты в такую же дату текущего года.
- `EntitySchemaStartOfCurrentMonthQueryFunction` — реализует функцию выражения даты начала текущего месяца.
- `EntitySchemaStartOfCurrentQuarterQueryFunction` — реализует функцию выражения даты начала

текущего квартала.

- `EntitySchemaStartOfCurrentHalfYearQueryFunction` — реализует функцию выражения даты начала текущего полугодия.
- `EntitySchemaStartOfCurrentYearQueryFunction` — реализует функцию выражения даты начала текущего года.
- `EntitySchemaBaseCurrentDateTimeQueryFunction` — базовый класс функции выражения базовых даты и времени.
- `EntitySchemaStartOfCurrentHourQueryFunction` — реализует функцию выражения начала текущего часа.
- `EntitySchemaCurrentTimeQueryFunction` — реализует функцию выражения текущего времени.
- `EntitySchemaCurrentUserQueryFunction` — реализует функцию выражения текущего пользователя.
- `EntitySchemaCurrentUserContactQueryFunction` — реализует функцию контакта текущего пользователя.
- `EntitySchemaCurrentUserAccountQueryFunction` — реализует функцию выражения контрагента текущего пользователя.
- `EntitySchemaDatePartQueryFunction` — реализует функцию запроса для части даты.
- `EntitySchemaUpperQueryFunction` — преобразовывает символы выражения аргумента к верхнему регистру.
- `EntitySchemaCastQueryFunction` — приводит выражение аргумента к заданному типу данных.
- `EntitySchemaTrimQueryFunction` — удаляет начальные и конечные пробелы из выражения.
- `EntitySchemaLengthQueryFunction` — возвращает длину выражения.
- `EntitySchemaConcatQueryFunction` — формирует строку, которая является результатом объединения строковых значений аргументов функции.
- `EntitySchemaWindowQueryFunction` — реализует функцию SQL окна.

Класс EntitySchemaQueryFunction c#

Пространство имен `Terrasoft.Core.Entities`.

Базовый класс функции выражения запроса к схеме объекта.

На заметку. Полный перечень методов класса `EntitySchemaQueryFunction`, его родительских классов, а также реализуемых им интерфейсов, можно найти в документации "[.NET библиотеки классов ядра платформы](#)".

Методы

```
abstract QueryColumnExpression CreateQueryColumnExpression(DBSecurityEngine dbSecurityEngine)
```

Возвращает выражение колонки запроса для текущей функции, сформированное с учетом заданных прав доступа.

Параметры

dbSecurityEngine	Объект <code>Terrasoft.Core.DB.DBSecurityEngine</code> , определяющий права доступа.
------------------	--

```
abstract DataValueType GetResultDataValueType(DataValueTypeManager valueTypeManager)
```

Возвращает тип данных возвращаемого функцией результата, используя переданный менеджер типов данных.

Параметры

valueTypeManager	Менеджер типов данных.
------------------	------------------------

```
abstract bool GetIsSupportsValueType(DataValueType valueType)
```

Определяет, имеет ли возвращаемый функцией результат указанный тип данных.

Параметры

valueType	Тип данных.
-----------	-------------

```
abstract string GetCaption()
```

Возвращает заголовок функции выражения.

```
virtual EntitySchemaQueryExpressionCollection GetArguments()
```

Возвращает коллекцию выражений аргументов функции.

```
void CheckIsSupportsValueType(DataValueType valueType)
```

Проверяет, имеет ли возвращаемый функцией результат указанный тип данных. В противном случае генерируется исключение.

Параметры

valueType	Тип данных.
-----------	-------------

Класс EntitySchemaAggregationQueryFunction C#

Пространство имен `Terrasoft.Core.Entities`.

Класс реализует агрегирующую функцию выражения.

На заметку. Полный перечень методов и свойств класса `EntitySchemaAggregationQueryFunction`, его родительских классов, а также реализуемых им интерфейсов, можно найти в документации "[.NET библиотеки классов ядра платформы](#)".

Конструкторы

`EntitySchemaAggregationQueryFunction(EntitySchemaQuery parentQuery)`

Инициализирует экземпляр `EntitySchemaAggregationQueryFunction` заданного типа агрегирующей функции для заданного запроса к схеме объекта.

Параметры

<code>parentQuery</code>	Запрос к схеме объекта, которому принадлежит функция.
--------------------------	---

`EntitySchemaAggregationQueryFunction(AggregationTypeStrict aggregationType, EntitySchemaQuery parentQuery)`

Инициализирует экземпляр `EntitySchemaAggregationQueryFunction` заданного типа агрегирующей функции для заданного запроса к схеме объекта.

Параметры

<code>aggregationType</code>	Тип агрегирующей функции.
<code>parentQuery</code>	Запрос к схеме объекта, которому принадлежит функция.

`EntitySchemaAggregationQueryFunction(AggregationTypeStrict aggregationType, EntitySchemaQueryExpression expression, EntitySchemaQuery parentQuery)`

Инициализирует новый экземпляр `EntitySchemaAggregationQueryFunction` для заданных типа агрегирующей функции, выражения и запроса к схеме объекта.

Параметры

<code>aggregationType</code>	Тип агрегирующей функции.
<code>expression</code>	Выражение запроса.
<code>parentQuery</code>	Запрос к схеме объекта, которому принадлежит функция.

`EntitySchemaAggregationQueryFunction(EntitySchemaAggregationQueryFunction source)`

Инициализирует новый экземпляр `EntitySchemaAggregationQueryFunction`, являющийся клоном переданного экземпляра агрегирующей функции выражения.

Параметры

<code>source</code>	Экземпляр агрегирующей функции выражения, клон которой создается.
---------------------	---

Свойства

`QueryAlias string`

Псевдоним функции в sql-запросе.

`AggregationType AggregationTypeStrict`

Тип агрегирующей функции.

`AggregationEvalType AggregationEvalType`

Область применения агрегирующей функции.

`Expression EntitySchemaQueryExpression`

Выражение аргумента агрегирующей функции.

Методы

`override void WriteMetaData(DataWriter writer)`

Выполняет сериализацию агрегирующей функции, используя заданный экземпляр `Terrasoft.Common.DataWriter`.

Параметры

<code>writer</code>	Экземпляр <code>Terrasoft.Common.DataWriter</code> , с помощью которого выполняется сериализация.
---------------------	---

`override QueryColumnExpression CreateQueryColumnExpression(DBSecurityEngine dbSecurityEngine)`

Возвращает выражение колонки запроса для агрегирующей функции, сформированное с учетом заданных прав доступа.

Параметры

<code>dbSecurityEngine</code>	Объект <code>Terrasoft.Core.DB.DBSecurityEngine</code> , определяющий права доступа.
-------------------------------	--

```
override EntitySchemaQueryExpressionCollection GetArguments()
```

Возвращает коллекцию выражений аргументов агрегирующей функции.

```
override DataValueType GetResultDataType(DataValueTypeManager dataTypeManager)
```

Возвращает тип данных возвращаемого агрегирующей функцией результата, используя заданный менеджер типов данных.

Параметры

<code>dataTypeManager</code>	Менеджер типов данных.
------------------------------	------------------------

```
override bool GetIsSupportedValueType(DataValueType dataType)
```

Определяет, имеет ли возвращаемый агрегирующей функцией результат указанный тип данных.

Параметры

<code>dataType</code>	Тип данных.
-----------------------	-------------

```
override string GetCaption()
```

Возвращает заголовок функции выражения.

```
override object Clone()
```

Создает клон текущего экземпляра `EntitySchemaAggregationQueryFunction`.

```
EntitySchemaAggregationQueryFunction All()
```

Устанавливает для текущей агрегирующей функции область применения [*Ко всем значениям*].

```
EntitySchemaAggregationQueryFunction Distinct()
```

Устанавливает для текущей агрегирующей функции область применения [*К уникальным значениям*].

Класс EntitySchemaIsNullOrEmptyQueryFunction C#

Пространство имен `Terrasoft.Core.Entities`.

Класс заменяет значения `null` замещающим выражением.

На заметку. Полный перечень методов и свойств класса `EntitySchemaIsNullOrEmptyQueryFunction`, его родительских классов, а также реализуемых им интерфейсов, можно найти в документации "[.NET библиотеки классов ядра платформы](#)".

Конструкторы

`EntitySchemaIsNullOrEmptyQueryFunction(EntitySchemaQuery parentQuery)`

Инициализирует экземпляр `EntitySchemaIsNullOrEmptyQueryFunction` для заданного запроса к схеме объекта.

Параметры

<code>parentQuery</code>	Запрос к схеме объекта, которому принадлежит функция.
--------------------------	---

`EntitySchemaIsNullOrEmptyQueryFunction(EntitySchemaQuery parentQuery, EntitySchemaQueryExpression check)`

Инициализирует новый экземпляр `EntitySchemaIsNullOrEmptyQueryFunction` для заданных запроса к схеме объекта, проверяемого выражения и замещающего выражения.

Параметры

<code>parentQuery</code>	Запрос к схеме объекта, которому принадлежит функция.
<code>checkExpression</code>	Выражение, которое проверяется на равенство <code>null</code> .
<code>replacementExpression</code>	Выражение, которое возвращается функцией, если <code>checkExpression</code> равно <code>null</code> .

`EntitySchemaIsNullOrEmptyQueryFunction(EntitySchemaIsNullOrEmptyQueryFunction source)`

Инициализирует новый экземпляр `EntitySchemaIsNullOrEmptyQueryFunction`, являющийся клоном переданной функции выражения.

Параметры

<code>source</code>	Экземпляр функции <code>EntitySchemaIsNullOrEmptyQueryFunction</code> , Клон которой создается.
---------------------	---

Свойства

`QueryAlias string`

Псевдоним функции в sql-запросе.

`CheckExpression EntitySchemaQueryExpression`

Выражение аргумента функции, которое проверяется на равенство значению `null`.

`ReplacementExpression EntitySchemaQueryExpression`

Выражение аргумента функции, которое возвращается, если проверяемое выражение равно `null`.

Методы

`override void WriteMetaData(DataWriter writer)`

Выполняет сериализацию функции выражения, используя переданный экземпляр `DataWriter`.

Параметры

<code>writer</code>	Экземпляр <code>DataWriter</code> , с помощью которого выполняется сериализация функции выражения.
---------------------	--

`override QueryColumnExpression CreateQueryColumnExpression(DBSecurityEngine dbSecurityEngine)`

Возвращает выражение колонки запроса для текущей функции, сформированное с учетом заданных прав доступа.

Параметры

<code>dbSecurityEngine</code>	Объект <code>Terrasoft.Core.DB.DBSecurityEngine</code> , определяющий права доступа.
-------------------------------	--

`override EntitySchemaQueryExpressionCollection GetArguments()`

Возвращает коллекцию выражений аргументов функции.

`override DataValueType GetResultDataValueType(DataValueTypeManager dataValueTypeManager)`

Возвращает тип данных возвращаемого функцией результата, используя переданный менеджер

типов данных.

Параметры

dataValueTypeManager	Менеджер типов данных.
----------------------	------------------------

Класс EntitySchemaCoalesceQueryFunction [c#](#)

Пространство имен `Terrasoft.Core.Entities`.

Класс возвращает первое выражение из списка аргументов, не равное `null`.

На заметку. Полный перечень методов и свойств класса `EntitySchemaCoalesceQueryFunction`, его родительских классов, а также реализуемых им интерфейсов, можно найти в документации "[.NET библиотеки классов ядра платформы](#)".

Конструкторы

`EntitySchemaCoalesceQueryFunction(EntitySchemaQuery parentQuery)`

Инициализирует новый экземпляр `EntitySchemaCoalesceQueryFunction` для заданного запроса к схеме объекта.

Параметры

aggregationType	Тип агрегирующей функции.
parentQuery	Запрос к схеме объекта, которому принадлежит функция.

`EntitySchemaCoalesceQueryFunction(EntitySchemaCoalesceQueryFunction source)`

Инициализирует новый экземпляр `EntitySchemaCoalesceQueryFunction`, являющийся клоном переданной функции.

Параметры

source	ФУНКЦИЯ <code>EntitySchemaCoalesceQueryFunction</code> , клон которой создается.
--------	--

Свойства

`QueryAlias string`

Псевдоним функции в sql-запросе.

`Expressions EntitySchemaQueryExpressionCollection`

Коллекция выражений аргументов функции.

`HasExpressions bool`

Признак, определяющий наличие хотя бы одного элемента в коллекции выражений аргументов функции.

Методы

`override bool GetIsSupportepataValueType(DataValueType dataType)`

Определяет, имеет ли возвращаемый функцией результат указанный тип данных.

Параметры

<code>dataType</code>	Тип данных.
-----------------------	-------------

Класс EntitySchemaCaseNotNullQueryFunctionWhenItem [C#](#)

Пространство имен `Terrasoft.Core.Entities`.

Класс, описывающий выражение условия sql-оператора `CASE`.

На заметку. Полный перечень методов класса `EntitySchemaCaseNotNullQueryFunctionWhenItem`, его родительских классов, а также реализуемых им интерфейсов, можно найти в документации "[.NET библиотеки классов ядра платформы](#)".

Конструкторы

`EntitySchemaCaseNotNullQueryFunctionWhenItem()`

Инициализирует новый экземпляр `EntitySchemaCaseNotNullQueryFunctionWhenItem`.

`EntitySchemaCaseNotNullQueryFunctionWhenItem(EntitySchemaQueryExpression whenExpression, EntityS`

Инициализирует экземпляр `EntitySchemaCaseNotNullQueryFunctionWhenItem` для заданных выражений предложений `WHEN` и `THEN`.

Параметры

whenExpression	Выражение предложения WHEN условия.
thenExpression	Выражение предложения THEN условия.

`EntitySchemaCaseNotNullQueryFunctionWhenItem(EntitySchemaCaseNotNullQueryFunctionWhenItem source)`
Инициализирует экземпляр `EntitySchemaCaseNotNullQueryFunctionWhenItem`, являющийся клоном переданной функции.

Параметры

source	ФУНКЦИЯ <code>EntitySchemaCaseNotNullQueryFunctionWhenItem</code> , КЛОН которой создается.
--------	---

Свойства

`WhenExpression EntitySchemaQueryExpression`

Выражение предложения WHEN .

`ThenExpression EntitySchemaQueryExpression`

Выражение предложения THEN .

Класс EntitySchemaCaseNotNullQueryFunctionWhenItems [C#](#)

Пространство имен `Terrasoft.Core.Entities`.

Класс реализует коллекцию выражений условий sql-оператора CASE .

На заметку. Полный перечень методов класса `EntitySchemaCaseNotNullQueryFunctionWhenItems`, его родительских классов, а также реализуемых им интерфейсов, можно найти в документации "[.NET библиотеки классов ядра платформы](#)".

Конструкторы

`EntitySchemaCaseNotNullQueryFunctionWhenItems()`

Инициализирует экземпляр `EntitySchemaCaseNotNullQueryFunctionWhenItems`.

`EntitySchemaCaseNotNullQueryFunctionWhenItems(EntitySchemaCaseNotNullQueryFunctionWhenItems source)`

Инициализирует новый экземпляр `EntitySchemaCaseNotNullQueryFunctionWhenItems`, являющийся клоном клоном переданной коллекции условий.

Параметры

<code>source</code>	Коллекция условий, клон которой создается.
---------------------	--

Класс EntitySchemaCaseNotNullQueryFunction [C#](#)

Пространство имен `Terrasoft.Core.Entities`.

Класс возвращает одно из множества возможных значений в зависимости от указанных условий.

На заметку. Полный перечень методов и свойств класса `EntitySchemaCaseNotNullQueryFunction`, его родительских классов, а также реализуемых им интерфейсов, можно найти в документации "[.NET библиотеки классов ядра платформы](#)".

Конструкторы

`CurrentDateTimeQueryFunction()`

Инициализирует новый экземпляр `CurrentDateTimeQueryFunction`.

`EntitySchemaCaseNotNullQueryFunction(EntitySchemaQuery parentQuery)`

Инициализирует новый экземпляр `EntitySchemaCaseNotNullQueryFunction` для заданного запроса к схеме объекта.

Параметры

<code>parentQuery</code>	Запрос к схеме объекта, которому принадлежит функция.
--------------------------	---

`EntitySchemaCaseNotNullQueryFunction(EntitySchemaCaseNotNullQueryFunction source)`

Инициализирует новый экземпляр `EntitySchemaCaseNotNullQueryFunction`, являющийся клоном переданной функции.

Параметры

<code>source</code>	ФУНКЦИЯ <code>EntitySchemaCaseNotNullQueryFunction</code> , клон которой создается.
---------------------	---

Свойства

`QueryAlias string`

Псевдоним функции в sql-запросе.

`WhenItems EntitySchemaCaseNotNullQueryFunctionWhenItems`

Коллекция условий функции выражения.

`HasWhenItems bool`

Признак, имеет ли функция хотя бы одно условие.

`ElseExpression EntitySchemaQueryExpression`

Выражение предложения `ELSE`.

Методы

`void SpecifyQueryAlias(string queryAlias)`

Определяет для текущей функции выражения заданный псевдоним в результирующем sql-запросе.

Параметры

<code>queryAlias</code>	Псевдоним, определяемый для текущей функции.
-------------------------	--

Класс EntitySchemaSystemValueQueryFunction [C#](#)

Пространство имен `Terrasoft.Core.Entities`.

Класс возвращает выражение системного значения.

На заметку. Полный перечень методов и свойств класса `EntitySchemaSystemValueQueryFunction`, его родительских классов, а также реализуемых им интерфейсов, можно найти в документации "[.NET библиотеки классов ядра платформы](#)".

Свойства

`QueryAlias string`

Псевдоним функции в sql-запросе.

`SystemValueName string`

Имя системного значения.

Класс EntitySchemaCurrentDateTimeQueryFunction c#

Пространство имен `Terrasoft.Core.Entities`.

Класс реализует функцию выражения текущей даты и времени.

На заметку. Полный перечень методов и свойств класса `EntitySchemaCurrentDateTimeQueryFunction`, его родительских классов, а также реализуемых им интерфейсов, можно найти в документации "[.NET библиотеки классов ядра платформы](#)".

Конструкторы

`EntitySchemaCurrentDateTimeQueryFunction(EntitySchemaQuery parentQuery)`

Инициализирует экземпляр `EntitySchemaCurrentDateTimeQueryFunction` для заданного запроса к схеме объекта.

Параметры

parentQuery	Запрос к схеме объекта, которому принадлежит функция.
-------------	---

`EntitySchemaCurrentDateTimeQueryFunction(EntitySchemaCurrentDateTimeQueryFunction source)`

Инициализирует экземпляр `EntitySchemaCurrentDateTimeQueryFunction`, являющийся клоном переданной функции.

Параметры

source	Экземпляр функции <code>EntitySchemaCurrentDateTimeQueryFunction</code> , клон которой создается.
--------	---

Свойства

`SystemValueName string`

Имя системного значения.

Методы

```
override string GetCaption()
```

Возвращает заголовок функции выражения.

```
override object Clone()
```

Создает клон текущего экземпляра `EntitySchemaCurrentDateTimeQueryFunction`.

Класс EntitySchemaBaseCurrentDateQueryFunction [C#](#)

Пространство имен `Terrasoft.Core.Entities`.

Базовый класс функции выражения для базовой даты.

На заметку. Полный перечень методов и свойств класса `EntitySchemaBaseCurrentDateQueryFunction`, его родительских классов, а также реализуемых им интерфейсов, можно найти в документации "[.NET библиотеки классов ядра платформы](#)".

Свойства

`SystemValueName` `string`

Имя системного значения.

`Offset` `int`

Смещение.

Класс EntitySchemaCurrentDateQueryFunction [C#](#)

Пространство имен `Terrasoft.Core.Entities`.

Класс реализует функцию выражения текущей даты.

На заметку. Полный перечень методов и свойств класса `EntitySchemaCurrentDateQueryFunction`, его родительских классов, а также реализуемых им интерфейсов, можно найти в документации "[.NET библиотеки классов ядра платформы](#)".

Конструкторы

```
EntitySchemaCurrentDateQueryFunction(EntitySchemaQuery parentQuery, int offset = 0) : this(parentQuery)
EntitySchemaCurrentDateQueryFunction(EntitySchemaQuery parentQuery, EntitySchemaQueryExpression
```

Инициализирует экземпляр `EntitySchemaCurrentDateQueryFunction` с указанным смещением относительно базовой даты для заданного запроса к схеме объекта.

Параметры

<code>parentQuery</code>	Запрос к схеме объекта, которому принадлежит функция.
<code>offset</code>	Смещение в днях относительно контрольной даты. Значение по умолчанию - <code>0</code> .
<code>expression</code>	Выражение запроса.

`EntitySchemaCurrentDateQueryFunction(EntitySchemaCurrentDateQueryFunction source)`

Инициализирует экземпляр `EntitySchemaCurrentDateQueryFunction`, являющийся клоном переданной функции.

Параметры

<code>source</code>	Экземпляр функции <code>EntitySchemaCurrentDateQueryFunction</code> , КЛОН которой создается.
---------------------	---

Методы

`override string GetCaption()`

Возвращает заголовок функции выражения.

`override object Clone()`

Создает клон текущего экземпляра `EntitySchemaCurrentDateQueryFunction`.

Класс EntitySchemaDateToCurrentYearQueryFunction C#

Пространство имен `Terrasoft.Core.Entities`.

Класс реализует функцию, которая конвертирует выражение даты в такую же дату текущего года.

На заметку. Полный перечень методов и свойств класса `EntitySchemaDateToCurrentYearQueryFunction`, его родительских классов, а также реализуемых им интерфейсов, можно найти в документации "[.NET библиотеки классов ядра платформы](#)".

Конструкторы

```
EntitySchemaDateToCurrentYearQueryFunction(EntitySchemaQuery parentQuery)
```

Инициализирует новый экземпляр `EntitySchemaDateToCurrentYearQueryFunction` для заданного запроса к схеме объекта.

Параметры

parentQuery	Запрос к схеме объекта, которому принадлежит функция.
-------------	---

```
EntitySchemaDateToCurrentYearQueryFunction(EntitySchemaQuery parentQuery, EntitySchemaQueryExpression expression)
```

Инициализирует новый экземпляр `EntitySchemaDateToCurrentYearQueryFunction` для заданного запроса к схеме объекта и переданного выражения даты.

Параметры

parentQuery	Запрос к схеме объекта, которому принадлежит функция.
expression	Выражение запроса.

```
EntitySchemaDateToCurrentYearQueryFunction(EntitySchemaDateToCurrentYearQueryFunction source)
```

Инициализирует новый экземпляр `EntitySchemaDateToCurrentYearQueryFunction`, являющийся клоном переданной функции.

Параметры

source	ФУНКЦИЯ <code>EntitySchemaDateToCurrentYearQueryFunction</code> , КЛОН которой создается.
--------	---

Свойства

`QueryAlias string`

Псевдоним функции в sql-запросе.

`Expression EntitySchemaQueryExpression`

Выражение аргументов функции.

Класс EntitySchemaStartOfCurrentWeekQueryFunction [C#](#)

Пространство имен `Terrasoft.Core.Entities`.

Класс реализует функцию выражения текущей даты.

На заметку. Полный перечень методов и свойств класса `EntitySchemaStartOfCurrentWeekQueryFunction`, его родительских классов, а также реализуемых им интерфейсов, можно найти в документации "[.NET библиотеки классов ядра платформы](#)".

Конструкторы

`EntitySchemaStartOfCurrentWeekQueryFunction(EntitySchemaQuery parentQuery, int offset = 0) : thi`
`EntitySchemaStartOfCurrentWeekQueryFunction(EntitySchemaQuery parentQuery, EntitySchemaQueryExpr`

Инициализирует экземпляр `EntitySchemaStartOfCurrentWeekQueryFunction` с указанным смещением относительно базовой даты для заданного запроса к схеме объекта.

Параметры

<code>parentQuery</code>	Запрос к схеме объекта, которому принадлежит функция.
<code>offset</code>	Смещение в днях относительно контрольной даты. Значение по умолчанию - <code>0</code> .
<code>expression</code>	Выражение запроса.

`EntitySchemaStartOfCurrentWeekQueryFunction(EntitySchemaStartOfCurrentWeekQueryFunction source)`

Инициализирует экземпляр `EntitySchemaStartOfCurrentWeekQueryFunction`, являющийся клоном переданной функции выражения.

Параметры

<code>source</code>	Экземпляр функции <code>EntitySchemaStartOfCurrentWeekQueryFunction</code> , клон которой создается.
---------------------	--

Методы

`override string GetCaption()`

Возвращает заголовок функции выражения.

`override object Clone()`

Создает клон текущего экземпляра `EntitySchemaStartOfCurrentWeekQueryFunction`.

Класс EntitySchemaStartOfCurrentMonthQueryFunction C#

Пространство имен `Terrasoft.Core.Entities`.

Класс реализует функцию выражения даты начала текущего месяца.

На заметку. Полный перечень методов и свойств класса

`EntitySchemaStartOfCurrentMonthQueryFunction`, его родительских классов, а также реализуемых им интерфейсов, можно найти в документации "[.NET библиотеки классов ядра платформы](#)".

Конструкторы

`EntitySchemaStartOfCurrentMonthQueryFunction(EntitySchemaQuery parentQuery, int offset = 0) : t`
`EntitySchemaStartOfCurrentMonthQueryFunction(EntitySchemaQuery parentQuery, EntitySchemaQueryExp`

Инициализирует экземпляр `EntitySchemaStartOfCurrentMonthQueryFunction` с указанным смещением относительно базовой даты для заданного запроса к схеме объекта.

Параметры

<code>parentQuery</code>	Запрос к схеме объекта, которому принадлежит функция.
<code>offset</code>	Смещение в днях относительно контрольной даты. Значение по умолчанию - <code>0</code> .
<code>expression</code>	Выражение запроса.

`EntitySchemaStartOfCurrentMonthQueryFunction(EntitySchemaStartOfCurrentMonthQueryFunction source)`

Инициализирует экземпляр `EntitySchemaStartOfCurrentMonthQueryFunction`, являющийся клоном переданной функции выражения.

Параметры

<code>source</code>	Экземпляр функции <code>EntitySchemaStartOfCurrentMonthQueryFunction</code> , клон которой создается.
---------------------	---

Методы

`override string GetCaption()`

Возвращает заголовок функции выражения.

```
override object Clone()
```

Создает клон текущего экземпляра `EntitySchemaStartOfCurrentMonthQueryFunction`.

Класс EntitySchemaStartOfCurrentQuarterQueryFunction [C#](#)

Пространство имен `Terrasoft.Core.Entities`.

Класс реализует функцию выражения даты начала текущего месяца.

На заметку. Полный перечень методов и свойств класса

`EntitySchemaStartOfCurrentQuarterQueryFunction`, его родительских классов, а также реализуемых им интерфейсов, можно найти в документации "[.NET библиотеки классов ядра платформы](#)".

Конструкторы

```
EntitySchemaStartOfCurrentQuarterQueryFunction(EntitySchemaQuery parentQuery, int offset = 0) : EntitySchemaStartOfCurrentQuarterQueryFunction(EntitySchemaQuery parentQuery, EntitySchemaQueryE
```

Инициализирует экземпляр `EntitySchemaStartOfCurrentQuarterQueryFunction` с указанным смещением относительно базовой даты для заданного запроса к схеме объекта.

Параметры

<code>parentQuery</code>	Запрос к схеме объекта, которому принадлежит функция.
<code>offset</code>	Смещение в днях относительно контрольной даты. Значение по умолчанию - <code>0</code> .
<code>expression</code>	Выражение запроса

```
EntitySchemaStartOfCurrentQuarterQueryFunction(EntitySchemaStartOfCurrentQuarterQueryFunction sc
```

Инициализирует экземпляр `EntitySchemaStartOfCurrentQuarterQueryFunction`, являющийся клоном переданной функции выражения.

Параметры

<code>source</code>	Экземпляр функции <code>EntitySchemaStartOfCurrentQuarterQueryFunction</code> , клон которой создается.
---------------------	---

Методы

`override string GetCaption()`

Возвращает заголовок функции выражения.

