

Интеграция и внешний API

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	7
Контракты данных	7
Ограничения при использовании DataService	7
Создать новую запись в разделе	8
Реализация создания новой записи в разделе [Контакты] через стороннее приложение	8
Реализация создания новой записи в разделе [Контакты] через Creatio IDE	12
Прочитать записи раздела	16
Реализация чтения записи раздела [Контакты] через стороннее приложение	16
Реализация чтения записи раздела [Контакты] через Creatio IDE	20
Отфильтровать записи раздела	24
Реализовать фильтры	25
Отфильтровать записи раздела с использованием макроса	29
Реализовать фильтр с макросом	29
Отфильтровать записи раздела по источнику вхождения пользователя в роль	32
Реализовать фильтры по источнику вхождения пользователя в роль	32
Обновить запись раздела	34
1. Создать и настроить проект консольного приложения C#	34
2. Добавить объявление полей и констант	35
3. Добавить метод, выполняющий аутентификацию	35
4. Реализовать запрос на добавление записи	35
5. Выполнить POST-запрос к DataService	38
Удалить запись раздела	38
1. Создать и настроить проект консольного приложения C#	39
2. Добавить объявление полей и констант	39
3. Добавить метод, выполняющий аутентификацию	40
4. Реализовать запрос на удаление записи	40
5. Выполнить POST-запрос к DataService	42
Добавить и изменить запись раздела	43
1. Создать и настроить проект консольного приложения C#	43
2. Добавить объявление полей и констант	44
3. Добавить метод, выполняющий аутентификацию	44
4. Реализовать запрос на добавление записи	44
5. Реализовать запрос на изменение записи	45
6. Реализовать пакетный запрос	46
7. Реализовать пакетный запрос	47
Класс InsertQuery	48

Свойства	49
Класс SelectQuery	50
Класс SelectQuery	50
Класс SelectQueryColumns	56
Класс ColumnExpression	57
Класс ServerESQCacheParameters	61
Класс Filters	61
Класс Filters	62
Класс BaseExpression	66
Перечисление EntitySchemaQueryMacrosType	70
Класс UpdateQuery	72
Свойства	73
Класс DeleteQuery	75
Свойства	76
Класс BatchQuery	77
Свойства	79
Класс ColumnValues	79
Свойства	79
Класс Parameter	80
Свойства	80
Методы	82
Возможности интеграции	83
Сервисы работы с данными	84
Сервис запуска бизнес-процессов	85
Пользовательский веб-сервис	85
Postman	86
Выполнить запрос к Creatio в Postman	86
1. Добавить запрос	86
2. Настроить запрос	88
3. Выполнить запрос	90
4. Сохранить запрос	90
Выполнить коллекцию запросов к Creatio в Postman	91
1. Добавить коллекцию запросов	91
2. Добавить запрос в коллекцию	93
3. Настроить переменные коллекции запроса	93
4. Выполнить коллекцию запросов	94
OData	97
Протокол OData 4	98
Протокол OData 3	98

Ограничения при использовании протокола OData	100
Примеры интеграций по OData	100
Примеры запросов с типом данных Stream	101
Получить данные	101
Добавить данные	102
Изменить данные	108
Удалить данные	110
Примеры пакетных запросов	111
Пакетный запрос (Content-Type: application/json)	112
Пакетный запрос (Content-Type: application/json и Prefer: continue-on-error)	115
Пакетный запрос (Content-Type: multipart/mixed)	117
Пакетный запрос (Content-Type: multipart/mixed и разными наборами запросов)	120
Примеры запросов в WCF-клиенте	122
Получить коллекцию контактов через LINQ-запрос	123
Получить коллекцию контактов неявным запросом	124
Получить коллекцию контактов явным запросом	124
Примеры CRUD-операций	125
Веб-сервис odata (OData 4)	127
Строка запроса	128
Заголовки запроса	130
Тело запроса	131
Код состояния ответа	131
Тело ответа	133
Веб-сервис EntityDataService.svc (OData 3)	133
Строка запроса	135
Заголовки запроса	137
Тело запроса	138
Код состояния ответа	138
Тело ответа	139
Аутентификация	140
Виды аутентификации	140
Защита от CSRF-атак	140
Реализовать аутентификацию на C#	141
Реализовать аутентификацию	141
Веб-сервис AuthService.svc	143
Строка запроса	144
Заголовки запроса	144
Тело запроса	144
Код состояния ответа	144

Заголовки ответа	145
Тело ответа	145
Fiddler	145
Выполнить запрос к Creatio в Fiddler	146
1. Сформировать запрос	146
2. Выполнить запрос	148
Результат выполнения	149
Сервис запуска бизнес-процессов	150
Запуск бизнес-процессов из внешнего приложения	150
Запуск бизнес-процесса из front-end части	150
Запустить бизнес-процесс через веб-сервис	151
1. Создать процесс добавления контакта	151
2. Создать процесс чтения контактов	154
3. Запустить выполнение бизнес-процессов из строки навигации браузера	156
4. Запустить выполнение бизнес-процессов из консольного приложения	157
Запустить процесс из клиентского модуля	159
1. Создать и настроить пользовательский бизнес-процесс "Проведение встречи"	160
2. Создать замещающую страницу контрагента и добавить на нее действие	162
3. Добавить необходимые методы в схемы	162
Результат выполнения примера	164
Веб-сервис ProcessEngineService.svc	165
Строка запроса	166
Заголовки запроса	167
Тело ответа	167

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. */
        DataType = DataType.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"
    });
```

