

Сервисы работы с данными

DataService

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

| | |
|--|-----------|
| DataService | 5 |
| Контракты данных | 5 |
| Ограничения при использовании DataService | 5 |
| Создать новую запись в разделе | 6 |
| Реализация создания новой записи в разделе [Контакты] через стороннее приложение | 6 |
| Реализация создания новой записи в разделе [Контакты] через Creatio IDE | 10 |
| Прочитать записи раздела | 14 |
| Реализация чтения записи раздела [Контакты] через стороннее приложение | 14 |
| Реализация чтения записи раздела [Контакты] через Creatio IDE | 18 |
| Отфильтровать записи раздела | 22 |
| Реализовать фильтры | 23 |
| Отфильтровать записи раздела с использованием макроса | 27 |
| Реализовать фильтр с макросом | 27 |
| Отфильтровать записи раздела по источнику вхождения пользователя в роль | 30 |
| Реализовать фильтры по источнику вхождения пользователя в роль | 30 |
| Обновить запись раздела | 32 |
| 1. Создать и настроить проект консольного приложения C# | 32 |
| 2. Добавить объявление полей и констант | 33 |
| 3. Добавить метод, выполняющий аутентификацию | 33 |
| 4. Реализовать запрос на добавление записи | 33 |
| 5. Выполнить POST-запрос к DataService | 36 |
| Удалить запись раздела | 36 |
| 1. Создать и настроить проект консольного приложения C# | 37 |
| 2. Добавить объявление полей и констант | 37 |
| 3. Добавить метод, выполняющий аутентификацию | 38 |
| 4. Реализовать запрос на удаление записи | 38 |
| 5. Выполнить POST-запрос к DataService | 40 |
| Добавить и изменить запись раздела | 41 |
| 1. Создать и настроить проект консольного приложения C# | 41 |
| 2. Добавить объявление полей и констант | 42 |
| 3. Добавить метод, выполняющий аутентификацию | 42 |
| 4. Реализовать запрос на добавление записи | 42 |
| 5. Реализовать запрос на изменение записи | 43 |
| 6. Реализовать пакетный запрос | 44 |
| 7. Реализовать пакетный запрос | 45 |
| Класс InsertQuery | 46 |

| | |
|--|-----------|
| Свойства | 47 |
| Класс SelectQuery | 48 |
| Класс SelectQuery | 48 |
| Класс SelectQueryColumns | 54 |
| Класс ColumnExpression | 55 |
| Класс ServerESQCacheParameters | 59 |
| Класс Filters | 59 |
| Класс Filters | 60 |
| Класс BaseExpression | 64 |
| Перечисление EntitySchemaQueryMacroType | 68 |
| Класс UpdateQuery | 70 |
| Свойства | 71 |
| Класс DeleteQuery | 73 |
| Свойства | 74 |
| Класс BatchQuery | 75 |
| Свойства | 77 |
| Класс ColumnValues | 77 |
| Свойства | 77 |
| Класс Parameter | 78 |
| Свойства | 78 |
| Методы | 80 |

DataService



Сервис работы с данными **DataService** приложения Creatio является RESTful-сервисом, т. е. поддерживает передачу состояния представления ([Representational State Transfer, REST](#)). В общем случае REST является интерфейсом управления информацией без использования дополнительных внутренних прослоек, т. е. данные не нужно преобразовывать в сторонний формат, например, XML. В RESTful-сервисе каждая единица информации определяется глобальным идентификатором, таким как **URL**. Каждый URL имеет строго заданный формат. Однако это не всегда удобно для передачи больших массивов данных. Доступ к объектам Creatio предоставляет веб-сервис `dataservice`.

Адрес сервиса `dataservice`

`https://mycreatio.com/0/dataservice`

В DataService данные автоматически могут быть сконфигурированы в различные форматы данных, такие как XML, JSON, HTML, CSV и JSV. Структура данных определяется [контрактами данных](#).

Контракты данных

Контракты данных сервиса работы с данными DataService, рекомендуемые для интеграции с Creatio, приведены в таблице.

Контракты данных сервиса DataService приложения Creatio

| Контракт данных | Описание |
|--------------------------|---|
| <code>InsertQuery</code> | Класс запроса на добавление записи раздела. |
| <code>UpdateQuery</code> | Класс запроса на обновление записи раздела. |
| <code>DeleteQuery</code> | Класс запроса на удаление записи раздела. |
| <code>SelectQuery</code> | Класс запроса на выбор записей раздела. |
| <code>BatchQuery</code> | Класс пакетного запроса. |
| <code>Filters</code> | Класс фильтров. |

Ограничения при использовании DataService

- 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. Рекомендуется использовать постраничный вывод, который реализован в свойствах `IsPageable` и `RowCount` контракта данных [SelectQuery](#).

- Количество запросов не ограничено.

Создать новую запись в разделе



Реализация создания новой записи в разделе [Контакты] через стороннее приложение

Пример. Создать консольное приложение, которое, используя класс `InsertQuery`, добавляет запись с колонками:

- [`ФИО`] ([`Name`]) — "Иванов Иван Иванович".
- [`Должность`] ([`Job`]) — "Разработчик".
- [`Рабочий телефон`] ([`Phone`]) — "+12 345 678 00 00".

1. Создать и настроить проект консольного приложения C#

1. Используя среду разработки Microsoft Visual Studio (версии не ниже 2017), создайте проект консольного приложения Visual C#.
Работа в Microsoft Visual Studio подробно описана в статье [Разработать C# код в пользовательском проекте](#).
2. Укажите в качестве названия проекта, например, `DataServiceInsertExample`.
3. Свойству проекта `Target framework` установите значение ".NET Framework 4.7".
4. В секцию `References` проекта добавьте зависимости от библиотек:
 - `System.Web.Extensions.dll` — библиотека классов, входящая в .NET Framework.
 - `Terrasoft.Core.dll` — библиотека основных классов серверного ядра приложения. Находится по пути `...\Terrasoft.WebApp\bin\Terrasoft.Core.dll`.
 - `Terrasoft.Nui.ServiceModel.dll` — библиотека классов служб приложения. Находится по пути `...\Terrasoft.WebApp\bin\Terrasoft.Nui.ServiceModel.dll`.
 - `Terrasoft.Common.dll` — библиотека основных классов серверного ядра приложения. Находится по

пути `...\Terrasoft.WebApp\bin\Terrasoft.Common.dll`.

5. В файл исходного кода приложения добавьте директивы `using`.

Добавление директив `using`

```
using System;
using System.Text;
using System.IO;
using System.Net;
using System.Collections.Generic;
using Terrasoft.Nui.ServiceModel.DataContract;
using Terrasoft.Core.Entities;
using System.Web.Script.Serialization;
```

2. Добавить объявление полей и констант

Добавление полей и констант в исходный код приложения необходимо выполнить для доступа к возможностям сервиса работы с данными DataService.

Добавление объявления полей и констант

```
/* Основной URL приложения Creatio. Необходимо заменить на пользовательский. */
private const string baseUrl = @"http://example.creatio.com";
/* Строка запроса к методу Login сервиса AuthService.svc. */
private const string authServiceUri = baseUrl + @"/ServiceModel/AuthService.svc/Login";
/* Строка пути запроса InsertQuery. */
private const string insertQueryUri = baseUrl + @"/0/DataService/json/reply/InsertQuery";
/* Cookie аутентификации Creatio. */
private static CookieContainer AuthCookie = new CookieContainer();
```

Здесь объявлены три строковых константных поля, с помощью которых формируются пути выполнения запросов на аутентификацию и запросов на чтение данных. Данные об аутентификации будут сохранены в поле `[AuthCookie]`.

3. Добавить метод, выполняющий аутентификацию

Для доступа создаваемого приложения к сервису работы с данными DataService необходимо выполнить [аутентификацию](#).

4. Реализовать запрос на добавление записи

Поскольку объявленная ранее константа `insertQueryUri` содержит путь для отправки данных в формате JSON, то отправляемые данные необходимо предварительно сконфигурировать в виде строки, содержащей описание JSON-объекта, соответствующего контракту данных `InsertQuery`. Это можно

сделать непосредственно в строчной переменной, однако намного удобнее и безопаснее с точки зрения возможности возникновения ошибок создать экземпляр класса `InsertQuery`, заполнить его свойства, а затем сериализовать его в строку.

Реализация запроса на добавление записи

```
/* Экземпляр класса запроса. */
var insertQuery = new InsertQuery()
{
    /* Название корневой схемы. */
    RootSchemaName = "Contact",
    /* Коллекция добавляемых значений колонок. */
    ColumnValues = new ColumnValues()
};

/* Экземпляр класса выражения запроса к схеме объекта.
Предназначен для конфигурирования колонки [ФИО]. */
var columnExpressionName = new ColumnExpression
{
    /* Тип выражения запроса к схеме объекта – параметр. */
    ExpressionType = EntitySchemaQueryExpressionType.Parameter,
    /* Параметр выражения запроса. */
    Parameter = new Parameter
    {
        /* Значение параметра. */
        Value = "Иванов Иван Иванович",
        /* Тип данных параметра – строка. */
        DataType = DataType.Text
    }
};

/* Экземпляр класса выражения запроса к схеме объекта.
Предназначен для конфигурирования колонки [Рабочий телефон]. */
var columnExpressionPhone = new ColumnExpression
{
    ExpressionType = EntitySchemaQueryExpressionType.Parameter,
    Parameter = new Parameter
    {
        Value = "+12 345 678 00 00",
        DataType = DataType.Text
    }
};

/* Экземпляр класса выражения запроса к схеме объекта.
Предназначен для конфигурирования колонки [Должность]. */
var columnExpressionJob = new ColumnExpression
{
    ExpressionType = EntitySchemaQueryExpressionType.Parameter,
    Parameter = new Parameter
    {
```



```

        /* GUID записи "Разработчик" системного справочника [Должность].
        Необходимо заменить на GUID записи в приложении bpm'online пользователя. */
        Value = "11D68189-CED6-DF11-9B2A-001D60E938C6",
        /* Тип данных параметра – уникальный идентификатор. */
        DataValueType = DataValueType.Guid
    }
};
/* Инициализация коллекции колонок запроса. */
insertQuery.ColumnValues.Items = new Dictionary<string, ColumnExpression>();
/* Добавление выражений запроса в коллекцию добавляемых колонок.
Колонка [ФИО]. */
insertQuery.ColumnValues.Items.Add("Name", columnExpressionName);
/* Колонка [Рабочий телефон]. */
insertQuery.ColumnValues.Items.Add("Phone", columnExpressionPhone);
/* Колонка [Должность]. */
insertQuery.ColumnValues.Items.Add("Job", columnExpressionJob);
/* Сериализация экземпляра класса запроса на добавление в JSON-строку. */
var json = new JavaScriptSerializer().Serialize(insertQuery);

```

5. Выполнить POST-запрос к DataService

1. Создайте экземпляр класса [HttpWebRequest](#).
2. Заполните свойства экземпляра.
3. Присоедините к запросу созданную ранее строку с JSON-объектом.
4. Выполните и обработайте результат запроса к сервису работы с данными DataService.

Реализация POST -запроса

```

/* Преобразование строки JSON-объекта в массив байтов. */
byte[] jsonArray = Encoding.UTF8.GetBytes(json);
/* Создание экземпляра HTTP-запроса. */
var insertRequest = HttpWebRequest.Create(insertQueryUri) as HttpWebRequest;
/* Определение метода запроса. */
insertRequest.Method = "POST";
/* Определение типа содержимого запроса. */
insertRequest.ContentType = "application/json";
/* Добавление полученных ранее аутентификационных cookie в запрос на получение данных. */
insertRequest.CookieContainer = AuthCookie;
/* Установить длину содержимого запроса. */
insertRequest.ContentLength = jsonArray.Length;

/* Добавление CSRF токена в заголовок запроса. */
CookieCollection cookieCollection = AuthCookie.GetCookies(new Uri(authServiceUri));
string csrfToken = cookieCollection["BPMCSRF"].Value;
insertRequest.Headers.Add("BPMCSRF", csrfToken);

```

```

/* Помещение в содержимое запроса JSON-объекта. */
using (var requestStream = insertRequest.GetRequestStream())
{
    requestStream.Write(jsonArray, 0, jsonArray.Length);
}
/* Выполнение HTTP-запроса и получение ответа от сервера. */
using (var response = (HttpWebResponse)insertRequest.GetResponse())
{
    /* Вывод ответа в консоль. */
    using (StreamReader reader = new StreamReader(response.GetResponseStream()))
    {
        Console.WriteLine(reader.ReadToEnd());
    }
}

```

Реализация создания новой записи в разделе [Контакты] через Creatio IDE

Пример. Добавить кнопку в раздел [*Контакты*] ([*Contacts*]). При нажатии на кнопку вызывать метод, который, используя класс `InsertQuery`, добавляет запись с колонками:


- [*ФИО*] ([*Name*]) — "Иванов Иван Иванович".
- [*Должность*] ([*Job*]) — "Разработчик".
- [*Рабочий телефон*] ([*Phone*]) — "+12 345 678 00 00".

1. Добавить кнопку в раздел [*Контакты*]

1. [Создайте пакет](#) и установите его в качестве текущего.
2. Создайте [схему замещающей модели представления](#) раздела [*Контакты*] ([*Contacts*]).


Module ×

Code
ContactSectionV2

Title *
Contacts section 

Parent object *
Contacts section (ContactSectionV2) ▼

Package
sdkInsertQueryPackage

Description 


CANCEL APPLY

3. На панели дизайнера нажмите кнопку **+** и создайте локализуемую строку.
Для созданной строки установите:

- [Код] ([Code]) — "InsertQueryContactButtonCaption".
- [Значение] ([Value]) — "Добавить контакт" ("Add contact").