`override object Clone()`

Создает клон текущего экземпляра `EntitySchemaStartOfCurrentQuarterQueryFunction`.

Класс EntitySchemaStartOfCurrentHalfYearQueryFunction [c#](#)

Пространство имен `Terrasoft.Core.Entities`.

Класс реализует функцию выражения даты начала текущего полугодия.

На заметку. Полный перечень методов и свойств класса

`EntitySchemaStartOfCurrentHalfYearQueryFunction`, его родительских классов, а также реализуемых им интерфейсов, можно найти в документации "[.NET библиотеки классов ядра платформы](#)".

Конструкторы

`EntitySchemaStartOfCurrentHalfYearQueryFunction(EntitySchemaQuery parentQuery, int offset = 0)` :
`EntitySchemaStartOfCurrentHalfYearQueryFunction(EntitySchemaQuery parentQuery, EntitySchemaQuery`

Инициализирует экземпляр `EntitySchemaStartOfCurrentHalfYearQueryFunction` с указанным смещением относительно базовой даты для заданного запроса к схеме объекта.

Параметры

<code>parentQuery</code>	Запрос к схеме объекта, которому принадлежит функция.
<code>offset</code>	Смещение в днях относительно контрольной даты. Значение по умолчанию - <code>0</code> .
<code>expression</code>	Выражение запроса.

`EntitySchemaStartOfCurrentHalfYearQueryFunction(EntitySchemaStartOfCurrentHalfYearQueryFunction`

Инициализирует экземпляр `EntitySchemaStartOfCurrentHalfYearQueryFunction`, являющийся клоном переданной функции выражения.

Параметры

source	Экземпляр функции <code>EntitySchemaStartOfCurrentHalfYearQueryFunction</code> , клон которой создается.
--------	--

Методы

`override string GetCaption()`

Возвращает заголовок функции выражения.

`override object Clone()`

Создает клон текущего экземпляра `EntitySchemaStartOfCurrentHalfYearQueryFunction`.

Класс EntitySchemaStartOfCurrentYearQueryFunction [C#](#)

Пространство имен `Terrasoft.Core.Entities`.

Класс реализует функцию выражения даты начала текущего года.

На заметку. Полный перечень методов и свойств класса `EntitySchemaStartOfCurrentYearQueryFunction`, его родительских классов, а также реализуемых им интерфейсов, можно найти в документации "[.NET библиотеки классов ядра платформы](#)".

Конструкторы

`EntitySchemaStartOfCurrentYearQueryFunction(EntitySchemaQuery parentQuery, int offset = 0) : thi`
`EntitySchemaStartOfCurrentYearQueryFunction(EntitySchemaQuery parentQuery, EntitySchemaQueryExpr`

Инициализирует экземпляр `EntitySchemaStartOfCurrentYearQueryFunction` с указанным смещением относительно базовой даты для заданного запроса к схеме объекта.

Параметры

<code>parentQuery</code>	Запрос к схеме объекта, которому принадлежит функция.
<code>offset</code>	Смещение в днях относительно контрольной даты. Значение по умолчанию - <code>0</code> .
<code>expression</code>	Выражение запроса.

`EntitySchemaStartOfCurrentYearQueryFunction(EntitySchemaStartOfCurrentYearQueryFunction source)`

Инициализирует экземпляр `EntitySchemaStartOfCurrentYearQueryFunction`, являющийся клоном переданной функции выражения.

Параметры

source	Экземпляр функции <code>EntitySchemaStartOfCurrentYearQueryFunction</code> , клон которой создается.
--------	--

Методы

`override string GetCaption()`

Возвращает заголовок функции выражения.

`override object Clone()`

Создает клон текущего экземпляра `EntitySchemaStartOfCurrentHalfYearQueryFunction`.

Класс EntitySchemaBaseCurrentDateTimeQueryFunction [c#](#)

Пространство имен `Terrasoft.Core.Entities`.

Базовый класс функции выражения базовых даты и времени.

На заметку. Полный перечень методов и свойств класса

`EntitySchemaBaseCurrentDateTimeQueryFunction`, его родительских классов, а также реализуемых им интерфейсов, можно найти в документации "[.NET библиотеки классов ядра платформы](#)".

Свойства

`SystemValueName string`

Имя системного значения.

Класс EntitySchemaStartOfCurrentHourQueryFunction [c#](#)

Пространство имен `Terrasoft.Core.Entities`.

Класс реализует функцию выражения начала текущего часа.

На заметку. Полный перечень методов и свойств класса `EntitySchemaStartOfCurrentHourQueryFunction`, его родительских классов, а также реализуемых им интерфейсов, можно найти в документации "[.NET библиотеки классов ядра платформы](#)".

Конструкторы

`EntitySchemaStartOfCurrentHourQueryFunction(EntitySchemaQuery parentQuery, int offset = 0) : bas`

Инициализирует экземпляр `EntitySchemaStartOfCurrentHourQueryFunction`, который является частью `parentQuery` и указан `offset` относительно базовой даты.

Параметры

<code>parentQuery</code>	Экземпляр <code>EntitySchemaQuery</code> .
<code>offset</code>	Смещение в часах относительно базовой даты.

`EntitySchemaStartOfCurrentHourQueryFunction(EntitySchemaQuery parentQuery, EntitySchemaQueryExpr`

Инициализирует экземпляр `EntitySchemaStartOfCurrentHourQueryFunction`, который является частью `parentQuery`, имеет указанные аргументы `expression` и `offset` относительно базовой даты.

Параметры

<code>parentQuery</code>	Экземпляр <code>EntitySchemaQuery</code> .
<code>expression</code>	Выражение аргумента функции.
<code>offset</code>	Смещение в часах относительно базовой даты.

`EntitySchemaStartOfCurrentHourQueryFunction(EntitySchemaStartOfCurrentHourQueryFunction source)`

Инициализирует экземпляр `EntitySchemaStartOfCurrentHourQueryFunction`, являющийся клоном переданной функции выражения.

Параметры

<code>source</code>	Экземпляр функции <code>EntitySchemaStartOfCurrentHourQueryFunction</code> , клон которой создается.
---------------------	--

Методы

`override string GetCaption()`

Возвращает заголовок функции выражения.

```
override object Clone()
```

Создает клон текущего экземпляра `EntitySchemaStartOfCurrentHourQueryFunction`.

Класс EntitySchemaCurrentTimeQueryFunction C#

Пространство имен `Terrasoft.Core.Entities`.

Класс реализует функцию выражения текущего времени.

На заметку. Полный перечень методов и свойств класса `EntitySchemaCurrentTimeQueryFunction`, его родительских классов, а также реализуемых им интерфейсов, можно найти в документации "[.NET библиотеки классов ядра платформы](#)".

Конструкторы

`EntitySchemaCurrentTimeQueryFunction(EntitySchemaQuery parentQuery) : base(parentQuery)`

Инициализирует новый экземпляр `EntitySchemaCurrentTimeQueryFunction` для заданного запроса к схеме объекта.

Параметры

<code>parentQuery</code>	Запрос к схеме объекта, которому принадлежит функция.
--------------------------	---

`EntitySchemaCurrentTimeQueryFunction(EntitySchemaCurrentTimeQueryFunction source) : base(source)`

Инициализирует новый экземпляр `EntitySchemaCurrentTimeQueryFunction`, являющийся клоном переданной функции.

Параметры

<code>source</code>	ФУНКЦИЯ <code>EntitySchemaCurrentTimeQueryFunction</code> , клон которой создается.
---------------------	---

Свойства

`SystemValueName string`

Имя системного значения.

Методы

```
override string GetCaption()
```

Возвращает заголовок функции выражения.

```
override object Clone()
```

Создает клон текущего экземпляра `EntitySchemaCurrentTimeQueryFunction`.

Класс EntitySchemaCurrentUserQueryFunction [C#](#)

Пространство имен `Terrasoft.Core.Entities`.

Класс реализует функцию выражения текущего пользователя.

На заметку. Полный перечень методов и свойств класса `EntitySchemaCurrentUserQueryFunction`, его родительских классов, а также реализуемых им интерфейсов, можно найти в документации "[.NET библиотеки классов ядра платформы](#)".

Конструкторы

```
EntitySchemaCurrentUserQueryFunction(EntitySchemaQuery parentQuery) : base(parentQuery)
```

Инициализирует новый экземпляр `EntitySchemaCurrentUserQueryFunction` для заданного запроса к схеме объекта.

Параметры

parentQuery	Запрос к схеме объекта, которому принадлежит функция.
-------------	---

```
EntitySchemaCurrentUserQueryFunction(EntitySchemaCurrentUserQueryFunction source) : base(source)
```

Инициализирует новый экземпляр `EntitySchemaCurrentUserQueryFunction`, являющийся клоном переданной функции.

Параметры

source	ФУНКЦИЯ <code>EntitySchemaCurrentUserQueryFunction</code> , клон которой создается.
--------	---

Свойства

Системное значение для

Имя системного значения.

Методы

`override string GetCaption()`

Возвращает заголовок функции выражения.

`override object Clone()`

Создает клон текущего экземпляра `EntitySchemaCurrentUserQueryFunction`.

Класс EntitySchemaCurrentUserContactQueryFunction c#

Пространство имен `Terrasoft.Core.Entities`.

Класс реализует функцию выражения контакта текущего пользователя.

На заметку. Полный перечень методов и свойств класса `EntitySchemaCurrentUserContactQueryFunction`, его родительских классов, а также реализуемых им интерфейсов, можно найти в документации "[.NET библиотеки классов ядра платформы](#)".

Конструкторы

`EntitySchemaCurrentUserContactQueryFunction(EntitySchemaQuery parentQuery) : base(parentQuery)`

Инициализирует новый экземпляр `EntitySchemaCurrentUserContactQueryFunction` для заданного запроса к схеме объекта.

Параметры

`parentQuery`

Запрос к схеме объекта, которому принадлежит функция.

`EntitySchemaCurrentUserContactQueryFunction(EntitySchemaCurrentUserContactQueryFunction source)`

Инициализирует новый экземпляр `EntitySchemaCurrentUserContactQueryFunction`, являющийся клоном переданной функции.

Параметры

`source`

ФУНКЦИЯ `EntitySchemaCurrentUserContactQueryFunction`, КЛОН которой создается.

Свойства

`SystemValueName string`

Имя системного значения.

Методы

`override string GetCaption()`

Возвращает заголовок функции выражения.

`override object Clone()`

Создает клон текущего экземпляра `EntitySchemaCurrentUserContactQueryFunction`.

Класс EntitySchemaCurrentUserAccountQueryFunction C#

Пространство имен `Terrasoft.Core.Entities`.

Класс реализует функцию выражения контрагента текущего пользователя.

На заметку. Полный перечень методов и свойств класса `EntitySchemaCurrentUserAccountQueryFunction`, его родительских классов, а также реализуемых им интерфейсов, можно найти в документации "[.NET библиотеки классов ядра платформы](#)".

Конструкторы

`EntitySchemaCurrentUserAccountQueryFunction(EntitySchemaQuery parentQuery)`

Инициализирует новый экземпляр `EntitySchemaCurrentUserAccountQueryFunction` для заданного запроса к схеме объекта.

Параметры

<code>parentQuery</code>	Запрос к схеме объекта, которому принадлежит функция.
--------------------------	---

`EntitySchemaCurrentUserAccountQueryFunction(EntitySchemaCurrentUserAccountQueryFunction source)`

Инициализирует новый экземпляр `EntitySchemaCurrentUserAccountQueryFunction`, являющийся клоном переданной функции.

Параметры

source	ФУНКЦИЯ <code>EntitySchemaCurrentUserAccountQueryFunction</code> , КЛОН которой создается.
--------	--

Свойства

`SystemValueName string`

Имя системного значения.

Класс EntitySchemaDatePartQueryFunction [c#](#)

Пространство имен `Terrasoft.Core.Entities`.

Класс реализует функцию запроса для части даты.

На заметку. Полный перечень методов и свойств класса `EntitySchemaDatePartQueryFunction`, его родительских классов, а также реализуемых им интерфейсов, можно найти в документации "[.NET библиотеки классов ядра платформы](#)".

Конструкторы

`EntitySchemaDatePartQueryFunction(EntitySchemaQuery parentQuery) : base(parentQuery)`

Инициализирует новый экземпляр `EntitySchemaDatePartQueryFunction` для заданного запроса к схеме объекта.

Параметры

parentQuery	Экземпляр <code>EntitySchemaQuery</code> .
-------------	--

`EntitySchemaDatePartQueryFunction(EntitySchemaQuery parentQuery, EntitySchemaDatePartQueryFunction expression)`

Инициализирует новый экземпляр `EntitySchemaDatePartQueryFunction`, который является частью `parentQuery` с указанной частью даты `interval` для запроса к схеме сущности и выражению запроса `expression`.

Параметры

parentQuery	Экземпляр <code>EntitySchemaQuery</code> .
interval	Часть даты.
expression	Выражение запроса.

`EntitySchemaDatePartQueryFunction(EntitySchemaDatePartQueryFunction source) : base(source)`

Инициализирует новый экземпляр `EntitySchemaDatePartQueryFunction`, являющийся клоном переданной функции.

Параметры

source	ФУНКЦИЯ <code>EntitySchemaDatePartQueryFunction</code> , клон которой создается.
--------	--

Свойства

`QueryAlias string`

Псевдоним функции в sql-запросе.

`EntitySchemaDatePartQueryFunctionInterval Interval`

Часть даты, возвращаемая функцией.

`EntitySchemaQueryExpression Expression`

Выражение аргумента функции.

Методы

`override void WriteMetaData(DataWriter writer)`

Выполняет сериализацию функции, используя заданный экземпляр `Terrasoft.Common.DataWriter`.

Параметры

writer	Экземпляр <code>Terrasoft.Common.DataWriter</code> , с помощью которого выполняется сериализация.
--------	---

```
override QueryColumnExpression CreateQueryColumnExpression(DBSecurityEngine dbSecurityEngine)
```

Возвращает выражение колонки запроса для текущей функции, сформированное с учетом заданных прав доступа.

Параметры

dbSecurityEngine	Объект <code>Terrasoft.Core.DB.DBSecurityEngine</code> , определяющий права доступа.
------------------	--

```
override DataValueType GetResultDataValueType(DataValueTypeManager dataTypeManager)
```

Возвращает тип данных возвращаемого функцией результата, используя переданный менеджер типов данных.

Параметры

dataTypeManager	Менеджер типов данных.
-----------------	------------------------

```
override bool GetIsSupportedDataValueType(DataValueType dataType)
```

Определяет, имеет ли возвращаемый функцией результат указанный тип данных.

Параметры

dataType	Тип данных.
----------	-------------

```
override string GetCaption()
```

Возвращает заголовок функции выражения.

```
override EntitySchemaQueryExpressionCollection GetArguments()
```

Возвращает коллекцию выражений аргументов функции.

```
override object Clone()
```

Создает клон текущего экземпляра `EntitySchemaUpperQueryFunction`.

Класс EntitySchemaUpperQueryFunction [c#](#)

Пространство имен `Terrasoft.Core.Entities`.

Класс преобразовывает символы выражения аргумента к верхнему регистру.

На заметку. Полный перечень методов и свойств класса `EntitySchemaUpperQueryFunction`, его родительских классов, а также реализуемых им интерфейсов, можно найти в документации "[.NET библиотеки классов ядра платформы](#)".

Конструкторы

`EntitySchemaUpperQueryFunction(EntitySchemaQuery parentQuery)`

Инициализирует новый экземпляр `EntitySchemaUpperQueryFunction` для заданного запроса к схеме объекта.

Параметры

<code>parentQuery</code>	Запрос к схеме объекта, которому принадлежит функция.
--------------------------	---

`EntitySchemaUpperQueryFunction(EntitySchemaQuery parentQuery, EntitySchemaQueryExpression expression)`

Инициализирует новый экземпляр `EntitySchemaUpperQueryFunction` для заданного запроса к схеме объекта и переданного выражения даты.

Параметры

<code>parentQuery</code>	Запрос к схеме объекта, которому принадлежит функция.
<code>expression</code>	Выражение запроса.

`EntitySchemaUpperQueryFunction(EntitySchemaUpperQueryFunction source)`

Инициализирует новый экземпляр `EntitySchemaUpperQueryFunction`, являющийся клоном переданной функции.

Параметры

<code>source</code>	ФУНКЦИЯ <code>EntitySchemaUpperQueryFunction</code> , КЛОН которой создается.
---------------------	---

Свойства

`QueryAlias string`

Псевдоним функции в sql-запросе.

`Expression EntitySchemaQueryExpression`

Выражение аргументов функции.

Класс EntitySchemaCastQueryFunction c#

Пространство имен `Terrasoft.Core.Entities`.

Класс приводит выражение аргумента к заданному типу данных.

На заметку. Полный перечень методов и свойств класса `EntitySchemaCastQueryFunction`, его родительских классов, а также реализуемых им интерфейсов, можно найти в документации "[.NET библиотеки классов ядра платформы](#)".

Конструкторы

`EntitySchemaCastQueryFunction(EntitySchemaQuery parentQuery, DBDataType castType)`

Инициализирует новый экземпляр `EntitySchemaCastQueryFunction` для заданного запроса к схеме объекта с указанным целевым типом данных.

Параметры

<code>parentQuery</code>	Запрос к схеме объекта, которому принадлежит функция.
<code>castType</code>	Целевой тип данных.

`EntitySchemaCastQueryFunction(EntitySchemaQuery parentQuery, EntitySchemaQueryExpression expression)`

Инициализирует новый экземпляр `EntitySchemaCastQueryFunction` с заданными выражением и целевым типом данных.

Параметры

<code>parentQuery</code>	Запрос к схеме объекта, которому принадлежит функция.
<code>expression</code>	Выражение запроса.
<code>castType</code>	Целевой тип данных.

`EntitySchemaCastQueryFunction(EntitySchemaCastQueryFunction source)`

Инициализирует новый экземпляр `EntitySchemaCastQueryFunction`, являющийся клоном переданной функции.

Параметры

<code>source</code>	ФУНКЦИЯ <code>EntitySchemaCastQueryFunction</code> , КЛОН КОТОРОЙ создается.
---------------------	--

Свойства

`QueryAlias string`

Псевдоним функции в sql-запросе.

`Expression EntitySchemaQueryExpression`

Выражение аргумента функции.

`CastType DBValueType`

Целевой тип данных.

Класс EntitySchemaTrimQueryFunction [c#](#)

Пространство имен `Terrasoft.Core.Entities`.

Класс удаляет начальные и конечные пробелы из выражения.

На заметку. Полный перечень методов и свойств класса `EntitySchemaTrimQueryFunction`, его родительских классов, а также реализуемых им интерфейсов, можно найти в документации "[.NET библиотеки классов ядра платформы](#)".

Конструкторы

`EntitySchemaTrimQueryFunction(EntitySchemaQuery parentQuery)`

Инициализирует новый экземпляр `EntitySchemaTrimQueryFunction` для заданного запроса к схеме объекта.

Параметры

parentQuery	Запрос к схеме объекта, которому принадлежит функция.
-------------	---

```
EntitySchemaTrimQueryFunction(EntitySchemaQuery parentQuery, EntitySchemaQueryExpression expression)
```

Инициализирует новый экземпляр `EntitySchemaTrimQueryFunction` для заданного запроса к схеме объекта и переданного выражения даты.

Параметры

parentQuery	Запрос к схеме объекта, которому принадлежит функция.
expression	Выражение запроса.

```
EntitySchemaTrimQueryFunction(EntitySchemaTrimQueryFunction source)
```

Инициализирует новый экземпляр `EntitySchemaTrimQueryFunction`, являющийся клоном переданной функции.

Параметры

source	ФУНКЦИЯ <code>EntitySchemaTrimQueryFunction</code> , клон которой создается.
--------	--

Свойства

```
QueryAlias string
```

Псевдоним функции в sql-запросе.

```
Expression EntitySchemaQueryExpression
```

Выражение аргументов функции.

Класс EntitySchemaLengthQueryFunction [c#](#)

Пространство имен `Terrasoft.Core.Entities`.

Класс возвращает длину выражения.

На заметку. Полный перечень методов и свойств класса `EntitySchemaLengthQueryFunction`, его родительских классов, а также реализуемых им интерфейсов, можно найти в документации "[.NET библиотеки классов ядра платформы](#)".

Конструкторы

`EntitySchemaLengthQueryFunction(EntitySchemaQuery parentQuery)`

Инициализирует новый экземпляр `EntitySchemaLengthQueryFunction` для заданного запроса к схеме объекта.

Параметры

<code>parentQuery</code>	Запрос к схеме объекта, которому принадлежит функция.
--------------------------	---

`EntitySchemaLengthQueryFunction(EntitySchemaQuery parentQuery, EntitySchemaQueryExpression expression)`

Инициализирует новый экземпляр `EntitySchemaLengthQueryFunction` для заданного запроса к схеме объекта и переданного выражения даты.

Параметры

<code>parentQuery</code>	Запрос к схеме объекта, которому принадлежит функция.
<code>expression</code>	Выражение запроса.

`EntitySchemaLengthQueryFunction(EntitySchemaLengthQueryFunction source)`

Инициализирует новый экземпляр `EntitySchemaLengthQueryFunction`, являющийся клоном переданной функции.

Параметры

<code>source</code>	ФУНКЦИЯ <code>EntitySchemaLengthQueryFunction</code> , КЛОН которой создается.
---------------------	--

Свойства

`QueryAlias string`

Псевдоним функции в sql-запросе.

`Expression EntitySchemaQueryExpression`

Выражение аргументов функции.

Класс EntitySchemaConcatQueryFunction C#

Пространство имен `Terrasoft.Core.Entities`.

Класс формирует строку, которая является результатом объединения строковых значений аргументов функции.

На заметку. Полный перечень методов и свойств класса `EntitySchemaConcatQueryFunction`, его родительских классов, а также реализуемых им интерфейсов, можно найти в документации "[.NET библиотеки классов ядра платформы](#)".

Конструкторы

`EntitySchemaConcatQueryFunction(EntitySchemaQuery parentQuery)`

Инициализирует новый экземпляр `EntitySchemaConcatQueryFunction` для заданного запроса к схеме объекта.

Параметры

<code>parentQuery</code>	Запрос к схеме объекта, которому принадлежит функция.
--------------------------	---

`EntitySchemaConcatQueryFunction(EntitySchemaQuery parentQuery, EntitySchemaQueryExpression[] exp)`

Инициализирует новый экземпляр `EntitySchemaConcatQueryFunction` для заданных массива выражений и запроса к схеме объекта.

Параметры

<code>parentQuery</code>	Запрос к схеме объекта, которому принадлежит функция.
<code>expressions</code>	Массив выражений.

`EntitySchemaConcatQueryFunction(EntitySchemaConcatQueryFunction source)`

Инициализирует новый экземпляр `EntitySchemaConcatQueryFunction`, являющийся клоном переданной функции.

Параметры

<code>source</code>	ФУНКЦИЯ <code>EntitySchemaConcatQueryFunction</code> , КЛОН которой создается.
---------------------	--

Свойства

`QueryAlias string`

Псевдоним функции в sql-запросе.

`Expressions EntitySchemaQueryExpressionCollection`

Коллекция выражений аргументов функции.

`HasExpressions bool`

Признак, определяющий наличие хотя бы одного элемента в коллекции выражений аргументов функции.

Класс EntitySchemaWindowQueryFunction [c#](#)

Пространство имен `Terrasoft.Core.Entities`.

Класс реализует функцию SQL окна.

На заметку. Полный перечень методов и свойств класса `EntitySchemaWindowQueryFunction`, его родительских классов, а также реализуемых им интерфейсов, можно найти в документации "[.NET библиотеки классов ядра платформы](#)".

Конструкторы

`EntitySchemaWindowQueryFunction(EntitySchemaQuery parentQuery)`

Инициализирует новый экземпляр `EntitySchemaWindowQueryFunction` для заданного запроса к схеме объекта.

Параметры

<code>parentQuery</code>	Запрос к схеме объекта, которому принадлежит функция.
--------------------------	---

`EntitySchemaWindowQueryFunction(EntitySchemaQueryExpression function, EntitySchemaQuery esq)`

Инициализирует новый экземпляр `EntitySchemaWindowQueryFunction` для заданного запроса к схеме объекта.

Параметры

function	Вложенная функция запроса.
esq	Запрос к схеме объекта.

`EntitySchemaWindowQueryFunction(EntitySchemaQueryExpression function, EntitySchemaQuery esq, Ent`
Инициализирует новый экземпляр `EntitySchemaWindowQueryFunction` для заданного запроса к схеме
объекта.

Параметры

function	Вложенная функция запроса.
parentQuery	Запрос к схеме объекта, которому принадлежит функция.
partitionBy	Выражение для разделения запроса.
orderBy	Выражение для сортировки запроса.

`EntitySchemaWindowQueryFunction(EntitySchemaQueryFunction source)`

Инициализирует новый экземпляр `EntitySchemaWindowQueryFunction`, являющийся клоном переданной
функции.

Параметры

source	ФУНКЦИЯ <code>EntitySchemaQueryFunction</code> , КЛОН которой создается.
--------	--

`EntitySchemaWindowQueryFunction(EntitySchemaWindowQueryFunction source)`

Инициализирует новый экземпляр `EntitySchemaWindowQueryFunction`, являющийся клоном переданной
функции.

Параметры

source	ФУНКЦИЯ <code>EntitySchemaWindowQueryFunction</code> , КЛОН которой создается.
--------	---

Свойства

`QueryAlias string`

Псевдоним функции в sql-запросе.

`InnerFunction EntitySchemaQueryExpression`

Функция для применения.

`PartitionByExpression EntitySchemaQueryExpression`

Разделение по пунктам.

`OrderByExpression EntitySchemaQueryExpression`

Сортировать по пункту.

Класс EntitySchemaQueryOptions c#



Сложный

Пространство имен `Terrasoft.Core.Entities`.

Класс `Terrasoft.Core.Entities.EntitySchemaQueryOptions` предназначен для настроек запроса к схеме объекта.

На заметку. Полный перечень методов и свойств класса `EntitySchemaQueryOptions`, его родительских классов, а также реализуемых им интерфейсов можно найти в [Библиотеке .NET классов](#).

Конструкторы

`EntitySchemaQueryOptions`

Инициализирует экземпляр класса. В конструкторе свойству `PageableRowCount` по умолчанию устанавливается значение 14.

Свойства

`PageableRowCount int`

Количество записей страницы результирующего набора данных, возвращаемого запросом.

`PageableDirection Terrasoft.Core.DB.PageableSelectDirection`

Направление постраничного вывода.

Возможные значения (`Terrasoft.Core.DB.PageableSelectDirection`)

Prior	Предыдущая страница.
First	Первая страница.
Current	Текущая страница.
Next	Следующая страница.

`PageableConditionValues Dictionary<string, object>`

Значения условий постраничного вывода.

`HierarchicalMaxDepth int`

Максимальный уровень вложенности иерархического запроса.

`HierarchicalColumnName string`

Имя колонки, которая используется для построения иерархического запроса.

`HierarchicalColumnValue Guid`

Начальное значение иерархической колонки, от которого будет строиться иерархия.

Пользовательские веб-сервисы



Средний

Веб-сервис — идентифицируемая уникальным веб-адресом (URL-адресом) программная система, которая обеспечивает взаимодействие между приложениями. **Назначение** веб-сервиса — настройка интеграции между Creatio и внешними приложениями и системами.

На основе пользовательской бизнес-логики Creatio сгенерирует и отправит запрос веб-сервису, получит ответ и предоставит необходимые данные. Эти данные можно использовать для создания или обновления записей в Creatio, а также для реализации пользовательской бизнес-логики или автоматизации.

Виды веб-сервисов в Creatio:

- **Внешние REST и SOAP-сервисы**, с которыми можно настроить интеграцию low-code инструментами. Подробнее читайте в блоке статей [Веб-сервисы](#) документации для пользователя.
- **Системные веб-сервисы**.
 - Системные веб-сервисы с аутентификацией на основе cookies.