```

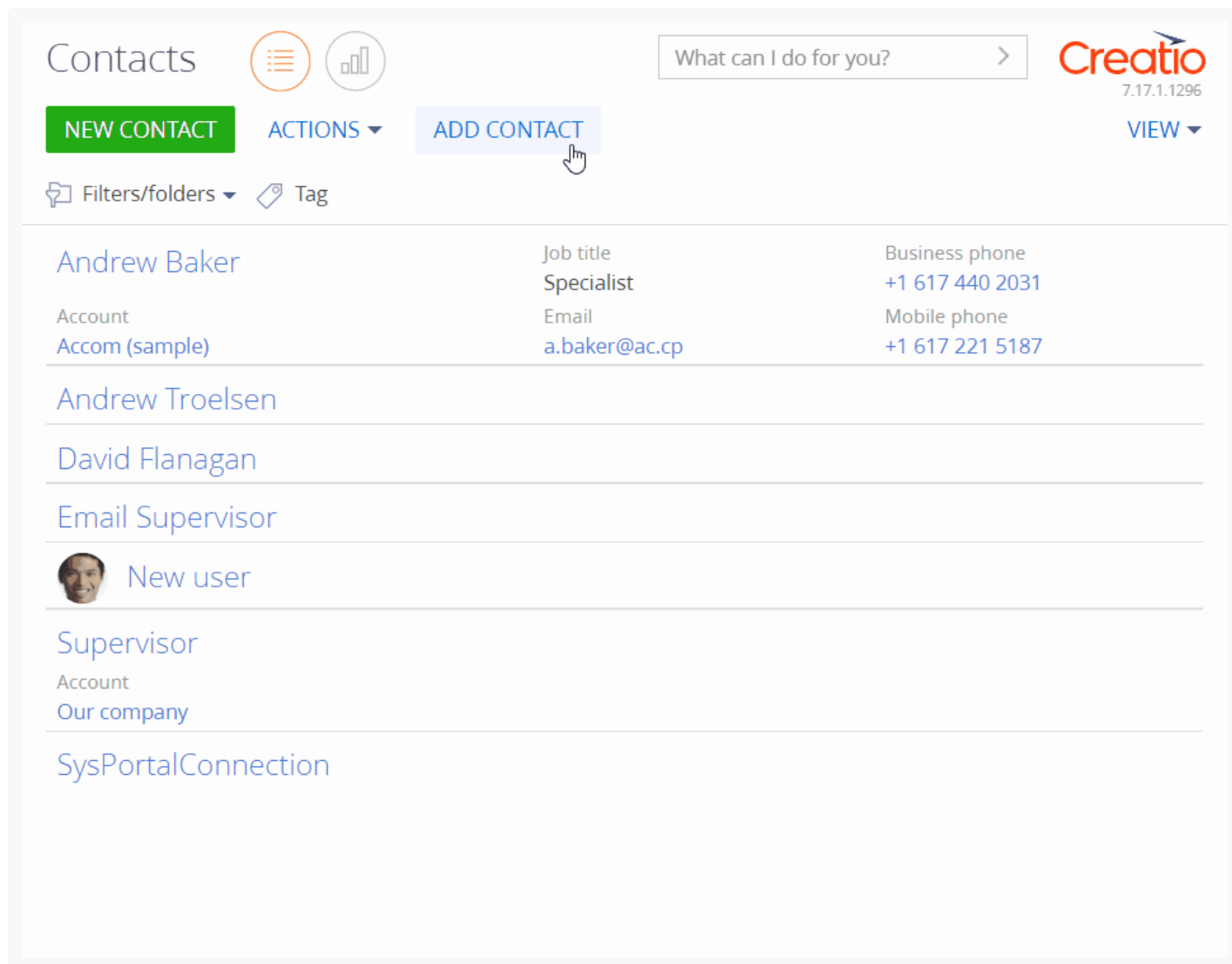
/* Установка значений-параметров Terrasoft.ParameterExpression.
Создается экземпляр значения-параметра и добавляется в коллекцию значений колонок.
Создание экземпляра значения-параметра для колонки [ФИО]. */
insert.setParameterValue("Name", "Иванов Иван Иванович", Terrasoft.DataValueType.TEXT);
/* Создание 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);
    });
  }
}
```

```

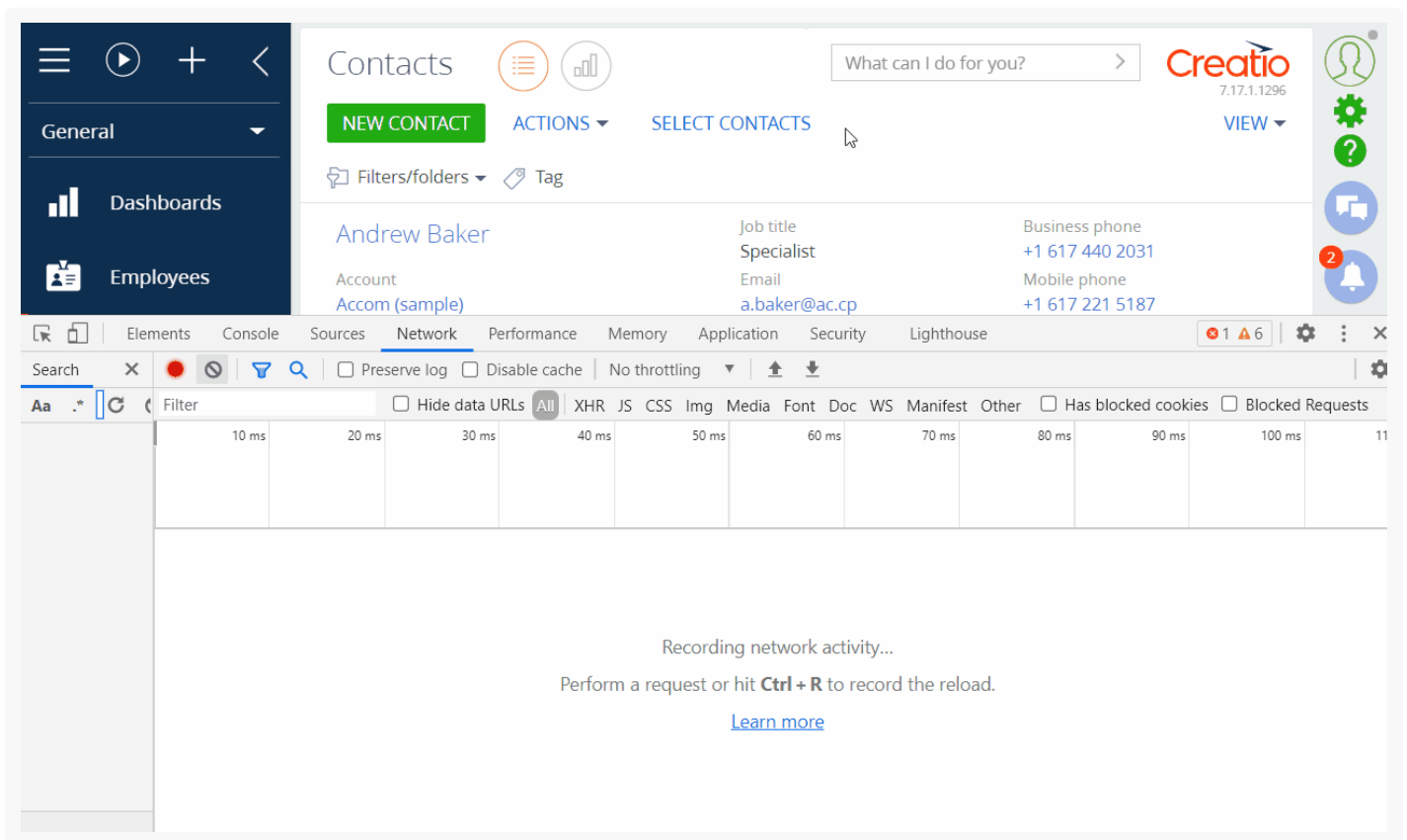
    }, this);
  }
}

```

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 [HttpRequest](#).
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 = HttpRequest.Create(updateQueryUri) as HttpRequest;
/* 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



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 `DataValueType` пространства имен `Terrasoft.Nui.ServiceModel.DataContract`.

Возможные значения (`DataValueType`)

Guid
Text
Color
CompositeObjectList
Object
ObjectList
SecureText
ShortText
HashText
Integer
LocalizableParameterValuesListDataValueType
LocalizableStringDataValueType
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 us
```

Получает значение параметра, при необходимости, с преобразованием типа.