Localizable Strings ×

Code *
InsertQueryContactButtonCaption

Value
Add contact 

CANCEL ADD

4. В массив `diff` добавьте конфигурационный объект с настройками расположения кнопки на странице.

diff

```
diff: /**SCHEMA_DIFF*/[
  /* Метаданные для добавления в раздел пользовательской кнопки. */
  {
    /* Выполняется операция добавления элемента на страницу. */
    "operation": "insert",
    /* Название родительского элемента управления, в который добавляется кнопка. */
    "parentName": "ActionButtonsContainer",
    /* Кнопка добавляется в коллекцию элементов управления родительского элемента (мета-и
    "propertyName": "items",
    /* Мета-имя добавляемой кнопки. */
    "name": "InsertQueryContactButton",
    /* Дополнительные свойства элемента. */
    "values": {
      /* Тип добавляемого элемента – кнопка. */
      itemType: Terrasoft.ViewItemType.BUTTON,
      /* Привязка заголовка кнопки к локализуемой строке схемы. */
      caption: { bindTo: "Resources.Strings.InsertQueryContactButtonCaption" },
      /* Привязка метода-обработчика нажатия кнопки. */
      click: { bindTo: "onInsertQueryContactClick" },
      "layout": {
        "column": 1,
        "row": 6,
        "colSpan": 1
      }
    }
  }
]/**SCHEMA_DIFF*/
```

2. Добавить метод-обработчик события нажатия кнопки

Чтобы при нажатии на созданную в разделе [*Контакты*] ([*Contacts*]) кнопку добавлялась запись с заданными параметрами, в секцию `methods` замещающей схемы модели представления добавьте метод-обработчик события нажатия кнопки `onInsertQueryContactClick`.

methods

```
methods: {
  /* Метод-обработчик нажатия кнопки. */
  onInsertQueryContactClick: function() {
    /* Создание экземпляра класса Terrasoft.InsertQuery. */
    var insert = Ext.create("Terrasoft.InsertQuery", {
      /* Название корневой схемы. */
      rootSchemaName: "Contact"
    });
```

```

/* Установка значений-параметров Terrasoft.ParameterExpression.
Создается экземпляр значения-параметра и добавляется в коллекцию значений колонок.
Создание экземпляра значения-параметра для колонки [ФИО]. */
insert.setParameterValue("Name", "Иванов Иван Иванович", Terrasoft.DataValueType.TEXT);
/* Создание экземпляра значения-параметра для колонки [Рабочий телефон]. */
insert.setParameterValue("Phone", "+12 345 678 00 00", Terrasoft.DataValueType.TEXT);
/* Создание экземпляра значения-параметра для колонки [Должность]. */
insert.setParameterValue("Job", "11D68189-CED6-DF11-9B2A-001D60E938C6", Terrasoft.DataVa
/* Запрос к серверу на обновление данных. */
insert.execute(function(response) {
    /* Вывод ответа от сервера в консоль браузера. */
    window.console.log(response);
});
/* Обновление данных реестра. */
this.reloadGridData();
}
}

```

Реализация класса `InsertQuery` для front-end части ядра приложения отличается от реализации класса `InsertQuery` для back-end части. Так, для создания параметров предусмотрен метод `setParameterValue`, а для выполнения запроса — метод `execute`. Узнать о свойствах и методах класса `InsertQuery`, реализованного в front-end части ядра приложения, можно в [Библиотеке JS классов](#).

Результат выполнения примера

После сохранения схемы и обновления страницы приложения с очисткой кэша в разделе [*Контакты*] ([*Contacts*]) появится кнопка [*Добавить контакт*] ([*Add contact*]). Кнопка добавляет контакт с заданными параметрами.

Прочитать записи раздела

 Сложный

Реализация чтения записи раздела [Контакты] через стороннее приложение

Пример. Создать консольное приложение, которое, используя класс `SelectQuery`, прочитает записи раздела [*Контакты*] ([*Contacts*]). Отобразить колонки:

- [*Id*].
- [*ФИО*] ([*Name*]).
- [*Количество активностей*] ([*ActivitiesCount*]) — агрегирующая колонка, отображающая количество активностей данного контакта.

1. Создать и настроить проект консольного приложения C#

1. Используя среду разработки Microsoft Visual Studio (версии не ниже 2017), создайте проект консольного приложения Visual C#.
Работа в Microsoft Visual Studio подробно описана в статье [Разработать C# код в пользовательском проекте](#).
2. Укажите в качестве названия проекта, например, `DataServiceSelectExample`.
3. Свойству проекта `Target framework` установите значение ".NET Framework 4.7".
4. В секцию `References` проекта добавьте зависимости от библиотек:
 - `System.Web.Extensions.dll` — библиотека классов, входящая в .NET Framework.
 - `Terrasoft.Core.dll` — библиотека основных классов серверного ядра приложения. Находится по пути `...\Terrasoft.WebApp\bin\Terrasoft.Core.dll`.
 - `Terrasoft.Nui.ServiceModel.dll` — библиотека классов служб приложения. Находится по пути `...\Terrasoft.WebApp\bin\Terrasoft.Nui.ServiceModel.dll`.
 - `Terrasoft.Common.dll` — библиотека основных классов серверного ядра приложения. Находится по пути `...\Terrasoft.WebApp\bin\Terrasoft.Common.dll`.
5. В файл исходного кода приложения добавьте директивы `using`.

Добавление директив `using`

```
using System;
using System.Text;
using System.IO;
using System.Net;
using System.Collections.Generic;
using Terrasoft.Nui.ServiceModel.DataContract;
using Terrasoft.Core.Entities;
using System.Web.Script.Serialization;
using Terrasoft.Common;
```

2. Добавить объявление полей и констант

Добавление полей и констант в исходный код приложения необходимо выполнить для доступа к возможностям сервиса работы с данными DataService.

Добавление объявления полей и констант

```
/* Основной URL приложения Creatio. Необходимо заменить на пользовательский. */
private const string baseUri = @"http://example.creatio.com";
/* Строка запроса к методу Login сервиса AuthService.svc. */
private const string authServiceUri = baseUri + @"/ServiceModel/AuthService.svc/Login";
/* Строка пути запроса SelectQuery. */
```

```
private const string selectQueryUri = baseUri + @"/0/DataService/json/SyncReply/SelectQuery";
/* Cookie аутентификации Creatio. */
private static CookieContainer AuthCookie = new CookieContainer();
```

Здесь объявлены три строковых константных поля, с помощью которых формируются пути выполнения запросов на аутентификацию и запросов на чтение данных. Данные об аутентификации будут сохранены в поле [*AuthCookie*].

3. Добавить метод, выполняющий аутентификацию

Для доступа создаваемого приложения к сервису работы с данными DataService необходимо выполнить [аутентификацию](#).

4. Реализовать запрос на добавление записи

Поскольку объявленная ранее константа `selectQueryUri` содержит путь для отправки данных в формате JSON, то отправляемые данные необходимо предварительно сконфигурировать в виде строки, содержащей описание JSON-объекта, соответствующего контракту данных `SelectQuery`. Это можно сделать непосредственно в строчной переменной, однако намного удобнее и безопаснее с точки зрения возможности возникновения ошибок создать экземпляр класса `SelectQuery`, заполнить его свойства, а затем сериализовать его в строку.

Реализация запроса на добавление записи

```
/* Экземпляр класса запроса. */
var selectQuery = new SelectQuery()
{
    /* Название корневой схемы. */
    RootSchemaName = "Contact",
    /* Коллекция колонок запроса. */
    Columns = new SelectQueryColumns()
};
/* Выражение, задающее тип колонки [ФИО]. */
var columnExpressionName = new ColumnExpression()
{
    /* Тип выражения – колонка схемы. */
    ExpressionType = EntitySchemaQueryExpressionType.SchemaColumn,
    /* Путь к колонке. */
    ColumnPath = "Name"
};
/* Конфигурирование колонки [Name]. */
var selectQueryColumnName = new SelectQueryColumn()
{
    /* Заголовок. */
    Caption = "ФИО",
    /* Направление сортировки – по возрастанию. */
    OrderDirection = OrderDirection.Ascending,
```



```

    /* Позиция порядка сортировки. */
    OrderPosition = 0,
    /* Выражение, задающее тип колонки. */
    Expression = columnName
};
/* Выражение, задающее тип колонки [Количество активностей]. */
var columnExpressionActivitiesCount = new ColumnExpression()
{
    /* Тип выражения – вложенный запрос. */
    ExpressionType = EntitySchemaQueryExpressionType.SubQuery,
    /* Путь к колонке относительно корневой схемы. */
    ColumnPath = "[Activity:Contact].Id",
    /* Тип функции – агрегирующая. */
    FunctionType = FunctionType.Aggregation,
    /* Тип агрегации – количество. */
    AggregationType = AggregationType.Count
};
/* Конфигурирование колонки [Количество активностей]. */
var selectQueryColumnActivitiesCount = new SelectQueryColumn()
{
    /* Заголовок. */
    Caption = "Количество активностей",
    /* Направление сортировки – по возрастанию. */
    OrderDirection = OrderDirection.Ascending,
    /* Позиция порядка сортировки. */
    OrderPosition = 1,
    /* Выражение, задающее тип колонки. */
    Expression = columnExpressionActivitiesCount
};

/* Добавление колонок в запрос. */
selectQuery.Columns.Items = new Dictionary<string, SelectQueryColumn>()
{
    {
        "Name",
        selectQueryColumnName
    },
    {
        "ActivitiesCount",
        selectQueryColumnActivitiesCount
    }
};
/* Сериализация экземпляра класса запроса на добавление в JSON-строку. */
var json = new JavaScriptSerializer().Serialize(selectQuery);

```

5. Выполнить POST-запрос к DataService

1. Создайте экземпляр класса [HttpWebRequest](#).
2. Заполните свойства экземпляра.
3. Присоедините к запросу созданную ранее строку с JSON-объектом.
4. Выполните и обработайте результат запроса к сервису работы с данными DataService.

Реализация POST -запроса

```

/* Преобразование строки JSON-объекта в массив байтов. */
byte[] jsonArray = Encoding.UTF8.GetBytes(json);
/* Создание экземпляра HTTP-запроса. */
var selectRequest = HttpWebRequest.Create(selectQueryUri) as HttpWebRequest;
/* Определение метода запроса. */
selectRequest.Method = "POST";
/* Определение типа содержимого запроса. */
selectRequest.ContentType = "application/json";
/* Добавление полученных ранее аутентификационных cookie в запрос на получение данных. */
selectRequest.CookieContainer = AuthCookie;
/* Установить длину содержимого запроса. */
selectRequest.ContentLength = jsonArray.Length;
/* Добавление CSRF токена в заголовок запроса. */
CookieCollection cookieCollection = AuthCookie.GetCookies(new Uri(authServiceUri));
string csrfToken = cookieCollection["BPMCSRF"].Value;
selectRequest.Headers.Add("BPMCSRF", csrfToken);

/* Помещение JSON-объекта в содержимое запроса. */
using (var requestStream = selectRequest.GetRequestStream())
{
    requestStream.Write(jsonArray, 0, jsonArray.Length);
}
/* Выполнение HTTP-запроса и получение ответа от сервера. */
using (var response = (HttpWebResponse)selectRequest.GetResponse())
{
    /* Вывод ответа в консоль. */
    using (StreamReader reader = new StreamReader(response.GetResponseStream()))
    {
        Console.WriteLine(reader.ReadToEnd());
    }
}

```

Реализация чтения записи раздела [Контакты] через Creatio IDE

Пример. Добавить кнопку в раздел [*Контакты*] ([*Contacts*]). При нажатии на кнопку вызывать метод, который, используя класс `SelectQuery`, прочитает записи раздела [*Контакты*] ([*Contacts*]).

Отобразить колонки:

- [*Id*].
- [*ФИО*] ([*Name*]).
- [*Количество активностей*] ([*ActivitiesCount*]) — агрегирующая колонка, отображающая количество активностей данного контакта.

1. Добавить кнопку в раздел [*Контакты*]

1. [Создайте пакет](#) и установите его в качестве текущего.
2. Создайте [схему замещающей модели представления](#) раздела [*Контакты*] ([*Contacts*]).

Module

Code
ContactSectionV2

Title *
Contacts section

Parent object *
Contacts section (ContactSectionV2)

Package
sdkSelectQueryPackage

Description

CANCEL APPLY

3. На панели дизайнера нажмите кнопку **+** и создайте локализуемую строку.
Для созданной строки установите:

- [*Код*] ([*Code*]) — "SelectQueryContactButtonCaption".
- [*Значение*] ([*Value*]) — "Выбор контактов" ("Select contacts").

Localizable Strings



Code *

SelectQueryContactButtonCaption

Value

Select contacts



CANCEL

ADD

4. В массив `diff` добавьте конфигурационный объект с настройками расположения кнопки на странице.

`diff`

```

/* Настройка визуализации кнопки в разделе. */
diff: /**SCHEMA_DIFF*/[
  /* Метаданные для добавления в раздел пользовательской кнопки. */
  {
    /* Указывает на то, что выполняется операция добавления элемента на страницу. */
    "operation": "insert",
    /* Мета-имя родительского элемента управления, в который добавляется кнопка. */
    "parentName": "ActionButtonsContainer",
    /* Указывает на то, что кнопка добавляется в коллекцию элементов управления родительс
    "propertyName": "items",
    /* Мета-имя добавляемой кнопки. */
    "name": "SelectQueryContactButton",
    /* Дополнительные свойства элемента. */
    "values": {
      /* Тип добавляемого элемента – кнопка. */
      itemType: Terrasoft.ViewItemType.BUTTON,
      /* Привязка заголовка кнопки к локализуемой строке схемы. */
      caption: { bindTo: "Resources.Strings.SelectQueryContactButtonCaption" },
      /* Привязка метода-обработчика нажатия кнопки. */
      click: { bindTo: "onSelectQueryContactClick" },
      "layout": {
        "column": 1,
        "row": 6,
        "colSpan": 1
      }
    }
  }
]
}

```

```
]/**SCHEMA_DIFF*/
```

2. Добавить метод-обработчик события нажатия кнопки

Чтобы при нажатии на созданную в разделе [*Контакты*] ([*Contacts*]) кнопку читались записи с заданными параметрами, в секцию `methods` замещающей схемы модели представления добавьте метод-обработчик события нажатия кнопки `onSelectQueryContactClick`.