- Системные веб-сервисы с анонимной аутентификацией.
- Пользовательские веб-сервисы.**
 - Пользовательские веб-сервисы с аутентификацией на основе cookies.
 - Пользовательские веб-сервисы с анонимной аутентификацией.

Системные веб-сервисы, разработанные на .NET Framework, реализованы на основе технологии [WCF](#) и управляются на уровне IIS. Системные веб-сервисы, разработанные на .NET Core, реализованы на основе технологии [ASP.NET Core Web API](#).

Виды аутентификации, которые поддерживаются веб-сервисами в Creatio, описаны в статье [Аутентификация](#). Рекомендуемым способом аутентификации является аутентификация на основе открытого протокола авторизации OAuth 2.0, которая описана в статье [Настройте авторизацию интегрированных приложений по протоколу OAuth 2.0](#).

Примеры **системных веб-сервисов с аутентификацией на основе cookies**, предоставляемых Creatio:

- `odata` — выполнение запросов от внешних приложений к серверу баз данных Creatio по протоколу OData 4. Описание использования протокола OData 4 в Creatio содержится в статье [OData](#).
- `EntityDataService.svc` — выполнение запросов от внешних приложений к серверу баз данных Creatio по протоколу OData 3. Описание использования протокола OData 3 в Creatio содержится в статье [OData](#).
- `ProcessEngineService.svc` — запуск бизнес-процессов Creatio из внешних приложений. Описание веб-сервиса содержится в статье [Сервис запуска бизнес-процессов](#).

Примеры **системных веб-сервисов с анонимной аутентификацией**, предоставляемых Creatio:

- `AuthService.svc` — выполнение запроса на аутентификацию в приложении Creatio. Описание веб-сервиса содержится в статье [Аутентификация](#).

В этой статье рассмотрены пользовательские веб-сервисы. Системные веб-сервисы описаны в разделе [Интеграция и внешний API](#).

Разработать пользовательский веб-сервис

Пользовательский веб-сервис — RESTful-сервис, реализованный на базе технологии WCF (для .NET Framework) или ASP.NET Core Web API (для .NET Core). **Отличие** пользовательского веб-сервиса от системного — возможность реализации специфических интеграционных задач.

В зависимости от платформы, на которой развернуто приложение, разработка пользовательского веб-сервиса имеет свои особенности. Ниже рассмотрены особенности разработки пользовательского веб-сервиса для платформ .NET Framework и .NET Core.

Разработать пользовательский веб-сервис с аутентификацией на основе cookies

- Создайте схему [[Исходный код](#)] ([\[Source code\]](#)). Для создания схемы воспользуйтесь статьей [Разработка конфигурационных элементов](#).

2. Создайте класс сервиса.

- a. В дизайнере схем добавьте пространство имен `Terrasoft.Configuration` или любое вложенное в него пространство имен. Название пространства имен может быть любым.
- b. С помощью директивы `using` добавьте пространства имен, типы данных которых будут задействованы в классе.
- c. Используйте пространство имен `Terrasoft.Web.Http.Abstractions`, которое позволит пользовательскому веб-сервису работать на платформах .NET Framework и .NET Core. Если при разработке веб-сервиса было использовано пространство имен `System.Web` и необходимо запустить веб-сервис на платформе .NET Core, то [выполните адаптацию веб-сервиса](#).
- d. Добавьте название класса, которое соответствует названию схемы (свойство [*Код*] ([*Code*])).
- e. В качестве родительского класса укажите класс `Terrasoft.Nui.ServiceModel.WebService.BaseService`.
- f. Для класса добавьте атрибуты `[ServiceContract]` и `[AspNetCompatibilityRequirements]` с необходимыми параметрами. Атрибут `[ServiceContract]` описан в официальной [документации Microsoft](#). Атрибут `[AspNetCompatibilityRequirements]` описан в официальной [документации Microsoft](#).

3. Реализуйте методы класса, которые соответствуют конечным точкам веб-сервиса.

Для методов добавьте атрибуты `[OperationContract]` и `[WebInvoke]` с необходимыми параметрами. Атрибут `[OperationContract]` описан в официальной [документации Microsoft](#). Атрибут `[WebInvoke]` описан в официальной [документации Microsoft](#).

4. Реализуйте дополнительные классы, экземпляры которых будут принимать или возвращать методы веб-сервиса (опционально). Выполняется при необходимости передачи данных сложных типов (экземпляры объектов, коллекции, массивы и т. д.).

Для класса добавьте атрибут `[DataContract]`, а для полей класса — атрибут `[DataMember]`. Атрибут `[DataContract]` описан в официальной [документации Microsoft](#). Атрибут `[DataMember]` описан в официальной [документации Microsoft](#).

5. Опубликуйте схему исходного кода.

В результате пользовательский веб-сервис с аутентификацией на основе cookies будет доступен для вызова из программного кода конфигурационных схем, а также из внешних приложений.

Разработать пользовательский веб-сервис с анонимной аутентификацией

Пользовательские веб-сервисы с анонимной аутентификацией — веб-сервисы, которые не требуют предварительной аутентификации пользователя, т. е. доступны для анонимного использования.

Важно. При реализации пользовательских веб-сервисов не рекомендуется использовать анонимную аутентификацию, поскольку это может снижать безопасность и производительность.

Разработать пользовательский веб-сервис с анонимной аутентификацией для платформы .NET Framework

1. Выполните шаги 1-5 инструкции [Разработать пользовательский веб-сервис с аутентификацией на основе cookies](#).
2. При создании класса сервиса добавьте системное подключение `SystemUserConnection`.
3. При создании метода класса укажите пользователя, от имени которого будет выполняться обработка http-запроса. Для этого после получения `SystemUserConnection` вызовите метод `SessionHelper.SpecifyWebOperationIdentity` пространства имен `Terrasoft.Web.Common`. Этот метод обеспечивает работоспособность бизнес-процессов при работе с сущностью (`Entity`) базы данных из пользовательского веб-сервиса с анонимной аутентификацией.

```
Terrasoft.Web.Common.SessionHelper.SpecifyWebOperationIdentity(HttpContextAccessor.GetInstanc
```

4. Зарегистрируйте пользовательский веб-сервис с анонимной аутентификацией:

- a. Перейдите в каталог `..\Terrasoft.WebApp\ServiceModel`.
- b. Создайте файл с названием веб-сервиса и расширением `*.svc`. Добавьте в него запись.

Шаблон регистрации пользовательского веб-сервиса с анонимной аутентификацией

```
<% @ServiceHost
    Service = "Service, ServiceNamespace"
    Factory = "Factory, FactoryNamespace"
    Debug = "Debug"
    Language = "Language"
    CodeBehind = "CodeBehind"
%>
```

Пример регистрации пользовательского веб-сервиса с анонимной аутентификацией

```
<% @ServiceHost
    Service = "Terrasoft.Configuration.UsrAnonymousConfigurationServiceNamespace.UsrAnonymous"
    Debug = "true"
    Language = "C#"
%>
```

Атрибут `Service` должен содержать полное имя класса веб-сервиса с указанием пространства имен.

WCF-директива `@ServiceHost` описана в официальной [документации Microsoft](#).

5. Настройте пользовательский веб-сервис с анонимной аутентификацией для работы по протоколам `http` и `https`:

 - a. Откройте файл `..\Terrasoft.WebApp\ServiceModel\http\services.config` и добавьте в него запись.

Пример изменений файла ..\Terrasoft.WebApp\ServiceModel\http\services.config

```

<services>
    ...
    <service name="Terrasoft.Configuration.[Custom namespace].[Service name]">
        <endpoint name="[Service name]EndPoint"
            address=""
            binding="webHttpBinding"
            behaviorConfiguration="RestServiceBehavior"
            bindingNamespace="http://Terrasoft.WebApp.ServiceModel"
            contract="Terrasoft.Configuration.[Custom namespace].[Service name]" />
    </service>
</services>

```

`<services>` — элемент, который содержит перечень конфигураций всех веб-сервисов приложения (вложенные элементы `<service>`).

`name` — атрибут, который содержит название типа (класса или интерфейса), реализующего контракт веб-сервиса.

`<endpoint>` — вложенный элемент, который содержит адрес, привязку и интерфейс, определяющий контракт веб-сервиса, указанного в атрибуте `name` элемента `<service>`.

Описание элементов конфигурирования веб-сервиса доступно в официальной [документации Microsoft](#).

- b. Аналогичную запись добавьте в файл `..\Terrasoft.WebApp\ServiceModel\https\services.config`.
6. Настройте доступ к пользовательскому веб-сервису с анонимной аутентификацией для всех пользователей:
 - a. Откройте файл `..\Terrasoft.WebApp\Web.config`.
 - b. Добавьте элемент `<location>`, определяющий относительный путь и права доступа к веб-сервису.

Пример изменений файла ..\Terrasoft.WebApp\Web.config

```

<configuration>
    ...
    <location path="ServiceModel/[Service name].svc">
        <system.web>
            <authorization>
                <allow users="*" />
            </authorization>
        </system.web>
    </location>
    ...
</configuration>

```

- с. В атрибут `value` ключа `AllowedLocations` элемента `<appSettings>` добавьте относительный путь к веб-сервису.

Пример изменений файла ..\Terrasoft.WebApp\Web.config

```
<configuration>
  ...
  <appSettings>
  ...
    <add key="AllowedLocations" value="[Предыдущие значения];ServiceModel/[Service name]" />
  ...
</appSettings>
  ...
</configuration>
```

7. Перезапустите приложение в IIS.

В результате пользовательский веб-сервис с анонимной аутентификацией будет доступен для вызова из программного кода конфигурационных схем, а также из внешних приложений. К веб-сервису можно обращаться, как с предварительной аутентификацией, так и без нее.

Разработать пользовательский веб-сервис с анонимной аутентификацией для платформы .NET Core

- Выполните шаги 1-5 инструкции [Разработать пользовательский веб-сервис с аутентификацией на основе cookies](#).
- Настройте доступ к пользовательскому веб-сервису с анонимной аутентификацией для всех пользователей:

Пример изменений файла ..\TerrasoftWebHost\appsettings.json

```
"Terrasoft.Configuration.[Service name]": [
  "/ServiceModel/[Service name].svc"
]
```

- Откройте конфигурационный файл ..\TerrasoftWebHost\appsettings.json.
- Добавьте информацию о веб-сервисе в блок `AnonymousRoutes` файла.
- Перезапустите приложение.

В результате пользовательский веб-сервис с анонимной аутентификацией будет доступен для вызова из программного кода конфигурационных схем, а также из внешних приложений. К веб-сервису можно обращаться, как с предварительным вводом логина и пароля, так и без их использования.

Важно. После обновления приложения необходимо выполнить повторную настройку веб-сервиса,

поскольку при обновлении приложения все конфигурационные файлы заменяются новыми.

Вызвать пользовательский веб-сервис

Пользовательский веб-сервис можно вызвать из браузера и из front-end части.

Вызвать пользовательский веб-сервис из браузера

Вызвать из браузера пользовательский веб-сервис с аутентификацией на основе cookies

Чтобы вызвать из браузера пользовательский веб-сервис с аутентификацией на основе cookies для платформы **.NET Framework**:

1. Для получения аутентификационных cookie используйте системный веб-сервис `AuthService.svc`.
2. Для вызова пользовательского веб-сервиса используйте строку запроса:

Шаблон адреса пользовательского веб-сервиса с аутентификацией на основе cookies

[Адрес приложения `Creatio/0/rest/[Название пользовательского веб-сервиса]/[Конечная точка пол`

Пример адреса пользовательского веб-сервиса с аутентификацией на основе cookies

`http://mycreatio.com/0/rest/UsrCustomConfigurationService/GetContactIdByName?Name=User1`

Для платформы **.NET Core** процедура вызова пользовательского веб-сервиса с аутентификацией на основе cookies аналогична. Различие — не нужно использовать приставку `/0`.

Вызвать из браузера пользовательский веб-сервис с анонимной аутентификацией

Чтобы вызвать из браузера пользовательский веб-сервис с анонимной аутентификацией для платформы **.NET Framework**, используйте строку запроса:

Шаблон адреса пользовательского веб-сервиса с анонимной аутентификацией

[Адрес приложения `Creatio]/0/ServiceModel/[Название пользовательского веб-сервиса]/[Конечная точ`

Пример адреса пользовательского веб-сервиса с анонимной аутентификацией

```
http://mycreatio.com/0/ServiceModel/UsrCustomConfigurationService.svc/GetContactIdByName?Name=Us
```

Для платформы **.NET Core** процедура вызова пользовательского веб-сервиса с анонимной аутентификацией аналогична. Различие — не нужно использовать приставку `/0`.

Вызвать пользовательский веб-сервис из front-end части

1. В модуль страницы, из которой вызывается веб-сервис, в качестве зависимости подключите модуль `ServiceHelper`. Этот модуль предоставляет удобный интерфейс для выполнения запросов к серверу через провайдер запросов `Terrasoft.AjaxProvider`, реализованный в клиентском ядре.
2. Вызовите пользовательский веб-сервис из модуля `ServiceHelper`.

Способы вызова пользовательского веб-сервиса:

- Вызовите метод `callService(serviceName, serviceMethodName, callback, serviceData, scope)`.
- Вызовите метод `callService(config)`, где `config` — конфигурационный объект со свойствами:
 - `serviceName` — имя пользовательского веб-сервиса.
 - `methodName` — имя вызываемого метода пользовательского сервиса.
 - `callback` — функция обратного вызова, в которой выполняется обработка ответа от веб-сервиса.
 - `data` — объект с проинициализированными входящими параметрами для метода веб-сервиса.
 - `scope` — контекст выполнения запроса.

Важно. Модуль `ServiceHelper` работает только с `POST`-запросами. Поэтому к методам пользовательского веб-сервиса необходимо добавить атрибут `[WebInvoke]` с параметром `Method = "POST"`.

Перенести существующий пользовательский веб-сервис на платформу .NET Core

Пользовательский веб-сервис, который был разработан на платформе .NET Framework и получает контекст без наследования базового класса `Terrasoft.Web.Common.BaseService`, можно перенести на платформу .NET Core. Для переноса необходимо **выполнить адаптацию пользовательского веб-сервиса**.

Свойство `HttpContextAccessor` класса `Terrasoft.Web.Common.BaseService` обеспечивает унифицированный доступ к контексту (`HttpContext`) в .NET Framework и .NET Core. Свойства `UserConnection` и `AppConnection` позволяют получить объект пользовательского подключения и объект подключения на уровне приложения. Это позволяет отказаться от использования свойства `HttpContext.Current` библиотеки `System.Web`.

Пример использования свойств родительского класса `Terrasoft.Web.Common BaseService`

```

namespace Terrasoft.Configuration.UsrCustomNamespace
{
    using Terrasoft.Web.Common;

    [ServiceContract]
    [AspNetCompatibilityRequirements(RequirementsMode = AspNetCompatibilityRequirementsMode.Requ
    public class UsrCustomConfigurationService: BaseService
    {
        /* Метод веб-сервиса. */
        [OperationContract]
        [WebInvoke(Method = "GET", RequestFormat = WebMessageFormat.Json, BodyStyle = WebMessageFormat
        ResponseFormat = WebMessageFormat.Json)]
        public void SomeMethod() {
            ...
            /* UserConnection – свойство BaseService. */
            var currentUser = UserConnection.CurrentUser;
            /* AppConnection – свойство BaseService. */
            var sdkHelpUrl = AppConnection.SdkHelpUrl;
            /* HttpContextAccessor – свойство BaseService. */
            var httpContext = HttpContextAccessor.GetInstance();
            ...
        }
    }
}

```

Для веб-сервиса, который был разработан без наследования класса `Terrasoft.Web.Common.BaseService` реализованы следующие **способы получения контекста**:

- Через `IHttpContextAccessor`, зарегистрированный в `DI` (`ClassFactory`).

Этот способ позволяет покрывать код тестами и отображает явные зависимости класса. Подробнее об использовании фабрики классов можно узнать из статьи [Принцип замещения классов](#).

- Через статическое свойство `HttpContext.Current`.

В исходный код с помощью директивы `using` необходимо добавить пространство имен `Terrasoft.Web.Http.Abstractions`. Статическое свойство `HttpContext.Current` реализует унифицированный доступ к `HttpContext`. Для адаптации кода веб-сервиса для платформы .NET Core замените пространство имен `System.Web` на `Terrasoft.Web.Http.Abstractions`.

Важно. В конфигурации запрещено использовать конкретную реализацию доступа к контексту запроса из .NET Framework (библиотека `System.Web`) или .NET Core (библиотека `Microsoft.AspNetCore.Http`).

Пример адаптации веб-сервиса к платформе .NET Core

```

namespace Terrasoft.Configuration.UsrCustomNamespace
{
    /* Использовать вместо System.Web. */
    using Terrasoft.Web.Http.Abstractions;

    [ServiceContract]
    [AspNetCompatibilityRequirements(RequirementsMode = AspNetCompatibilityRequirementsMode.Req
    public class UsrCustomConfigurationService
    {
        /* Метод веб-сервиса. */
        [OperationContract]
        [WebInvoke(Method = "GET", RequestFormat = WebMessageFormat.Json, BodyStyle = WebMessageF
        ResponseFormat = WebMessageFormat.Json)]
        public void SomeMethod() {
            ...
            var httpContext = HttpContext.Current;
            ...
        }
    }
}

```

Разработать пользовательский веб-сервис с аутентификацией на основе cookies



Средний

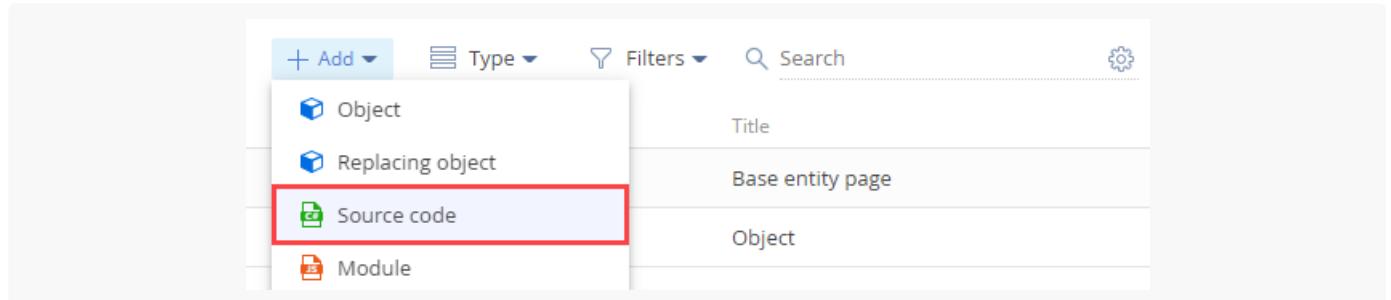
Пример. Создать пользовательский веб-сервис с аутентификацией на основе cookies, который возвращает идентификатор контакта по указанному имени.

- Если контакт найден, то вернуть идентификатор контакта.
- Если найденных контактов несколько, то вернуть идентификатор первого найденного контакта.
- Если контакт найден, то вернуть пустую строку.

1. Создать схему [Исходный код]

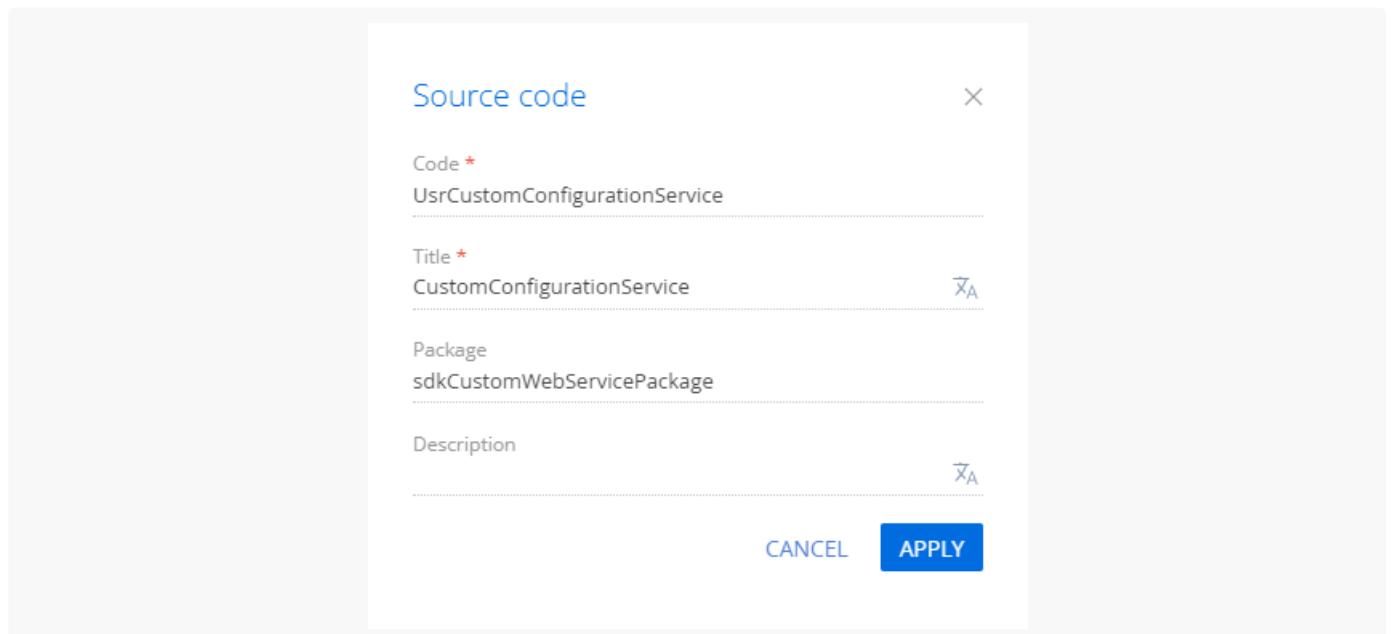
1. [Перейдите в раздел \[Конфигурация \]](#) ([Configuration]) и выберите пользовательский [пакет](#), в который будет добавлена схема.
2. На панели инструментов реестра раздела нажмите [Добавить] —> [Исходный код] ([Add] —>

[Source code]).



3. В дизайнере схем заполните свойства схемы:

- [Код] ([Code]) — "UsrCustomConfigurationService".
- [Заголовок] ([Title]) — "CustomConfigurationService".



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

2. Создать класс сервиса

1. В дизайнере схем добавьте пространство имен, вложенное в `Terrasoft.Configuration`. Название может быть любым, например, `UsrCustomConfigurationServiceNamespace`.
2. С помощью директивы `using` добавьте пространства имен, типы данных которых будут задействованы в классе.
3. Добавьте название класса, которое соответствует названию схемы (свойство [Код] ([Code])).
4. В качестве родительского класса укажите класс `Terrasoft.Nui.ServiceModel.WebService.BaseService`.
5. Для класса добавьте атрибуты `[ServiceContract]` и `[AspNetCompatibilityRequirements(RequirementsMode = AspNetCompatibilityRequirementsMode.Required)]`.

3. Реализовать метод класса

В дизайнере схем добавьте в класс метод `public string GetContactIdByName(string Name)`, который реализует конечную точку пользовательского веб-сервиса. С помощью `EntitySchemaQuery` метод отправит запрос к базе данных. В зависимости от значения параметра `Name`, отправленного в строке запроса, тело ответа на запрос будет содержать:

- Идентификатор контакта (типа строка) — если контакт найден.
- Идентификатор первого найденного контакта (типа строка) — если найдено несколько контактов.
- Пустую строку — если контакт не найден.

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

UsrCustomConfigurationService

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

    [ServiceContract]
    [AspNetCompatibilityRequirements(RequirementsMode = AspNetCompatibilityRequirementsMode.Req
    public class UsrCustomConfigurationService: BaseService
    {

        /* Метод, возвращающий идентификатор контакта по имени контакта. */
        [OperationContract]
        [WebInvoke(Method = "GET", RequestFormat = WebMessageFormat.Json, BodyStyle = WebMessageFormat
        ResponseFormat = WebMessageFormat.Json)]
        public string GetContactIdByName(string Name) {
            /* Результат по умолчанию. */
            var result = "";
            /* Экземпляр EntitySchemaQuery, обращающийся в таблицу Contact базы данных. */
            var esq = new EntitySchemaQuery(UserConnection.EntitySchemaManager, "Contact");
            /* Добавление колонок в запрос. */
            var colId = esq.AddColumn("Id");
            var colName = esq.AddColumn("Name");
            /* Фильтрация данных запроса. */
            var esqFilter = esq.CreateFilterWithParameters(FilterComparisonType.Equal, "Name", N
            esq.Filters.Add(esqFilter);
            /* Получение результата запроса. */
            var entities = esq.GetEntityCollection(UserConnection);
            /* Если данные получены. */
            if (entities != null && entities.Count > 0)
                result = entities[0].Id;
            else
                result = "";
            return result;
        }
    }
}
```

```

if (entities.Count > 0)
{
    /* Возвратить значение колонки "Id" первой записи результата запроса. */
    result = entities[0].GetColumnValue(colId.Name).ToString();
    /* Также можно использовать такой вариант:
    result = entities[0].GetTypedColumnValue<string>(colId.Name); */
}
/* Возвратить результат. */
return result;
}
}
}

```

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

Результат выполнения примера

В результате выполнения примера в Creatio появится пользовательский веб-сервис

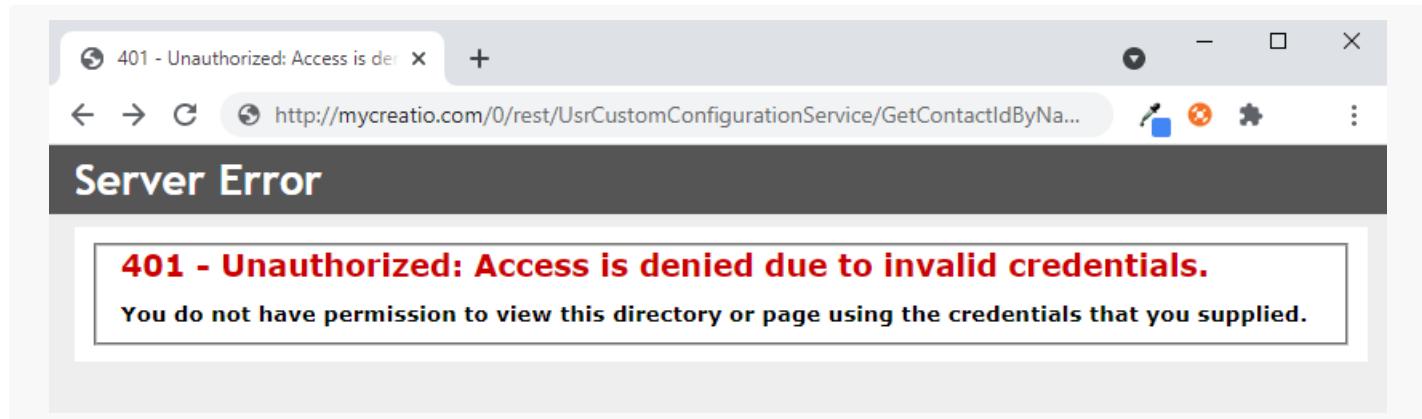
`UsrCustomConfigurationService` типа REST с конечной точкой `GetContactIdByName`.

Из браузера обратитесь к конечной точке `GetContactIdByName` веб-сервиса и в параметре `Name` передайте имя контакта.

Строка запроса с именем существующего контакта