Параметры

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

Возможности интеграции



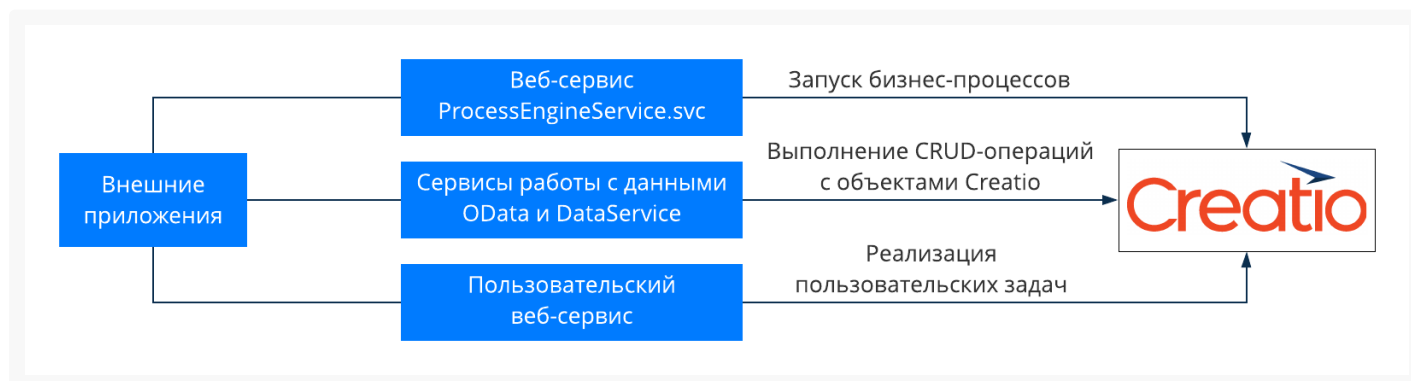
Средний

Платформа Creatio предлагает широкие возможности для интеграции сторонних программных продуктов с Creatio.

Выбор способа интеграции зависит от следующих факторов:

- Потребностей клиента.
- Типа и архитектуры сторонних программных продуктов.
- Компетенции разработчика.

Основные способы интеграции представлены на схеме ниже.



Сравнение основных характеристик способов интеграции с Creatio приведено в таблице.

Сравнение основных способов интеграции с Creatio

Характеристика	DataService	OData	Process Engine Service	Пользовательский веб-сервис
Способы аутентификации	Forms	Forms	Forms	Anonymous, Forms
Решаемые задачи	CRUD-операции с объектами Creatio, фильтрация данных и использование встроенных макросов Creatio.	CRUD-операции с объектами, добавление и удаление связей, получение метаданных, коллекций, полей объектов, сортировка и др.	Запуск бизнес-процессов, передача и получение параметров запускаемого бизнес-процесса, запуск отдельных элементов бизнес-процесса.	Любые требуемые пользовательские задачи, которые можно решить в рамках открытого API Creatio.
Форматы обмена данными	XML, JSON, JSV, CSV	XML, JSON	XML, JSON	XML, JSON
Разработчик	Creatio	Microsoft	Creatio	Creatio
Наличие клиентских библиотек	Можно использовать *.dll-библиотеки Creatio только для .NET приложений.	Доступны в официальной документации OData	Не нужны	Не нужны
Сложность использования	Высокая	Средняя	Низкая	Средняя

Сервисы работы с данными

Сервис DataService

[DataService](#) (разработан Creatio) — сервис, который реализует связь между клиентской и серверной частями платформы. С помощью DataService выполняется передача данных, введенных в пользовательском интерфейсе, в серверную часть приложения для последующей обработки и сохранения в базу данных. Это основное связующее звено front-end и back-end уровней Creatio.

Кроме выполнения CRUD-операций, сервис работы с данными DataService позволяет использовать [встроенные макросы](#) и [фильтры](#). Сложные запросы можно [выполнять пакетно](#).

Протокол OData

[OData \(Open Data Protocol\)](#) — это утвержденный ISO/IEC стандарт OASIS, который определяет набор лучших практик для построения и использования REST API. Доступ к данным и объектам Creatio по протоколу OData может получить любое стороннее приложение, которое поддерживает обмен сообщениями HTTP и может обрабатывать данные XML или JSON. При этом данные доступны в виде ресурсов, адресуемых через URI. Доступ к данным и их изменение осуществляется с помощью стандартных HTTP-методов GET, PUT/PATCH, POST и DELETE.

Кроме выполнения CRUD-операций, протокол OData позволяет использовать функции работы со строками, датами и временем. Также протокол предоставляет большое количество клиентских библиотек по работе с OData для популярных прикладных и мобильных платформ.

Приложение Creatio поддерживает протоколы OData 4 и OData 3.

OData 4 предоставляет больше возможностей, чем OData 3. Основное **отличие протоколов** — ответ на запрос, возвращаемый сервером, имеет разный формат данных. Различия протоколов OData 3 и OData 4 описаны в [официальной документации OData](#).

При планировании интеграции с Creatio по протоколу OData необходимо использовать протокол версии 4.

Детальное описание протокола содержится в [документации OData](#).

Сервис запуска бизнес-процессов

Одной из задач интеграции внешнего приложения с Creatio является запуск бизнес-процессов. С этой целью в сервисной модели Creatio реализован веб-сервис `ProcessEngineService.svc`, позволяющий запускать бизнес-процессы извне.

Кроме выполнения запуска бизнес-процессов из внешнего приложения, сервис `ProcessEngineService.svc` позволяет выполнять обмен данными между Creatio и внешними приложениями.

Пользовательский веб-сервис

В конфигурации Creatio существует возможность создавать [пользовательские веб-сервисы](#), которые используются для реализации специфических интеграционных задач. Конфигурационный веб-сервис представляет собой RESTful-сервис, реализованный на базе технологии [WCF](#).

Пользовательский веб-сервис позволяет выполнять обмен данными, который реализуется разработчиком удобным способом. Также веб-сервис предоставляет возможность реализации разработчиком любых операций с объектами Creatio, в том числе CRUD-операций.

Для доступа пользователя к таким сервисам авторизация не обязательна.

Postman



Postman — это набор инструментов для тестирования API. Он является средой разработки для создания, тестирования, контроля и публикации API-документации. **Назначение** Postman — тестирование отправки запросов с клиента на сервер и получения ответа от сервера.

Postman рекомендуется использовать для тестирования запросов при интеграции с Creatio по протоколу [OData 4](#) и [OData 3](#).

Этапы работы с запросами в Postman:

- Добавление запроса.
- Настройка запроса.
- Выполнение запроса.
- Сохранение запроса.

Коллекция запросов позволяет выполнять несколько запросов один за другим. Использование коллекций значительно сокращает время тестирования запросов.

Коллекция запросов к Creatio должна включать:

1. `post` -запрос к сервису аутентификации `AuthService.svc`. Работа с сервисом аутентификации подробно описана в статье [Аутентификация](#).
2. Пользовательский запрос для работы с данными.

Этапы работы с коллекциями запросов в Postman:

- Добавление коллекции запросов.
- Добавление запроса в коллекцию.
- Настройка переменных коллекции запроса.
- Выполнение коллекции запросов.