methods

```
methods: {
  /* Метод-обработчик нажатия кнопки. */
  onSelectQueryContactClick: function() {
    /* Создание экземпляра класса Terrasoft.InsertQuery. */
    var select = Ext.create("Terrasoft.EntitySchemaQuery", {
      /* Название корневой схемы. */
      rootSchemaName: "Contact"
    });
    /* Добавление в запрос колонки [ФИО]. */
    select.addColumn("Name");
    /* Добавление в запрос агрегирующей колонки [Количество активностей]. */
    select.addAggregationSchemaColumn(
      /* Путь к колонке относительно корневой схемы. */
      "[Activity:Contact].Id",
      /* Тип агрегации – количество. */
      Terrasoft.AggregationType.COUNT,
      /* Заголовок колонки. */
      "ActivitiesCount",
      /* Область применения агрегирующей функции – для всех элементов. */
      Terrasoft.AggregationEvalType.ALL);
    /* Запрос к серверу на обновление данных.
    Получение всей коллекции записей и ее отображение в консоли браузера. */
    select.getEntityCollection(function(result) {
      if (!result.success) {
        /* Обработка/логирование ошибки. */
        this.showInformationDialog("Ошибка запроса данных");
        return;
      }
      /* Выводимое сообщение. */
      var message = "";
      /* Перебор результирующей коллекции и построение выводимого сообщения. */
      result.collection.each(function(item) {
        message += "ФИО: " + item.get("Name") +
          ". Количество активностей: " + item.get("ActivitiesCount") + "\n";
      });
      /* Вывод сообщения в консоль. */
      window.console.log(message);
    });
  }
}
```

```

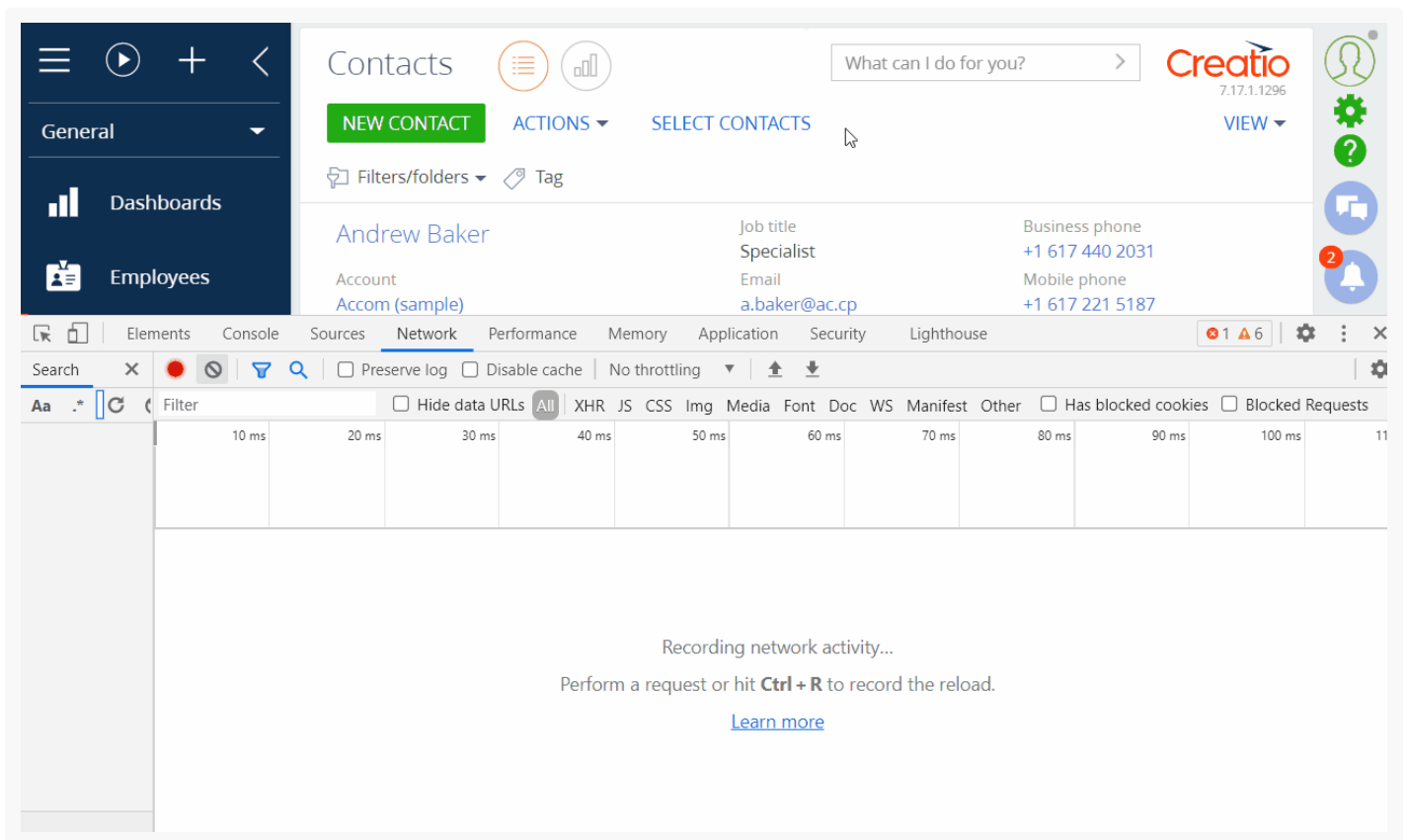
    }, this);
  }
}

```

В front-end части ядра приложения отсутствует класс, аналогичный классу `SelectQuery` для back-end части. Для выбора данных раздела необходимо использовать класс `Terrasoft.EntitySchemaQuery`. Подробнее о свойствах и методах этого класса можно узнать в [Библиотеке JS классов](#).

Результат выполнения примера

После сохранения схемы и обновления страницы приложения с очисткой кэша в разделе [*Контакты*] ([*Contacts*]) появится кнопка [*Выбор контактов*] ([*Select contacts*]). В консоли браузера кнопка отобразит записи с заданными параметрами.



Отфильтровать записи раздела

 Сложный

Пример. Создать консольное приложение, которое, используя класс `SelectQuery`, прочитает записи раздела [*Контакты*] ([*Contacts*]). Отобразить колонки:

- [*Id*].

- [ФИО] ([Name]).
- [Количество активностей] ([ActivitiesCount]) — агрегирующая колонка, отображающая количество активностей данного контакта.

Отфильтровать данные таким образом, чтобы были прочитаны только те контакты, у которых количество активностей находится в диапазоне от 1 до 3, а значение колонки [ФИО] ([Name]) начинается с символа "ч".

Реализовать фильтры

Предварительно выполните пример чтения записей в стороннем приложении, описанный в статье [Добавить кнопку для чтения записей раздела \[Контакты\]](#).

Результат реализации экземпляра класса запроса на чтение записей в сокращенной форме представлен ниже.

SelectQuery()

```
/* Экземпляр класса запроса. */
var selectQuery = new SelectQuery()
{
    /* Название корневой схемы. */
    RootSchemaName = "Contact",
    /* Добавление колонок в запрос. */
    Columns = new SelectQueryColumns()
    {
        /* Коллекция колонок. */
        Items = new Dictionary<string, SelectQueryColumn>()
        {
            /* Колонка [ФИО]. */
            {
                /* Ключ. */
                "Name",
                /* Значение. */
                new SelectQueryColumn()
                {
                    /* Выражение, задающее тип колонки. */
                    Expression = new ColumnExpression()
                    {
                        /* Тип выражения – колонка схемы. */
                        ExpressionType = EntitySchemaQueryExpressionType.SchemaColumn,
                        /* Путь к колонке. */
                        ColumnPath = "Name"
                    }
                }
            },
            /* Колонка [Количество активностей]. */
```

```

        {
            "ActivitiesCount",
            new SelectQueryColumn()
            {
                Expression = new ColumnExpression()
                {
                    /* Тип выражения – вложенный запрос. */
                    ExpressionType = EntitySchemaQueryExpressionType.SubQuery,
                    /* Путь к колонке относительно корневой схемы. */
                    ColumnPath = "[Activity:Contact].Id",
                    /* Тип функции – агрегирующая. */
                    FunctionType = FunctionType.Aggregation,
                    /* Тип агрегации – количество. */
                    AggregationType = AggregationType.Count
                }
            }
        }
    }
};

```

Чтобы **реализовать фильтры**:

1. Создайте экземпляр класса коллекции фильтров `Filters`.
2. Заполните необходимые свойства значениями.
3. В свойство `Filters` экземпляра класса запроса, созданного на предыдущем шаге, передайте ссылку на этот экземпляр.

Пример реализации класса коллекции фильтров

```

/* Фильтры запроса. */
var selectFilters = new Filters()
{
    /* Тип фильтра – группа. */
    FilterType = Terrasoft.Nui.ServiceModel.DataContract.FilterType.FilterGroup,
    /* Коллекция фильтров. */
    Items = new Dictionary<string, Filter>
    {
        /* Реализация фильтров. */
    }
};
/* Добавление фильтров в запрос. */
selectQuery.Filters = selectFilters;
/* Сериализация экземпляра класса запроса на чтение в JSON-строку. */
var json = new JavaScriptSerializer().Serialize(selectQuery);

```


Свойство `Items` должно содержать коллекцию типа "ключ-значение". В качестве ключа указывается строка, содержащая название фильтра, а в качестве значения — экземпляр класса `Filters`, содержащий непосредственную реализацию фильтра.

4. Реализуйте фильтр, обеспечивающий выбор только тех контактов, у которых количество активностей попадает в диапазон от 1 до 3. Для этого добавьте экземпляр в коллекцию фильтров.

Пример нового экземпляра коллекции фильтров

```
/* Фильтрация по активностям. */
{
    /* Ключ. */
    "FilterActivities",
    /* Значение. */
    new Filter
    {
        /* Тип фильтра – фильтр диапазона. */
        FilterType = Terrasoft.Nui.ServiceModel.DataContract.FilterType.Between,
        /* Тип сравнения – диапазон. */
        ComparisonType = FilterComparisonType.Between,
        /* Выражение, подлежащее проверке. */
        LeftExpression = new BaseExpression()
        {
            /* Тип выражения – вложенный запрос. */
            ExpressionType = EntitySchemaQueryExpressionType.SubQuery,
            /* Путь к колонке относительно корневой схемы. */
            ColumnPath = "[Activity:Contact].Id",
            /* Тип функции – агрегирующая. */
            FunctionType = FunctionType.Aggregation,
            /* Тип агрегации – количество. */
            AggregationType = AggregationType.Count
        },
        /* Конечное выражение диапазона фильтрации. */
        RightGreaterExpression = new BaseExpression()
        {
            /* Тип выражения – параметр. */
            ExpressionType = EntitySchemaQueryExpressionType.Parameter,
            /* Параметр выражения. */
            Parameter = new Parameter()
            {
                /* Тип данных параметра – целое число. */
                DataValueType = DataValueType.Integer,
                /* Значение параметра. */
                Value = 3
            }
        },
        /* Начальное выражение диапазона фильтрации. */
        RightLessExpression = new BaseExpression()
```

```

        {
            ExpressionType = EntitySchemaQueryExpressionType.Parameter,
            Parameter = new Parameter()
            {
                DataValueType = DataValueType.Integer,
                Value = 1
            }
        }
    }
}

```

5. Реализуйте фильтр, обеспечивающий выбор только тех контактов, у которых значение колонки [ФИО] ([Name]) начинается с символа "Ч". Для этого добавьте экземпляр в коллекцию фильтров.

Пример нового экземпляра коллекции фильтров

```

/* Фильтрация по имени. */
{
    /* Ключ. */
    "FilterName",
    /* Значение. */
    new Filter
    {
        /* Тип фильтра – фильтр сравнения. */
        FilterType = Terrasoft.Nui.ServiceModel.DataContract.FilterType.CompareFilter,
        /* Тип сравнения – начинается выражением. */
        ComparisonType = FilterComparisonType.StartsWith,
        /* Выражение, подлежащее проверке. */
        LeftExpression = new BaseExpression()
        {
            /* Тип выражения – колонка схемы. */
            ExpressionType = EntitySchemaQueryExpressionType.SchemaColumn,
            /* Путь к колонке. */
            ColumnPath = "Name"
        },
        /* Выражение фильтрации. */
        RightExpression = new BaseExpression()
        {
            /* Тип выражения – параметр. */
            ExpressionType = EntitySchemaQueryExpressionType.Parameter,
            /* Параметр выражения. */
            Parameter = new Parameter()
            {
                /* Тип данных параметра – текст. */
                DataValueType = DataValueType.Text,
                /* Значение параметра. */
                Value = "Ч"
            }
        }
    }
}

```

```

    }
  }
}

```

Отфильтровать записи раздела с использованием макроса



Пример. Создать консольное приложение, которое, используя класс `SelectQuery`, прочитает записи раздела [*Контакты*] ([*Contacts*]). Отобразить колонки:

- [*Id*].
- [*ФИО*] ([*Name*]).
- [*Дата рождения*] ([*Birthday*]).

Отфильтровать данные таким образом, чтобы были прочитаны только те контакты, у которых год рождения равен "1992".

Реализовать фильтр с макросом

Предварительно выполните пример чтения записей в стороннем приложении, описанный в статье [Добавить кнопку для чтения записей раздела \[Контакты\]](#).

Результат реализации экземпляра класса запроса на чтение записей в сокращенной форме представлен ниже.

`SelectQuery()`

```

/* Экземпляр класса запроса. */
var selectQuery = new SelectQuery()
{
    /* Название корневой схемы. */
    RootSchemaName = "Contact",
    /* Добавление колонок в запрос. */
    Columns = new SelectQueryColumns()
    {
        /* Коллекция колонок. */
        Items = new Dictionary<string, SelectQueryColumn>()
        {
            /* Колонка [ФИО]. */
            {
                /* Ключ. */
                "Name",

```

```

        /* Значение. */
        new SelectQueryColumn()
        {
            /* Выражение, задающее тип колонки. */
            Expression = new ColumnExpression()
            {
                /* Тип выражения – колонка схемы. */
                ExpressionType = EntitySchemaQueryExpressionType.SchemaColumn,
                /* Путь к колонке. */
                ColumnPath = "Name"
            }
        },
        /* Колонка [Количество активностей]. */
        {
            "ActivitiesCount",
            new SelectQueryColumn()
            {
                Expression = new ColumnExpression()
                {
                    /* Тип выражения – вложенный запрос. */
                    ExpressionType = EntitySchemaQueryExpressionType.SubQuery,
                    /* Путь к колонке относительно корневой схемы. */
                    ColumnPath = "[Activity:Contact].Id",
                    /* Тип функции – агрегирующая. */
                    FunctionType = FunctionType.Aggregation,
                    /* Тип агрегации – количество. */
                    AggregationType = AggregationType.Count
                }
            }
        }
    }
};

```

Чтобы **реализовать фильтр с макросом**:

1. Создайте экземпляр класса коллекции фильтров `Filters`.
2. Заполните необходимые свойства значениями.
3. В свойство `Filters` экземпляра класса запроса, созданного на предыдущем шаге, передайте ссылку на этот экземпляр.

Пример реализации класса коллекции фильтров

```

/* Фильтры запроса. */
var selectFilters = new Filters()
{

```

```

/* Тип фильтра – группа. */
FilterType = Terrasoft.Nui.ServiceModel.DataContract.FilterType.FilterGroup,
/* Коллекция фильтров. */
Items = new Dictionary<string, Filter>
{

    /* Фильтрация по году рождения. */
    {
        /* Ключ. */
        "FilterYear",
        /* Значение. */
        new Filter
        {
            /* Тип фильтра – фильтр сравнения. */
            FilterType = Terrasoft.Nui.ServiceModel.DataContract.FilterType.CompareFilter
            /* Тип сравнения – равенство. */
            ComparisonType = FilterComparisonType.Equal,
            /* Выражение, подлежащее проверке. */
            LeftExpression = new BaseExpression()
            {
                /* Тип выражения – колонка схемы. */
                ExpressionType = EntitySchemaQueryExpressionType.SchemaColumn,
                /* Путь к схеме. */
                ColumnPath = "BirthDate"
            },
            /* Выражение, с которым сравнивается проверяемое значение. */
            RightExpression = new BaseExpression
            {
                /* Тип выражения – функция. */
                ExpressionType = EntitySchemaQueryExpressionType.Function,
                /* Тип функции – макрос. */
                FunctionType = FunctionType.Macros,
                /* Тип макроса – год. */
                MacroType = EntitySchemaQueryMacroType.Year,
                /* Аргумент функции. */
                FunctionArgument = new BaseExpression
                {
                    /* Тип выражения, определяющего аргумент – параметр. */
                    ExpressionType = EntitySchemaQueryExpressionType.Parameter,
                    /* Инициализация параметра. */
                    Parameter = new Parameter
                    {
                        /* Тип параметра – целое число. */
                        DataValueType = DataValueType.Integer,
                        /* Значение параметра. */
                        Value = "1992"
                    }
                }
            }
        }
    }
}

```

```

    }
  }
}
};
/* Добавление фильтров в запрос. */
selectQuery.Filters = selectFilters;
/* Сериализация экземпляра класса запроса на чтение в JSON-строку. */
var json = new JavaScriptSerializer().Serialize(selectQuery);

```

Коллекция содержит единственный фильтр с ключом `FilterYear`. Поскольку из коллекции записей необходимо выбрать только те, у которых год рождения контакта равен 1992, то тип фильтра устанавливается как фильтр сравнения, а тип сравнения — равенство значений. В качестве выражения, подлежащего проверке, устанавливается значение колонки `[Дата рождения]` (`[Birthday]`), а в качестве выражения, с которым сравнивается проверяемое выражение — функция-макрос.