```
http://mycreatio.com/0/rest/UsrCustomConfigurationService/GetContactIdByName?Name=Andrew%20Baker
```

При обращении к веб-сервису без предварительной авторизации возникнет ошибка.



Авторизуйтесь в приложении и выполните запрос еще раз. Если контакт, указанный в параметре `Name`, найден в базе данных, то в свойстве `GetContactIdByNameResult` будет возвращено значение идентификатора контакта.

The screenshot shows a browser window with the URL <http://mycreatio.com/0/rest/UsrCustomConfigurationService/GetContactIdByName?Name=Andrew%20Bake>. The response body contains the JSON object: {"GetContactIdByNameResult": "c4ed336c-3e9b-40fe-8b82-5632476472b4"}

Если контакт, указанный в параметре `Name`, не найден в базе данных, то в свойстве `GetContactIdByNameResult` будет возвращена пустая строка.

Строка запроса с именем несуществующего контакта

`http://mycreatio.com/0/rest/UsrCustomConfigurationService/GetContactIdByName?Name=Andrew%20Bake`

The screenshot shows a browser window with the URL http://mycreatio.com/SalesEnterpriseENU_MKysla/0/rest/UsrCustomConfig.... The response body contains the JSON object: {"GetContactIdByNameResult": ""}

Разработать пользовательский веб-сервис с анонимной аутентификацией

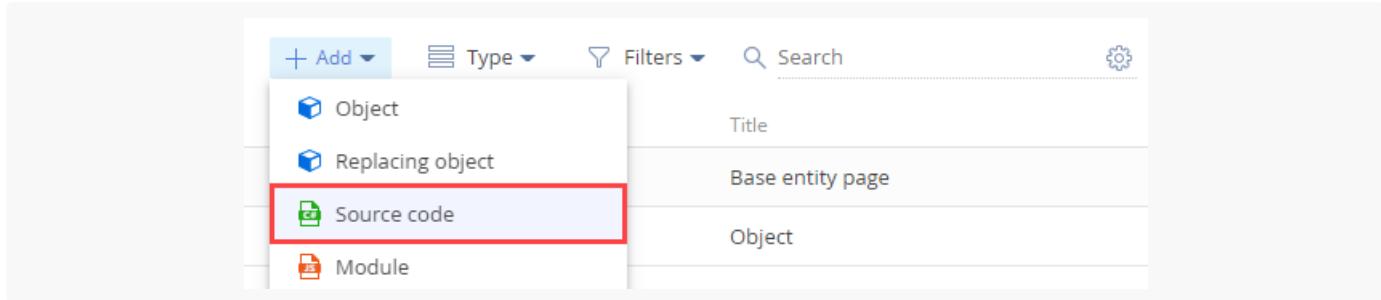


Пример. Создать пользовательский веб-сервис с анонимной аутентификацией, который возвращает идентификатор контакта по указанному имени.

- Если контакт найден, то вернуть идентификатор контакта.
- Если найденных контактов несколько, то вернуть идентификатор первого найденного контакта.
- Если контакт найден, то вернуть пустую строку.

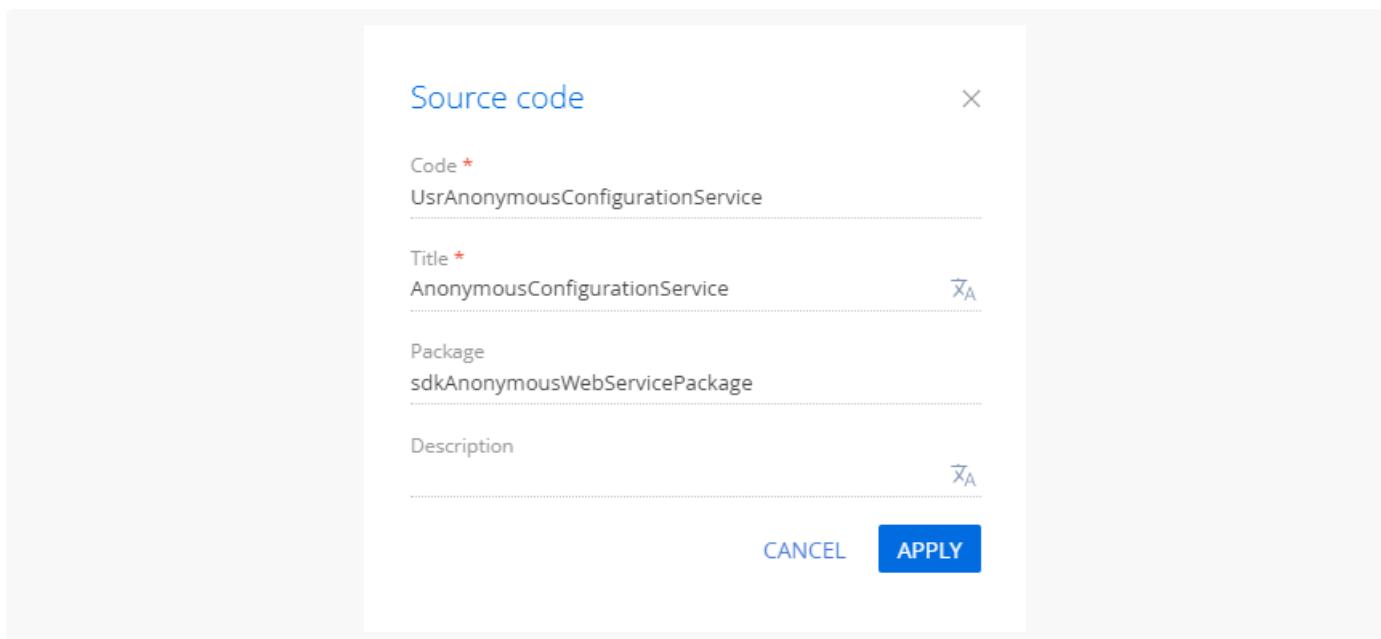
1. Создать схему [Исходный код]

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]) — "UsrAnonymousConfigurationService".
- [Заголовок] ([Title]) — "AnonymousConfigurationService".



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

2. Создать класс сервиса

1. В дизайнере схем добавьте пространство имен, вложенное в `Terrasoft.Configuration`. Название может быть любым, например, `UsrAnonymousConfigurationNamespace`.
2. С помощью директивы `using` добавьте пространства имен, типы данных которых будут задействованы в классе.
3. Добавьте название класса, которое соответствует названию схемы (свойство [Код] ([Code])).
4. В качестве родительского класса укажите класс `Terrasoft.Nui.ServiceModel.WebService.BaseService`.
5. Для класса добавьте атрибуты `[ServiceContract]` и `[AspNetCompatibilityRequirements(RequirementsMode = AspNetCompatibilityRequirementsMode.Required)]`.
6. Для анонимного доступа к пользовательскому веб-сервису добавьте системное подключение `SystemUserConnection`.

3. Реализовать метод класса

В дизайнере схем добавьте в класс метод `public string GetContactIdByName(string Name)`, который реализует конечную точку пользовательского веб-сервиса. С помощью `EntitySchemaQuery` метод отправит запрос к базе данных. В зависимости от значения параметра `Name`, отправленного в строке запроса, тело ответа на запрос будет содержать:

- Идентификатор контакта (типа строка) — если контакт найден.
- Идентификатор первого найденного контакта (типа строка) — если найдено несколько контактов.
- Пустую строку — если контакт не найден.

Укажите пользователя, от имени которого будет выполняться обработка данного http-запроса. Для этого после получения `SystemUserConnection` вызовите метод `SessionHelper.SpecifyWebOperationIdentity` пространства имен `Terrasoft.Web.Common`. Этот метод обеспечивает работоспособность бизнес-процессов при работе с сущностью (`Entity`) базы данных из пользовательского веб-сервиса с анонимной аутентификацией.

```
Terrasoft.Web.Common.SessionHelper.SpecifyWebOperationIdentity(HttpContextAccessor.GetInstance())
```

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

UsrAnonymousConfigurationService

```
/* Пользовательское пространство имен. */
namespace Terrasoft.Configuration.UsrAnonymousConfigurationServiceNamespace
{
    using System;
    using System.ServiceModel;
    using System.ServiceModel.Web;
    using System.ServiceModel.Activation;
    using Terrasoft.Core;
    using Terrasoft.Web.Common;
    using Terrasoft.Core.Entities;

    [ServiceContract]
    [AspNetCompatibilityRequirements(RequirementsMode = AspNetCompatibilityRequirementsMode.Req
    public class UsrAnonymousConfigurationService: BaseService
    {
        /* Ссылка на экземпляр UserConnection, требуемый для обращения к базе данных. */
        private SystemUserConnection _systemUserConnection;
        private SystemUserConnection SystemUserConnection {
            get {
                return _systemUserConnection ?? (_systemUserConnection = (SystemUserConnection)(A
            }
        }
    }
}
```

```

/* Метод, возвращающий идентификатор контакта по имени контакта. */
[OperationContract]
[WebInvoke(Method = "GET", RequestFormat = WebMessageFormat.Json, BodyStyle = WebMessageFormat.Boundary)]
public string GetContactIdByName(string Name){
    /* Указывается пользователь, от имени которого выполняется обработка данного http-запроса.
    SessionHelper.SpecifyWebOperationIdentity(HttpContextAccessor.GetInstance(), SystemUserConnection);
    /* Результат по умолчанию.*/
    var result = "";
    /* Экземпляр EntitySchemaQuery, обращающийся в таблицу Contact базы данных.*/
    var esq = new EntitySchemaQuery(SystemUserConnection.EntitySchemaManager, "Contact");
    /* Добавление колонок в запрос.*/
    var colId = esq.AddColumn("Id");
    var colName = esq.AddColumn("Name");
    /* Фильтрация данных запроса.*/
    var esqFilter = esq.CreateFilterWithParameters(FilterComparisonType.Equal, "Name", Name);
    esq.Filters.Add(esqFilter);
    /* Получение результата запроса.*/
    var entities = esq.GetEntityCollection(SystemUserConnection);
    /* Если данные получены.*/
    if (entities.Count > 0)
    {
        /* Возвратить значение колонки "Id" первой записи результата запроса.*/
        result = entities[0].GetColumnValue(colId.Name).ToString();
        /* Также можно использовать такой вариант:
        result = entities[0].GetTypedColumnValue<string>(colId.Name); */
    }
    /* Возвратить результат.*/
    return result;
}
}
}

```

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

4. Зарегистрировать пользовательский веб-сервис с анонимной аутентификацией

- Перейдите в каталог ..\Terrasoft.WebApp\ServiceModel .
- Создайте файл `UsrAnonymousConfigurationService.svc` и добавьте в него запись.

```

<% @ServiceHost
    Service = "Terrasoft.Configuration.UsrAnonymousConfigurationServiceNamespace.UsrAnonymousConfigurationService"
    Debug = "true"
    Language = "C#"

```

```
%>
```

Атрибут `Service` содержит полное имя класса веб-сервиса с указанием пространства имен.

5. Настроить пользовательский веб-сервис с анонимной аутентификацией для работы по протоколам http и https

- Откройте файл `..\Terrasoft.WebApp\ServiceModel\http\services.config` и добавьте в него запись.

Файл `..\Terrasoft.WebApp\ServiceModel\http\services.config`

```
<services>
    ...
    <service name="Terrasoft.Configuration.UsrAnonymousConfigurationServiceNamespace.UsrAnonymous">
        <endpoint name="[Service name]EndPoint"
            address=""
            binding="webHttpBinding"
            behaviorConfiguration="RestServiceBehavior"
            bindingNamespace="http://Terrasoft.WebApp.ServiceModel"
            contract="Terrasoft.Configuration.UsrAnonymousConfigurationServiceNamespace.UsrAnonymous" />
    </service>
</services>
```

- Аналогичную запись добавьте в файл `..\Terrasoft.WebApp\ServiceModel\https\services.config`.

6. Настроить доступ к пользовательскому веб-сервису с анонимной аутентификацией для всех пользователей

- Откройте файл `..\Terrasoft.WebApp\Web.config`.
- Добавьте элемент `<location>`, определяющий относительный путь и права доступа к веб-сервису.

Файл `..\Terrasoft.WebApp\Web.config`

```
<configuration>
    ...
    <location path="ServiceModel/UsrAnonymousConfigurationService.svc">
        <system.web>
            <authorization>
                <allow users="*" />
            </authorization>
        </system.web>
    </location>
    ...

```

```
</configuration>
```

3. В атрибут `value` ключа `AllowedLocations` элемента `<appSettings>` добавьте относительный путь к веб-сервису.

Файл ..\Terrasoft.WebApp\Web.config

```
<configuration>
  ...
  <appSettings>
    ...
      <add key="AllowedLocations" value="[Предыдущие значения];ServiceModel/UsrAnonymousCon...
    ...
  </appSettings>
  ...
</configuration>
```

7. Перезапустить приложение в IIS

Для применения изменений перезапустите приложение в IIS.

Результат выполнения примера

В результате выполнения примера в Creatio появится пользовательский веб-сервис `UsrAnonymousConfigurationService` С конечной точкой `GetContactIdByName`. К веб-сервису можно обращаться из браузера, как с предварительным вводом логина и пароля, так и без их использования.

Из браузера обратитесь к конечной точке `GetContactIdByName` веб-сервиса и в параметре `Name` передайте имя контакта.

Строка запроса с именем существующего контакта

```
http://mycreatio.com/0/ServiceModel/UsrAnonymousConfigurationService/GetContactIdByName?Name=Anc...
```

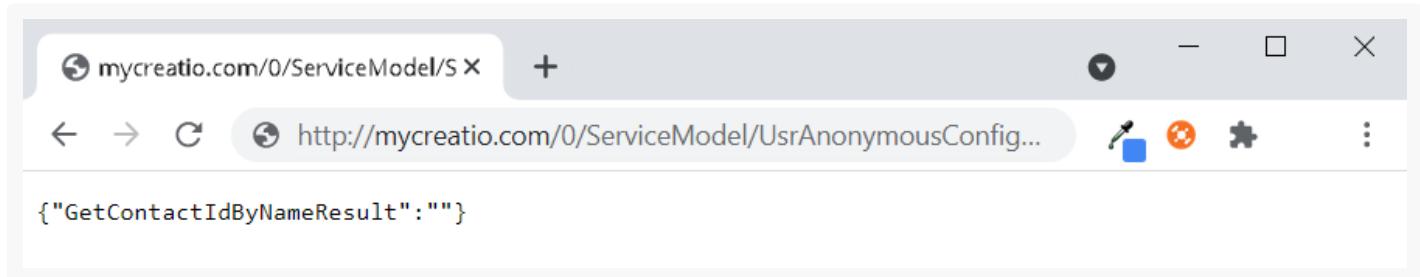
Если контакт, указанный в параметре `Name`, найден в базе данных, то в свойстве `GetContactIdByNameResult` будет возвращено значение идентификатора контакта.



Если контакт, указанный в параметре `Name`, не найден в базе данных, то в свойстве `GetContactIdByNameResult` будет возвращена пустая строка.

Строка запроса с именем несуществующего контакта

```
http://mycreatio.com/0/ServiceModel/UsrAnonymousConfigurationService/GetContactIdByName?Name=An...
```



Вызвать пользовательский веб-сервис из front-end части



Пример. На страницу добавления нового контакта добавить кнопку, по клику на которую будет вызываться пользовательский веб-сервис. В информационном окне страницы отобразить результат, возвращаемый методом веб-сервиса.

1. Создать пользовательский веб-сервис

В примере используется пользовательский веб-сервис `UsrCustomConfigurationService`, разработка которого описана в статье [Разработать пользовательский веб-сервис с аутентификацией на основе cookies](#).

В веб-сервисе `UsrCustomConfigurationService` измените параметр `Method` атрибута `WebInvoke` на `POST`.

Исходный код пользовательского веб-сервиса, используемого в примере, представлен ниже.

UsrCustomConfigurationService

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

```

using Terrasoft.Web.Common;
using Terrasoft.Core.Entities;

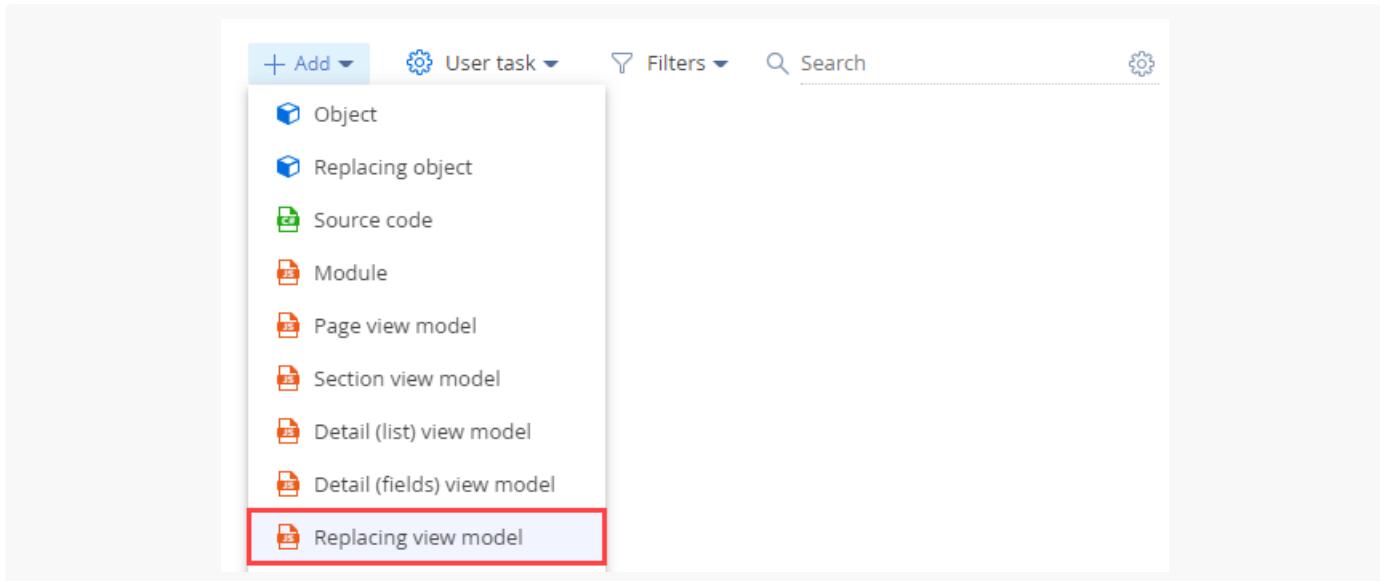
[ServiceContract]
[AspNetCompatibilityRequirements(RequirementsMode = AspNetCompatibilityRequirementsMode.Requ
public class UsrCustomConfigurationService: BaseService
{

    /* Метод, возвращающий идентификатор контакта по имени контакта. */
    [OperationContract]
    [WebInvoke(Method = "POST", RequestFormat = WebMessageFormat.Json, BodyStyle = WebMessag
    ResponseFormat = WebMessageFormat.Json)]
    public string GetContactIdByName(string Name) {
        /* Результат по умолчанию. */
        var result = "";
        /* Экземпляр EntitySchemaQuery, обращающийся в таблицу Contact базы данных. */
        var esq = new EntitySchemaQuery(UserConnection.EntitySchemaManager, "Contact");
        /* Добавление колонок в запрос. */
        var colId = esq.AddColumn("Id");
        var colName = esq.AddColumn("Name");
        /* Фильтрация данных запроса. */
        var esqFilter = esq.CreateFilterWithParameters(FilterComparisonType.Equal, "Name", N
        esq.Filters.Add(esqFilter);
        /* Получение результата запроса. */
        var entities = esq.GetEntityCollection(UserConnection);
        /* Если данные получены. */
        if (entities.Count > 0)
        {
            /* Возвратить значение колонки "Id" первой записи результата запроса. */
            result = entities[0].GetColumnValue(colId.Name).ToString();
            /* Также можно использовать такой вариант:
            result = entities[0].GetTypedColumnValue<string>(colId.Name); */
        }
        /* Возвратить результат. */
        return result;
    }
}
}

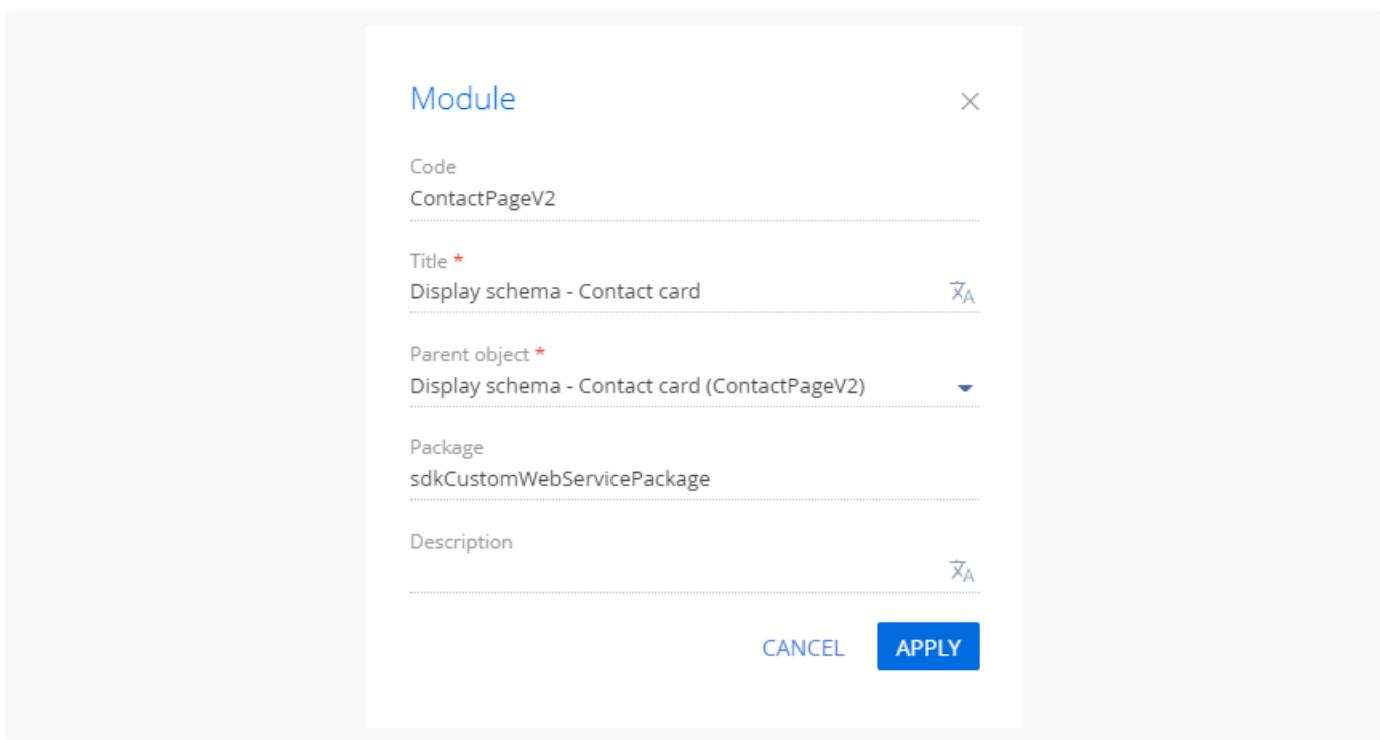
```

2. Создать замещающую страницу записи контакта

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 свойстве [Родительский объект] ([Parent object]) выберите схему модели представления [Схема отображения карточки контакта] ([Display schema — Contact card]) пакета `ContactPageV2`, которую необходимо заменить. После подтверждения выбранного родительского объекта остальные свойства будут заполнены автоматически.



4. В объявлении модуля страницы записи в качестве зависимости подключите модуль `ServiceHelper`. Зависимости модуля описаны в статье [Функция define\(\)](#).

3. Добавить кнопку на страницу записи контакта

1. В блоке [Локализуемые строки] ([Localizable strings]) панели свойств нажмите кнопку и заполните **свойства локализуемой строки**:

- [Код] ([Code]) — "GetServiceInfoButtonCaption".
- [Значение] ([Value]) — "Вызвать сервис" ("Call service").

2. Добавьте обработчик кнопки.

Для вызова веб-сервиса воспользуйтесь методом `callService()` модуля `ServiceHelper`. **Параметры функции** `callService()`, которые необходимо передать:

- `UsrCustomConfigurationService` — имя класса пользовательского веб-сервиса.
- `GetContactIdByName` — имя вызываемого метода пользовательского веб-сервиса.
- Функцию обратного вызова, в которой выполните обработку результатов сервиса.
- `serviceData` — объект с проинициализированными входящими параметрами для метода пользовательского веб-сервиса.
- Контекст выполнения.

Исходный код схемы замещающей модели представления `ContactPageV2` представлен ниже.

ContactPageV2

```
define("ContactPageV2", ["ServiceHelper"],
function(ServiceHelper) {
    return {
        /* Название схемы объекта страницы записи. */
        entitySchemaName: "Contact",
        details: /**SCHEMA_DETAILS*/{}/**SCHEMA_DETAILS*/,
        /* Методы модели представления страницы записи. */
        methods: {
            /* Проверяет, заполнено ли поле [ФИО] страницы. */
            isContactNameSet: function() {
                return this.get("Name") ? true : false;
            },
            /* Метод-обработчик нажатия кнопки. */
            onGetServiceInfoClick: function() {
                var name = this.get("Name");
                /* Объект, инициализирующий входящие параметры для метода сервиса. */
                var serviceData = {
                    /* Название свойства совпадает с именем входящего параметра метода сервиса */
                    Name: name
                };
                /* Вызов веб-сервиса и обработка результатов. */
                ServiceHelper.callService("UsrCustomConfigurationService", "GetContactIdByName",
                    function(response) {
                        var result = response.GetContactIdByNameResult;
                        this.showInformationDialog(result);
                    }, serviceData, this);
            }
        },
    },
});
```