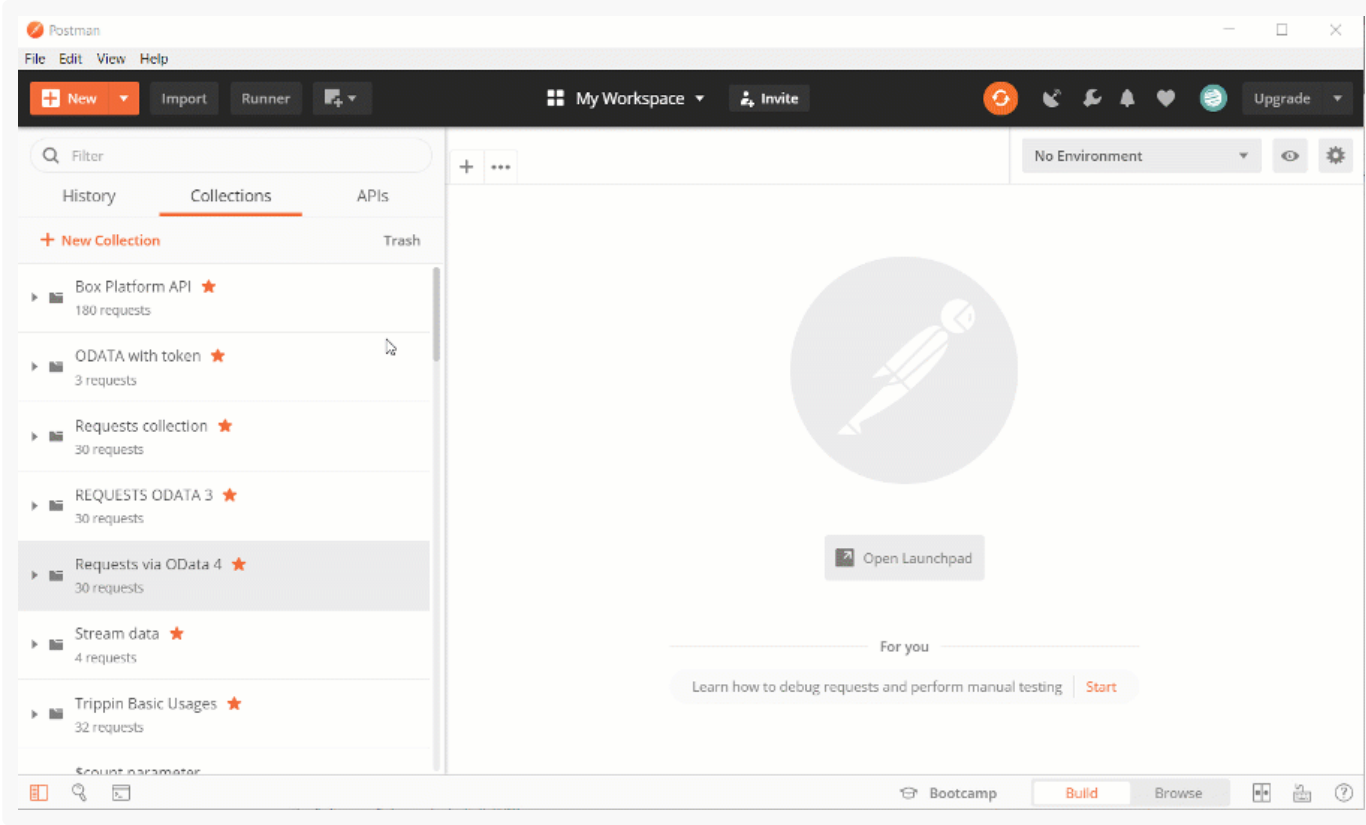
Работа с Postman подробно описана в официальной [документации Postman](#).

Выполнить запрос к Creatio в Postman



1. Добавить запрос

1. На панели инструментов в Postman нажмите кнопку [*New*].
2. В блоке [*Building blocks*] вкладки [*Create New*] выберите [*Request*].



3. Заполните поля запроса.

Поля для создания запроса

Имя поля	Описание	Пример заполнения поля
[<i>Request name</i>]	Имя запроса.	Test request
[<i>Request description (Optional)</i>]	Описание запроса (при необходимости).	Test request for getting Creatio object collection instances
[<i>Search for a collection or folder to save to:</i>]	Поиск ранее созданной коллекции запросов или создание новой.	Test collection

Окно создания запроса

SAVE REQUEST

Request name

Test request

Request description (Optional)

Test request for getting Creatio object collection instances

Descriptions support **Markdown**

Select a collection or folder to save to:

Search for a collection or folder

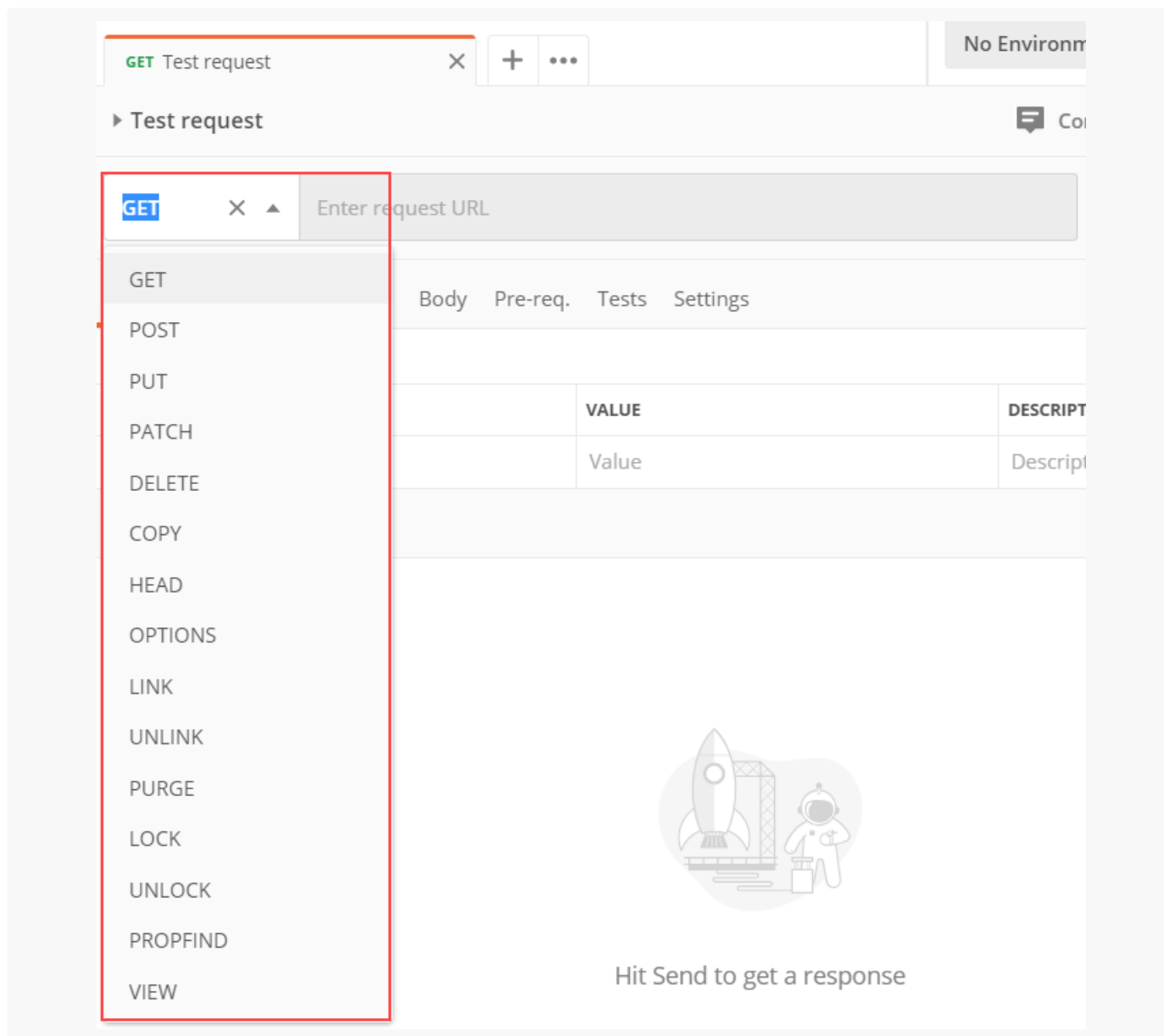
◀ Test collection + Create Folder

Cancel Save to Test collection

4. Нажмите [Save]. Кнопка становится активной после заполнения поля [*Search for a collection or folder to save to:*].