В данном случае использовать макрос удобно потому, что дата рождения хранится в базе данных в формате `гггг-мм-дд`. Макрос автоматически определяет год из этого значения, следовательно, разработчику нет необходимости самостоятельно писать дополнительный программный код.

Поскольку макрос `EntitySchemaQueryMacrosType.Year` является параметрическим, то необходимо инициализировать свойство `FunctionArgument`, которому присваивается ссылка на экземпляр класса `BaseExpression`. В нем и определяется целочисленный параметр со значением "1992".

Отфильтровать записи раздела по источнику вхождения пользователя в роль



Пример. Создать консольное приложение, которое, используя класс `SelectQuery`, прочитает записи раздела `[Контакты]` (`[Contacts]`). Отобразить колонки:

- `[Id]`.
- `[ФИО]` (`[Name]`).

Отфильтровать данные по источнику вхождения в роль таким образом, чтобы пользователь увидел все записи, за исключением тех, права на которые он только унаследовал от своих подчиненных.

Реализовать фильтры по источнику вхождения пользователя в роль

Для использования свойства `adminUnitRoleSources` необходимо добавить директиву

```
using Terrasoft.Common; .
```

Предварительно выполните пример чтения записей в стороннем приложении, описанный в статье [Прочитать записи раздела \[Контакты\]](#).

Результат реализации экземпляра класса запроса на чтение записей в сокращенной форме представлен ниже.

SelectQuery()

```
// Экземпляр класса запроса.
var selectQuery = new SelectQuery()
{
    // Название корневой схемы.
    RootSchemaName = "Contact",
    // Добавление колонок в запрос.
    Columns = new SelectQueryColumns()
    {
        // Коллекция колонок.
        Items = new Dictionary<string, SelectQueryColumn>()
        {
            // Колонка [ФИО].
            {
                // Ключ.
                "Name",
                // Значение.
                new SelectQueryColumn()
                {
                    // Выражение, задающее тип колонки.
                    Expression = new ColumnExpression()
                    {
                        // Тип выражения – колонка схемы.
                        ExpressionType = EntitySchemaQueryExpressionType.SchemaColumn,
                        // Путь к колонке.
                        ColumnPath = "Name"
                    }
                }
            },
        }
    }
};
```

Чтобы **реализовать фильтры по источнику вхождения пользователя в роль**:

Заполните значение свойства `adminUnitRoleSources` с помощью перечисления констант из JS-класса `Terrasoft.AdminUnitRoleSources` через побитовое **или** (`" | "`).

Реализация фильтров по источнику вхождения в роли

```
// Фильтр по источникам вхождения в роль.
selectQuery.adminUnitRoleSources = Terrasoft.AdminUnitRoleSources.ExplicitEntry
    | Terrasoft.AdminUnitRoleSources.Delegated
    | Terrasoft.AdminUnitRoleSources.FuncRoleFromOrgRole
    | Terrasoft.AdminUnitRoleSources.UpHierarchy;

// Сериализация экземпляра класса запроса на чтение в JSON-строку.
var json = new JavaScriptSerializer().Serialize(selectQuery);
```

Обновить запись раздела



Пример. Создать консольное приложение, которое, используя класс `UpdateQuery`, обновит контакт "Иванов Иван Иванович" раздела [*Контакты*] ([*Contacts*]). Для этой записи добавьте в колонку [*Email*] значение "i.ivanov@creatio.com".

1. Создать и настроить проект консольного приложения C#

Предварительно выполните пример чтения записей в стороннем приложении, описанный в статье [Добавить кнопку для чтения записей раздела \[Контакты\]](#).

Чтобы **создать и настроить проект консольного приложения C#**:

- Используя среду разработки Microsoft Visual Studio (версии не ниже 2017), создайте проект консольного приложения Visual C#.
Работа в Microsoft Visual Studio подробно описана в статье [Разработать C# код в пользовательском проекте](#).
- Укажите в качестве названия проекта, например, `DataServiceUpdateExample`.
- Свойству проекта `Target framework` установите значение ".NET Framework 4.7".
- В секцию `References` проекта добавьте зависимости от библиотек:
 - `System.Web.Extensions.dll` — библиотека классов, входящая в .NET Framework.
 - `Terrasoft.Core.dll` — библиотека основных классов серверного ядра приложения. Находится по пути `...\Terrasoft.WebApp\bin\Terrasoft.Core.dll`.
 - `Terrasoft.Nui.ServiceModel.dll` — библиотека классов служб приложения. Находится по пути `...\Terrasoft.WebApp\bin\Terrasoft.Nui.ServiceModel.dll`.
 - `Terrasoft.Common.dll` — библиотека основных классов серверного ядра приложения. Находится по пути `...\Terrasoft.WebApp\bin\Terrasoft.Common.dll`.
- В файл исходного кода приложения добавьте директивы `using`.

Добавление директив using

```
using System;
using System.Text;
using System.IO;
using System.Net;
using System.Collections.Generic;
using Terrasoft.Nui.ServiceModel.DataContract;
using Terrasoft.Core.Entities;
using System.Web.Script.Serialization;
using Terrasoft.Common;
```

2. Добавить объявление полей и констант

Добавление полей и констант в исходный код приложения необходимо выполнить для доступа к возможностям сервиса работы с данными DataService.

Добавление объявления полей и констант

```
/* Основной URL приложения Creatio. Необходимо заменить на пользовательский. */
private const string baseUrl = @"http://example.creatio.com";
/* Строка запроса к методу Login сервиса AuthService.svc. */
private const string authServiceUri = baseUrl + @"/ServiceModel/AuthService.svc/Login";
/* Строка пути запроса UpdateQuery. */
private const string updateQueryUri = baseUrl + @"/0/DataService/json/reply/UpdateQuery";
/* Cookie аутентификации Creatio. */
private static CookieContainer AuthCookie = new CookieContainer();
```

Здесь объявлены три строковых константных поля, с помощью которых формируются пути выполнения запросов на аутентификацию и запросов на чтение данных. Данные об аутентификации будут сохранены в поле [*AuthCookie*].

3. Добавить метод, выполняющий аутентификацию

Для доступа создаваемого приложения к сервису работы с данными DataService необходимо выполнить [аутентификацию](#).

4. Реализовать запрос на добавление записи

Поскольку объявленная ранее константа `updateQueryUri` содержит путь для отправки данных в формате JSON, то отправляемые данные необходимо предварительно сконфигурировать в виде строки, содержащей описание JSON-объекта, соответствующего контракту данных `UpdateQuery`. Это можно сделать непосредственно в строчной переменной, однако намного удобнее и безопаснее с точки зрения

возможности возникновения ошибок создать экземпляр класса `UpdateQuery`, заполнить его свойства, а затем сериализовать его в строку.

Реализация запроса на добавление записи

```
/* Экземпляр класса запроса. */
var updateQuery = new UpdateQuery()
{
    /* Название корневой схемы. */
    RootSchemaName = "Contact",
    /* Новые значения колонок. */
    ColumnValues = new ColumnValues()
    {
        /* Коллекция ключ-значение. */
        Items = new Dictionary<string, ColumnExpression>()
        {
            /* Колонка Email. */
            {
                /* Ключ. */
                "Email",
                /* Значение – экземпляр класса выражения запроса к схеме объекта.
                Конфигурирование колонки [Email]. */
                new ColumnExpression()
                {
                    /* Тип выражения запроса к схеме объекта – параметр. */
                    ExpressionType = EntitySchemaQueryExpressionType.Parameter,
                    /* Параметр выражения запроса. */
                    Parameter = new Parameter()
                    {
                        /* Значение параметра. */
                        Value = "i.ivanov@creatio.com",
                        /* Тип данных параметра – строка. */
                        DataValueType = DataValueType.Text
                    }
                }
            }
        }
    },
    /* Фильтры запроса. */
    Filters = new Filters()
    {
        /* Тип фильтра – группа. */
        FilterType = Terrasoft.Nui.ServiceModel.DataContract.FilterType.FilterGroup,
        /* Коллекция фильтров. */
        Items = new Dictionary<string, Filter>()
        {
            /* Фильтрация по имени. */
            {
```

```

/* Ключ. */
"FilterByName",
/* Значение. */
new Filter
{
    /* Тип фильтра – фильтр сравнения. */
    FilterType = Terrasoft.Nui.ServiceModel.DataContract.FilterType.CompareFilter
    /* Тип сравнения – начинается выражением. */
    ComparisonType = FilterComparisonType.Equal,
    /* Выражение, подлежащее проверке. */
    LeftExpression = new BaseExpression()
    {
        /* Тип выражения – колонка схемы. */
        ExpressionType = EntitySchemaQueryExpressionType.SchemaColumn,
        /* Путь к колонке. */
        ColumnPath = "Name"
    },
    /* Выражение фильтрации. */
    RightExpression = new BaseExpression()
    {
        /* Тип выражения – параметр. */
        ExpressionType = EntitySchemaQueryExpressionType.Parameter,
        /* Параметр выражения. */
        Parameter = new Parameter()
        {
            /* Тип данных параметра – текст. */
            DataValueType = DataValueType.Text,
            /* Значение параметра. */
            Value = "Иванов Иван Иванович"
        }
    }
}
}
}
}
}
};
/* Сериализация экземпляра класса запроса на добавление в JSON-строку. */
var json = new JavaScriptSerializer().Serialize(updateQuery);

```

Здесь создается экземпляр класса `UpdateQuery`, в свойстве `ColumnValues` которого устанавливается значение `"i.ivanov@creatio.com"` для колонки `[Email]`. Чтобы это значение было применено только для определенной записи или группы записей, необходимо свойству `Filters` присвоить значение ссылки на корректно инициализированный экземпляр класса `Filters`. В данном случае в коллекцию фильтров добавлен единственный фильтр, который отбирает только те записи, у которых значение колонки `[ФИО]` (`[Name]`) равно `"Иванов Иван Иванович"`.

5. Выполнить POST-запрос к DataService

1. Создайте экземпляр класса [HttpWebRequest](#).
2. Заполните свойства экземпляра.
3. Присоедините к запросу созданную ранее строку с JSON-объектом.
4. Выполните и обработайте результат запроса к сервису работы с данными DataService.

Реализация POST -запроса

```

/* Преобразование строки JSON-объекта в массив байтов. */
byte[] jsonArray = Encoding.UTF8.GetBytes(json);
/* Создание экземпляра HTTP-запроса. */
var updateRequest = HttpWebRequest.Create(updateQueryUri) as HttpWebRequest;
/* Определение метода запроса. */
updateRequest.Method = "POST";
/* Определение типа содержимого запроса. */
updateRequest.ContentType = "application/json";
/* Добавление полученных ранее аутентификационных cookie в запрос на получение данных. */
updateRequest.CookieContainer = AuthCookie;
/* Установить длину содержимого запроса. */
updateRequest.ContentLength = jsonArray.Length;

/* Добавление CSRF токена в заголовок запроса. */
CookieCollection cookieCollection = AuthCookie.GetCookies(new Uri(authServiceUri));
string csrfToken = cookieCollection["BPMCSRF"].Value;
updateRequest.Headers.Add("BPMCSRF", csrfToken);

/* Помещение JSON-объекта в содержимое запроса. */
using (var requestStream = updateRequest.GetRequestStream())
{
    requestStream.Write(jsonArray, 0, jsonArray.Length);
}
/* Выполнение HTTP-запроса и получение ответа от сервера. */
using (var response = (HttpWebResponse)updateRequest.GetResponse())
{
    /* Вывод ответа в консоль. */
    using (StreamReader reader = new StreamReader(response.GetResponseStream()))
    {
        Console.WriteLine(reader.ReadToEnd());
    }
}

```

Удалить запись раздела



Сложный

Пример. Создать консольное приложение, которое, используя класс `DeleteQuery`, удалит контакт "Иванов Иван Иванович" раздела [*Контакты*] ([*Contacts*]).

1. Создать и настроить проект консольного приложения C#

Предварительно выполните пример чтения записей в стороннем приложении, описанный в статье [Добавить кнопку для чтения записей раздела \[Контакты\]](#).

Чтобы **создать и настроить проект консольного приложения C#**:

1. Используя среду разработки Microsoft Visual Studio (версии не ниже 2017), создайте проект консольного приложения Visual C#.
Работа в Microsoft Visual Studio подробно описана в статье [Разработать C# код в пользовательском проекте](#).
2. Укажите в качестве названия проекта, например, `DataServiceDeleteExample`.
3. Свойству проекта `Target framework` установите значение ".NET Framework 4.7".
4. В секцию `References` проекта добавьте зависимости от библиотек:
 - `System.Web.Extensions.dll` — библиотека классов, входящая в .NET Framework.
 - `Terrasoft.Core.dll` — библиотека основных классов серверного ядра приложения. Находится по пути `...\Terrasoft.WebApp\bin\Terrasoft.Core.dll`.
 - `Terrasoft.Nui.ServiceModel.dll` — библиотека классов служб приложения. Находится по пути `...\Terrasoft.WebApp\bin\Terrasoft.Nui.ServiceModel.dll`.
 - `Terrasoft.Common.dll` — библиотека основных классов серверного ядра приложения. Находится по пути `...\Terrasoft.WebApp\bin\Terrasoft.Common.dll`.
5. В файл исходного кода приложения добавьте директивы `using`.

Добавление директив `using`

```
using System;
using System.Text;
using System.IO;
using System.Net;
using System.Collections.Generic;
using Terrasoft.Nui.ServiceModel.DataContract;
using Terrasoft.Core.Entities;
using System.Web.Script.Serialization;
using Terrasoft.Common;
```

2. Добавить объявление полей и констант

Добавление полей и констант в исходный код приложения необходимо выполнить для доступа к возможностям сервиса работы с данными DataService.