```

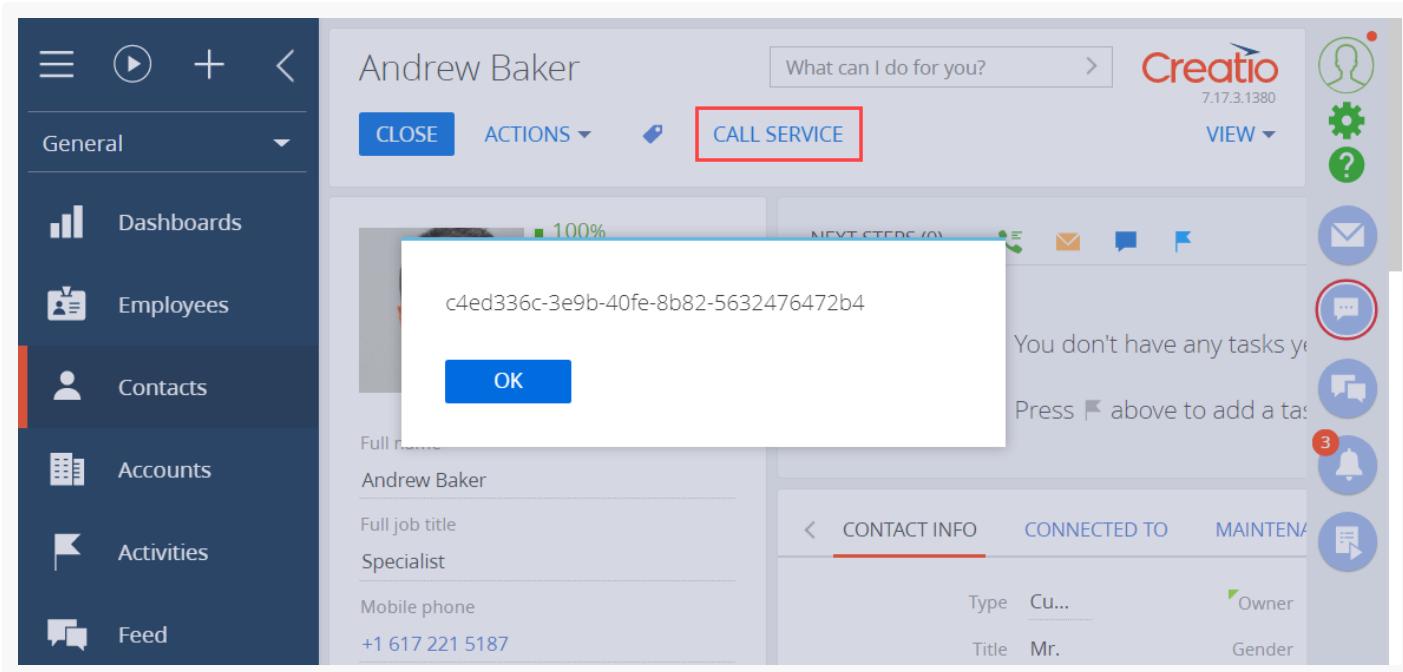
diff: /**SCHEMA_DIFF*/[
    /* Метаданные для добавления на страницу пользовательской кнопки. */
    {
        /* Выполняется операция добавления элемента на страницу. */
        "operation": "insert",
        /* Имя родительского элемента управления, в который добавляется кнопка. */
        "parentName": "LeftContainer",
        /* Кнопка добавляется в коллекцию элементов управления родительского элемента */
        "propertyName": "items",
        /* Имя добавляемой кнопки. */
        "name": "GetServiceInfoButton",
        /* Дополнительные свойства поля. */
        "values": {
            /* Тип добавляемого элемента - кнопка. */
            itemType: Terrasoft.ViewItemType.BUTTON,
            /* Привязка заголовка кнопки к локализуемой строке схемы. */
            caption: {bindTo: "Resources.Strings.GetServiceInfoButtonCaption"},
            /* Привязка метода-обработчика нажатия кнопки. */
            click: {bindTo: "onGetServiceInfoClick"},
            /* Привязка свойства доступности кнопки. */
            enabled: {bindTo: "isContactNameSet"},
            /* Настройка расположения поля. */
            "layout": {"column": 1, "row": 6, "colSpan": 2, "rowSpan": 1}
        }
    }
];
/**SCHEMA_DIFF*/
);
});

```

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

Результат выполнения примера

В результате выполнения примера после обновления страницы приложения на странице контакта появится кнопка [Вызвать сервис] ([Call service]). При нажатии на эту кнопку вызывается метод `GetContactIdByName` пользователя веб-сервиса `UsrCustomConfigurationService`, который возвращает значение идентификатора текущего контакта.



Вызвать пользовательский веб-сервис с помощью Postman

 Средний

Для интеграции внешних приложений с пользовательскими веб-сервисами Creatio необходимо выполнять HTTP-запросы к этим сервисам. Для понимания принципа формирования запросов удобно использовать такие инструменты редактирования и отладки, как [Postman](#) или [Fiddler](#).

Postman — это набор инструментов для тестирования запросов. **Назначение** Postman — тестирование отправки запросов с клиента на сервер и получения ответа от сервера. В этой статье рассмотрен пример вызова пользовательского веб-сервиса с аутентификацией на основе cookies с помощью Postman.

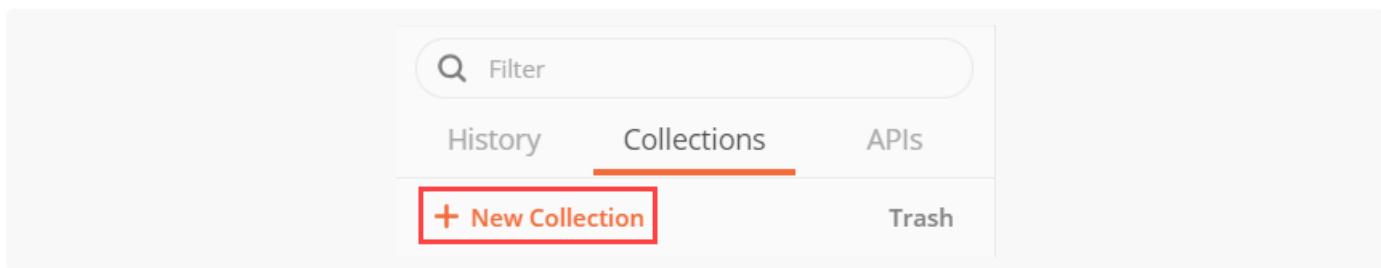
Пример. Вызвать пользовательский веб-сервис с аутентификацией на основе cookies с помощью Postman.

В примере используется пользовательский веб-сервис `UsrCustomConfigurationService`, разработка которого описана в статье [Разработать пользовательский веб-сервис с аутентификацией на основе cookies](#).

Поскольку в примере используется пользовательский веб-сервис с аутентификацией на основе cookies, то предварительно необходимо выполнить аутентификацию в приложении, выполнив запрос к системному веб-сервису `AuthService.svc`. Описание аутентификации содержится в статье [Аутентификация](#).

1. Создать коллекцию запросов

- На панели работы с запросами в Postman перейдите на вкладку [*Collections*] и нажмите [*+ New Collection*].



- Заполните **поля коллекции запросов**:

- [*Name*] — "Test configuration web service".

Окно создания коллекции запросов

CREATE A NEW COLLECTION

Name
Test configuration web service

Description Authorization Pre-request Scripts Tests Variables

This description will show in your collection's documentation, along with the descriptions of its folders and requests.
Make things easier for your teammates with a complete collection description.

Descriptions support [Markdown](#)

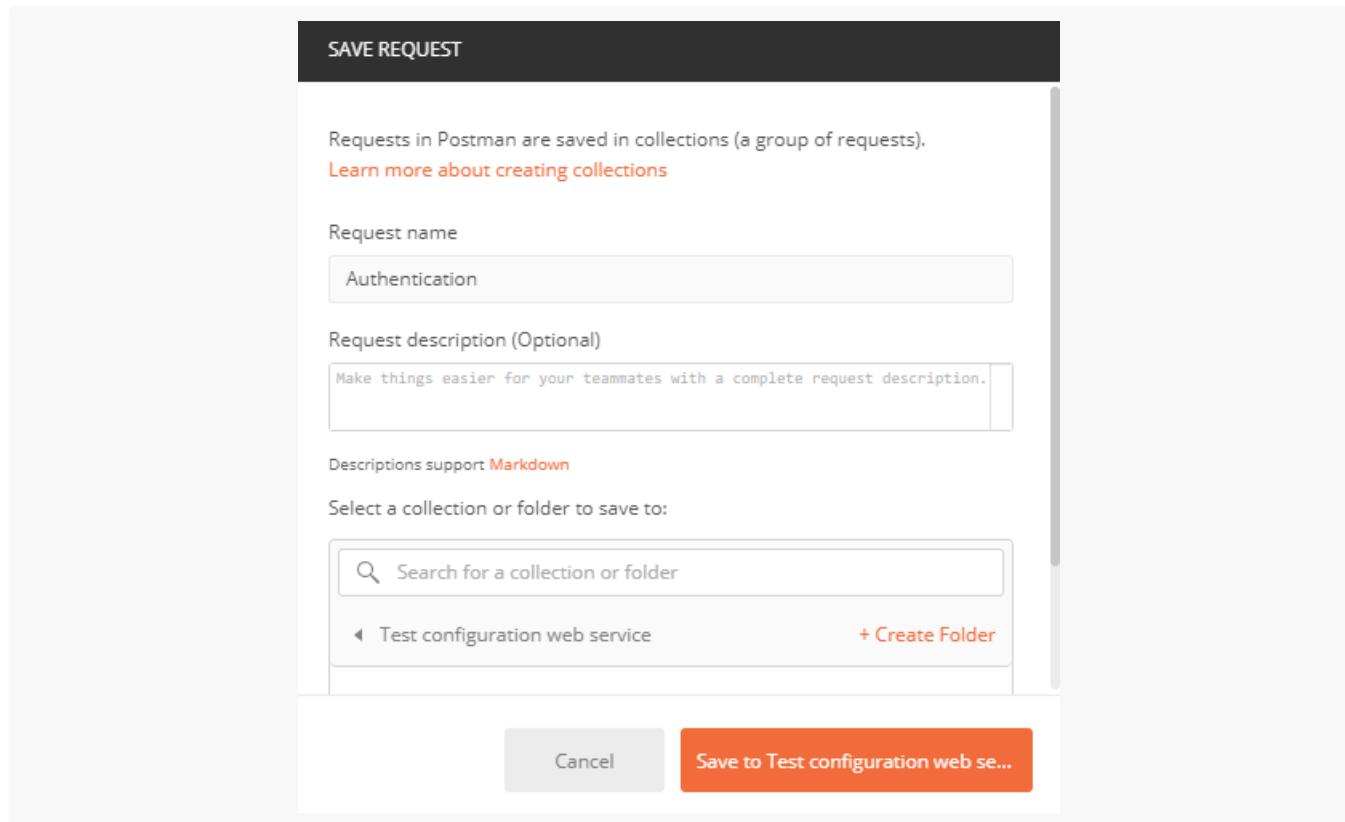
Cancel Create

- Чтобы создать коллекцию запросов, нажмите [*Create*].

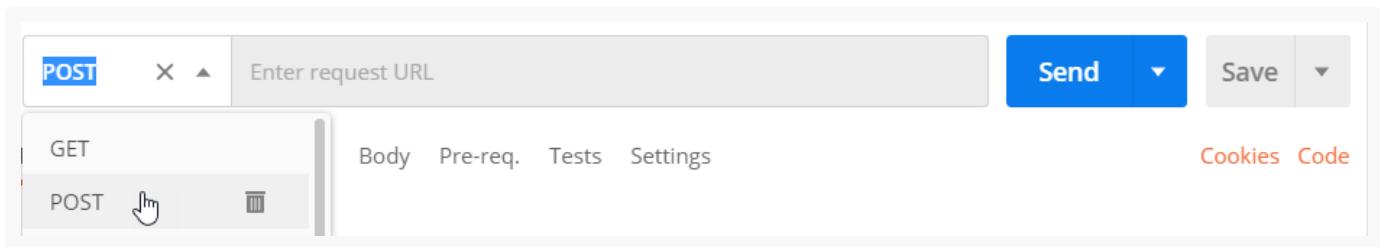
2. Настроить аутентификационный запрос

- На панели работы с запросами в Postman правой кнопкой мыши кликните по имени коллекции `Test configuration web service` —> [*Add request*].
 - Заполните **поля запроса**:
- [*Request name*] — "Authentication".

Окно создания запроса



3. Чтобы добавить запрос в коллекцию, нажмите [*Save to Test configuration web service*].
4. В выпадающем списке панели инструментов рабочей области Postman выберите метод запроса **POST**.



5. В поле запроса панели инструментов рабочей области Postman введите строку запроса к сервису аутентификации.

Шаблон адреса сервиса AuthService.svc
[Адрес приложения Creatio]/ServiceModel/AuthService.svc/Login

Пример адреса сервиса AuthService.svc
<http://mycreatio.com/creatio/ServiceModel/AuthService.svc/Login>

6. Установите **формат данных запроса**:

- Перейдите на вкладку [*Body*].
- Установите опцию "raw".
- Выберите тип "JSON".

The screenshot shows the Postman application interface. At the top, there's a header with a clock icon, a search bar, and several buttons. Below the header, the title 'Untitled Request' is displayed, along with a 'POST' method and an empty 'Enter request URL' field. To the right of the URL field are 'Send' and 'Save' buttons. Below the URL field, there are tabs for 'Params', 'Authorization', 'Headers (8)', 'Body', 'Pre-request Script', 'Tests', and 'Settings'. The 'Body' tab is currently selected. Under the 'Body' tab, there's a sub-tab for 'Query Params' with a table. The table has columns for 'KEY', 'VALUE', and 'DESCRIPTION'. There is one row with 'Key' and 'Value' fields filled, and a 'Description' field below it. Below the table, there's a section titled 'Response' with a placeholder image of a rocket launching. At the bottom of the interface, there are buttons for 'Find and Replace', 'Console', 'Bootcamp', 'Build', 'Browse', and help icons.

- В рабочей области Postman перейдите на вкладку [*Body*] и заполните тело `POST`-запроса — JSON-объект, который содержит аутентификационные данные (логин и пароль).

Тело POST -запроса

```
{
  "UserName": "User01",
  "UserPassword": "User01"
}
```

The screenshot shows the Postman interface with the following details:

- Method:** POST
- URL:** http://mycreatio.com/ServiceModel/AuthService.svc/Login
- Body Tab:** Selected, showing JSON content:


```

1 {
2   "UserName": "User01",
3   "UserPassword": "User01"
4 }
```
- Params, Authorization, Headers (9), Pre-request Script, Tests, Settings, Cookies, Code, Build, Save, Send Buttons:** Standard Postman UI elements.

3. Выполнить аутентификационный запрос

Чтобы выполнить запрос в Postman, на панели инструментов рабочей области нажмите [Send].

В результате выполнения запроса будет получен ответ, который содержит JSON-объект. Тело ответа отображается на вкладке Body в Postman.

The screenshot shows the Postman interface after sending the request, displaying the response body:

```

1 {
2   "Code": 0,
3   "Message": "",
4   "Exception": null,
5   "PasswordChangeUrl": null,
6   "RedirectUrl": null
7 }
```

Other visible sections include Cookies (4), Headers (14), Test Results, and a status bar indicating Status: 200 OK, Time: 79 ms, Size: 1.42 KB.

Признаки успешного выполнения запроса:

- Получен код состояния **200 OK**.
- Параметр **Code** тела ответа содержит значение "0".

Ответ на запрос также содержит cookie **BPMLOADER**, **.ASPXAUTH**, **BPMCSRF** и **UserName**, которые отображаются на вкладке **Cookies**, а также продублированы на вкладке **Headers** в Postman.

The screenshot shows the Postman interface with the 'Cookies' tab selected. There are two entries listed:

Name	Value	Domain	Path	Expires	HttpOnly	Secure
BPMLOADER	4qnu3suiqkotug myqd30hmlw	mycreatio.com	creatio	Session	true	false
.ASPxAuth	EFE15AC92C7B0E DEE4F9EB0BE794 A6809972A5E5B D467C16F3115D B9CDFBAF359078 66243C86839B37 F5994F18435F20	mycreatio.com	creatio	Session	true	false

Below the table, there are buttons for 'Find and Replace', 'Console', 'Bootcamp', 'Build', 'Browse', and help icons.

Эти cookie необходимо использовать в дальнейших запросах к сервисам Creatio, которые используют аутентификацию на основе cookies.

При включенной [зашите от CSRF-атак](#) использование cookie `BPMCSRF` является обязательным. Если cookie `BPMCSRF` не будет использован, сервер вернет код состояния **403 Forbidden**. Для [демо-сайтов Creatio](#) использование cookie `BPMCSRF` необязательно, поскольку защита от CSRF-атак по умолчанию отключена.

Запрос выполняется неуспешно, если была допущена ошибка при построении запроса или тела запроса.

Признаки неуспешного выполнения запроса:

- Параметр `Code` тела ответа содержит значение "1".
- Параметр `Message` тела ответа содержит описание причины неуспешной аутентификации.

The screenshot shows the Postman interface with the 'Test Results' tab selected. The response body is displayed in JSON format:

```

1 {
2   "Code": 1,
3   "Message": "Invalid username or password specified. Verify that you have entered correct data or contact your system administrator. A system administrator can change the password on the user page",
4   "Exception": {
5     "HelpLink": null,
6     "InnerException": null,
7     "Message": "Invalid username or password specified. Verify that you have entered correct data or contact your system administrator. A system administrator can change the password on the user page",
8     "StackTrace": "",
9     "Type": "System.Security.SecurityException"
10 },
11 "PasswordChangeUrl": null,
12 "RedirectUrl": null
13 }

```

Below the JSON, there are buttons for 'Find and Replace', 'Console', 'Bootcamp', 'Build', 'Browse', and help icons.

4. Настроить запрос к пользовательскому веб-сервису с аутентификацией на основе cookies

Пользовательский веб-сервис `UsrCustomConfigurationService` работает с запросами только по методу `GET`.

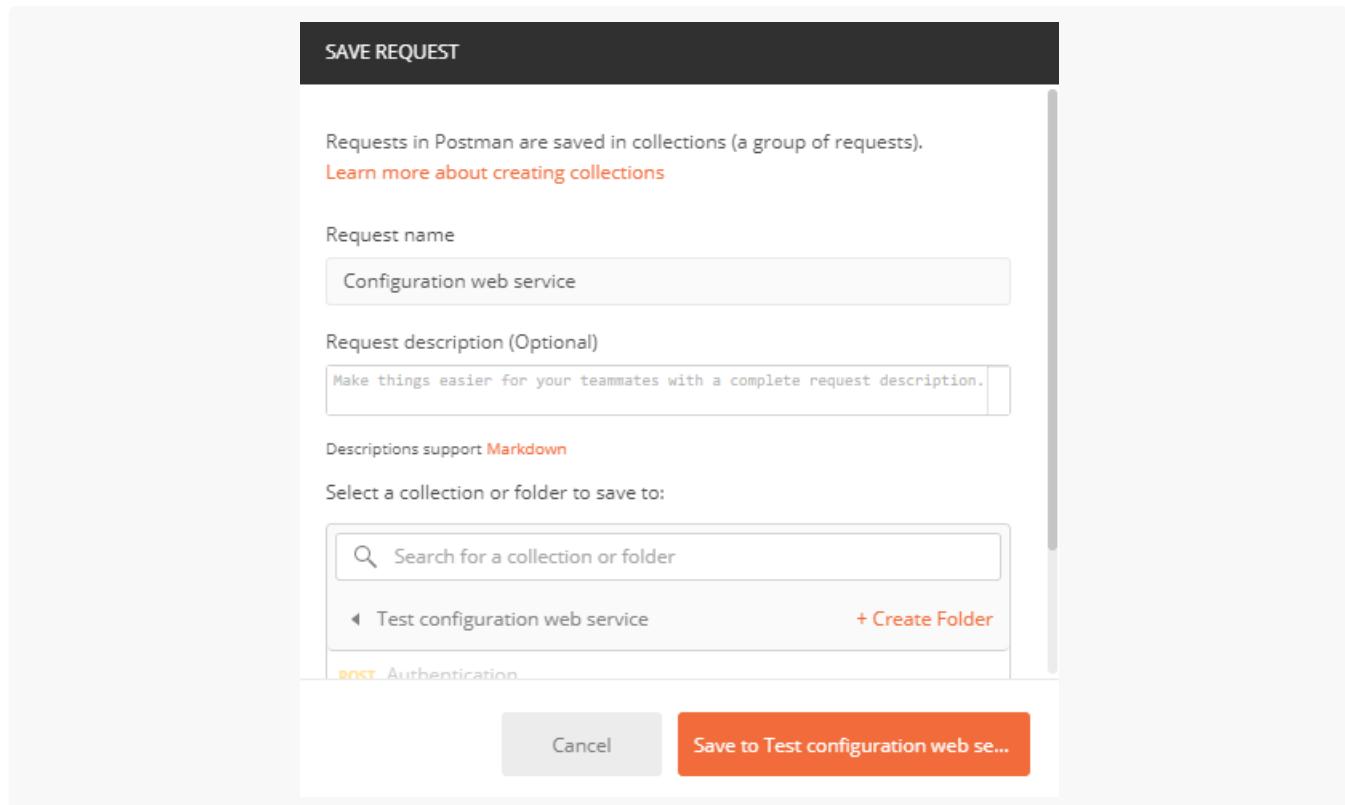
Чтобы **настроить запрос к пользовательскому веб-сервису с аутентификацией на основе cookies**:

1. На панели работы с запросами в Postman правой кнопкой мыши кликните по имени коллекции `Test configuration web service` —> [*Add request*].

2. Заполните **поля запроса**:

- [*Request name*] — "Configuration web service".

Окно создания запроса



3. Чтобы добавить запрос в коллекцию, нажмите [*Save to Test configuration web service*].

4. По умолчанию в Postman выбран метод `GET`. В поле запроса панели инструментов рабочей области Postman введите строку запроса к пользовательскому веб-сервису `UsrCustomConfigurationService`.

Шаблон адреса пользовательского веб-сервиса

[Адрес приложения `Creatio`]/0/rest/UsrCustomConfigurationService/GetContactIdByName?Name=[Имя

Пример адреса пользовательского веб-сервиса

`http://mycreatio.com/creatio/0/rest/UsrCustomConfigurationService/GetContactIdByName?Name=And`

5. В рабочей области Postman перейдите на вкладку [*Headers*] и в заголовки запроса к пользовательскому веб-сервису добавьте cookie, полученные в ответ на авторизационный запрос. В поле [*Key*] добавьте имя cookie, а в поле [*Value*] скопируйте соответствующее значение cookie.

The screenshot shows the Postman interface with the following details:

- Request Type:** GET
- URL:** http://mycreatio.com/creatio/0/rest/UsrCustomConfigurationService/GetContactIdByName?Name=Andrew
- Headers Tab:** The "Headers (11)" tab is selected, showing the following cookie settings:

KEY	VALUE	DESCRIPTION
BPMLOADER	4qnu3suiqkotugmyqd30hmlw	
.ASPxAUTH	B2B9181BF4568C4C6F98960C82CE43290BB2A5C...	
UserName	83%7C117%7C112%7C101%7C114%7C118%7C10...	
BPMCSRF	w4zCxpurXdUff2As2cpk6e	
Key	Value	Description
- Buttons:** Send, Save, Examples, BUILD, Find and Replace, Console, Bootcamp, Build, Browse.

5. Выполнить запрос к пользовательскому веб-сервису с аутентификацией на основе cookies

Чтобы выполнить запрос в Postman, на панели инструментов рабочей области нажмите [Send].

Результат выполнения примера

В результате выполнения запроса будет получен ответ, который содержит JSON-объект. Тело ответа отображается на вкладке Body в Postman.

Если контакт, указанный в параметре Name, найден в базе данных, то в свойстве GetContactIdByNameResult будет возвращено значение идентификатора контакта.

The screenshot shows the Postman interface with a sequence of requests. The first request is a POST to 'Authentication'. The second request is a GET to 'Configuration web service'. The third request is a GET to 'Configuration web service' with the URL 'http://mycreatio.com/creatio/0/rest/UsrCustomConfigurationService/GetContactIdByName?Name=AndrewBaker'. The 'Params' tab is selected, showing a query parameter 'Name' with the value 'Andrew%20Baker'. The 'Body' tab shows the JSON response: "GetContactIdByNameResult": "c4ed336c-3e9b-40fe-8b82-5632476472b4". The status bar at the bottom indicates 'Status: 200 OK'.

Если контакт, указанный в параметре `Name`, не найден в базе данных, то в свойстве `GetContactIdByNameResult` будет возвращена пустая строка.

Сложные Select-запросы



Средний

Некоторые сложные запросы к базе данных могут нагружать ресурсы сервера базы данных на 100% в течение продолжительного времени. Это приводит к затруднению или невозможности работы других пользователей.

Виды сложных запросов:

- Неоптимально составленные запросы в динамических группах, блоках итогов.
- Сложные аналитические выборки в блоках итогов.

Способы выполнения сложных `Select`-запросов:

- С использованием отдельного пула запросов (доступно для Microsoft SQL Server Enterprise Edition).
- С использованием read-only реплики.

Использование вышеуказанных способов выполнения сложных `Select`-запросов позволяет:

- Ограничить ресурсы, которые выделяются сервером базы данных, на обработку сложных `Select`-запросов.
- Уменьшить влияние обработки сложных `Select`-запросов на работу других пользователей и частей системы.

Отдельный пул запросов

Назначение отдельного пула запросов — обработка сложных `Select`-запросов, которые не являются частью транзакции и вынесены в отдельный пул.

1. Настроить соединение отдельного пула запросов

MS SQL Server позволяет ограничивать выделяемые ресурсы с помощью встроенного инструмента — [Resource Governor](#). Ранжирование соединений в Resource Governor базируется на информации о подключении, а не о конкретном запросе. Работу инструмента сложно увидеть на незагруженном сервере и на "коротких" запросах. Эффект от использования Resource Governor наблюдается, когда сервер баз данных загружен на 100%, а сложный запрос выполняется продолжительное время.

Чтобы **настроить соединение отдельного пула запросов**, откройте конфигурационный файл `ConnectionString.config` и для свойства `App` или `Application Name` допишите суффикс `_Limited`. Эта настройка позволяет использовать специальное соединение, которое разделит запросы на простые и потенциально сложные.

Виды соединений:

- Если в строке соединения конфигурационного файла `ConnectionString.config` для свойства `App` **не указано значение**, то для соединений будут использованы значения по умолчанию. Значение по умолчанию для общего соединения — ".Net SqlClient DataProvider", значение по умолчанию для соединения отдельного пула запросов — ".Net SqlClient DataProvider_Limited".
- Если в строке соединения конфигурационного файла `ConnectionString.config` для свойства `App` **указано значение** "creatio", то значение свойства для соединения отдельного пула запросов будет заменено значением "creatio_Limited".

Пример настройки свойства App строки соединения

```
<add name="db" connectionString="App=creatio; Data Source=dbserver\mssql12016; Initial Catalog
```

Таким образом, при загрузке дашбордов или фильтрации разделов с помощью динамических групп приложение создает дополнительные соединения с базой данных. В отличие от основных соединений, эти соединения в названиях содержат суффикс `_Limited`.

Важно. При использовании отдельного пула запросов не выполняется ограничение ресурсов. Использование суффикса `_Limited` позволяет выполнять ранжирование соединений средствами Resource Governor.

2. Включить функциональность отдельного пула запросов

Чтобы **включить функциональность отдельного пула запросов**, в файле

`..\Terrasoft.WebApp\Web.config` установите значение `true` для ключа `UseQueryKinds` элемента `<appSettings>`. Ключ `UseQueryKinds` обеспечивает отправку запросов из дашбордов и динамических групп в соединения, которые в названии содержат суффикс `_Limited`.