2. Настроить запрос

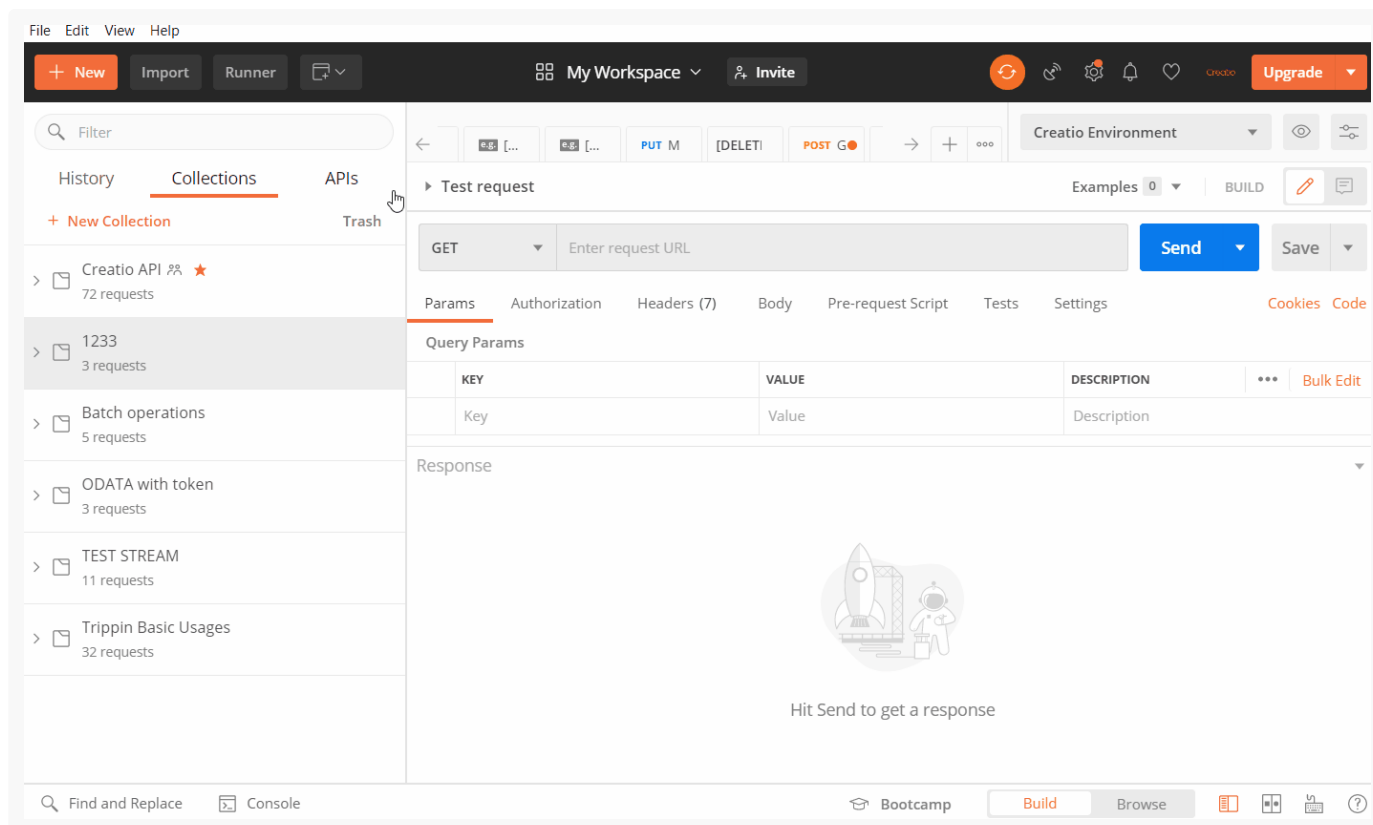
1. В выпадающем списке панели инструментов рабочей области Postman выберите метод запроса.



2. В поле запроса панели инструментов рабочей области Postman введите строку запроса.

3. Установите **формат данных** запроса:

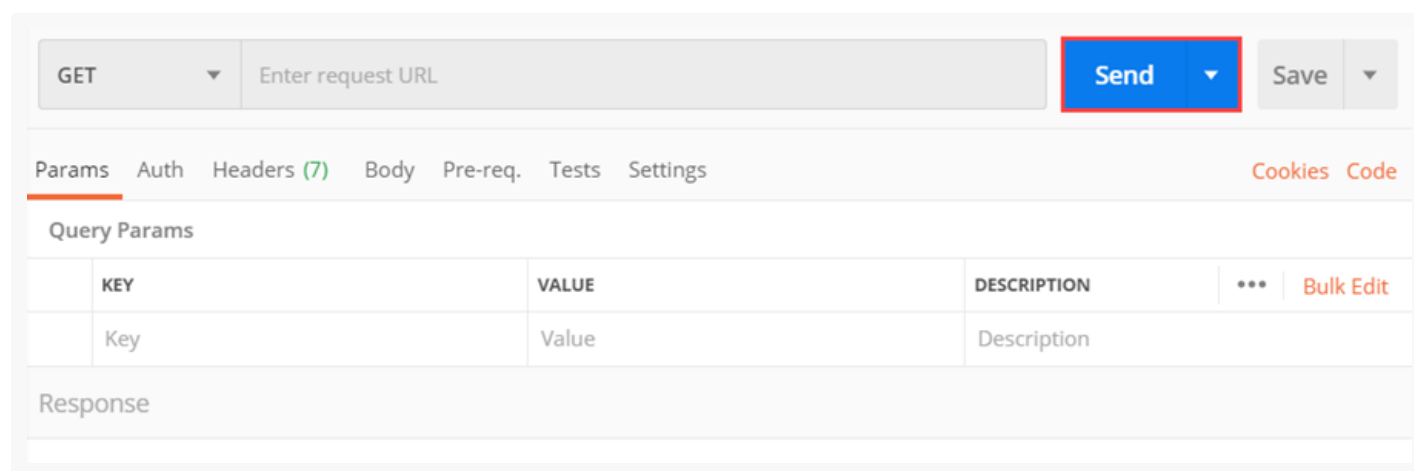
- a. Перейдите на вкладку [*Body*].
- b. Установите опцию "raw".
- c. Выберите тип "JSON".



4. В рабочей области Postman перейдите на вкладку [*Body*] и для методов `POST` и `PATCH` заполните тело запроса.
5. В рабочей области Postman перейдите на вкладку [*Headers*] и установите заголовки.

3. Выполнить запрос

Чтобы **выполнить запрос** в Postman, на панели инструментов рабочей области нажмите [*Send*].



4. Сохранить запрос

Чтобы **сохранить запрос** в Postman, на панели инструментов рабочей области нажмите [*Save*].

GET Enter request URL Send Save

Params Auth Headers (7) Body Pre-req. Tests Settings Cookies Code

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

Response

Работа с Postman подробно описана в официальной [документации Postman](#).