Добавление объявления полей и констант

```
/* Основной URL приложения Creatio. Необходимо заменить на пользовательский. */
private const string baseUrl = @"http://example.creatio.com";
/* Строка запроса к методу Login сервиса AuthService.svc. */
private const string authServiceUri = baseUrl + @"/ServiceModel/AuthService.svc/Login";
/* Строка пути запроса DeleteQuery. */
private const string deleteQueryUri = baseUrl + @"/0/DataService/json/reply/DeleteQuery";
/* Cookie аутентификации Creatio. */
private static CookieContainer AuthCookie = new CookieContainer();
```

Здесь объявлены три строковых константных поля, с помощью которых формируются пути выполнения запросов на аутентификацию и запросов на удаление данных. Данные об аутентификации будут сохранены в поле [*AuthCookie*].

3. Добавить метод, выполняющий аутентификацию

Для доступа создаваемого приложения к сервису работы с данными DataService необходимо выполнить [аутентификацию](#).

4. Реализовать запрос на удаление записи

Поскольку объявленная ранее константа `deleteQueryUri` содержит путь для отправки данных в формате JSON, то отправляемые данные необходимо предварительно сконфигурировать в виде строки, содержащей описание JSON-объекта, соответствующего контракту данных `DeleteQuery`. Это можно сделать непосредственно в строчной переменной, однако намного удобнее и безопаснее с точки зрения возможности возникновения ошибок создать экземпляр класса `DeleteQuery`, заполнить его свойства, а затем сериализовать его в строку.

Реализация запроса на удаление записи

```
/* Экземпляр класса запроса. */
var deleteQuery = new DeleteQuery()
{
    /* Название корневой схемы. */
    RootSchemaName = "Contact",
    /* Фильтры запроса. */
    Filters = new Filters()
    {
        /* Тип фильтра – группа. */
        FilterType = Terrasoft.Nui.ServiceModel.DataContract.FilterType.FilterGroup,
        /* Коллекция фильтров. */
```

```

Items = new Dictionary<string, Filter>()
{
    /* Фильтрация по имени. */
    {
        /* Ключ. */
        "FilterByName",
        /* Значение. */
        new Filter
        {
            /* Тип фильтра – фильтр сравнения. */
            FilterType = Terrasoft.Nui.ServiceModel.DataContract.FilterType.CompareFilter,
            /* Тип сравнения – начинается выражением. */
            ComparisonType = FilterComparisonType.Equal,
            /* Выражение, подлежащее проверке. */
            LeftExpression = new BaseExpression()
            {
                /* Тип выражения – колонка схемы. */
                ExpressionType = EntitySchemaQueryExpressionType.SchemaColumn,
                /* Путь к колонке. */
                ColumnPath = "Name"
            },
            /* Выражение фильтрации. */
            RightExpression = new BaseExpression()
            {
                /* Тип выражения – параметр. */
                ExpressionType = EntitySchemaQueryExpressionType.Parameter,
                /* Параметр выражения. */
                Parameter = new Parameter()
                {
                    /* Тип данных параметра – текст. */
                    DataValueType = DataValueType.Text,
                    /* Значение параметра. */
                    Value = "Иванов Иван Иванович"
                }
            }
        }
    }
}

};
/* Сериализация экземпляра класса запроса на добавление в JSON-строку. */
var json = new JavaScriptSerializer().Serialize(updateQuery);

```

Здесь создается экземпляр класса `DeleteQuery`, в свойстве `RootSchemaName` которого устанавливается значение `Contact`. Для того чтобы удалена была только определенная запись или группа записей, необходимо свойству `Filters` присвоить значение ссылки на корректно инициализированный экземпляр класса `Filters`. В данном случае в коллекцию фильтров добавлен единственный фильтр, который

отбирает только те записи, у которых значение колонки [ФИО] ([Name]) равно "Иванов Иван Иванович".

5. Выполнить POST-запрос к DataService

1. Создайте экземпляр класса [HttpWebRequest](#).
2. Заполните свойства экземпляра.
3. Присоедините к запросу созданную ранее строку с JSON-объектом.
4. Выполните и обработайте результат запроса к сервису работы с данными DataService.

Реализация POST -запроса

```

/* Преобразование строки JSON-объекта в массив байтов. */
byte[] jsonArray = Encoding.UTF8.GetBytes(json);
/* Создание экземпляра HTTP-запроса. */
var updateRequest = HttpWebRequest.Create(updateQueryUri) as HttpWebRequest;
/* Определение метода запроса. */
updateRequest.Method = "POST";
/* Определение типа содержимого запроса. */
updateRequest.ContentType = "application/json";
/* Добавление полученных ранее аутентификационных cookie в запрос на получение данных. */
updateRequest.CookieContainer = AuthCookie;
/* Установить длину содержимого запроса. */
updateRequest.ContentLength = jsonArray.Length;

/* Добавление CSRF токена в заголовок запроса. */
CookieCollection cookieCollection = AuthCookie.GetCookies(new Uri(authServiceUri));
string csrfToken = cookieCollection["BPMCSRF"].Value;
updateRequest.Headers.Add("BPMCSRF", csrfToken);

/* Помещение JSON-объекта в содержимое запроса. */
using (var requestStream = updateRequest.GetRequestStream())
{
    requestStream.Write(jsonArray, 0, jsonArray.Length);
}
/* Выполнение HTTP-запроса и получение ответа от сервера. */
using (var response = (HttpWebResponse)updateRequest.GetResponse())
{
    /* Вывод ответа в консоль. */
    using (StreamReader reader = new StreamReader(response.GetResponseStream()))
    {
        Console.WriteLine(reader.ReadToEnd());
    }
}

```


Добавить и изменить запись раздела



Пример. Создать консольное приложение, которое, используя класс `BatchQuery` :

- Добавить в раздел [*Контакты*] ([*Contacts*]) запись, которая в колонке [*ФИО*] ([*Name*]) содержит значение "Петров Петр Петрович".
- Изменит значение колонки [*Рабочий телефон*] ([*Phone*]) на "012 345 67 89" для всех записей раздела [*Контакты*] ([*Contacts*]), у которых колонка [*ФИО*] ([*Name*]) содержит значение "Петров Петр Петрович".

1. Создать и настроить проект консольного приложения C#

1. Используя среду разработки Microsoft Visual Studio (версии не ниже 2017), создайте проект консольного приложения Visual C#.
Работа в Microsoft Visual Studio подробно описана в статье [Разработать C# код в пользовательском проекте](#).
2. Укажите в качестве названия проекта, например, `DataServiceDeleteExample`.
3. Свойству проекта `Target framework` установите значение ".NET Framework 4.7".
4. В секцию `References` проекта добавьте зависимости от библиотек:
 - `System.Web.Extensions.dll` — библиотека классов, входящая в .NET Framework.
 - `Terrasoft.Core.dll` — библиотека основных классов серверного ядра приложения. Находится по пути `...\Terrasoft.WebApp\bin\Terrasoft.Core.dll`.
 - `Terrasoft.Nui.ServiceModel.dll` — библиотека классов служб приложения. Находится по пути `...\Terrasoft.WebApp\bin\Terrasoft.Nui.ServiceModel.dll`.
 - `Terrasoft.Common.dll` — библиотека основных классов серверного ядра приложения. Находится по пути `...\Terrasoft.WebApp\bin\Terrasoft.Common.dll`.
5. В файл исходного кода приложения добавьте директивы `using`.

Добавление директив `using`

```
using System;
using System.Text;
using System.IO;
using System.Net;
using System.Collections.Generic;
using Terrasoft.Nui.ServiceModel.DataContract;
using Terrasoft.Core.Entities;
```

```
using System.Web.Script.Serialization;
using Terrasoft.Common;
```

2. Добавить объявление полей и констант

Добавление полей и констант в исходный код приложения необходимо выполнить для доступа к возможностям сервиса работы с данными DataService.

Добавление объявления полей и констант

```
/* Основной URL приложения Creatio. Необходимо заменить на пользовательский. */
private const string baseUrl = @"http://example.creatio.com";
/* Строка запроса к методу Login сервиса AuthService.svc. */
private const string authServiceUri = baseUrl + @"/ServiceModel/AuthService.svc/Login";
/* Строка пути запроса BatchQuery. */
private const string batchQueryUri = baseUrl + @"/0/DataService/json/reply/BatchQuery";
/* Cookie аутентификации Creatio. */
private static CookieContainer AuthCookie = new CookieContainer();
```

Здесь объявлены три строковых константных поля, с помощью которых формируются пути выполнения запросов на аутентификацию и запросов на чтение данных. Данные об аутентификации будут сохранены в поле [*AuthCookie*].

3. Добавить метод, выполняющий аутентификацию

Для доступа создаваемого приложения к сервису работы с данными DataService необходимо выполнить [аутентификацию](#).

4. Реализовать запрос на добавление записи

Поскольку объявленная ранее константа `batchQueryUri` содержит путь для отправки данных в формате JSON, то отправляемые данные необходимо предварительно сконфигурировать в виде строки, содержащей описание JSON-объекта. Для формирования содержимого единичных запросов удобно воспользоваться классами-контрактами данных, а затем сериализовать их в строку.

Реализация запроса на добавление записи

```
/* Запрос на добавление данных. */
var insertQuery = new InsertQuery()
{
    RootSchemaName = "Contact",
    ColumnValues = new ColumnValues()
    {
        Items = new Dictionary<string, ColumnExpression>()
```

```

    {
        {
            "Name",
            new ColumnExpression()
            {
                ExpressionType = EntitySchemaQueryExpressionType.Parameter,
                Parameter = new Parameter()
                {
                    Value = "Петров Петр Петрович",
                    DataValueType = DataValueType.Text
                }
            }
        }
    }
}
};

```

5. Реализовать запрос на изменение записи

Измените значение колонки [*Рабочий телефон*] на "012 345 67 89" для всех записей раздела [*Контакты*] ([*Contacts*]), которые в колонке [*ФИО*] ([*Name*]) содержат значение "Петров Петр Петрович".

Реализация запроса на изменение записи

```

/* Запрос на обновление данных. */
var updateQuery = new UpdateQuery()
{
    RootSchemaName = "Contact",
    ColumnValues = new ColumnValues()
    {
        Items = new Dictionary<string, ColumnExpression>()
        {
            {
                "Phone",
                new ColumnExpression()
                {
                    ExpressionType = EntitySchemaQueryExpressionType.Parameter,
                    Parameter = new Parameter()
                    {
                        Value = "0123456789",
                        DataValueType = DataValueType.Text
                    }
                }
            }
        }
    },
    Filters = new Filters()

```

```

{
    FilterType = Terrasoft.Nui.ServiceModel.DataContract.FilterType.FilterGroup,
    Items = new Dictionary<string, Filter>()
    {
        {
            "FilterByName",
            new Filter
            {
                FilterType = Terrasoft.Nui.ServiceModel.DataContract.FilterType.CompareFilter,
                ComparisonType = FilterComparisonType.Equal,
                LeftExpression = new BaseExpression()
                {
                    ExpressionType = EntitySchemaQueryExpressionType.SchemaColumn,
                    ColumnPath = "Name"
                },
                RightExpression = new BaseExpression()
                {
                    ExpressionType = EntitySchemaQueryExpressionType.Parameter,
                    Parameter = new Parameter()
                    {
                        DataValueType = DataValueType.Text,
                        Value = "Петров Петр Петрович"
                    }
                }
            }
        }
    }
};

```

6. Реализовать пакетный запрос

1. После сериализации созданных экземпляров классов запросов, в строки с JSON-объектами добавьте сведения о полном квалифицированном имени типа соответствующего контракта данных.
2. Сформируйте строку, содержащую пакетный запрос.

Пакетный запрос

```

/* Сериализация экземпляра класса запроса на добавление в JSON-строку. */
var jsonInsert = new JavaScriptSerializer().Serialize(insertQuery);
/* Вставка типа запроса в в JSON-строку. */
jsonInsert = jsonInsert.Insert(1, @"\"__type\": \"Terrasoft.Nui.ServiceModel.DataContract.Ins
/* Сериализация экземпляра класса запроса на обновление в JSON-строку. */
var jsonUpdate = new JavaScriptSerializer().Serialize(updateQuery);
/* Вставка типа запроса в в JSON-строку. */
jsonUpdate = jsonUpdate.Insert(1, @"\"__type\": \"Terrasoft.Nui.ServiceModel.DataContract.Upd

```

```
/* Формирование пакетного запроса. */
var json = @"{"items": [" + jsonInsert + "," + jsonUpdate + "]}";
```

7. Реализовать пакетный запрос

1. Создайте экземпляр класса [HttpWebRequest](#).
2. Заполните свойства экземпляра.
3. Присоедините к запросу созданную ранее строку с JSON-объектом.
4. Выполните и обработайте результат запроса к сервису работы с данными DataService.

Реализация POST -запроса

```
/* Преобразование строки JSON-объекта в массив байтов. */
byte[] jsonArray = Encoding.UTF8.GetBytes(json);
/* Создание экземпляра HTTP-запроса. */
var batchRequest = HttpWebRequest.Create(deleteQueryUri) as HttpWebRequest;
/* Определение метода запроса. */
batchRequest.Method = "POST";
/* Определение типа содержимого запроса. */
batchRequest.ContentType = "application/json";
/* Добавление полученных ранее аутентификационных cookie в запрос на получение данных. */
batchRequest.CookieContainer = AuthCookie;
/* Установить свойство ContentLength запроса. */
batchRequest.ContentLength = jsonArray.Length;

/* Добавление токена BPMCSRF в запрос. */
CookieCollection cookieCollection = AuthCookie.GetCookies(new Uri(authServiceUri));
string csrfToken = cookieCollection["BPMCSRF"].Value;
batchRequest.Headers.Add("BPMCSRF", csrfToken);

/* Помещение в содержимое запроса JSON-объекта. */
using (var requestStream = batchRequest.GetRequestStream())
{
    requestStream.Write(jsonArray, 0, jsonArray.Length);
}
/* Выполнение HTTP-запроса и получение ответа от сервера. */
using (var response = (HttpWebResponse)batchRequest.GetResponse())
{
    /* Вывод ответа в консоль. */
    using (StreamReader reader = new StreamReader(response.GetResponseStream()))
    {
        Console.WriteLine(reader.ReadToEnd());
    }
}
/* Задержка выполнения приложения. */
```

```
Console.ReadKey();
```

Класс InsertQuery C#



Пространство имен `Terrasoft.Nui.ServiceModel.DataContract` библиотеки классов

`Terrasoft.Nui.ServiceModel.dll`.

Назначение класса `InsertQuery` — добавление записей в раздел. Передача данных непосредственно в сервис работы с данными DataService осуществляется по HTTP протоколу при помощи `POST`-запроса по URL.

Структура запроса на добавление данных

```
/* Формат URL для POST-запроса к DataService на добавление данных. */
http(s)://[Адрес приложения Creatio]/[Номер конфигурации]/dataservice/[Формат данных]/reply/InsertQuery
```

Пример запроса на добавление данных

```
/* Пример URL для POST-запроса к DataService на добавление данных. */
http(s)://example.creatio.com/0/dataservice/json/reply/InsertQuery
```

Контракт данных `InsertQuery` имеет иерархическую структуру с несколькими уровнями вложенности. Для простоты восприятия иерархическую структуру контракта данных `InsertQuery` удобно представить в формате объекта JSON.