```
..\Terrasoft.WebApp\Web.config
```

```
<add key="UseQueryKinds" value="true" />
```

3. Настроить инструмент Resource Governor

Настройка инструмента Resource Governor подразумевает настройку групп и пула.

Чтобы **настроить инструмент Resource Governor**, выполните SQL-скрипт.

Пример настройки групп и пула

```
ALTER RESOURCE POOL poolLimited WITH (
    MAX_CPU_PERCENT = 20,
    MIN_CPU_PERCENT = 0
    -- REQUEST_MAX_MEMORY_GRANT_PERCENT = value
    -- REQUEST_MAX_CPU_TIME_SEC = value
    -- REQUEST_MEMORY_GRANT_TIMEOUT_SEC = value
    -- MAX_DOP = value
    -- GROUP_MAX_REQUESTS = value
);
GO
--- Create a workload group for off-hours processing
--- and configure the relative importance.
CREATE WORKLOAD GROUP groupLimited WITH (IMPORTANCE = LOW) USING poolLimited
GO ALTER RESOURCE GOVERNOR RECONFIGURE;
GO
```

Настройка инструмента подробно описана в официальной [документации Resource Governor](#).

Для каждого нового соединения выполняется функция-классификатор, которая возвращает название группы.

Пример функции-классификатора

```
USE [master]
GO

ALTER FUNCTION [dbo].[fnProtoClassifier]()
RETURNS sysname
WITH SCHEMABINDING
AS
BEGIN
    IF(app_name() like '%_Limited')
    BEGIN
        RETURN N'groupLimited'
    END
END
```

```

END
RETURN N'default'
END;

```

Read-only реплика

Начиная с версии 7.18.4, Creatio позволяет читать данные из read-only реплики. **Назначение** read-only реплики — обработка сложных `Select`-запросов.

Запросы, перенаправление которых позволяет настроить Creatio:

- Пользовательские `SelectQuery`-запросов из интерфейса приложения.
- `Select`-запросы с back-end части приложения, например, в элементе процесса [Задание-сценарий] ([*Script-task*]).

Как и для [отдельного пула запросов](#), на read-only реплику позволяет направлять только `Select`-запросы, которые не являются частью транзакции. Creatio позволяет использовать только одну read-only реплику.

Настройка read-only реплики описана в статье [Ускорить обработку сложных запросов к базе данных](#).

Выполнить сложный Select-запрос

Чтобы **выполнить** `Select`-запрос, получите специальный `DBExecutor`, передав в качестве дополнительного параметра значение `Limited` из перечисления `QueryKind`.

В приведенном ниже примере `QueryKind` — это аргумент метода `EnsureDBConnection()`. Значение аргумента приходит в клиентском `ESQ`-запросе, устанавливается в серверный `ESQ`-запрос и в `Select`-запрос.

Получение `DBExecutor` в зависимости от полученного `QueryKind`

```

using (DBExecutor executor = userConnection.EnsureDBConnection(QueryKind)) {
    /* ... */
};

```

Вызов `EnsureDBConnection(QueryKind.General)` эквивалентен вызову `EnsureDBConnection()` без указания `QueryKind`.

Таким образом, если при создании экземпляра класса `Terrasoft.EntitySchemaQuery` во front-end части приложения установить признак `QueryKind.Limited`, то это значение будет передано на сервер и запросу будет обеспечен специальный `DBExecutor`, который использует отдельный пул запросов.

Пример установки признака `QueryKind.Limited` клиентскому ESQ-запросу в схеме `ChartModule`

```

...
getChartDataESQ: function() {

```

```

        return this.Ext.create("Terrasoft.EntitySchemaQuery", {
            rootSchema: this.entitySchema,
            queryKind: Terrasoft.QueryKind.LIMITED
        });
    },
    ...

```

Важно. Если в программном коде присутствуют вложенные вызовы `userConnection.EnsureDBConnection(QueryKind)`, то необходимо убедиться, что на всех уровнях вложенности используется одно и то же значение `QueryKind`.

Бизнес-логика объектов



Сложный

Способы настройки событий (сохранение, изменение, удаление и т. д.) объекта:

- На уровне интерфейса приложения через событийные подпроцессы дизайнера объекта.
- На уровне back-end части приложения с использованием средств разработки.

Механизм событийного слоя Entity

Назначение событийного слоя `Entity` — настройка обработчиков событий объекта на уровне back-end части приложения с использованием средств разработки. Приложение поддерживает работу только с обработчиками, которые определены в основной конфигурации и сборках файлового контента. Не поддерживаются обработчики из внешних сборок.

Важно. Механизм событийного слоя `Entity` срабатывает после выполнения событийных подпроцессов объекта.

События объекта, которые могут быть обработаны с использованием событийного слоя:

- `OnDeleted` — после удаления записи.
- `OnInserted` — после добавления записи.
- `OnInserting` — перед добавлением записи.
- `OnDeleting` — перед удалением записи.
- `OnSaved` — после сохранения записи.
- `OnSaving` — перед сохранением записи.
- `OnUpdated` — после обновления записи.
- `OnUpdating` — перед обновлением записи.

Составляющие, которые реализуют механизм событийного слоя `Entity`:

- `BaseEntityEventListener` — предоставляет методы-обработчики событий сущности.
- `EntityEventArgs` — предоставляет свойства с аргументами метода-обработчика, который выполняется после возникновения события.
- Класс `EntityBeforeEventArgs` — предоставляет свойства с аргументами метода-обработчика, который выполняется до возникновения события.
- Атрибут `EntityEventListener` — регистрация слушателя.

Класс `BaseEntityEventListener`

Назначение класса `Terrasoft.Core.Entities.Events.BaseEntityEventListener` — предоставляет методы-обработчики различных событий сущности, которые представлены в таблице ниже.

Методы-обработчики событий сущности

Метод-обработчик	Описание метода
<code>OnDeleted(object sender, EntityEventArgs e)</code>	Обработчик события после удаления записи.
<code>OnDeleting(object sender, EntityBeforeEventArgs e)</code>	Обработчик события перед удалением записи.
<code>OnInserted(object sender, EntityEventArgs e)</code>	Обработчик события после добавления записи.
<code>OnInserting(object sender, EntityBeforeEventArgs e)</code>	Обработчик события перед добавлением записи.
<code>OnSaved(object sender, EntityEventArgs e)</code>	Обработчик события после сохранения записи.
<code>OnSaving(object sender, EntityBeforeEventArgs e)</code>	Обработчик события перед сохранением записи.
<code>OnUpdated(object sender, EntityEventArgs e)</code>	Обработчик события после обновления записи.
<code>OnUpdating(object sender, EntityBeforeEventArgs e)</code>	Обработчик события перед обновлением записи.

Параметры методов класса `BaseEntityEventListener`:

- `sender` — ссылка на экземпляр объекта, который генерирует событие.
- `e` — аргументы события. Может принимать значения `EntityEventArgs` (после события) или `EntityBeforeEventArgs` (перед событием).

Последовательность вызова методов-обработчиков событий приведена в таблице ниже.

Последовательность вызова методов-обработчиков

Создание объекта	Изменение объекта	Удаление объекта
OnSaving()	OnSaving()	OnDeleting()
OnInserting()	OnUpdating()	OnDeleted()
OnInserted()	OnUpdated()	
OnSaved()	OnSaved()	

Экземпляр `UserConnection` в обработчиках событий необходимо получать из параметра `sender`. Пример получения `UserConnection` приведен ниже.

Пример получения UserConnection

```
[EntityEventListener(SchemaName = "Activity")]
public class ActivityEntityEventListener : BaseEntityEventListener
{
    public override void OnSaved(object sender, EntityEventArgs e) {
        base.OnSaved(sender, e);
        var entity = (Entity) sender;
        var userConnection = entity.UserConnection;
    }
}
```

Класс EntityEventArgs

Назначение класса `Terrasoft.Core.Entities.EntityEventArgs` — предоставляет свойства с аргументами метода-обработчика, который выполняется после возникновения события.

Свойства класса `EntityEventArgs`:

- `ModifiedColumnValues` — коллекция измененных колонок.
- `PrimaryColumnName` — идентификатор записи.

Класс EntityBeforeEventArgs

Назначение класса `Terrasoft.Core.Entities.EntityBeforeEventArgs` — предоставляет свойства с аргументами метода-обработчика, который выполняется до возникновения события.

Свойства класса `EntityBeforeEventArgs`:

- `KeyValue` — идентификатор записи.
- `IsCanceled` — позволяет отменить дальнейшее выполнение события.

- `AdditionalCondition` — позволяет дополнительно описать условия фильтрации сущности перед действием.

Атрибут EntityEventListener

Назначение атрибута `EntityEventListener` — регистрация слушателя. Слушатель может быть связан со всеми объектами (`IsGlobal = true`) или с конкретным объектом (например, `SchemaName = "Contact"`). Один класс-слушатель можно помечать множеством атрибутов для определения необходимого набора "прослушиваемых" сущностей.

Настроить обработчик события объекта

Чтобы **настроить обработчик** события объекта (наследника класса `Entity`):

1. Создайте класс-наследник класса `BaseEntityEventListener`.
2. Декорируйте класс атрибутом `[EntityEventListener]` с указанием имени сущности, для которой необходимо выполнить подписку событий.
3. Переопределите метод-обработчик настраиваемого события.

Пример переопределения метода-обработчика события

```
/* Слушатель событий сущности "Активность". */
[EntityEventListener(SchemaName = "Activity")]
public class ActivityEntityEventListener : BaseEntityEventListener
{
    /* Переопределение обработчика события сохранения сущности. */
    public override void OnSaved(object sender, EntityEventArgs e) {
        /* Вызов родительской реализации. */
        base.OnSaved(sender, e);
        /* Дополнительные действия.
        ... */
    }
}
```

Асинхронность в событийном слое Entity

Дополнительная бизнес-логика на объекте продолжительна во времени и выполняется последовательно. Это замедляет производительность front-end части приложения, например, при сохранении или изменении сущности. Для решения этой проблемы разработан **механизм асинхронного выполнения операций**, который основан на событийном слое `Entity`.

Составляющие, которые реализуют асинхронность в событийном слое `Entity`:

- Интерфейс `IEntityEventAsyncExecutor` — предоставляет метод для асинхронного выполнения операций.

- Интерфейс `IEntityEventAsyncOperation` — предоставляет метод для запуска асинхронной операции.
- Класс `EntityEventAsyncEventArgs` — экземпляры класса используются в качестве аргумента для передачи в асинхронную операцию.

Интерфейс `IEntityEventAsyncExecutor`

Назначение интерфейса `Terrasoft.Core.Entities.AsyncOperations.Interfaces.IEntityEventAsyncExecutor` — предоставляет метод для асинхронного выполнения операций.

`ExecuteAsync<TOperation>(object parameters)` — типизированный метод для запуска операции с параметрами. `TOperation` — конфигурационный класс, который реализует интерфейс `IEntityEventAsyncOperation`.

Интерфейс `IEntityEventAsyncOperation`

Назначение интерфейса `Terrasoft.Core.Entities.AsyncOperations.Interfaces.IEntityEventAsyncOperation` — предоставляет метод для запуска асинхронной операции.

`Execute(UserConnection userConnection, EntityEventAsyncEventArgs arguments)` — метод для запуска.

Важно. В классе, который реализует интерфейс `IEntityEventAsyncOperation`, не рекомендуется реализовывать логику изменения основной сущности. Это может привести к неверному формированию данных. Также не стоит выполнять легковесные операции (например, подсчет значения поля), поскольку создание отдельного потока может занимать больше времени, чем выполнение самой операции.

Класс `EntityEventAsyncEventArgs`

Назначение класса `Terrasoft.Core.Entities.AsyncOperations.EntityEventAsyncEventArgs` — экземпляры класса используются в качестве аргумента для передачи в асинхронную операцию.

Свойства класса `EntityEventAsyncEventArgs`:

- `EntityId` — идентификатор записи.
- `EntitySchemaName` — название схемы.
- `EntityColumnValues` — словарь текущих значений колонок сущности.
- `OldEntityColumnValues` — словарь старых значений колонок сущности.

Реализовать асинхронность в событийном слое

Чтобы **реализовать асинхронность в событийном слое**:

- Создайте класс, который реализует интерфейс `IEntityEventAsyncOperation`. В нем реализуйте дополнительную логику, которая может выполняться асинхронно.

Пример класса, который реализует дополнительную логику, приведен ниже.

Пример класса, который реализует дополнительную логику

```
/* Класс, который реализует асинхронный вызов операций. */
public class DoSomethingActivityAsyncOperation: IEntityEventAsyncOperation
{
    /* Стартовый метод класса. */
    public void Execute(UserConnection userConnection, EntityEventAsyncOperationArgs argument
        /* ... */
    }
}
```

2. В методе-обработчике слушателя объекта активности используйте [фабрику классов](#).

Назначение фабрики классов:

- Получить экземпляр класса, который реализует интерфейс `IEntityEventAsyncExecutor`.
- Подготовить параметры.
- Передать на выполнение класс, который реализует дополнительную логику.

Пример вызова асинхронной операции в событийном слое приведен ниже.

Пример вызова асинхронной операции в событийном слое

```
[EntityEventListener(SchemaName = "Activity")]
public class ActivityEntityEventListener : BaseEntityEventListener
{
    /* Метод-обработчик события после сохранения сущности. */
    public override void OnSaved(object sender, EntityAfterEventArgs e) {
        base.OnSaved(sender, e);
        /* Экземпляр класса для асинхронного выполнения. */
        var asyncExecutor = ClassFactory.Get<IEntityEventAsyncExecutor>(
            new ConstructorArgument("userConnection", ((Entity)sender).UserConnection));
        /* Параметры для асинхронного выполнения. */
        var operationArgs = new EntityEventAsyncOperationArgs((Entity)sender, e);
        /* Выполнение в асинхронном режиме. */
        asyncExecutor.ExecuteAsync<DoSomethingActivityAsyncOperation>(operationArgs);
    }
}
```

Хранилища данных и кэш



В Creatio реализована поддержка двух видов хранилищ — данные и кэш.

Назначение разделения хранилищ — логическое разделение помещаемой в хранилище информации и упрощение ее дальнейшего использования в программном коде.

Разделение хранилищ позволяет:

- Изолировать данные между рабочими пространствами и между сессиями пользователей.
- Условно классифицировать данные.
- Управлять временем жизни данных.

Физически все хранилища данных и кэша могут располагаться на абстрактном сервере хранения данных. Исключение составляют данные уровня `Request`, которые хранятся непосредственно в памяти.

Сервером хранилищ Creatio является Redis. В общем случае это может быть произвольное хранилище, доступ к которому осуществляется через унифицированные интерфейсы. Необходимо учитывать, что операции обращения к хранилищу являются ресурсоемкими, поскольку связаны с сериализацией / десериализацией данных и сетевым обменом.

Возможности работы с данными, которые предоставляют хранилища Creatio:

- Доступ к данным по ключу для чтения/записи.
- Удаление данных из хранилища по ключу.

Хранилище данных

Назначение хранилища данных — промежуточное хранение редко изменяемых (т. н. "долгосрочных") данных.

Уровни хранилища данных

Уровень	Описание
<code>Request</code>	Уровень запроса. Данные доступны в течение времени обработки текущего запроса. Соответствует значению перечисления <code>Terrasoft.Core.Store.DataLevel.Request</code> .
<code>Session</code>	Уровень сессии. Данные доступны в сессии текущего пользователя. Соответствует значению перечисления <code>Terrasoft.Core.Store.DataLevel.Session</code> .
<code>Application</code>	Уровень приложения. Данные доступны для всего приложения. Соответствует значению перечисления <code>Terrasoft.Core.Store.DataLevel.Application</code> .

Данные, помещаемые в хранилище, будут содержаться в нем до момента их явного удаления.

Ограничения времени жизни объектов:

- Для данных хранилища уровня `Request` время жизни объектов ограничено **временем выполнения запроса**.
- Для данных хранилища уровня `Session` время жизни объектов ограничено **временем существования сессии**.
- Данные хранилища уровня `Application` сохраняются на протяжении всего периода существования приложения и могут быть удалены из него только путем непосредственной **очистки внешнего хранилища**.

Хранилище кэша

Назначение хранилища кэша — хранение оперативной информации.

Уровни хранилища кэша

Уровень	Описание
Session	Уровень сессии. Данные доступны в сессии текущего пользователя. Соответствует значению перечисления <code>Terrasoft.Core.Store.CacheLevel.Session</code> .
Workspace	Уровень рабочего пространства. Данные доступны для всех пользователей одного и того же рабочего пространства. Соответствует значению перечисления <code>Terrasoft.Core.Store.CacheLevel.Workspace</code> .
Application	Уровень приложения. Данные доступны всем пользователям приложения вне зависимости от рабочего пространства. Соответствует значению перечисления <code>Terrasoft.Core.Store.CacheLevel.Application</code> .

Для данных, хранящихся в кэше, существует **время устаревания** — граничный срок актуальности конкретного элемента кэша. Независимо от времени устаревания, все элементы удаляются из кэша при завершении времени их жизни.

Ограничения времени жизни объектов:

- Для данных уровня `Session` объекты удаляются при завершении сессии.
- Для данных уровня `Workspace` объекты удаляются при явном удалении **рабочего пространства**.
- Для данных уровня `Application` объекты сохраняются на протяжении всего периода существования приложения и могут быть удалены из него только путем непосредственной **очистки внешнего хранилища**.

Данные могут быть удалены из кэша в произвольный момент времени. В связи с этим могут возникать ситуации, когда программный код пытается получить закэшированные данные, которые на момент обращения уже удалены из кэша. В этом случае вызывающему коду необходимо просто получить эти данные из постоянного хранилища и поместить их в кэш.

Объектная модель хранилищ

Для работы с хранилищем данных и кэшем в Creatio реализован ряд [классов и интерфейсов](#) пространства имен `Terrasoft.Core.Store`.

Интерфейс IBaseStore

Интерфейс определяет **базовые возможности** всех типов хранилищ и позволяет реализовать:

- Доступ к данным по ключу для чтения/записи (индексатор `this[string key]`).
- Удаление данных из хранилища по заданному ключу (метод `Remove(string key)`).

- Инициализация хранилища заданным перечнем параметров (метод `Initialize(IDictionary parameters)`). Параметры для инициализации хранилищ Creatio вычитываются из конфигурационного файла. Перечни параметров задаются в секциях `storeDataAdapter` (для хранилища данных) и `storeCacheAdapter` (для кэша). В общем случае параметры могут задаваться произвольным образом.

Интерфейс IDataStore

Интерфейс определяет специфику работы с **хранилищами данных**. Является наследником базового интерфейса хранилищ `IBaseStore`. Дополнительно в интерфейсе определена возможность получения списка всех ключей хранилища (свойство `Keys`).

Важно. Свойство `Keys` при работе с хранилищами данных рекомендуется использовать только в исключительных случаях, когда решение задачи альтернативными способами невозможно.

Интерфейс ICacheStore

Интерфейс определяет специфику **работы с кэшем**. Является наследником базового интерфейса хранилищ `IBaseStore`. Дополнительно в интерфейсе определен метод `GetValues(IEnumerable keys)`, который возвращает словарь объектов кэша с заданными ключами. Метод позволяет оптимизировать работу с хранилищем при одновременном получении набора данных.

Класс Store

Статический класс для доступа к кэшу и хранилищам данных различных уровней. Уровни хранилища данных и кэша определены в перечислениях `Terrasoft.Core.Store.DataLevel` и `Terrasoft.Core.Store.CacheLevel` соответственно.

Статические свойства класса `Store`:

- `Data` — возвращает экземпляр провайдера хранилища данных.
- `Cache` — возвращает экземпляр провайдера кэша.

Важно. Для корректной работы хранилищ в .Net Core, необходимо заменить использование статических свойств на работу через `UserConnection` (см. ниже).

Доступ к хранилищам данных и кэшам из UserConnection

Доступ к хранилищам данных и кэшам приложения из конфигурационного кода предоставляют свойства статического класса `Store` пространства имен `Terrasoft.Core.Store`.

Альтернативным вариантом доступа к хранилищу данных и кэшу в конфигурационной логике, является доступ через экземпляр класса `UserConnection`.

Этот вариант позволяет избежать использования длинных имен свойств и подключения дополнительных сборок. Для обеспечения миграции от фреймворка .NET Framework к .Net Core необходимо заменить использование статических свойств на работу через `UserConnection`.

В классе `UserConnection` реализован ряд вспомогательных свойств, позволяющих получить быстрый доступ к хранилищам данных и кэшу различных уровней:

- `ApplicationCache` возвращает ссылку на кэш уровня приложения.
- `WorkspaceCache` возвращает ссылку на кэш уровня рабочего пространства.
- `SessionCache` возвращает ссылку на кэш уровня сессии.
- `RequestData` возвращает ссылку на хранилище данных уровня запроса.
- `SessionData` возвращает ссылку на хранилище данных уровня сессии.
- `ApplicationData` возвращает ссылку на хранилище данных уровня приложения.

Пример работы с кэшем через класс `UserConnection`

```
// Ключ, с которым в кэш будет помещаться значение.
string cacheKey = "SomeKey";

// Помещение значения в кэш уровня сессии через свойство UserConnection.
UserConnection.SessionCache[cacheKey] = "SomeValue";

// Получение значения из кэша через свойство класса Store.
// В результате переменная valueFromCache будет содержать значение "SomeValue".
string valueFromCache = UserConnection.SessionCache[cacheKey] as String;
```

Использование кэша в `EntitySchemaQuery`

В [классе](#) `EntitySchemaQuery` реализован механизм работы с хранилищем (кэшем `Creatio` либо произвольным хранилищем, определенным пользователем). Работа с кэшем позволяет оптимизировать эффективность выполнения операций за счет обращения к закэшированным результатам запроса без дополнительного обращения к базе данных. При выполнении запроса `EntitySchemaQuery` данные, полученные из базы данных, помещаются в кэш, который определяется свойством `Cache` с ключом, который задается свойством `CacheItemName`. По умолчанию в качестве кэша запроса `EntitySchemaQuery` выступает кэш `Creatio` **уровня сессии** с локальным хранением данных. В общем случае в качестве кэша запроса может выступать произвольное хранилище, которое реализует интерфейс `ICacheStore`.

Пример работы с кэшем приложения при выполнении запроса `EntitySchemaQuery`

```
// Создание экземпляра запроса EntitySchemaQuery с корневой схемой City.
var esqResult = new EntitySchemaQuery(UserConnection.EntitySchemaManager, "City");

// Добавление в запрос колонки с наименованием города.
esqResult.AddColumn("Name");

// Определение ключа, под которым в кэше будут храниться результаты выполнения запроса.
// В качестве кэша выступает кэш уровня сессии с локальным кэшированием данных (так как не
```

```
// переопределяется свойство Cache объекта).
esqResult.CacheItemName = "EsqResultItem";

// Выполнение запроса к базе данных для получения результирующей коллекции объектов.
// После выполнения этой операции результаты запроса будут помещены в кэш. При дальнейшем обращении
// esqResult для получения коллекции объектов запроса (если сам запрос не был изменен) эти объекты
// браться из сессионного кэша.
esqResult.GetEntityCollection(UserConnection);
```

Прокси-классы хранилища данных и кэша

Доступ к хранилищам и кэшам в Creatio может быть осуществлен как напрямую (через свойства класса `Store`), так и через прокси-классы.

Прокси-классы — это специальные объекты, которые представляют собой промежуточное звено между хранилищами и вызывающим кодом. **Назначение** прокси-классов — выполнение промежуточных действий над данными перед их чтением/записью в хранилище.

Особенность прокси-классов — каждый из них является хранилищем.

Области применения прокси-классов:

- **Первоначальная настройка и конфигурирование приложения.**

В конфигурационном файле `Web.config`, который находится в корневом каталоге приложения, можно настроить использование прокси-классов для хранилищ данных и кэшей. Настройка прокси-классов для соответствующего хранилища данных или кэша осуществляется в секциях `storeDataAdapters` и `storeCacheAdapters` соответственно. В них добавляется секция `proxies`, в которой перечисляются все прокси-классы, применяемые к хранилищу.

При загрузке приложения настройкичитываются из конфигурационного файла и применяются к соответствующему виду хранилища. Таким образом можно выстраивать **цепочки прокси-классов**, которые будут последовательно выполняться. Порядок прокси-классов в цепочке выполнения соответствует их порядку в конфигурационном файле. При этом первым в цепочке выполнения выступает прокси-класс, перечисленный последним в секции `proxies`, то есть выполнение осуществляется "снизу вверх".

Особенности настройки цепочки прокси-классов:

- Конечной точкой применения цепочки прокси-классов является конкретный кэш или хранилище данных, для которого эта цепочка определяется.
- Каждый прокси-класс работает или с хранилищем данных, или с хранилищем кэша. Вид хранилища определяется с помощью свойств `ICacheStoreProxy.CacheStore` ИЛИ `IDataStoreProxy.DataStore`. Однако, на что именно ссылается это свойство (на другой прокси-класс, на конечное хранилище или кэш), для прокси-класса неизвестно. При этом сам прокси-класс может выступать в качестве хранилища, с которым будут работать другие прокси-классы.

Пример настройки прокси-классов

```
<storeDataAdapters>
    <storeAdapter levelName="Request" type="RequestDataAdapterClassName">
```

```

<proxies>
    <proxy name="requestDataProxyName1" type="requestDataProxyClassName1" />
    <proxy name="requestDataProxyName2" type="requestDataProxyClassName2" />
    <proxy name="requestDataProxyName3" type="requestDataProxyClassName3" />
</proxies>
</storeAdapter>
</storeDataAdapters>

<storeCacheAdapters>
    <storeAdapter levelName="Session" type="SessionCacheAdapterClassName">
        <proxies>
            <proxy name="sessionCacheProxyName1" type="SessionCacheProxyClassName1" />
            <proxy name="sessionCacheProxyName2" type="SessionCacheProxyClassName2" />
        </proxies>
    </storeAdapter>
</storeCacheAdapters>

```

В соответствии с настройками, приведенными в примере, цепочка выполнения прокси-классов, например, хранилища данных, будет следующей:

requestDataProxyName3 → requestDataProxyName2 → requestDataProxyName1 → requestDataAdapterClassName
(конечное хранилище данных уровня запроса).

- **Разграничение данных различных пользователей приложения.**

С помощью прокси-классов решается задача **изоляции данных** между пользователями. Наиболее простым решением этой задачи является трансформация ключей значений перед помещением их в хранилище (например, путем добавления к ключу дополнительного префикса, специфичного для конкретного пользователя). Использование таких прокси-классов обеспечивает уникальность ключей хранилища. Это позволяет избежать потери и искажения данных при одновременной попытке разных пользователей записать в хранилище различные значения с одинаковым ключом. Пример работы с прокси-классами трансформации ключей [приведен ниже](#). В общем случае с помощью прокси-классов можно реализовать сложную логику.

- **Выполнение других промежуточных действий с данными перед помещением их в хранилище.**

В прокси-классах можно реализовывать логику выполнения любых произвольных действий с данными перед помещением их в хранилище или получения их из него. Вынесение логики обработки данных в прокси-класс позволяет избежать дублирования кода, что, в свою очередь, облегчает его модификацию и сопровождение.

Базовые интерфейсы прокси-классов

Чтобы класс можно было **использовать в качестве прокси-класса** для работы с хранилищами, он должен реализовывать один или оба интерфейса пространства имен `Terrasoft.Core.Store`:

- `IDataStoreProxy` — интерфейс прокси-классов хранилища данных.
- `ICacheStoreProxy` — интерфейс прокси-классов кэша.

Каждый из этих интерфейсов имеет одно свойство — ссылку на то хранилище (или кэш), с которым

работает данный прокси-класс. Для интерфейса `IDataStoreProxy` это свойство `DataStore`, а для интерфейса `ICacheStoreProxy` — свойство `CacheStore`.

Прокси-классы трансформации ключей

Прокси-классы, которые **реализуют логику трансформации ключей значений**, помещаемых в хранилища.

- Класс `KeyTransformerProxy` — абстрактный класс, который является базовым классом для всех прокси-классов, преобразующих ключи кэша. Реализует методы и свойства интерфейса `ICacheStoreProxy`. Чтобы избежать дублирования логики, при создании пользовательских прокси-классов для трансформации ключей рекомендуется наследоваться от этого класса.
- Класс `PrefixKeyTransformerProxy` — прокси-класс, преобразующий ключи кэша путем добавления к ним заданного префикса.

Пример работы с кэшем уровня сессии через прокси-класс `PrefixKeyTransformerProxy`

```
// Ключ, с которым значение будет помещаться в кэш через прокси.
string key = "Key";

// Префикс, который будет добавляться к ключу значения прокси-классом.
string prefix = "customPrefix";

// Создание прокси-класса, который будет использоваться для записи значений в кэш уровня сессии.
ICacheStore proxyCache = new PrefixKeyTransformerProxy(prefix, Store.Cache[CacheLevel.Session]

// Запись значения с ключом key в кэш через прокси-класс.
// Фактически это значение записывается в глобальный кэш уровня сессии с ключом prefix + key.
proxyCache[key] = "CachedValue";

// Получение значения по ключу key через прокси.
var valueFromProxyCache = (string)proxyCache[key];

// Получение значения по ключу prefix + key непосредственно из кэша уровня сессии.
var valueFromGlobalCache = (string)UserConnection.SessionCache[prefix + key];
```

В итоге переменные `valueFromProxyCache` и `valueFromGlobalCache` будут содержать одинаковое значение `CachedValue`.

- Класс `DataStoreKeyTransformerProxy` — прокси-класс, преобразующий ключи хранилища данных путем добавления к ним заданного префикса.

Пример работы с хранилищем данных через прокси-класс `DataStoreKeyTransformerProxy`

```
// Ключ, с которым значение будет помещаться в хранилище через прокси.
string key = "Key";
```

```

// Префикс, который будет добавляться к ключу значения прокси-классом.
string prefix = "customPrefix";

// Создание прокси-класса, который будет использоваться для записи значений в хранилище уровня
IDataStore proxyStorage = new DataStoreKeyTransformerProxy(prefix) { DataStore = Store.Data[D

// Запись значения с ключом key в хранилище через прокси-класс.
// Фактически это значение записывается в глобальное хранилище уровня сессии с ключом prefix
proxyStorage[key] = "StoredValue";

// Получение значения по ключу key через прокси.
var valueFromProxyStorage = (string)proxyStorage[key];

// Получение значения по ключу prefix + key непосредственно из хранилища уровня сессии.
var valueFromGlobalStorage = (string)UserConnection.SessionData[prefix + key];

```

В итоге переменные `valueFromProxyStorage` и `valueFromGlobalStorage` будут содержать одинаковое значение `StoredValue`.

Локальное кэширование данных

На базе прокси-классов в Creatio реализован механизм **локального кэширования данных**.

Основная цель кэширования данных — снижение нагрузки на сервер хранения данных и времени выполнения запросов при работе с редко изменяемыми данными.

Логику механизма локального кэширования реализует внутренний прокси-класс `LocalCachingProxy`. Этот прокси-класс выполняет кэширование данных на текущем узле веб-фермы. Класс выполняет мониторинг времени жизни кэшированных объектов и получает данные из глобального кэша только в том случае, если кэшированные данные не актуальны.

Для применения механизма локального кэширования на практике используются **методы расширения** для интерфейса `ICacheStore` из статического класса `CacheStoreUtilities`:

- `WithLocalCaching()` — переопределенный метод, который возвращает экземпляр класса `LocalCachingProxy`, выполняющего локальное кэширование.
- `WithLocalCachingOnly(string)` — метод, выполняющий локальное кэширование данных заданной группы элементов с мониторингом их актуальности.
- `ExpireGroup(string)` — метод устанавливает признак устаревания для заданной группы элементов. При вызове этого метода все элементы заданной группы становятся неактуальными и не возвращаются при запросе данных.

Пример работы с кэшем рабочего пространства с использованием локального кэширова..