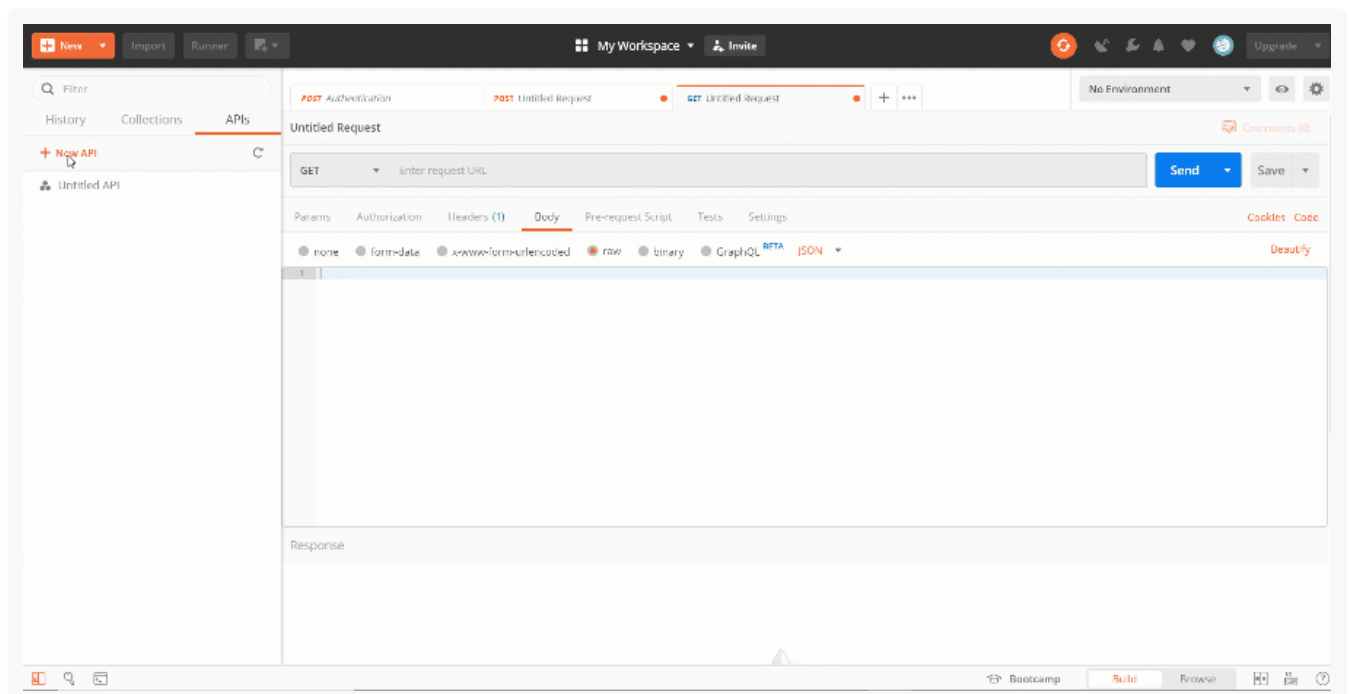
Выполнить коллекцию запросов к Creatio в Postman

Средний

1. Добавить коллекцию запросов

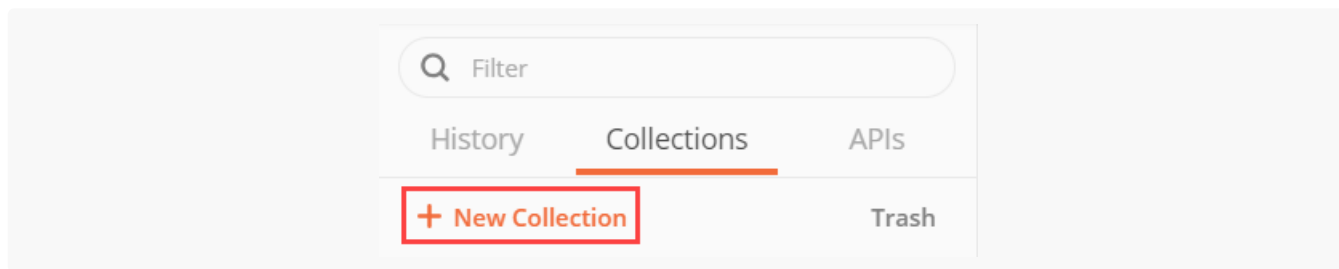
1. Создайте коллекцию запросов. **Способы** создания коллекции запросов в Postman:

- На панели инструментов в Postman нажмите кнопку [*New*]. В блоке [*Building blocks*] вкладки [*Create New*] нажмите [*Collection*].



- На панели работы с запросами в Postman перейдите на вкладку [*Collections*]. Нажмите [+ New

Collection].



2. Заполните поля коллекции запросов.

Поля для создания коллекции запросов

Имя поля	Описание	Пример заполнения поля
[Name]	Имя коллекции.	Test collection
[Description]	Описание коллекции (при необходимости).	Test collection for test requests

Окно создания коллекции запросов

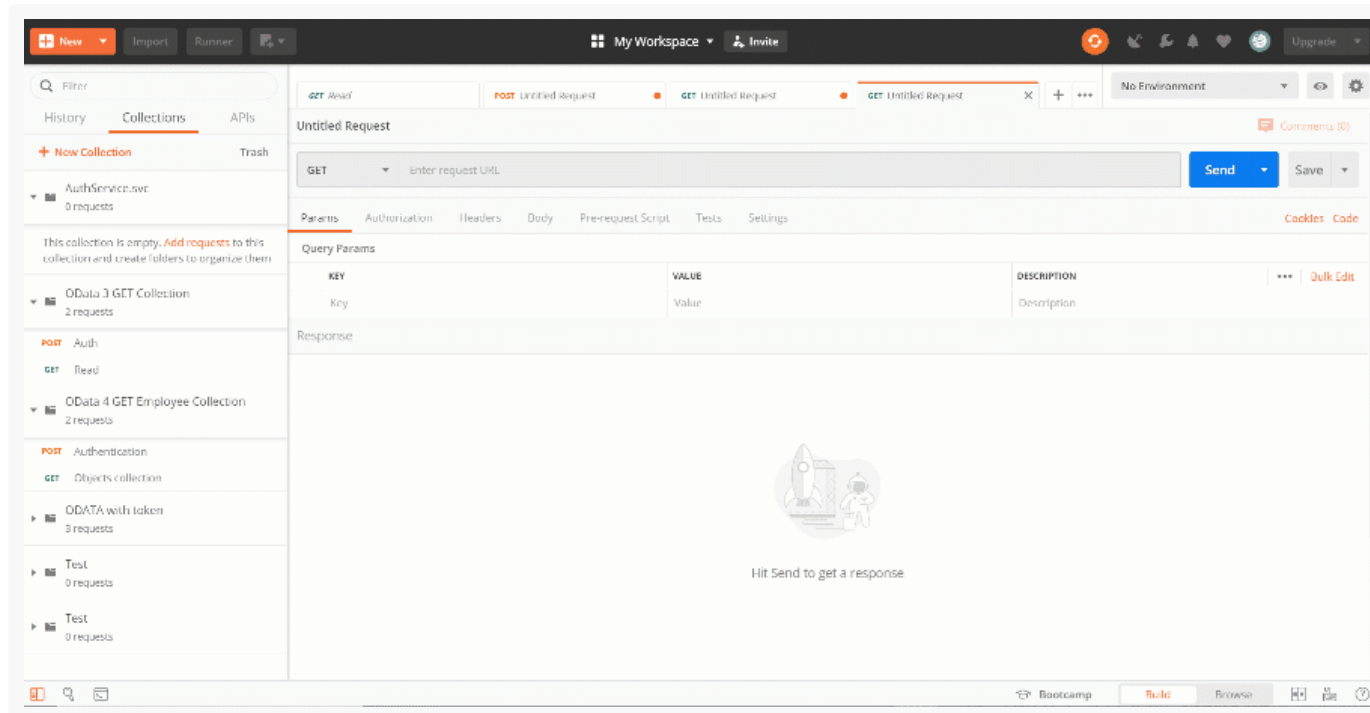
The screenshot shows the 'CREATE A NEW COLLECTION' dialog box. It has a title bar with a close button. Below the title bar, there is a 'Name' field containing 'Test collection'. Below the name field, there are tabs for 'Description', 'Authorization', 'Pre-request Scripts', 'Tests', and 'Variables'. The 'Description' tab is selected. Below the tabs, there is a text area containing 'Test collection for test requests'. At the bottom of the dialog, there are 'Cancel' and 'Create' buttons. A note at the bottom left states 'Descriptions support Markdown'.