Структура контракта данных `InsertQuery`

```
{
  "RootSchemaName": "[Имя корневой схемы объекта]",
  "OperationType": [Тип операции с записью],
  "QueryId": [Идентификатор запроса],
  "QueryKind": [Информация о запросе для DBExecutor],
  "IncludeProcessExecutionData": [Данные о выполнении процесса],
  "ColumnValues": {
    "Items": {
      "Имя добавляемой колонки": {
        "ExpressionType": [Тип выражения],
        "Parameter": {
          "DataValueType": [Тип данных],
          "Value": "[Значение колонки]"
        }
      }
    }
  }
}
```

```

    }...
  }
}
}

```

На заметку. Полный перечень свойств класса `InsertQuery` и его родительских классов можно найти в [Библиотеке .NET классов](#).

Свойства

`QueryId` `string`

Идентификатор запроса.

`QueryKind` `QueryKind`

Дополнительная информация о запросе, которая может быть использована для отправки запроса [DBExecutor](#).

Возможные значения (`QueryKind`)

| | |
|---------|--|
| General | Используется, как значение по умолчанию. |
| Limited | Используется, чтобы показать, что запрос может быть отправлен исполнителю с ограниченными ресурсами. |

`ColumnValues` `ColumnValues`

Содержит коллекцию значений колонок добавляемой записи. Унаследовано от родительского класса `BaseQuery`. Имеет тип `ColumnValues`, определенный в пространстве имен `Terrasoft.Nui.ServiceModel.DataContract`.

`RootSchemaName` `string`

Строка, которая содержит название корневой схемы объекта добавляемой записи.

`OperationType` `QueryOperationType`

Тип операции с записью. Задается значением перечисления `QueryOperationType` пространства имен `Terrasoft.Nui.ServiceModel.DataContract`. Для `InsertQuery` устанавливается значение `QueryOperationType.Insert`.

Возможные значения (`QueryOperationType`)

| | |
|--------|---|
| Select | 0 |
| Insert | 1 |
| Update | 2 |
| Delete | 3 |
| Batch | 4 |

`IncludeProcessExecutionData` `bool`

Флаг, который включает данные о выполнении процесса.

Класс SelectQuery C#

 Сложный

Класс SelectQuery C#

Пространство имен `Terrasoft.Nui.ServiceModel.DataContract` библиотеки классов

`Terrasoft.Nui.ServiceModel.dll`.

Назначение класса `SelectQuery` — чтение записей раздела. Передача данных непосредственно в сервис работы с данными `DataService` осуществляется по HTTP протоколу при помощи `POST`-запроса по URL.

Структура запроса на чтение данных

```
/* Формат URL для POST-запроса к DataService на чтение данных. */
http(s)://[Адрес приложения Creatio]/[Номер конфигурации]/dataservice/[Формат данных]/reply/Sele
```

Пример запроса на чтение данных

```
/* Пример URL для POST-запроса к DataService на чтение данных. */
http(s)://example.creatio.com/0/dataservice/json/reply/SelectQuery
```

Контракт данных `SelectQuery` имеет иерархическую структуру с несколькими уровнями вложенности. Для простоты восприятия иерархическую структуру контракта данных `SelectQuery` удобно представить в формате объекта JSON.

Структура контракта данных SelectQuery

```

{
  "RootSchemaName": "[Имя корневой схемы объекта]",
  "OperationType": [Тип операции с записью],
  "QueryId": [Идентификатор запроса],
  "QueryKind": [Информация о запросе для DBExecutor],
  "IncludeProcessExecutionData": [Данные о выполнении процесса],
  "ColumnValues": [Значения колонок],
  "ChunkSize": [Количество сущностей в блоке],
  "IgnoreDisplayValues": [Использовать запрос для отображения значений столбцов],
  "QueryOptimize": [Использовать оптимизацию запросов],
  "QuerySource": [Источник запроса],
  "QueryType": [Тип запроса],
  "RowsOffset": [Количество пропущенных строк],
  "UseLocalization": [Использовать локализованные данные],
  "UseMetrics": [Использовать метрики для запроса],
  "UseRecordDeactivation": [Отключить данные при фильтрации],
  "Columns": {
    "Items": {
      "Name": {
        "OrderDirection": [Порядок сортировки],
        "OrderPosition": [Позиция колонки],
        "Caption": "[Заголовок]",
        "Expression": {
          "ExpressionType": [Тип выражения],
          "ColumnPath": "[Путь к колонке]",
          "Parameter": [Параметр],
          "FunctionType": [Тип функции],
          "MacroType": [Тип макроса],
          "FunctionArgument": [Аргумент функции],
          "DatePartType": [Тип части даты],
          "AggregationType": [Тип агрегации],
          "AggregationEvalType": [Область применения агрегации],
          "SubFilters": [Вложенные фильтры]
        }
      }
    }
  },
  "AllColumns": [Признак выбора всех колонок],
  "ServerESQCacheParameters": {
    "CacheLevel": [Уровень кеширования],
    "CacheGroup": [Группа кеширования],
    "CacheItemName": [Ключ записи в хранилище]
  },
  "IsPageable": [Признак разбиения на страницы],
  "IsDistinct": [Признак уникальности],
  "RowCount": [Количество выбираемых записей],

```

```

"ConditionalValues": [Условия для построения постраничного запроса],
"IsHierarchical": [Признак иерархической выборки данных],
"HierarchicalMaxDepth": [Максимальный уровень вложенности иерархического запроса],
"HierarchicalColumnName": [Имя колонки, используемой для построения иерархического запроса],
"HierarchicalColumnValue": [Начальное значение иерархической колонки],
"Filters": [Фильтры],
"AdminUnitRoleSources": [Источник вхождения пользователя в роль, который используется для фил
}

```

На заметку. Полный перечень свойств класса `InsertQuery` и его родительских классов можно найти в [Библиотеке .NET классов](#).

Свойства

`Filters` `Filters`

Коллекция фильтров запросов. Имеет тип `Filters`, определенный в пространстве имен `Terrasoft.Nui.ServiceModel.DataContract`.

`QueryId` `string`

Идентификатор запроса.

`QueryKind` `QueryKind`

Дополнительная информация о запросе, которая может быть использована для отправки запроса [DBExecutor](#).

Возможные значения (`QueryKind`)

| | |
|---------|--|
| General | Используется, как значение по умолчанию. |
| Limited | Используется, чтобы показать, что запрос может быть отправлен исполнителю с ограниченными ресурсами. |

`ColumnValues` `ColumnValues`

Значения колонок.

`RootSchemaName` `string`

Строка, содержащая название корневой схемы объекта добавляемой записи.

OperationType QueryOperationType

Тип операции с записью. Задается значением перечисления QueryOperationType пространства имен Terrasoft.Nui.ServiceModel.DataContract. Для SelectQuery устанавливается значение QueryOperationType.Select.

Возможные значения (QueryOperationType)

| | |
|--------|---|
| Select | 0 |
| Insert | 1 |
| Update | 2 |
| Delete | 3 |
| Batch | 4 |

IncludeProcessExecutionData bool

Флаг, который включает данные о выполнении процесса.

AdminUnitRoleSources AdminUnitRoleSources

Целочисленное свойство, которое соответствует критериям фильтрации записей по источнику вхождения пользователя в роли. Значение по умолчанию: 0. Перечисление AdminUnitRoleSources находится в пространстве имен Terrasoft.Common. Чтобы сформировать AdminUnitRoleSources, необходимо с помощью побитового или (" | ") перечислить константы из JS-класса Terrasoft.AdminUnitRoleSources. В результате, на серверной стороне запись будет возвращена, если у пользователя есть хоть одна роль, которой доступна запись, и пользователь входит в эту роль в соответствии с источниками, указанными в условиях фильтрации.

Возможные значения (AdminUnitRoleSources)

| | |
|---------------------|--|
| All | Получает все роли. |
| AsManager | Получает роль менеджера. |
| Delegated | Делегированная роль. |
| ExplicitEntry | Явное вхождение в роль. |
| FuncRoleFromOrgRole | Получает функциональную роль с организационной роли. |
| None | Пустое значение. |
| Self | Самостоятельно. |
| UpHierarchy | Возвращает роль вверх по иерархии. |

AllColumns `bool`

Признак выбора всех колонок. Если значение установлено как `true`, в результате выполнения запроса будут выбраны все колонки корневой схемы.

ChunkSize `int`

Количество сущностей в блоке.

Columns `SelectQueryColumns`

Содержит коллекцию колонок для считываемых записей. Имеет тип `SelectQueryColumns`, определенный в пространстве имен `Terrasoft.Nui.ServiceModel.DataContract`. Следует конфигурировать, если значение признака `AllColumns` установлено как `false` и нужен определенный набор колонок корневой схемы, не включающий колонку `[Id]`.

ConditionalValues `ColumnValues`

Условия для построения постраничного запроса. Тип `ColumnValues` определен в пространстве имен `Terrasoft.Nui.ServiceModel.DataContract`.

HierarchicalColumnName `string`

Имя колонки, использующейся для построения иерархического запроса.

HierarchicalColumnValue `string`

Начальное значение иерархической колонки, от которого будет строиться иерархия.

`HierarchicalMaxDepth int`

Максимальный уровень вложенности иерархического запроса.

`IgnoreDisplayValues bool`

Флаг, который указывает, что запрос не будет использоваться для отображения значений столбцов.

`IsDistinct bool`

Признак, указывающий убирать или нет дубли в результирующем наборе данных.

`IsHierarchical bool`

Признак иерархической выборки данных.

`IsPageable bool`

Признак постраничной выборки данных.

`QueryOptimize bool`

Позволяет использовать оптимизацию запросов.

`QuerySource QuerySource`

Источник запроса.

Возможные значения (`QuerySource`)

| | |
|---------------|--------------------------|
| Filter | Из фильтра. |
| FilterSummary | Из аннотации с фильтром. |
| Undefined | Не задано. |

`QueryType QueryType`

Тип запроса.

Возможные значения (`QueryType`)

| | |
|--------|-------------------|
| Delete | Удаление данных. |
| Select | Чтение данных. |
| Update | Изменение данных. |

RowCount `int`

Количество выбираемых строк. По умолчанию содержит значение `-1`, т. е. выбираются все строки.

RowsOffset `int`

Количество пропущенных строк.

ServerESQCacheParameters `ServerESQCacheParameters`

Параметры кэширования `EntitySchemaQuery` на сервере. Тип `ServerESQCacheParameters` определен в пространстве имен `Terrasoft.Nui.ServiceModel.DataContract`.

UseLocalization `bool`

Параметр, который определяет использование локализованных данных.

UseMetrics `bool`

Использует метрики для запроса.

UseRecordDeactivation `bool`

Определяет отключение данных при фильтрации.

Класс SelectQueryColumns C#

Пространство имен `Terrasoft.Nui.ServiceModel.DataContract`.

На заметку. Полный перечень свойств класса `SelectQueryColumns` и его родительских классов можно найти в [Библиотеке .NET классов](#).

Свойства

Items `Dictionary<string, SelectQueryColumn>`

Коллекция ключей и значений `Dictionary<string, SelectQueryColumn>`. Ключом является строка с названием добавляемой колонки, а значением — экземпляр класса `SelectQueryColumn`, который определен в пространстве имен `Terrasoft.Nui.ServiceModel.DataContract`.

Параметры

| | |
|----------------------------------|--|
| OrderDirection OrderDirection | Направление сортировки. Задается значением перечисления OrderDirection пространства имен <code>Terrasoft.Common</code> , определенного в библиотеке классов Terrasoft.Common . |
| OrderPosition int | Задаёт номер позиции в коллекции колонок запроса, по которой производится сортировка. |
| Caption string | Заголовок колонки. |
| Expression ColumnExpression | Свойство, определяющее выражение типа выбираемой колонки. |

Класс ColumnExpression C#

Пространство имен `Terrasoft.Nui.ServiceModel.DataContract`.

Класс `ColumnExpression` определяет выражение, задающее тип колонки схемы. Он определен в пространстве имен `Terrasoft.Nui.ServiceModel.DataContract` библиотеки `Terrasoft.Nui.ServiceModel`. Свойства экземпляра этого класса заполняются в зависимости от свойства `ExpressionType`, которое и задает тип выражения.

На заметку. Полный перечень свойств класса `SelectQueryColumns` и его родительских классов можно найти в [Библиотеке .NET классов](#).

Свойства

PrimaryColumnMacroType int

Тип макроса основной колонки.

PrimaryDisplayColumnMacroType int

Тип макроса отображаемой колонки.

PrimaryImageColumnMacroType int

Тип макроса основной колонки изображения.

ExpressionType EntitySchemaQueryExpressionType

Тип выражения, определяющий значение, которое будет содержаться в добавляемой колонке.

Задается значением перечисления EntitySchemaQueryExpressionType пространства имен Terrasoft.Core.Entities, определенного в библиотеке классов Terrasoft.Core. Для SelectQuery устанавливается значение EntitySchemaQueryExpressionType.Parameter.

Возможные значения (EntitySchemaQueryExpressionType)

| | | |
|---------------------|---|--------------------------|
| SchemaColumn | 0 | Колонка схемы. |
| Function | 1 | Функция. |
| Parameter | 2 | Параметр. |
| SubQuery | 3 | Вложенный запрос. |
| ArithmeticOperation | 4 | Арифметическая операция. |

IsBlock bool

Блокировка.

ColumnPath string

Путь к колонке относительно корневой схемы.

Parameter Parameter

Определяет значение, которое будет содержаться в добавляемой колонке. Имеет тип Parameter, определенный в пространстве имен Terrasoft.Nui.ServiceModel.DataContract.

FunctionType FunctionType

Тип функции. Задается значением из перечисления FunctionType, определенного в пространстве имен Terrasoft.Nui.ServiceModel.DataContract.

Возможные значения (FunctionType)

| | | |
|-------------|---|-----------------------|
| None | 0 | Не определен. |
| Macros | 1 | Макрос. |
| Aggregation | 2 | Агрегирующая функция. |
| DatePart | 3 | Часть даты. |
| Length | 4 | Длина. |

MacroType EntitySchemaQueryMacroType

Тип макроса. Задается значением перечисления EntitySchemaQueryMacroType , определенного в пространстве имен Terrasoft.Core.Entities .

FunctionArgument BaseExpression

Аргумент функции. Принимает значение, если функция определена с параметром. Класс BaseExpression определен в пространстве имен Terrasoft.Nui.ServiceModel.DataContract , является предком для класса ColumnExpresion и имеет такой же набор свойств.

FunctionArguments BaseExpression[]

Массив аргументов функции.

DateDiffInterval DateDiffQueryFunctionInterval

Интервал разницы дат.

DatePartType DatePart

Часть даты. Задается значением из перечисления DatePart , определенного в пространстве имен Terrasoft.Nui.ServiceModel.DataContract .