```

// Создание первого прокси-класса, который выполняет локальное кэширование данных. Все элементы,
// записываемые в кэш через этот прокси, принадлежат к группе контроля актуальности Group1.
ICacheStore cacheStore1 = Store.Cache[CacheLevel.Workspace].WithLocalCaching("Group1");

```

```
// Добавление элемента в кэш через прокси-класс.
cacheStore1["Key1"] = "Value1";

// Создание второго прокси-класса, который выполняет локальное кэширование данных. Все элементы,
// записываемые в кэш через этот прокси, также принадлежат к группе контроля актуальности Group1
ICacheStore cacheStore2 = Store.Cache[CacheLevel.Workspace].WithLocalCaching("Group1");
cacheStore2["Key2"] = "Value2";

// Для всех элементов группы Group1 устанавливается признак устаревания. Устаревшими считаются эл-
// с ключами Key1 и Key2, так как они принадлежат к одной группе контроля актуальности Group1, н-
// то, что добавлены в кэш через разные прокси.
cacheStore2.ExpireGroup("Group1");

// Попытка получения значений из кэша по ключам Key1 и Key2 после того, как эти элементы были по-
// устаревшие. В результате переменные cachedValue1 и cachedValue2 будут содержать значение null
var cachedValue1 = cacheStore1["Key1"];
var cachedValue2 = cacheStore1["Key2"];
```

Особенности использования хранилищ данных и кэша

Для повышения эффективности работы с хранилищами данных и кэшами в конфигурационной логике и в логике реализации бизнес-процессов необходимо учитывать особенности их использования.

- Все объекты, помещаемые в хранилища, должны быть **сериализуемыми**. Это обусловлено спецификой работы с хранилищами ядра приложения. При сохранении данных в хранилища (за исключением хранилища данных уровня `Request`) объект предварительно **сериализуется**, а при получении — **десериализуется**.
- Обращение к хранилищу является относительно ресурсоемкой операцией. В связи с этим в коде необходимо избегать излишних обращений к хранилищу. В примерах 1 и 2 приводятся корректные и некорректные варианты работы с кэшем.

Пример 1

Неоптимальный код

```
// Выполняется сетевое обращение и десериализация данных.
if (UserConnection.SessionData["SomeKey"] != null)
{
    // Повторно выполняется сетевое обращение и десериализация данных.
    return (string)UserConnection.SessionData["SomeKey"];
}
```

Оптимальный код (вариант 1)

```
// Получение объекта из хранилища в промежуточную переменную.
object value = UserConnection.SessionData["SomeKey"];

// Проверка значения промежуточной переменной.
if (value != null)
{
    // Возврат значения.
    return (string)value;
}
```

Оптимальный код (вариант 2)

```
// Использование расширяющего метода GetValue().
return UserConnection.SessionData.GetValue<string>("SomeKey");
```

Пример 2

Неоптимальный код

```
// Выполняется сетевое обращение и десериализация данных.
if (UserConnection.SessionData["SomeKey"] != null)
{
    // Повторно выполняется сетевое обращение и десериализация данных.
    UserConnection.SessionData.Remove("SomeKey");
}
```

Оптимальный код

```
// Удаление выполняется сразу, без предварительной проверки.
UserConnection.SessionData.Remove("SomeKey");
```

- Любые изменения состояния объекта, полученного из хранилища данных или кэша, происходят локально в памяти и не фиксируются в хранилище автоматически. Поэтому, чтобы эти изменения отобразились в хранилище, измененный объект необходимо явно в него записать.

Пример записи данных в хранилище

```
// Получение словаря значений из хранилища данных сессии по ключу "SomeDictionary".
Dictionary<string, string> dic = (Dictionary<string, string>)UserConnection.SessionData["Some
```

```
// Изменение значения элемента словаря. Изменения в хранилище не зафиксировались.
dic["Key"] = "ChangedValue";

// Добавление нового элемента в словарь. Изменения в хранилище не зафиксировались.
dic.Add("NewKey", "NewValue");

// В хранилище данных по ключу "SomeDictionary" записывается словарь. Теперь все внесенные из
// зафиксированы в хранилище.
UserConnection.SessionData["SomeDictionary"] = dic;
```

Копирование иерархических данных



Сложный

Для копирования записей таблицы базы данных и записей связанных таблиц используется **копирование иерархических данных**. В Creatio иерархическое копирование данных доступно к использованию для копирования таблиц `[ProductHierarchyDataStructure0btainer]` и `[ProductConditionHierarchyDataStructure0btainer]`. Чтобы скопировать данные с других таблиц, необходимо выполнить кастомизацию иерархического копирования данных.

Иерархическое копирование можно использовать в [пользовательском веб-сервисе](#), например, при копировании данных из внешнего сервиса в приложение Creatio или при подключении к внешней базе данных.

Например, есть раздел `[Продукты]` (`[Products]`), который сформирован на основе таблицы `[Product]` базы данных. Страница продукта содержит пользовательские детали. При иерархическом копировании записи раздела будут скопированы и данные записи, и данные связанных деталей.

При копировании учитываются [права доступа](#) к таблицам. Например, пользователь не имеет прав на чтение и добавление записей в таблицу `[contact]`. При вызове копирования иерархических данных запись не будет скопирована и приложение вернет сообщение о невозможности выполнения данной операции.

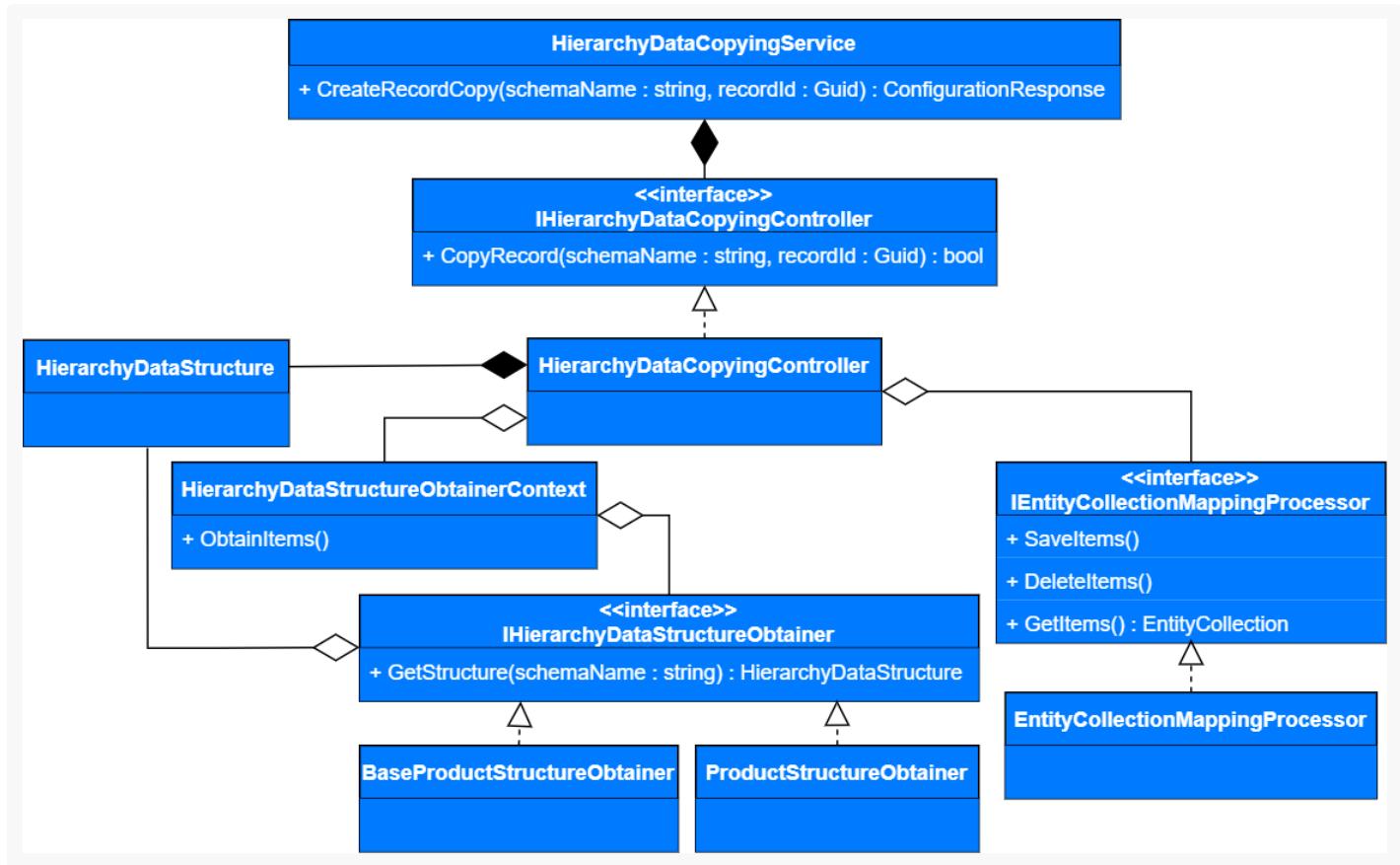
Структура и алгоритм работы копирования иерархических данных

Составляющие копирования иерархических данных представлены в таблице.

Составляющие копирования иерархических данных

Название	Описание	Классы и интерфейсы
Контроллер	Контролирует процесс копирования	IHierarchyDataCopyingController HierarchyDataCopyingController — контроль создания копии записи таблицы и связанных данных из базы данных.
Получатель	Получает из базы данных структуру текущей таблицы и связанных таблиц	HierarchyDataStructureObtainerContext — выбор алгоритма для получения иерархической структуры таблицы и связанных таблиц. IHierarchyDataStructureObtainer HierarchyDataStructureObtainer — получение иерархической структуры таблицы и связанных таблиц. BaseHierarchyDataStructureObtainer ProductHierarchyDataStructureObtainer — получение иерархической структуры таблицы [Product] и связанных таблиц.
Контейнер	Сохраняет структуру текущей таблицы и связанных таблиц	HierarchyDataStructure — контейнер для сохранения информации о структуре иерархических данных.
Мапер	Работает со структурой	IEntityCollectionMappingProcessor EntityCollectionMappingProcessor — получение структуры из базы данных, копирование.

Диаграмма классов копирования иерархических данных представлено на рисунке ниже.



Алгоритм работы копирования иерархических данных:

- Класс сервиса вызывает контроллер процесса копирования и передает в него название таблицы и идентификатор записи, данные которой будут копироваться.
- Контроллер начинает поэтапное создание копии:
 - Получает структуру таблицы и связанных таблиц в унифицированной форме.
 - Сохраняет полученную структуру таблицы в унифицированной форме.
- Контроллер копирует записи в соответствии со структурой, полученной на этапе сохранения.

Унифицированная форма — это сохранение полученной структуры таблицы в объект с типом `HierarchyDataStructure`. Если таблица имеет связанные таблицы (колонка по внешнему ключу ссылается на запись другой таблицы), то они помещаются в созданный объект (в коллекцию из объектов аналогичного типа). При необходимости расширения или обновления механизма получения структуры использование унифицированной формы позволяет обработать новую структуру без дополнительных изменений кода в контроллере.

Шаблон унифицированной формы, которая используется при сохранении полученной структуры таблицы, приведен ниже.

Шаблон унифицированной формы структуры таблицы

```

/* Class holds structure of hierarchical data. */
public class HierarchyDataStructure
  
```

```

{
    public string SchemaName;
    public List<string> Columns;

    /* If current structure object does not have a parent foreign table name than here need to be
    public string ParentColumnName;

    /* List of child structures. */
    public List<HierarchyDataStructure> Structures;

    /* List filters. */
    public HierarchyDataStructureFilterGroup Filters;
}

```

Кастомизировать копирование иерархических данных

Кастомизация копирования иерархических данных позволяет:

- Добавить пользовательскую реализацию получателя данных (класс `HierarchyDataStructureObtainer`).
- Изменить реализацию получателя данных (класс `HierarchyDataStructureObtainer`).
- Изменить реализацию контроллера (класс `HierarchyDataCopyingController`).
- Добавить пользовательскую реализацию копирования иерархических данных.

Добавить пользовательскую реализацию получателя данных

Способы добавления пользовательской реализации получателя данных (класс `HierarchyDataStructureObtainer`):

- Через базовый интерфейс.
- Через наследование базового класса.

Добавить пользовательскую реализацию получателя данных через базовый интерфейс

1. Создайте класс, который реализует интерфейс `IHierarchyDataStructureObtainer`. Шаблон названия класса: `[НазваниеОбъектаКопирования]HierarchyDataStructureObtainer`.
2. Добавьте пользовательскую реализацию метода интерфейса `ObtainStructure()`. Обязательно укажите модификатор `virtual`.

Пример реализации получателя данных через базовый интерфейс содержится в пакете `[ProductBankCustomerJourney]` —> классы `ProductHierarchyDataStructureObtainer` и `ProductConditionHierarchyDataStructureObtainer`.

Добавить пользовательскую реализацию получателя данных через наследование

базового класса

Под базовой реализацией необходимо понимать стандартное копирование записи без связанных записей. Получатель реализован в классе `BaseHierarchyDataStructureObtainer` базового пакета [*NUI*].

1. Создайте класс, который реализует интерфейс `BaseHierarchyDataStructureObtainer` (пакет [*NUI*] —> класс `BaseHierarchyDataStructureObtainer`). Шаблон названия класса: `[НазваниеОбъектаКопирования]HierarchyDataStructureObtainer`.
2. Расширьте базовую реализацию получателя.

Пример реализации получателя данных через наследование базового класса содержится в пакете [*ProductBankCustomerJourney*] —> класс `ProductHierarchyDataStructureObtainer`.

Изменить реализацию получателя данных

1. Создайте класс, который замещает один из классов `BaseHierarchyDataStructureObtainer` (пакет [*NUI*]), `ProductConditionHierarchyDataStructureObtainer` (пакет [*ProductBankCustomerJourney*]), `ProductHierarchyDataStructureObtainer` (пакет [*ProductBankCustomerJourney*]).
2. В замещающий класс добавьте пользовательскую реализацию замещающего метода `ObtainStructure()` базового класса.

Изменить реализацию контроллера

1. Создайте класс, который реализует интерфейс `IHierarchyDataCopyingController`. Шаблон названия класса: `[НазваниеОбъекта]HierarchyDataController`.
2. В метод интерфейса `copyRecord` добавьте пользовательский алгоритм копирования.

Один шаг алгоритма должен содержать вызов одного метода другого класса. Шаг алгоритма также должен включать в себя создание объекта класса, который необходимо вызвать, или минимальную подготовку данных, которые будут передаваться в метод.

Добавить пользовательскую реализацию копирования иерархических данных

1. Добавьте реализацию контроллера процесса копирования (класс `HierarchyDataCopyingController`). Контроллер должен поэтапно вызывать получателя структуры (класс `HierarchyDataStructureObtainer`), обработчика структуры (класс `EntityCollectionMappingProcessor`), контейнера структуры (класс `HierarchyDataStructure`).
2. Добавьте реализацию получателя иерархической структуры данных (класс `HierarchyDataStructureObtainer`).
3. Создайте класс, который реализует интерфейс. Шаблон названия класса: `[НазваниеОбъекта]HierarchyDataProcessor`.
Рекомендуется добавить интерфейс для класса обработчика. Это позволяет добавить другую реализацию и заменить существующую, а также используется для унификации всех обработчиков.
4. Создайте класс, реализующий интерфейс `IEntityCollectionMappingHandler`.

5. В метод контроллера `CopyRecord` добавьте вызовы методов получателя структуры (класс `HierarchyDataStructureObtainer`), обработчика структуры (класс `EntityCollectionMappingProcessor`), контейнера структуры (класс `HierarchyDataStructure`).
6. Создайте в пользовательском классе объект класса `HierarchyDataCopyingController`.

Пример создания объекта контроллера

```
var copyController = ClassFactory.Get<HierarchyDataCopyingController>(new ConstructorArgument
```

7. Вызовите метод копирования `copyController`.

Пример вызова метода копирования

```
copyController.CopyRecord(schemaName, recordId);
```

Вызвать копирование иерархических данных

Копирование иерархических данных можно вызвать из front-end и из back-end части.

Вызвать иерархическое копирование из front-end части

Чтобы **вызвать копирование иерархических данных из front-end части**, используйте метод `callService()`.

Пример вызова содержится в пакете [`ProductBankCustomerJourney`] —> схема `ProductConditionDetailV2` —> метод `callCopyRecordService()`.

Пример вызова сервиса копирования из front-end части

```
/**  
 * Call service that creates records copy.  
 * @protected  
 */  
callCopyRecordService: function() {  
    this.showBodyMask();  
    var config = this.getCopyRecordConfig();  
    this.callService(config, this.copyRecordServiceCallback, this);  
}
```

Вызвать иерархическое копирование из back-end части

Чтобы **вызвать копирование иерархических данных из back-end части**, в пользовательском классе создайте объект класса `HierarchyDataCopyingController`.

Пример вызова сервиса копирования из back-end части

```
var copyController = ClassFactory.Get<HierarchyDataCopyingController>(new ConstructorArgument("L
```

Чтобы **работать с данными таблиц по маппингу колонок**:

1. В пользовательском классе создайте объект класса маппера, который реализует интерфейс `IEntityCollectionMappingHandler`.

Пример создания объекта класса маппера

```
var entityCollectionMappingHandler = ClassFactory.Get<IEntityCollectionMappingHandler>(new Co
```

2. Вызовите методы маппера через объект.

Пример вызова метода копирования

```
entityCollectionMappingHandler.CopyItems(data.SchemaName, columns, filterGroup, relatedColumn
```

На заметку. Создание объекта по названию интерфейса позволяет разработчику заменить существующую реализацию маппера на пользовательскую. При изменении реализации существующего маппера будет перекомпилирован только класс маппера. Все классы, которые используют данную реализацию, не требуют перекомпиляции. Подробнее о создании объектов с использованием механизма внедрения зависимостей описано в статье [Замещение конфигурационных элементов](#).

Чтобы **получить структуру определенной таблицы** в виде объекта с типом `HierarchyDataStructure`:

1. В пользовательском классе создайте объект класса `HierarchyDataStructureObtainerContext`.

Пример создания получателя структуры таблицы

```
var _hierarchyDataStructureObtainer = ClassFactory.Get<HierarchyDataStructureObtainerContext>
```

2. Получите структуру определенной таблицы.

Способы получения структуры:

- Вызовите метод `ObtainStructureByObtainerStrategy` и передайте ему параметр `schemaName` — название таблицы, запись которой необходимо скопировать.
- Вызовите реализацию существующего получателя структуры — `ProductHierarchyDataStructureObtainer` ИЛИ `ProductConditionHierarchyDataStructureObtainer`.

Автоматическая привязка данных



Сложный

Автоматическая привязка данных позволяет в автоматическом режиме создать привязки данных объектов со сложной объектной моделью. В Creatio автоматическая привязка данных доступна к использованию для [каталога продуктов](#) в продуктах линейки Financial Services Creatio.

Структура автоматической привязки данных

Составляющие автоматической привязки данных представлены в таблице.

Составляющие автоматической привязки данных

Название	Назначение	Классы и интерфейсы
Сервис	По заданной структуре запускает генерацию привязок данных. Генерация выполняется в фоновом режиме .	<code>Service</code> — интерфейс для вызова автоматической привязки данных из front-end части.
Контроллер	Обращение к классам <code>Obtainer</code> (получение структуры связанных записей) и <code>Manufacturer</code> (генерация привязок данных).	<code>Controller</code> — класс для управления созданием привязок данных.
Генератор привязок	Является оберткой ядрового класса <code>PackageElementUtilities</code> , представляет собой удобный интерфейс для выполнения CRUD-операций с привязками данных. Операции с данными описаны в статьях Доступ к данным через ORM и Прямой доступ к данным .	<code>Manufacturer</code> — промежуточный класс для взаимодействия между контроллером, классом <code>PackageElementUtilities</code> и классом <code>PackageValidator</code> .
Получатель структуры	Возвращает в контроллер структуру связанных записей для основной сущности.	<code>Obtainer</code> — при формировании привязок данных позволяет гибко изменять структуру связанных объектов для основной сущности. Для определения требуемой реализации использует шаблон <code>Strategy</code> .

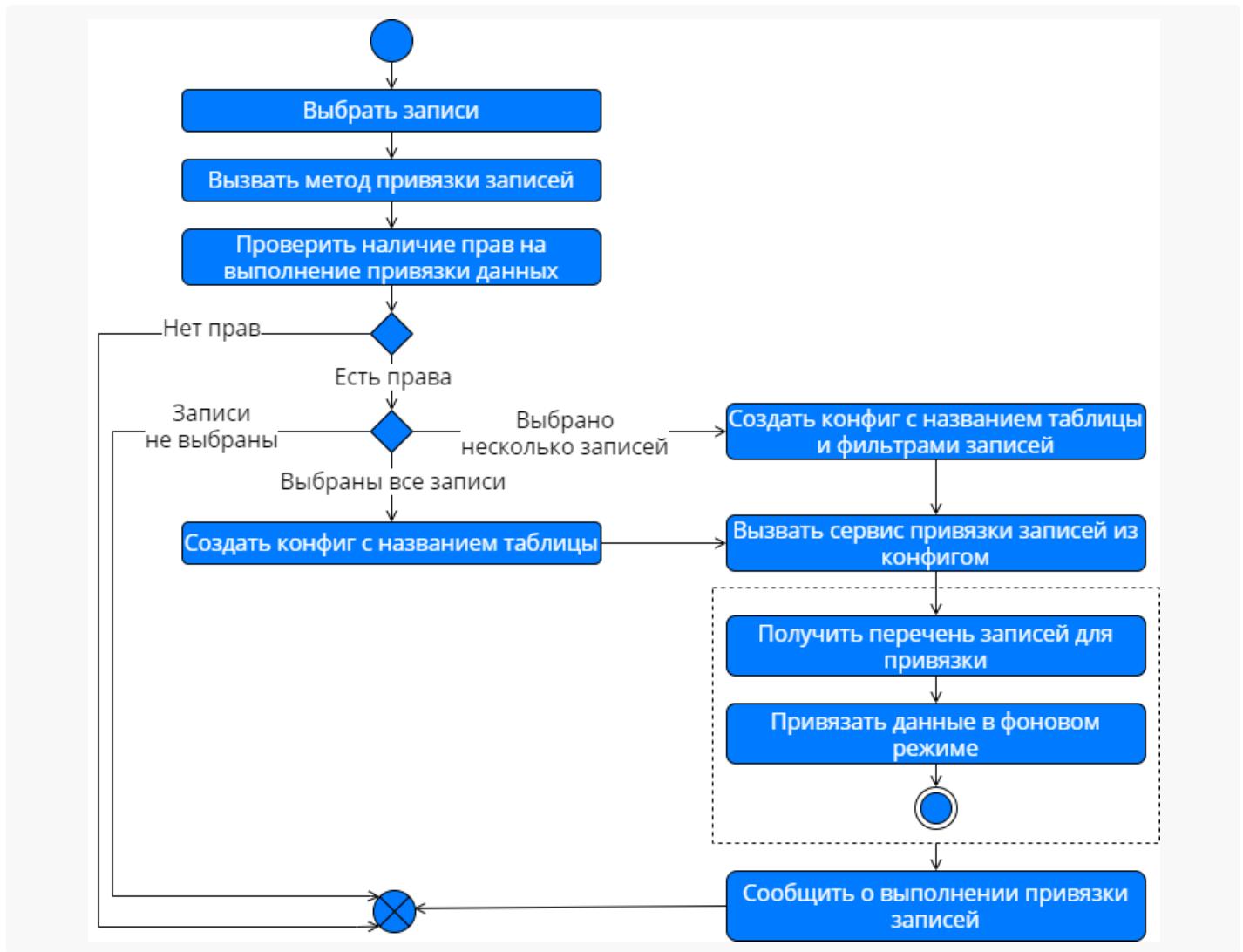
В системе реализована **базовая структура**, в которой возвращается только текущая сущность со всеми своими колонками, кроме системных. Свойства базовой структуры представлены в таблице ниже.

Свойства базовой структуры связанных сущностей

Свойство	Описание
EntitySchemaName	Название сущности.
FilterColumnPathes	Массив путей к первичной колонке основной сущности (при использовании фильтрации выполняется объединение через логический оператор <code>or</code>).
IsBundle	<p>Способ сохранения записей.</p> <p>Способы сохранения записей:</p> <ul style="list-style-type: none"> Сохранение записей в одну привязку. Создание по одной привязке на каждую запись, связанную по фильтру с одной записью основной сущности.
Columns	Перечень колонок для привязки.
InnerStructures	Перечень структур вложенных объектов (например, детали, которые относятся к основной сущности).
DependsStructures	Перечень структур, от которых зависит текущая структура (например, справочники, на которые через справочные поля ссылается запись в текущей структуре).

Алгоритм работы автоматической привязки данных

Алгоритм работы автоматической привязки данных представлен на рисунке ниже.



Обработка структуры начинается с основной сущности и выполняется для каждого экземпляра класса структуры `SchemaDataBindingStructure`.

Алгоритм обработки структуры:

- Рекурсивно обрабатывается каждая структура перечня структур, которые содержатся в свойстве `DependsStructures`. Это выполняется, чтобы при привязке основной сущности все объекты, на которые ссылается текущий, уже были привязаны.
- По фильтру, который содержится в свойстве `FilterColumnPathes`, создается привязка к записи основной сущности.
- Рекурсивно обрабатывается каждая структура из перечня структур, которые содержатся в свойстве `InnerStructures`. Поскольку структуры ссылаются на основную сущность, то их необходимо привязывать после основной сущности.

Способы именования автоматических привязок данных:

- `agb_EntitySchemaName`, если свойство `IsBundle` установлено в значение `true`. `EntitySchemaName` — наименование сущности текущей привязки.
- `agb_EntitySchemaName_PrimaryEntityId`, если свойство `IsBundle` установлено в значение `false`.

`PrimaryEntityId` — идентификатор записи основной сущности, с которой связаны записи в текущей привязке.

Важно. Не рекомендуется переименовывать автоматически созданные привязки, поскольку автоматически обновляются только привязки, имена которых соответствуют указанному шаблону.

Добавить пользовательскую структуру для привязки объекта

1. Создайте схему [*Исходный код*] ([*Source code*]). Для создания схемы воспользуйтесь статьей [Разработка конфигурационных элементов](#).
2. Создайте класс сервиса.
 - a. В дизайнере схем добавьте пространство имен `Terrasoft.Configuration`.
 - b. Создайте класс. Шаблон названия класса: `[НазваниеСущности]SchemaDataBindingStructureObtainer` (например, `ProductSchemaDataBindingStructureObtainer`). По указанному шаблону будет выполняться поиск классов при определении используемой реализации получателя (компонент `Obtainer`).
 - c. В качестве родительского класса укажите класс `BaseSchemaDataBindingStructureObtainer`.
 - d. Добавьте пользовательскую реализацию метода интерфейса `ObtainStructure()`. В этом методе сформируйте и верните необходимую структуру сущностей.

Метод `ObtainStructure()`