3. Нажмите [Create].

2. Добавить запрос в коллекцию

Способы добавления запроса в коллекцию Postman:

- На панели работы с запросами в Postman перетащить курсором мыши ранее созданный запрос в коллекцию.
- На панели работы с запросами в Postman по нажатию правой кнопки мыши по имени созданной коллекции —> пункт [*Add Request*].

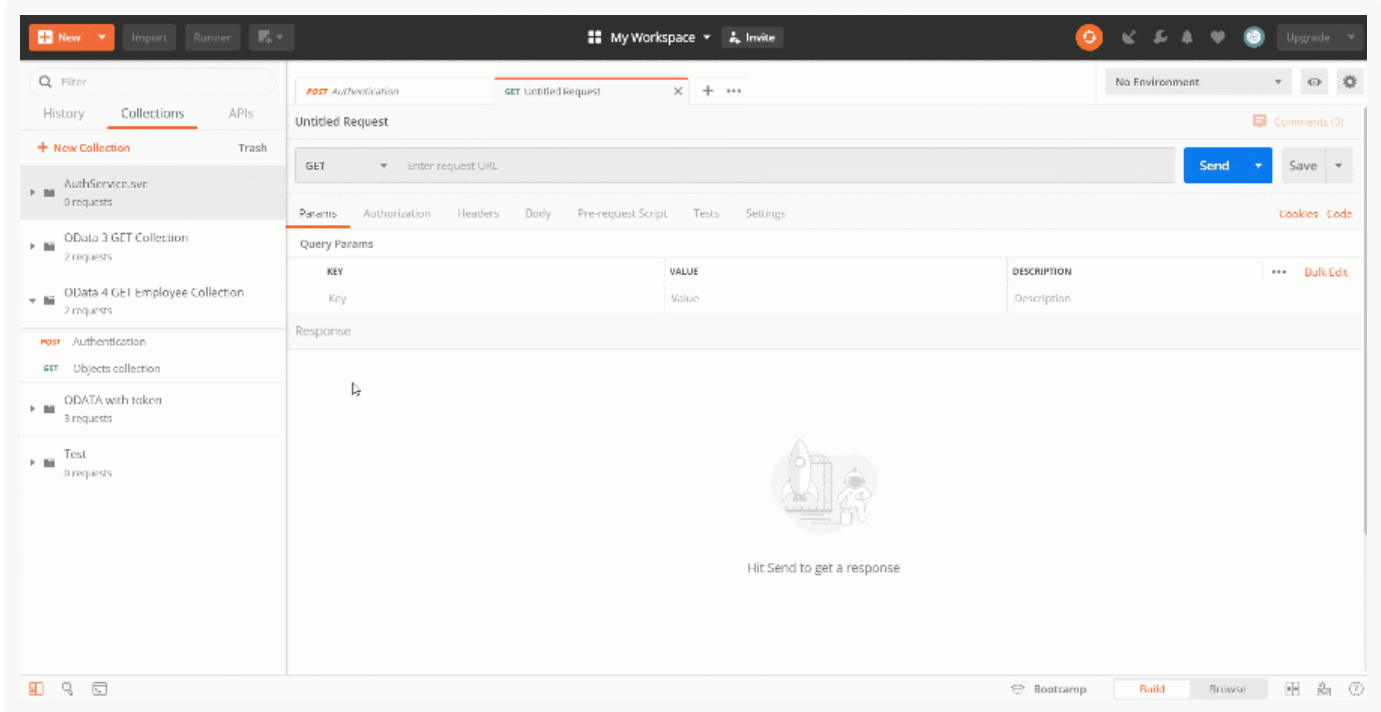


3. Настроить переменные коллекции запроса

Коллекция позволяет установить общие переменные и параметры для всех запросов коллекции.

Чтобы **настроить переменные** коллекции запроса в Postman:

1. На панели работы с запросами в Postman нажмите правой кнопкой мыши по имени созданной коллекции.
2. Выберите пункт [*Edit*] и перейдите на вкладку [*Variables*].




3. Создайте переменные коллекции.

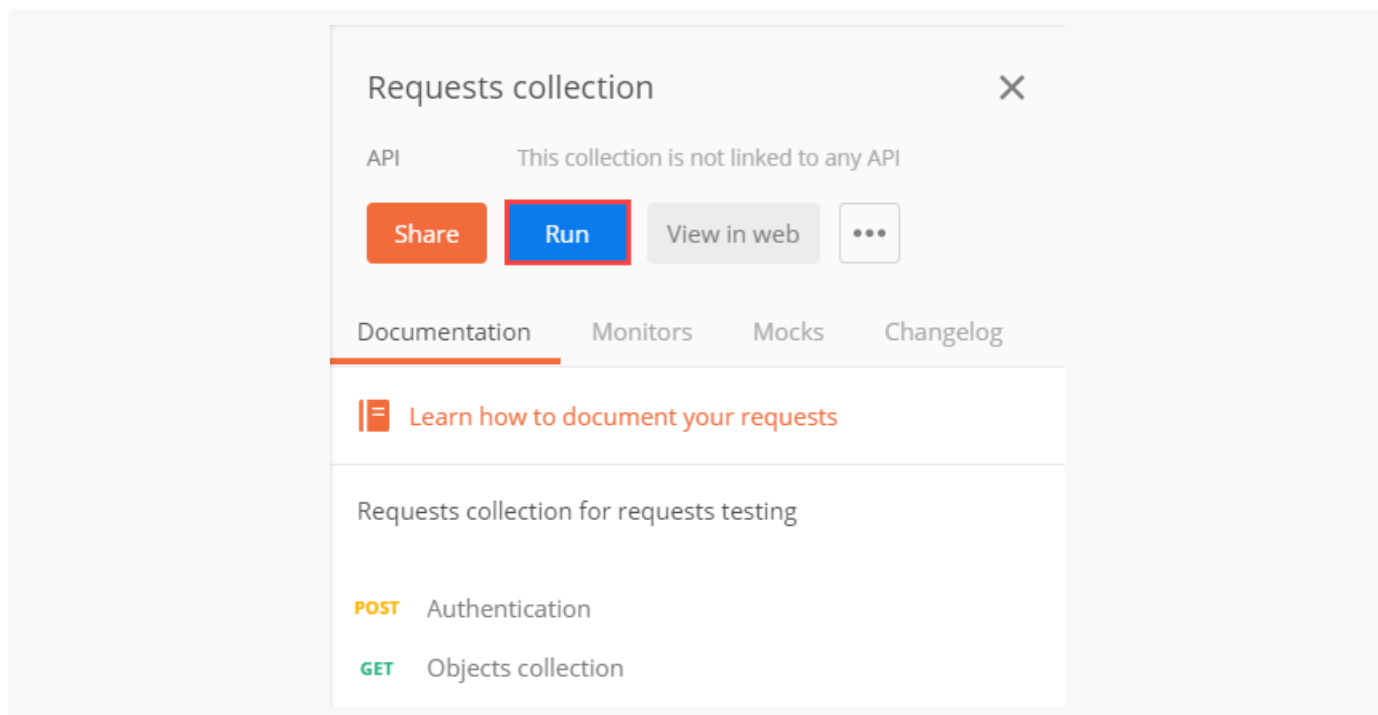
Переменные коллекции

Имя переменной	Описание
BaseURI	Адрес приложения Creatio.
UserName	Логин пользователя Creatio.
UserPassword	Пароль пользователя Creatio.
BPMCSRF	Токен для защиты от CSRF-атак.
CollectionName	Имя коллекции объектов (таблицы базы данных), с которой необходимо работать.