Возможные значения (DatePart)

| | | |
|------------|---|---------------|
| None | 0 | Не определен. |
| Day | 1 | День. |
| Week | 2 | Неделя. |
| Month | 3 | Месяц. |
| Year | 4 | Год. |
| Weekday | 5 | День недели. |
| Hour | 6 | Час. |
| HourMinute | 7 | Минута. |

AggregationType AggregationType

Тип агрегирующей функции. Задается значением из перечисления `AggregationType`, определенного в пространстве имен `Terrasoft.Common`, определенного в библиотеке классов `Terrasoft.Common`.

AggregationEvalType AggregationEvalType

Область применения агрегирующей функции. Задается значением из перечисления `AggregationEvalType`, определенного в пространстве имен `Terrasoft.Core.DB`, определенного в библиотеке классов `Terrasoft.Core`.

SubFilters Filters

Коллекция фильтров вложенных запросов. Имеет тип `Filters`, определенный в пространстве имен `Terrasoft.Nui.ServiceModel.DataContract`.

ArithmeticOperation ArithmeticOperation

Тип арифметической операции.

Возможные значения (`ArithmeticOperation`)

| | |
|----------------|------------|
| Addition | Сложение. |
| Division | Деление. |
| Multiplication | Умножение. |
| Subtraction | Вычитание. |

LeftArithmeticOperand BaseExpression

Левый операнд.

RightArithmeticOperand BaseExpression

Правый операнд.

Класс ServerESQCacheParameters C#

Пространство имен `Terrasoft.Nui.ServiceModel.DataContract`.

На заметку. Полный перечень свойств класса `SelectQueryColumns` и его родительских классов можно найти в [Библиотеке .NET классов](#).

Свойства

CacheGroup string

Группа кэширования.

CacheItemName string

Ключ записи в хранилище.

CacheLevel int

Уровень размещения данных в кэше `EntitySchemaQuery`.

Класс Filters C#

 Сложный

Класс Filters C#

Пространство имен `Terrasoft.Nui.ServiceModel.DataContract` библиотеки классов

`Terrasoft.Nui.ServiceModel.dll`.

Назначение класса `Filters` — выполнение фильтрации данных во время выполнения запросов к сервису работы с данными `DataService`. Например, при чтении записей раздела необходимо выполнить выборку только тех записей, которые соответствуют определенному критерию или нескольким критериям.

Для простоты восприятия иерархическую структуру контракта данных `Filters` удобно представить в формате объекта JSON.

Структура контракта данных `Filters`

```
"Filters":{
  "RootSchemaName":"[Имя корневой схемы объекта]",
  "FilterType":[Тип фильтра],
  "ComparisonType":[Тип сравнения],
  "LogicalOperation":[Логическая операция],
  "IsNull":[Признак проверки на заполненность],
  "IsEnabled":[Признак активности],
  "IsNot":[Признак использования оператора отрицания],
  "SubFilters":[Фильтры подзапроса],
  "Items":[Коллекция группы фильтров],
  "LeftExpression":[Выражение, подлежащее проверке],
  "RightExpression":[Выражение фильтрации],
  "RightExpressions":[Массив выражений фильтрации],
  "RightLessExpression":[Начальное выражение диапазона фильтрации],
  "RightGreaterExpression":[Конечное выражение диапазона фильтрации],
  "TrimDateTimeParameterToDate":[Признак отсекаания времени для параметров типа дата/время],
  "Key":"[Ключ фильтра в коллекции фильтров]",
  "IsAggregative":[Признак агрегирующего фильтра],
  "LeftExpressionCaption":"[Заголовок выражения, подлежащего проверке]",
  "ReferenceSchemaName":"[Название ссылочной схемы]",
  "DateDiffInterval":[Интервал разницы дат],
  "FunctionArguments":[Массив аргументов функции],
  "IsBlock":[Блокировка],
  "LeftArithmeticOperand":[Левый операнд],
  "RightArithmeticOperand":[Правый операнд]
}
```

На заметку. Полный перечень свойств класса `Filters` и его родительских классов можно найти в [Библиотеке .NET классов](#).

Свойства

FilterType FilterType

Тип фильтра. Задается значением перечисления `FilterType` пространства имен

`Terrasoft.Nui.ServiceModel.DataContract`.

Возможные значения (`FilterType`)

| | | |
|---------------|---|---|
| None | 0 | Тип фильтра не определен. |
| CompareFilter | 1 | Фильтр сравнения. Используется для сравнения результатов выражений. |
| IsNullFilter | 2 | Фильтр, определяющий, является ли проверяемое выражение пустым или нет. |
| Between | 3 | Фильтр, проверяющий, входит ли проверяемое выражение в диапазон выражений. |
| InFilter | 4 | Фильтр, проверяющий, равно ли проверяемое выражение одному из выражений. |
| Exists | 5 | Фильтр существования по заданному полю. |
| FilterGroup | 6 | Группа фильтров. Группы фильтров могут вкладываться друг в друга, т. е. коллекция сама может быть элементом другой коллекции. |

ComparisonType FilterComparisonType

Тип операции сравнения. Задается значением перечисления `FilterComparisonType` пространства имен

`Terrasoft.Core.Entities`.

Возможные значения (`FilterComparisonType`)

| | |
|----------------|--|
| Between | Диапазон значений. |
| Contain | Содержит выражение. |
| EndWith | Заканчивается выражением. |
| Equal | Равно. |
| Exists | Существует по заданному условию. |
| Greater | Больше. |
| GreaterOrEqual | Больше или равно. |
| IsNull | Является <code>null</code> в базе данных. |
| IsNotNull | Не является <code>null</code> в базе данных. |
| Less | Меньше. |
| LessOrEqual | Меньше или равно. |
| NotContain | Не содержит выражение. |
| NotEndWith | Не заканчивается выражением. |
| NotEqual | Не равно. |
| NotExists | Не существует по заданному условию. |
| NotStartWith | Не начинается выражением. |
| StartWith | Начинается выражением. |

LogicalOperation LogicalOperationStrict

Логическая операция. Тип не допускает значение `None`, определен в перечислении `LogicalOperationStrict` пространства имен `Terrasoft.Common`.

IsNull bool

Признак проверки на заполненность проверяемого выражения.

IsEnabled bool

Признак того, что фильтр активен и будет учитываться при построении запроса.

`IsNot bool`

Определяет, использовать ли логический оператор отрицания.

`SubFilters Filters`

Фильтры подзапроса. Не могут содержать фильтры с другими подзапросами.

`Items Dictionary<string, Filter>`

Коллекция, содержащая группу фильтров.

`LeftExpression BaseExpression`

Выражение в левой части сравнения, т. е. выражение, подлежащее проверке. Класс `BaseExpression` определен в пространстве имен `Terrasoft.Nui.ServiceModel.DataContract`.

`RightExpression BaseExpression`

Выражение фильтрации, которое будет сравниваться с выражением, содержащимся в свойстве `LeftExpression`.

`RightExpressions BaseExpression[]`

Массив выражений, которые будут сравниваться с выражением, содержащимся в свойстве `LeftExpression`.

`RightLessExpression BaseExpression`

Начальное выражение диапазона фильтрации.

`RightGreaterExpression BaseExpression`

Конечное выражение диапазона фильтрации.

`TrimDateTimeParameterToDate bool`

Признак, указывающий отсекают ли время для параметров типа Дата-время.

`Key string`

Ключ фильтра в коллекции фильтров `Items`.

`IsAggregative` `bool`

Признак того, что фильтр является агрегирующим.

`LeftExpressionCaption` `string`

Заголовок левой части сравнения.

`ReferenceSchemaName` `string`

Имя схемы объекта, на которую ссылается левая часть фильтра, если тип колонки — справочник.

Класс BaseExpression C#

Пространство имен `Terrasoft.Nui.ServiceModel.DataContract`.

Класс `BaseExpression` является базовым классом выражений. Он определен в пространстве имен `Terrasoft.Nui.ServiceModel.DataContract` библиотеки `Terrasoft.Nui.ServiceModel.dll`. Свойства экземпляра этого класса заполняются в зависимости от свойства `ExpressionType`, которое и задает тип выражения.

На заметку. Полный перечень свойств класса `BaseExpression` и его родительских классов можно найти в [Библиотеке .NET классов](#).

Свойства

`AggregationEvalType` `AggregationEvalType`

Область применения агрегирующей функции. Задается значением из перечисления `AggregationEvalType`, определенного в пространстве имен `Terrasoft.Core.DB`, определенного в библиотеке классов `Terrasoft.Core`.

Возможные значения (`AggregationEvalType`)

| | |
|----------|-------------------------------------|
| All | Применяется ко всем значениям. |
| Distinct | Применяется к уникальным значениям. |
| None | Область не задана. |

`AggregationType` `AggregationType`

Тип агрегирующей функции. Задается значением из перечисления `AggregationType`, определенного в пространстве имен `Terrasoft.Common`, определенного в библиотеке классов `Terrasoft.Common`.

Возможные значения (`AggregationType`)

| | |
|-------|---|
| Avg | Среднее значение всех элементов. |
| Count | Количество всех элементов. |
| Max | Максимальное значение среди всех элементов. |
| Min | Минимальное значение среди всех элементов. |
| None | Тип агрегирующей функции не определен. |
| Sum | Сумма значений всех элементов. |

ArithmeticOperation `ArithmeticOperation`

Тип арифметической операции.

Возможные значения (`ArithmeticOperation`)

| | |
|----------------|------------|
| Addition | Сложение. |
| Division | Деление. |
| Multiplication | Умножение. |
| Subtraction | Вычитание. |

ColumnPath `string`

Путь к колонке относительно корневой схемы.

DateDiffInterval `DateDiffQueryFunctionInterval`

Интервал разницы дат.

DatePartType `DatePart`

Часть даты. Задается значением из перечисления `DatePart`, определенного в пространстве имен `Terrasoft.Nui.ServiceModel.DataContract`.

Возможные значения (`DatePart`)

| | | |
|------------|---|---------------|
| None | 0 | Не определен. |
| Day | 1 | День. |
| Week | 2 | Неделя. |
| Month | 3 | Месяц. |
| Year | 4 | Год. |
| Weekday | 5 | День недели. |
| Hour | 6 | Час. |
| HourMinute | 7 | Минута. |

`ExpressionType` `EntitySchemaQueryExpressionType`

Тип выражения, определяющий значение, которое будет содержаться в добавляемой колонке. Задается значением перечисления `EntitySchemaQueryExpressionType` пространства имен `Terrasoft.Core.Entities`, определенного в библиотеке классов `Terrasoft.Core`. Для `InsertQuery` устанавливается значение `EntitySchemaQueryExpressionType.Parameter`.

Возможные значения (`EntitySchemaQueryExpressionType`)

| | | |
|---------------------|---|--------------------------|
| SchemaColumn | 0 | Колонка схемы. |
| Function | 1 | Функция. |
| Parameter | 2 | Параметр. |
| SubQuery | 3 | Вложенный запрос. |
| ArithmeticOperation | 4 | Арифметическая операция. |

`FunctionArgument` `BaseExpression`

Аргумент функции. Принимает значение, если функция определена с параметром. Класс `BaseExpression` определен в пространстве имен `Terrasoft.Nui.ServiceModel.DataContract`, является предком для класса `ColumnExpresion` и имеет такой же набор свойств.

`FunctionArguments` `BaseExpression[]`

Массив аргументов функции.

`FunctionType` `FunctionType`

Тип функции. Задается значением из перечисления `FunctionType`, определенного в пространстве имен `Terrasoft.Nui.ServiceModel.DataContract`.

Возможные значения (`FunctionType`)

| | |
|-------------|-----------------------|
| None | Не определен. |
| Macros | Макрос. |
| Aggregation | Агрегирующая функция. |
| DatePart | Часть даты. |
| Length | Длина. |
| DateAdd | Добавление даты. |
| DateDiff | Разница дат. |
| Window | Окно. |

`IsBlock` `bool`

Блокировка.

`LeftArithmeticOperand` `BaseExpression`

Левый операнд.

`MacroType` `EntitySchemaQueryMacroType`

Тип макроса. Задается значением перечисления `EntitySchemaQueryMacroType`, определенного в пространстве имен `Terrasoft.Core.Entities`.

`Parameter` `Parameter`

Определяет значение, которое будет содержаться в добавляемой колонке. Имеет тип `Parameter`,

определенный в пространстве имен `Terrasoft.Nui.ServiceModel.DataContract`.

`RightArithmeticOperand` `BaseExpression`

Правый операнд.

`SubFilters` `Filters`

Коллекция фильтров вложенных запросов. Имеет тип `Filters`, определенный в пространстве имен `Terrasoft.Nui.ServiceModel.DataContract`.

Перечисление EntitySchemaQueryMacroType C#



Пространство имен `Terrasoft.Core.Entities`.

При создании запросов к сервису работы с данными DataService могут использоваться как параметризованные (требующие аргумент), так и непараметризованные макросы. Типы макросов, которые можно использовать в выражениях запросов, определены перечислением `EntitySchemaQueryMacroType`.

На заметку. Полный перечень значений перечисления `EntitySchemaQueryMacroType` можно найти в [Библиотеке .NET классов](#).

Возможные значения (`EntitySchemaQueryMacroType`)

| | |
|---------------------------------|--------------------------------|
| <code>CurrentHalfYear</code> | Текущее полугодие. |
| <code>CurrentHour</code> | Текущий час. |
| <code>CurrentMonth</code> | Текущий месяц. |
| <code>CurrentQuarter</code> | Текущий квартал. |
| <code>CurrentUser</code> | Текущий пользователь. |
| <code>CurrentUserContact</code> | Контакт текущего пользователя. |
| <code>CurrentWeek</code> | Текущая неделя. |
| <code>CurrentYear</code> | Текущий год. |

| | |
|------------------------------|---|
| DayOfMonth | День месяца. |
| DayOfWeek | День недели. |
| DayOfYearToday | Годовщина в сегодняшнюю дату. |
| DayOfYearTodayPlusDaysOffset | Годовщина в сегодняшнюю дату с учетом смещения дне. |
| Hour | Час. |
| HourMinute | Время. |
| Month | Месяц. |
| NextHalfYear | Следующее полугодие. |
| NextHour | Следующий час. |
| NextMonth | Следующий месяц. |
| NextNDays | Следующие N дней. |
| NextNDaysOfYear | Юбилей на следующие N дней. |
| NextNHours | Следующие N часов. |
| NextQuarter | Следующий квартал. |
| NextWeek | Следующая неделя. |
| NextYear | Следующий год. |
| None | Тип макроса не определен. |
| PreviousHalfYear | Предыдущее полугодие. |
| PreviousHour | Предыдущий час. |
| PreviousMonth | Предыдущий месяц. |
| PreviousNDays | Предыдущие N дней. |
| PreviousNDaysOfYear | Юбилей в предыдущие N дней. |
| PreviousNHours | Предыдущие N часов. |
| PreviousQuarter | Предыдущий квартал. |

| | |
|--------------|--------------------|
| PreviousWeek | Предыдущая неделя. |
| PreviousYear | Предыдущий год. |
| Today | Сегодня. |
| Tomorrow | Завтра. |
| Year | Год. |
| Yesterday | Вчера. |

Класс UpdateQuery C#



Пространство имен `Terrasoft.Nui.ServiceModel.DataContract` библиотеки классов

`Terrasoft.Nui.ServiceModel.dll`.