```
public override SchemaDataBindingStructure ObtainStructure(UserConnection userConnection)
```

Важно. При необходимости модификации реализации структуры, которая содержится в недоступных для редактирования пакетах, в пользовательском пакете создайте класс и используйте атрибут `[Override]`. Описание атрибута содержится в статье [Атрибут \[Override\]](#).

3. Опубликуйте схему исходного кода.

Вызвать автоматическую привязку данных

Автоматическую привязку данных можно вызвать из front-end или из back-end части.

Вызвать автоматическую привязку данных из front-end части

1. Вызовите сервис генерации привязок.
2. Установите настройки для выполнения генерации привязок.

Способы установки настроек для генерации привязок:

- Передайте параметры.
 - Пакет для генерации привязки.
 - Название сущности.
 - Перечень идентификаторов для привязки.
- Передайте фильтр.

Ниже представлен пример вызова автоматической привязки данных из front-end части с помощью параметров.

Пример вызова автоматической привязки данных из front-end части с помощью параметров

```
ServiceHelper.callService("SchemaDataBindingService",
    "GenerateBindings",
    callback,
    {
        "schemaName": this.entitySchemaName,
        "sysPackageUID": sysPackageUID
        "recordIds": [..., ...]
    },
    this
);
```

Ниже представлен пример вызова автоматической привязки данных из front-end части с помощью фильтра.

Пример вызова автоматической привязки данных из front-end части с помощью фильтра

```
ServiceHelper.callService("SchemaDataBindingService",
    "GenerateBindingsByFilter",
    callback,
    {
        "schemaName": this.entitySchemaName,
        "sysPackageUID": sysPackageUID
        "filterConfig": ...
    },
    this
);
```

Важно. Автоматическая привязка данных, вызванная из front-end части, запускается в фоновом режиме. По выполнению привязки данных на коммуникационной панели отобразится уведомление, которое будет содержать информацию о количестве успешно и неуспешно привязанных записях продуктов.

Вызвать автоматическую привязку данных из back-end части

1. Создайте экземпляр класса `SchemaDataBindingController`.
2. Вызовите метод `GenerateBindings`, в который передайте:

- Идентификатор пакета.
- Название сущности.
- Перечень идентификаторов для привязки.

Ниже представлен пример вызова автоматической привязки данных из back-end части.

Пример вызова автоматической привязки данных из back-end части

```
var dataBindingController = ClassFactory.Get<IDataBindingController>(
    new ConstructorArgument("userConnection", UserConnection));
var result = dataBindingController.GenerateBindings(schemaName, recordIds, sysPackageUiId);
```

API для работы с файлами



Начиная с версии 7.17.2 ядро Creatio предоставляет набор классов и интерфейсов по работе с файлами:

- пространство имен `Terrasoft.File.Abstractions` — интерфейсы и абстрактные классы, описывающие логику работы с файлами в Creatio.
- пространство имен `Terrasoft.File` — конкретные реализации абстракций, использующиеся в системе.

Местоположение файла и файловые локаторы

Местоположение файла в файловом хранилище задается с помощью **файлового локатора** — объекта, который реализует интерфейс `Terrasoft.File.Abstractions.IFileLocator`.

Файловый локатор обязательно содержит **уникальный идентификатор файла** `RecordId`.

Можно создавать разные реализации файловых локаторов для различных файловых хранилищ. В зависимости от специфики хранилища файловый локатор может содержать дополнительные свойства, позволяющие определить место хранения файла. Например, класс `Terrasoft.File.EntityFileLocator` — это реализация файлового локатора для текущего файлового хранилища Creatio [*Файлы и ссылки*] ([*Attachments*]). Объект класса `EntityFileLocator`, кроме свойства `RecordId`, имеет свойство `EntitySchemaName` — имя схемы объекта, в котором хранится файл, например: "ActivityFile" или "CaseFile".

Все методы по работе с файлами оперируют с файловыми локаторами.

Файлы и файловые хранилища

Структура файла в Creatio:

- метаданные файла;
- контент файла.

Метаданные описывают свойства файла:

- имя;
- размер в байтах;
- дата создания и т. п.

Основой для метаданных файла является абстрактный класс

`Terrasoft.File.Abstractions.Metadata.FileMetadata`. Примером его конкретной реализации является класс `Terrasoft.File.Metadata.EntityFileMetadata` для описания метаданных файлов в объекте [Файлы и ссылки] ([Attachments]).

Контент — это непосредственно содержимое файла.

Метаданные файла и его контент хранятся в Creatio в разных **хранилищах**:

- Хранилище метаданных файлов должно реализовывать интерфейс `Terrasoft.File.Abstractions.IFileMetadataStorage`.
- Хранилище контента файла должно реализовывать интерфейс `Terrasoft.File.Abstractions.IFileContentStorage`.

Конкретные реализации этих интерфейсов скрывают в себе нюансы взаимодействия с различными системами хранения файлов: [Файлы и ссылки] ([Attachments]) Creatio, файловая система сервера, Amazon S3, Google Drive и т. п.

Интерфейс `Terrasoft.File.Abstractions.IFile` предоставляет необходимые методы для работы с файлами, хранящимися в любых типах файловых хранилищ. Реализация этого интерфейса означает "файл" в терминах Creatio. Методы в этом интерфейсе обеспечивают асинхронную работу с файлами. Синхронные версии этих методов находятся в расширяющем классе `Terrasoft.File.Abstractions.FileUtils`.

Получить файл (IBuilderFactory)

Интерфейс `Terrasoft.File.Abstractions.IBuilderFactory` предоставляет методы для создания и получения объектов некоторого класса, реализующего интерфейс `Terrasoft.File.Abstractions.IFile`. Этот интерфейс реализует фабрику, доступ к которой обеспечивается через методы класса `Terrasoft.File.FileFactoryUtils`, расширяющего класс `UserConnection`. Соответственно, для успешной работы с файлами необходимо иметь экземпляра `UserConnection` или `SystemUserConnection`.

Место хранения нового или существующего файла однозначно определяется его файловым локатором. Для получения доступа к существующему файлу необходимо знать его уникальный идентификатор `RecordId`. Для нового файла этот идентификатор формируется самостоятельно и передается в метод по созданию локатора.

Реализовать и зарегистрировать новый тип файлового хранилища

Для реализации нового типа файлового хранилища необходимо:

- Создать свою реализацию интерфейса `Terrasoft.File.Abstractions.Content.IFileContentStorage`, который описывает необходимое API для работы с хранилищем файлового контента.
- Если текущее хранилище метаданных файлов (`Terrasoft.File.Metadata.EntityFileMetadataStorage`) по каким-то причинам не подходит, необходимо реализовать собственное хранилище метаданных, свой тип метаданных и свой тип файлового локатора:
 - Хранилище данных должно реализовывать интерфейс `Terrasoft.File.Abstractions.Metadata.IFileMetadataStorage`.
 - Файловый локатор должен реализовывать интерфейс `Terrasoft.File.Abstractions.IFileLocator`.
 - Класс метаданных должен быть наследником абстрактного класса `Terrasoft.File.Abstractions.Metadata.FileMetadata`.
- Новые хранилища контента и метаданных файлов необходимо зарегистрировать в соответствующих справочниках "SysFileContentStorage" и "SysFileMetadataStorage".

Исключения при работе с файлами

Тип исключения	Сообщение	Условия возникновения
<code>Terrasoft.File.Abstractions.FileNotFoundException</code>	File not found by locator '{тип_локатора}' {локатор.ToString}'	При доступе к любому из свойств или методов интерфейса <code>IFile</code> , если метаданные файла не найдены.
<code>System.InvalidOperationException</code>	Can't delete new file: '{тип_локатора}', {локатор.ToString}'	При попытке удаления только что созданного файла: <code>FileMetadata.StoringState == FileStoringState.New</code>
<code>Terrasoft.Common.NullOrEmptyException</code>	File name cannot be null or empty	При попытке сохранения файла с пустым полем <code>Name</code>
<code>System.InvalidOperationException</code>	Can't find a metadata storage for the '{тип_локатора}' locator type	Если подходящее по типу локатора хранилище метаданных файла не найдено
<code>System.InvalidOperationException</code>	Can't find a content storage for the '{тип_метаданных}' metadata type	Если подходящее по типу метаданных хранилище контента файла не найдено.

Настройка активного хранилища

Настройка активного хранилища происходит путем установки значения системной настройки [Активное хранилище содержимого файлов] (код "ActiveFileContentStorage").

Примеры работы с файлами

Средний

Важно. Для корректной работы приведенных ниже примеров кода необходимо подключить пространство имен Terrasoft.Common.

```
using Terrasoft.Common;
```

Пример. Получить экземпляр класса, реализующего интерфейс IFile.

Получение IFile для существующего файла (вариант 1)

```
// Получаем фабрику файлов.
IFileFactory fileFactory = UserConnection.GetFileFactory();
// Создаем файловый локатор для файла с идентификатором recordId, который хранится в таблице БД
// "ActivityFile" ("Файлы и ссылки" для объекта "Activity").
var fileLocator = new EntityFileLocator("ActivityFile", recordId);
// Передаем сформированный локатор в метод Get фабрики.
IFile file = fileFactory.Get(fileLocator);
```

```
// В результате в file получаем экземпляр некоего класса, который реализует интерфейс IFile, через
// можно производить другие манипуляции с файлом и его содержимым.
```

Получение IFile для существующего файла (вариант 2)

```
// Создаем файловый локатор для файла с идентификатором recordId, который хранится в таблице БД
// "ActivityFile" ("Файлы и ссылки" для объекта "Activity").
var fileLocator = new EntityFileLocator("ActivityFile", recordId);
// Передаем сформированный локатор в расширяющий метод GetFile.
IFile file = UserConnection.GetFile(fileLocator);
// В результате в file получаем экземпляр некоего класса, который реализует интерфейс IFile, через
// можно производить другие манипуляции с файлом и его содержимым.
```

Получение IFile для нового файла (вариант 1)

```
// Получаем фабрику файлов.
IFileFactory fileFactory = UserConnection.GetFileFactory();
// Создаем уникальный идентификатор нового файла.
Guid recordId = Guid.NewGuid();
// Создаем файловый локатор для нового файла с идентификатором recordId, который хранится в таблице
// "ActivityFile" ("Файлы и ссылки" для объекта "Activity").
var fileLocator = new EntityFileLocator("ActivityFile", recordId);
// Передаем сформированный локатор в метод Create фабрики.
IFile file = fileFactory.Create(fileLocator);
// В результате в file получаем экземпляр некоего класса, который реализует интерфейс IFile, через
// можно производить другие манипуляции с файлом и его содержимым.
```

Получение IFile для нового файла (вариант 2)

```
// Создаем уникальный идентификатор нового файла.
Guid recordId = Guid.NewGuid();
// Создаем файловый локатор для нового файла с идентификатором recordId, который хранится в таблице
// "ActivityFile" ("Файлы и ссылки" для объекта "Activity").
var fileLocator = new EntityFileLocator("ActivityFile", recordId);
// Передаем сформированный локатор в расширяющий метод CreateFile.
IFile file = UserConnection.CreateFile(fileLocator);
// В результате в file получаем экземпляр некоего класса, который реализует интерфейс IFile, через
// можно производить другие манипуляции с файлом и его содержимым.
```

Пример. Создать новый файл с привязкой к записи раздела [Активности] ([Activities]).

Создание нового файла

```
// Создаем уникальный идентификатор нового файла.
Guid fileId = Guid.NewGuid();
// Создаем файловый локатор для нового файла.
var fileLocator= new EntityFileLocator("ActivityFile", fileId);
// Получаем объект IFile для нового файла.
IFile file = UserConnection.CreateFile(fileLocator);
// В хранилищах еще не сохранены ни метаданные нового файла, ни его контент.
// Задаем имя файлу в его метаданных.
file.Name = "New file";
// Устанавливаем новому файлу атрибут: привязываем этот файл к записи Активности с ключом activityId.
// Это тоже метаданные файла.
file.SetAttribute("ActivityId", activityId);
// Сохраняем метаданные файла. Это нужно делать обязательно ПЕРЕД сохранением его контента.
file.Save();
// byte[] content – это контент файла.
var content = new byte[] {0x12, 0x34, 0x56};
using (var stream = new MemoryStream(content)) {
    // Сохраняем контент в БД.
    // FileWriteOptions.SinglePart означает, что весь контент передается одним непрерывным куском
    file.Write(stream, FileWriteOptions.SinglePart);
}
```

Пример. Получить содержимое файла.

Чтение содержимого файла

```
var content = new byte[]();
// Получаем файл по его локатору.
var fileLocator= new EntityFileLocator("ActivityFile", recordId);
IFile file = UserConnection.GetFile(fileLocator);
// Читаем все содержимое файла в массив байт content. Не забудьте освобождать объект потока с помощью Dispose()
using (Stream stream = file.Read()) {
    content = stream.ReadAllBytes();
}
```

Пример. Скопировать файл, затем переместить его на новое место и удалить.

Копирование, перемещение, удаление файла

```

var content = new byte[]();
// Получаем файл по его локатору.
var fileLocator= new EntityFileLocator("ActivityFile", fileId);
IFile file = UserConnection.GetFile(fileLocator);
// Читаем все содержимое файла в массив байт content. Не забудьте освобождать объект потока с помощью
using (Stream stream = file.Read()) {
    content = stream.ReadAllBytes();
}

// Копирование файла.

// Создаем новый IFile для копирования текущего.
Guid copyFileDialogId = Guid.NewGuid();
var copyFileLocator = new EntityFileLocator("ActivityFile", copyFileDialogId);
IFile copyFile = UserConnection.CreateFile(copyFileLocator);
copyFile.Name = file.Name + " - Copy";
copyFile.Save();

// Копируем содержимого первого файла в новый файл.
copyFile.Write(new MemoryStream(content), FileModeOptions.SinglePart);

// Альтернативный способ копирования файла.
file.Copy(copyFile);

// Перемещение файла.

// Создаем новый файл для перемещения текущего.
Guid moveFileDialogId = Guid.NewGuid();
var moveFileLocator = new EntityFileLocator("ContactFile", moveFileDialogId);
IFile moveFile = UserConnection.CreateFile(moveFileLocator);
moveFile.Save();

// Перемещаем на новое место.
file.Move(moveFile);

// Удаление исходного файла.
file.Delete();

```

Пример реализации файлового хранилища контента



Средний

Пример. Создать класс, реализующий интерфейс `IFileContentStorage` для создания хранилища контента в файловой системе.

Пример реализации файлового хранилища контента

```
namespace Terrasoft.Configuration
{
    using System.IO;
    using System.Threading.Tasks;
    using Terrasoft.File.Abstractions;
    using Terrasoft.File.Abstractions.Content;
    using Terrasoft.File.Abstractions.Metadata;
    using Terrasoft.File.Metadata;

    /// <summary>
    /// Хранилище контента в файловой системе.
    /// </summary>
    public class FsFileBlobStorage : IFileContentStorage
    {

        // Корневой путь к хранилищу.
        private const string BaseFsPath = "C:\\FsStore\\\";

        private static string GetPath(FileMetadata fileMetadata) {
            var md = (EntityFileMetadata)fileMetadata;
            string key = $"{md.EntitySchemaName}\\{md.RecordId}_{fileMetadata.Name}";
            return Path.Combine(BaseFsPath, key);
        }

        public Task<Stream> ReadAsync(IFileContentReadContext context) {
            string filePath = GetPath(context.FileMetadata);
            Stream stream = System.IO.File.OpenRead(filePath);
            return Task.FromResult(stream);
        }

        public async Task WriteAsync(IFileContentWriteContext context) {
            string filePath = GetPath(context.FileMetadata);
            FileMode flags = context.WriteOptions != FileMode.SinglePart
                ? FileMode.Append
                : FileMode.OpenOrCreate;
            string dirPath = Path.GetDirectoryName(filePath);
            if (!Directory.Exists(dirPath)) {
                Directory.CreateDirectory(dirPath);
            }
            using (var fileStream = System.IO.File.Open(filePath, flags)) {
                context.Stream.CopyToAsync(fileStream);
            }
        }
    }
}
```

```
        }

    public Task DeleteAsync(IFileContentDeleteContext context) {
        string filePath = GetPath(context.FileMetadata);
        System.IO.File.Delete(filePath);
        return Task.CompletedTask;
    }

    public Task CopyAsync(IFileContentCopyMoveContext context) {
        string sourceFilePath = GetPath(context.SourceMetadata);
        string targetFilePath = GetPath(context.TargetMetadata);
        System.IO.File.Copy(sourceFilePath, targetFilePath);
        return Task.CompletedTask;
    }

    public Task MoveAsync(IFileContentCopyMoveContext context) {
        string sourceFilePath = GetPath(context.SourceMetadata);
        string targetFilePath = GetPath(context.TargetMetadata);
        System.IO.File.Move(sourceFilePath, targetFilePath);
        return Task.CompletedTask;
    }
}
```

Интерфейс IFile



Средний

Пространство имен `Terrasoft.File.Abstractions`.

Интерфейс `Terrasoft.File.Abstractions.IFile` предоставляет необходимые методы для работы с файлами, хранящимися в любых типах файловых хранилищ. Методы этого интерфейса обеспечивают асинхронную работу с файлами. Синхронные версии этих методов находятся в расширяющем классе `Terrasoft.File.Abstractions.FileUtils`.

На заметку. Полное описание интерфейса `IFile` можно найти в документации "[.NET библиотеки классов ядра платформы](#)".

Свойства

FileLocator **IFileLocator**

Файловый локатор, который связан с текущим экземпляром класса, реализующего интерфейс `IFile`.

Name string

Имя файла.

Length long

Размер текущего файла в байтах.

CreatedOn DateTime

Дата и время создания текущего файла.

ModifiedOn DateTime

Дата и время модификации текущего файла.

Exists bool

Показывает существует ли текущий файл.

Методы

Task CopyAsync(IFile target)

Асинхронно копирует текущий файл в новый `target`.

Task MoveAsync(IFile target)

Асинхронно перемещает текущий файл в новый `target`.

Task DeleteAsync()

Асинхронно удаляет текущий файл.

Task WriteAsync(Stream stream, FileMode writeOptions)

Асинхронно записывает содержимое текущего файла в поток `stream`.

Task<Stream> ReadAsync()

Асинхронно считывает содержимое текущего файла.

```
Task SaveAsync()
```

Асинхронно сохраняет метаданные текущего файла.

```
void SetAttribute< TValue >(string name, TValue value)
```

Устанавливает значение `value` атрибута `name` для текущего файла.

```
TValue GetAttribute< TValue >(string name, TValue defaultValue)
```

Возвращает значение `defaultValue` атрибута `name` либо значение по умолчанию для текущего файла.

Интерфейс IFileContentStorage

 Средний

Пространство имен `Terrasoft.File.Abstractions`.

Интерфейс `Terrasoft.File.Abstractions.IFileContentStorage` предоставляет необходимые методы для работы с хранилищем контента файла.

На заметку. Полное описание интерфейса `IFileContentStorage` можно найти в документации "[.NET библиотеки классов ядра платформы](#)".

Методы

```
Task<Stream> ReadAsync(IFileContentReadContext context)
```

Считывает файловый контент.

```
Task WriteAsync(IFileContentWriteContext context)
```

Записывает файловый контент.

```
Task DeleteAsync(IFileContentDeleteContext context)
```

Удаляет файловый контент.

```
Task CopyAsync(IFileContentCopyMoveContext context)
```

Копирует файловый контент.

```
Task MoveAsync(IFileContentCopyMoveContext context)
```

Перемещает файловый контент.

Интерфейс IFileFactory c#



Средний

Пространство имен `Terrasoft.File.Abstractions`.

Интерфейс `Terrasoft.File.Abstractions.IFileFactory` предоставляет набор методов для получения или создания экземпляра класса, реализующего интерфейс `Terrasoft.File.Abstractions.IFile`.

На заметку. Полное описание интерфейса `IFileFactory` можно найти в документации "[.NET библиотеки классов ядра платформы](#)".

Свойства

`UseRights bool`

Определяет должны ли учитываться права пользователя при создании файла или нет.

Методы

`IFile Get(IFileLocator fileLocator, FileOptions options)`

Возвращает экземпляр класса, реализующего интерфейс `IFile`, с заданными параметрами `options` из определенного файлового локатора `fileLocator`.

`IFile Create(IFileLocator fileLocator, FileOptions options)`

Создает новый экземпляр класса, реализующего интерфейс `IFile`, с заданными параметрами `options` для определенного файлового локатора `fileLocator`.

Класс EntityFileLocator c#



Средний

Пространство имен `Terrasoft.File`.

Класс предоставляет реализацию интерфейса `IFileLocator` для текущего файлового хранилища `Creatio`.

На заметку. Полный перечень методов и свойств класса `EntityFileLocator`, его родительских классов, а также реализуемых им интерфейсов, можно найти в документации "[.NET библиотеки классов ядра платформы](#)".

Конструкторы

`EntityFileLocator()`

Создает новый экземпляр класса `EntityFileLocator`.

`EntityFileLocator(string entitySchemaName, Guid recordId)`

Создает новый экземпляр класса `EntityFileLocator` для заданного файла `recordId`, привязанного к конкретной схеме объекта `entitySchemaName`.

Свойства

`EntitySchemaName string`

Метаданные — имя схемы объекта, в котором хранится файл.

`RecordId Guid`

Метаданные — идентификатор файла.

Класс EntityFileMetadata C#



Средний

Пространство имен `Terrasoft.File`.

Класс `Terrasoft.File.EntityFileMetadata` реализует абстрактный класс `Terrasoft.File.Abstractions.Metadata.FileMetadata`. Этот класс описывает метаданные файлов в объекте [Файлы и ссылки] ([Attachments]) и предоставляет методы для работы с ними.

На заметку. Полный перечень методов и свойств класса `EntityFileMetadata`, его родительских классов, а также реализуемых им интерфейсов, можно найти в документации "[.NET библиотеки классов ядра платформы](#)".

Конструкторы

```
EntityFileMetadata(EntityFileLocator fileLocator)
```

Создает новый экземпляр класса `EntityFileMetadata` для файлового локатора `fileLocator`.

Свойства

`Attributes IReadOnlyDictionary<string, object>`

Коллекция значений атрибутов.

`RecordId Guid`

Идентификатор файла.

`EntitySchemaName string`

Имя схемы объекта, в котором хранится файл.

Методы

```
override void SetAttribute< TValue >(string name, TValue value)
```

Устанавливает для файла дополнительный значение `value` дополнительного атрибута `name`.

```
override TValue GetAttribute< TValue >(string name, TValue defaultValue)
```

Возвращает установленное значение либо значение по умолчанию `defaultValue` определенного дополнительного атрибута `name`.

Класс FileFactoryUtils



Пространство имен `Terrasoft.File`.

Класс `Terrasoft.File.FileFactoryUtils` предоставляет расширяющие методы класса `UserConnection` и класса-фабрики, реализующего интерфейс `Terrasoft.File.AbstractionsIBuilderFactory`. Таким образом класс обеспечивает доступ к фабрике создания новых или получения существующих файлов. Соответственно, для успешной работы с файлами необходимо иметь экземпляр `UserConnection` ИЛИ `SystemUserConnection`.

На заметку. Полный перечень методов и свойств класса `FileFactoryUtils`, его родительских классов, а также реализуемых им интерфейсов, можно найти в документации "[.NET библиотеки классов ядра платформы](#)".

Методы

```
static IFileFactory GetFileFactory(this UserConnection source)
```

Расширяющий метод класса `UserConnection`, который возвращает экземпляр класса, реализующего интерфейс `IFileFactory`.

```
static IFile GetFile(this UserConnection source, IFileLocator fileLocator)
```

Расширяющий метод класса `UserConnection`, который возвращает экземпляр класса, реализующего интерфейс `IFile` из определенного файлового локатора `fileLocator`.

```
static IFile CreateFile(this UserConnection source, IFileLocator fileLocator)
```

Расширяющий метод класса `UserConnection`, который создает новый экземпляр класса, реализующего интерфейс `IFile` для определенного файлового локатора `fileLocator`.

```
static IFile Get(this IFileFactory source, IFileLocator fileLocator)
```

Расширяющий метод класса, реализующего интерфейс `IFileFactory`. Возвращает экземпляр класса, реализующего интерфейс `IFile` для определенного файлового локатора `fileLocator`.

```
static IFile Create(this IFileFactory source, IFileLocator fileLocator)
```

Расширяющий метод класса, реализующего интерфейс `IFileFactory`. Создает новый экземпляр класса, реализующего интерфейс `IFile` для определенного файлового локатора `fileLocator`.

```
static IFileFactory WithRightsDisabled(this IFileFactory source)
```

Расширяющий метод класса, реализующего интерфейс `IFileFactory`. Возвращает экземпляр класса, реализующего интерфейс `IFileFactory`, настроенного без учета прав пользователя.

Класс FileMetadata c#



Средний

Пространство имен `Terrasoft.File.Abstractions.Metadata`.

Абстрактный класс `Terrasoft.File.Abstractions.Metadata.FileMetadata` предоставляет свойства метаданных файла и методы для работы с метаданными.

На заметку. Полный перечень методов и свойств класса `FileMetadata`, его родительских классов, а также реализуемых им интерфейсов, можно найти в документации "[.NET библиотеки классов ядра](#)

[платформы](#)".

Свойства

Name string

Имя файла.

Length long

Размер файла в байтах.

CreatedOn DateTime

Дата и время создания файла.

ModifiedOn DateTime

Дата и время изменения файла.

FileContentStorageId Guid

Идентификатор хранилища контента файла.

StoringState FileStoringState

Состояние файла ("Новый", "Изменен", "Неизменен", "Удален").

Методы

abstract void SetAttribute< TValue >(string name, TValue value)

Устанавливает для файла дополнительный значение `value` дополнительного атрибута `name`.

abstract TValue GetAttribute< TValue >(string name, TValue defaultValue)

Возвращает установленное значение либо значение по умолчанию `defaultValue` определенного дополнительного атрибута `name`.

void SetStoringState(FileStoringState newState)

Устанавливает состояние файла `FileStoringState.Modified` если предыдущее состояние не равно

```
FileStoringState.New .
```

Класс FileUtils



Средний

Пространство имен `Terrasoft.File.Abstractions`.

Класс `Terrasoft.File.Abstractions.FileUtils` предоставляет расширяющие методы для работы с файлами.

На заметку. Полный перечень методов и свойств класса `FileUtils`, его родительских классов, а также реализуемых им интерфейсов, можно найти в документации "[.NET библиотеки классов ядра платформы](#)".

Методы

```
static void SetAttributes(this IFile source, IReadOnlyDictionary<string, object> attributes)
```

Устанавливает для файла значения атрибутов, переданных в коллекции `attributes`.

```
static void Save(this IFile source)
```

Сохраняет метаданные файла.

```
static Stream Read(this IFile source)
```

Считывает содержимое файла.

```
static void Write(this IFile source, Stream stream, FileModeOptions writeOptions)
static void Write(this IFile source, byte[] content)
```

Записывает содержимое файла.

Параметры

<code>source</code>	Файл, содержимое которого необходимо записать.
<code>stream</code>	Поток, предоставляющий содержимое файла.
<code>writeOptions</code>	Параметры для записи файла.
<code>content</code>	Содержимое файла в виде массива байтов.

```
static void Delete(this IFile source)
```

Удаляет определенный файл.

```
static void Copy(this IFile source, IFile target)
```

Копирует существующий файл `source` в новый `target`.

```
static void Move(this IFile source, IFile target)
```

Перемещает существующий файл `source` в новое место `target`.

Библиотека .NET классов



Сложный

Документация по классам back-end части ядра платформы (.NET Core API) доступна на отдельном web-ресурсе.