Назначение класса `UpdateQuery` — обновление содержимого записей раздела. Передача данных непосредственно в сервис работы с данными DataService осуществляется по HTTP протоколу при помощи `POST`-запроса по URL.

Структура запроса на обновление данных

```
/* Формат URL для POST-запроса к DataService на обновление данных. */
http(s)://[Адрес приложения Creatio]/[Номер конфигурации]/dataservice/[Формат данных]/reply/UpdateQuery
```

Пример запроса на обновление данных

```
/* Пример URL для POST-запроса к DataService на обновление данных. */
http(s)://example.creatio.com/0/dataservice/json/reply/UpdateQuery
```

Контракт данных `UpdateQuery` имеет иерархическую структуру с несколькими уровнями вложенности. Для простоты восприятия иерархическую структуру контракта данных `UpdateQuery` удобно представить в формате объекта JSON.

Структура контракта данных `UpdateQuery`

```
{
  "RootSchemaName": "[Имя корневой схемы объекта]",
  ...
}
```

```

"OperationType":[Тип операции с записью],
"QueryId":"[Идентификатор запроса]",
"QueryKind":[Информация о запросе для DBExecutor],
"IncludeProcessExecutionData":[Данные о выполнении процесса],
"IsUpsert":[Добавить запись при ее отсутствии в базе данных],
"QueryType":[Тип запроса],
"IsForceUpdate":[Принудительное обновление],
"ColumnValues":{
    "Items":{
        "Имя добавляемой колонки":{
            "ExpressionType":[Тип выражения],
            "Parameter":{
                "DataValueType":[Тип данных],
                "Value":"[Значение колонки]"
            }
        }
    }...
},
"Filters":[Фильтры запроса]
}

```

На заметку. Полный перечень свойств класса `UpdateQuery` и его родительских классов можно найти в [Библиотеке .NET классов](#).

Свойства

`Filters` `Filters`

Коллекция фильтров запросов. Имеет тип `Filters`, определенный в пространстве имен `Terrasoft.Nui.ServiceModel.DataContract`.

Экземпляр класса запроса `UpdateQuery` в свойстве `Filters` обязательно должен содержать ссылку на корректно инициализированный экземпляр класса `Filters`. Иначе новые значения колонок из свойства `ColumnValues` будут установлены для всех записей раздела.

`QueryId` `string`

Идентификатор запроса.

`QueryKind` `QueryKind`

Дополнительная информация о запросе, которая может быть использована для отправки запроса [DBExecutor](#).

Возможные значения (`QueryKind`)

| | |
|---------|--|
| General | Используется, как значение по умолчанию. |
| Limited | Используется, чтобы показать, что запрос может быть отправлен исполнителю с ограниченными ресурсами. |

`ColumnValues` `ColumnValues`

Содержит коллекцию значений колонок добавляемой записи. Унаследовано от родительского класса `BaseQuery`. Имеет тип `ColumnValues`, определенный в пространстве имен `Terrasoft.Nui.ServiceModel.DataContract`.

`RootSchemaName` `string`

Строка, содержащая название корневой схемы объекта добавляемой записи.

`OperationType` `QueryOperationType`

Тип операции с записью. Задается значением перечисления `QueryOperationType` пространства имен `Terrasoft.Nui.ServiceModel.DataContract`. Для `UpdateQuery` устанавливается значение `QueryOperationType.Update`.

Возможные значения (`QueryOperationType`)

| | |
|--------|---|
| Select | 0 |
| Insert | 1 |
| Update | 2 |
| Delete | 3 |
| Batch | 4 |

`IncludeProcessExecutionData` `bool`

Флаг, который включает данные о выполнении процесса.

`IsForceUpdate` `bool`

Признак принудительного обновления. Если имеет значение `true`, то сущность будет принудительно сохранена на сервере, даже если значения колонок не были изменены. По умолчанию имеет значение

`false`.`IsUpsert bool`

Признак необходимости добавления записи при ее отсутствии в базе данных.

`QueryType QueryType`

Тип запроса.

Возможные значения (`QueryType`)

| | |
|--------|-------------------|
| Delete | Удаление данных. |
| Select | Чтение данных. |
| Update | Изменение данных. |

Класс DeleteQuery C#

 Сложный

Пространство имен `Terrasoft.Nui.ServiceModel.DataContract` библиотеки классов
`Terrasoft.Nui.ServiceModel.dll`.

Назначение класса `DeleteQuery` — удаление записи раздела. Передача данных непосредственно в сервис работы с данными DataService осуществляется по HTTP протоколу при помощи `POST`-запроса по URL.

Структура запроса на удаление данных

```
/* Формат URL для POST-запроса к DataService на удаление данных. */
http(s)://[Адрес приложения Creatio]/[Номер конфигурации]/dataservice/[Формат данных]/reply/DeleteQuery
```

Пример запроса на удаление данных

```
/* Пример URL для POST-запроса к DataService на удаление данных. */
http(s)://example.creatio.com/0/dataservice/json/reply/DeleteQuery
```

Контракт данных `DeleteQuery` имеет иерархическую структуру с несколькими уровнями вложенности. Для простоты восприятия иерархическую структуру контракта данных `DeleteQuery` удобно представить в формате объекта JSON.

Структура контракта данных DeleteQuery

```
{
  "RootSchemaName": "[Имя корневой схемы объекта]",
  "OperationType": [Тип операции с записью],
  "QueryId": "[Идентификатор запроса]",
  "QueryKind": [Информация о запросе для DBExecutor],
  "IncludeProcessExecutionData": [Данные о выполнении процесса],
  "QueryType": [Тип запроса],
  "ColumnValues": [Значения колонок (не используется)],
  "Filters": [Фильтры запроса]
}
```

На заметку. Полный перечень свойств класса `DeleteQuery` и его родительских классов можно найти в [Библиотеке .NET классов](#).

Свойства

`Filters Filters`

Коллекция фильтров запросов. Имеет тип `Filters`, определенный в пространстве имен `Terrasoft.Nui.ServiceModel.DataContract`.

Экземпляр класса запроса `DeleteQuery` в свойстве `Filters` обязательно должен содержать ссылку на корректно инициализированный экземпляр класса `Filters`. Иначе будут удалены все записи раздела.

`QueryId string`

Идентификатор запроса.

`QueryKind QueryKind`

Дополнительная информация о запросе, которая может быть использована для отправки запроса [DBExecutor](#).

Возможные значения (`QueryKind`)

| | |
|---------|--|
| General | Используется, как значение по умолчанию. |
| Limited | Используется, чтобы показать, что запрос может быть отправлен исполнителю с ограниченными ресурсами. |

ColumnValues ColumnValues

Содержит коллекцию значений колонок добавляемой записи. Унаследовано от родительского класса `BaseQuery`. В данном типе запросов не используется.

RootSchemaName string

Строка, содержащая название корневой схемы объекта удаляемой записи.

OperationType QueryOperationType

Тип операции с записью. Задается значением перечисления `QueryOperationType` пространства имен `Terrasoft.Nui.ServiceModel.DataContract`. Для `DeleteQuery` устанавливается значение `QueryOperationType.Delete`.

Возможные значения (`QueryOperationType`)

| | |
|--------|---|
| Select | 0 |
| Insert | 1 |
| Update | 2 |
| Delete | 3 |
| Batch | 4 |

IncludeProcessExecutionData bool

Флаг, который включает данные о выполнении процесса.

QueryType QueryType

Тип запроса.

Возможные значения (`QueryType`)

| | |
|--------|-------------------|
| Delete | Удаление данных. |
| Select | Чтение данных. |
| Update | Изменение данных. |

Класс BatchQuery

Класс BatchQuery



Пространство имен `Terrasoft.Nui.ServiceModel.DataContract` библиотеки классов `Terrasoft.Nui.ServiceModel.dll`.

Назначение класса `BatchQuery` — выполнение пакетных запросов.

Пакетный запрос — коллекция произвольных запросов к сервису работы с данными DataService. Пакетные запросы используются для минимизации обращений к сервису работы с данными DataService, что позволяет повысить производительность приложения. Передача данных непосредственно в сервис работы с данными DataService осуществляется по HTTP протоколу при помощи `POST`-запроса по URL.

Структура пакетного запроса

```
/* Формат URL для пакетного POST-запроса к DataService. */
http(s)://[Адрес приложения Creatio]/[Номер конфигурации]/dataservice/[Формат данных]/reply/BatchQuery
```

Пример пакетного запроса

```
/* Пример URL для пакетного POST-запроса к DataService. */
http(s)://example.creatio.com/0/dataservice/json/reply/BatchQuery
```

Данные пакетного запроса могут передаваться в различных форматах. Одним из удобных для восприятия форматов является формат JSON.

Структура контракта данных `BatchQuery`

```
{
  "ContinueOnError": [Продолжить выполнение запроса в случае ошибки],
  "IncludeProcessExecutionData": [Данные о выполнении процесса],
  "Queries": [
    {
      "__type": "[Полное квалифицированное имя типа запроса]",
      /* Содержимое единичного запроса. */
      ...
    },
    /* Другие единичные запросы. */
    ...
  ]
}
```

На заметку. Полный перечень свойств класса `BatchQuery` и его родительских классов можно найти

в [Библиотеке .NET классов](#).

Свойства

`ContinueOnError` `bool`

Продолжить выполнение запроса в случае ошибки.

`IncludeProcessExecutionData` `bool`

Флаг, который включает данные о выполнении процесса.

`Queries` `List<BaseQuery>`

Коллекция единичных запросов. Для формирования содержимого единичных запросов, которые включены в пакетный запрос, можно воспользоваться контрактами данных `InsertQuery`, `SelectQuery`, `UpdateQuery` и `DeleteQuery`.

Класс ColumnValues C#

 Сложный

Пространство имен `Terrasoft.Nui.ServiceModel.DataContract`.

Класс `ColumnValues` содержит коллекцию значений колонок добавляемой записи.

На заметку. Полный перечень свойств класса `ColumnValues` и его родительских классов можно найти в [Библиотеке .NET классов](#).

Свойства

`Items` `Dictionary<string, ColumnExpression>`

Коллекция ключей и значений `Dictionary<string, ColumnExpression>`. Ключом является строка с названием добавляемой колонки, а значением — объект типа `ColumnExpression`, определенный в пространстве имен `Terrasoft.Nui.ServiceModel.DataContract`.

[Параметры](#)

| | | | | | | | | | | | | | | | | |
|---------------------|---|--------------------------|---|----------------|----------|---|----------|-----------|---|-----------|----------|---|-------------------|---------------------|---|--------------------------|
| ExpressionType | <p>Тип выражения, определяющий значение, которое будет содержаться в добавляемой колонке. Задается значением перечисления <code>EntitySchemaQueryExpressionType</code> пространства имен <code>Terrasoft.Core.Entities</code> , определенного в библиотеке классов <code>Terrasoft.Core</code> . Для <code>InsertQuery</code> устанавливается значение <code>EntitySchemaQueryExpressionType.Parameter</code> .</p> <p>Возможные значения (<code>EntitySchemaQueryExpressionType</code>)</p> <table><tr><td>SchemaColumn</td><td>0</td><td>Колонка схемы.</td></tr><tr><td>Function</td><td>1</td><td>Функция.</td></tr><tr><td>Parameter</td><td>2</td><td>Параметр.</td></tr><tr><td>SubQuery</td><td>3</td><td>Вложенный запрос.</td></tr><tr><td>ArithmeticOperation</td><td>4</td><td>Арифметическая операция.</td></tr></table> | SchemaColumn | 0 | Колонка схемы. | Function | 1 | Функция. | Parameter | 2 | Параметр. | SubQuery | 3 | Вложенный запрос. | ArithmeticOperation | 4 | Арифметическая операция. |
| SchemaColumn | 0 | Колонка схемы. | | | | | | | | | | | | | | |
| Function | 1 | Функция. | | | | | | | | | | | | | | |
| Parameter | 2 | Параметр. | | | | | | | | | | | | | | |
| SubQuery | 3 | Вложенный запрос. | | | | | | | | | | | | | | |
| ArithmeticOperation | 4 | Арифметическая операция. | | | | | | | | | | | | | | |
| Parameter | <p>Определяет значение, которое будет содержаться в добавляемой колонке. Имеет тип <code>Parameter</code> , определенный в пространстве имен <code>Terrasoft.Nui.ServiceModel.DataContract</code> .</p> | | | | | | | | | | | | | | | |

Класс Parameter C#

 Сложный

Пространство имен `Terrasoft.Nui.ServiceModel.DataContract`.

На заметку. Полный перечень свойств класса `Parameter` и его родительских классов можно найти в [Библиотеке .NET классов](#).

Свойства

ArrayValue `string[]`

Массив значений добавляемой колонки. Используется при сериализации массивов и BLOB данных.

DataValueType `DataValueType`

Тип данных значения, которое будет содержаться в добавляемой колонке. Задается значением перечисления `DataType` пространства имен `Terrasoft.Nui.ServiceModel.DataContract`.

Возможные значения (`DataType`)

| |
|--|
| Guid |
| Text |
| Color |
| CompositeObjectList |
| Object |
| ObjectList |
| SecureText |
| ShortText |
| HashText |
| Integer |
| LocalizableParameterValuesListDataType |
| LocalizableStringDataType |
| LongText |
| Float |
| Float1 |
| Float2 |
| Float3 |
| Float4 |
| Float8 |
| Money |
| DateTime |

| |
|--|
| Date |
| EntityCollectionDataValueType |
| EntityColumnMappingCollectionDataValueType |
| EntityDataValueType |
| Time |
| ValueList |
| Lookup |
| Enum |
| File |
| Boolean |
| Blob |
| Image |
| ImageLookup |
| Mapping |
| MaxSizeText |
| MediumText |
| MetaDataTextDataValueType |

Value `object`

Объект, содержащий значение добавляемой колонки.

ShouldSkipConversion `bool`

Признак, отображающий необходимость пропустить процесс приведения типа для свойства `Value`.

Методы


```
DataValueType GetDataType(UserConnection userConnection)
```

Получить тип данных, используя `UserConnection`.

```
LocalizableString GetLocalizableValue()
```

Получает локализуемое значение параметра.

```
object GetValue(UserConnection userConnection, DataValueType forcedDataValueType = null, bool useUtcTime = false)
```

Получает значение параметра, при необходимости, с преобразованием типа.

Параметры

| | |
|----------------------------------|--|
| <code>userConnection</code> | Пользовательское подключение. |
| <code>forcedDataValueType</code> | Тип данных значения, которое необходимо преобразовать. |
| <code>useUtcTime</code> | Установите значение <code>true</code> для параметра, если метод должен возвращать дату <code>DateTimeKind.Utc</code> . |