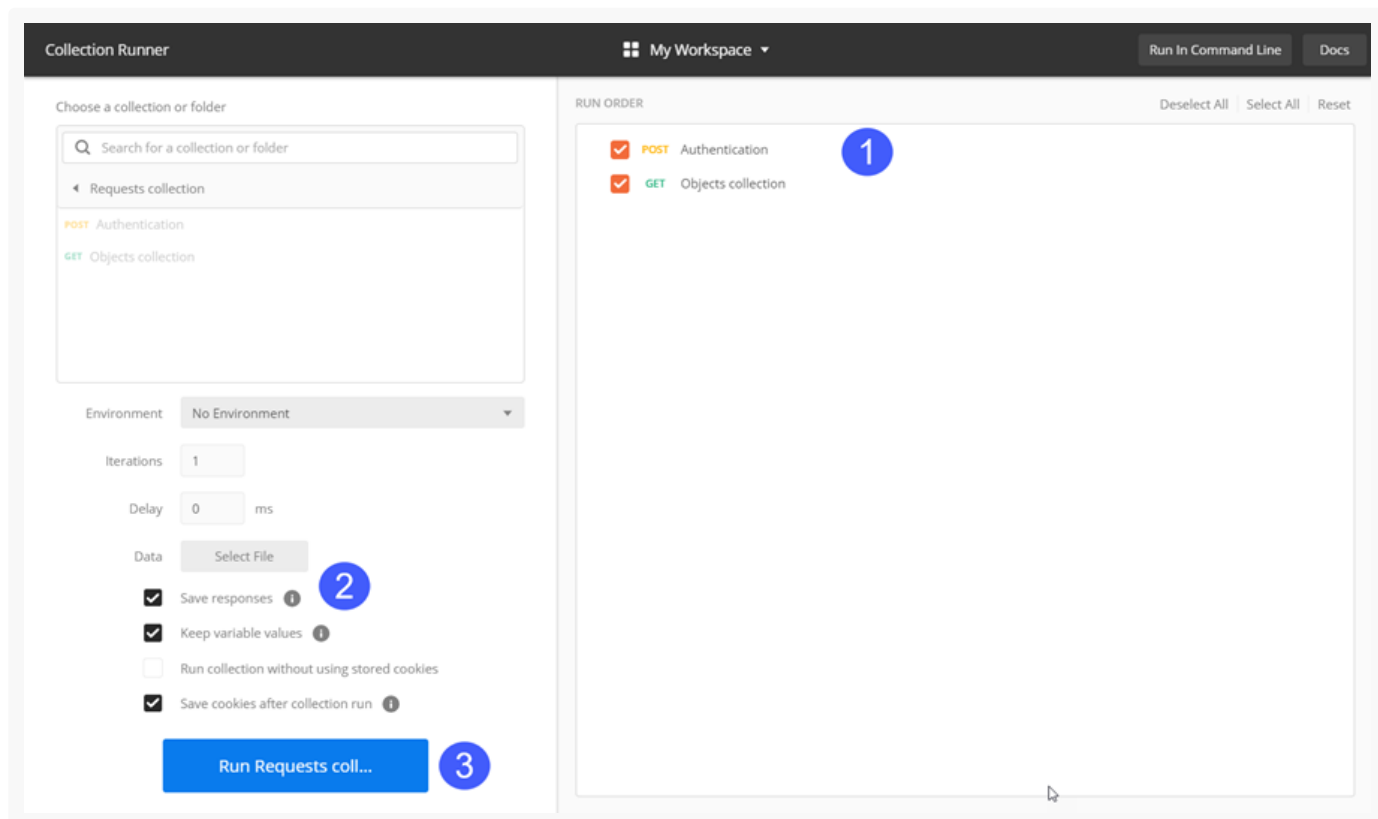
4. Продублируйте значения переменных в колонках [Initial value] и [Current value].

4. Выполнить коллекцию запросов

- 1. На панели работы с запросами в Postman рядом с именем коллекции нажмите кнопку .
- 2. Нажмите кнопку [Run].



3. В блоке [*Run order*] выберите запросы, которые необходимо запустить, и настройте порядок их выполнения (1).
4. Установите признак [*Save responses*] (2).
5. Нажмите [*Run*] (3).



После выполнения коллекции запросов Postman позволяет посмотреть структурные элементы каждого

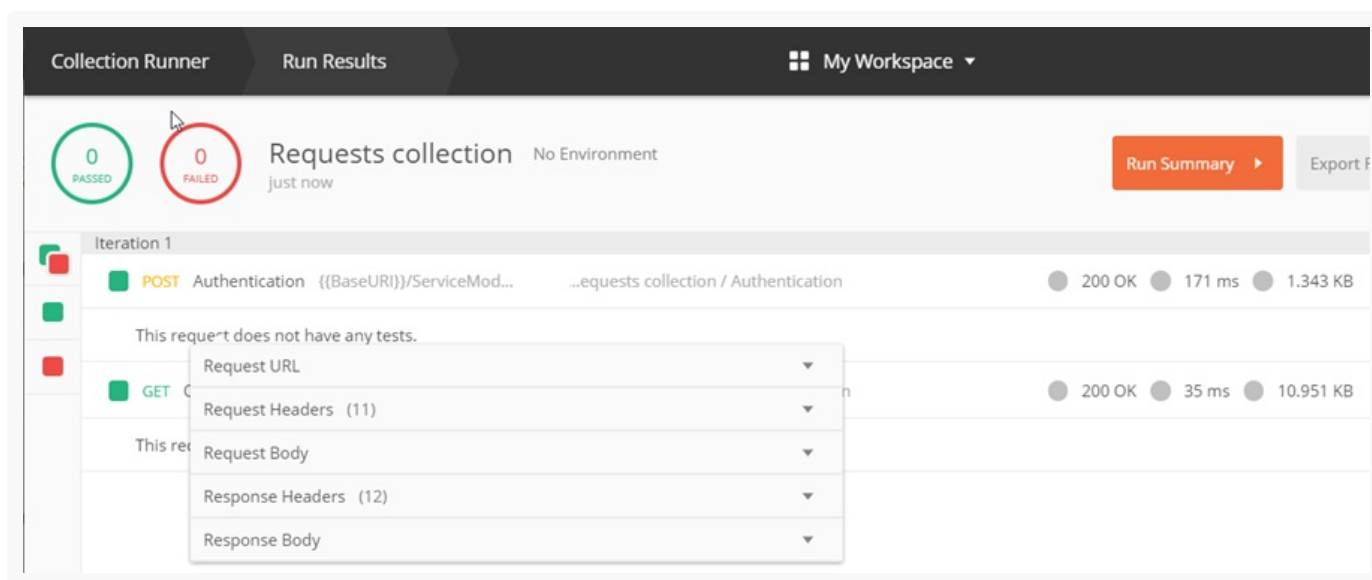
запроса коллекции.

Структурные **элементы запроса**:

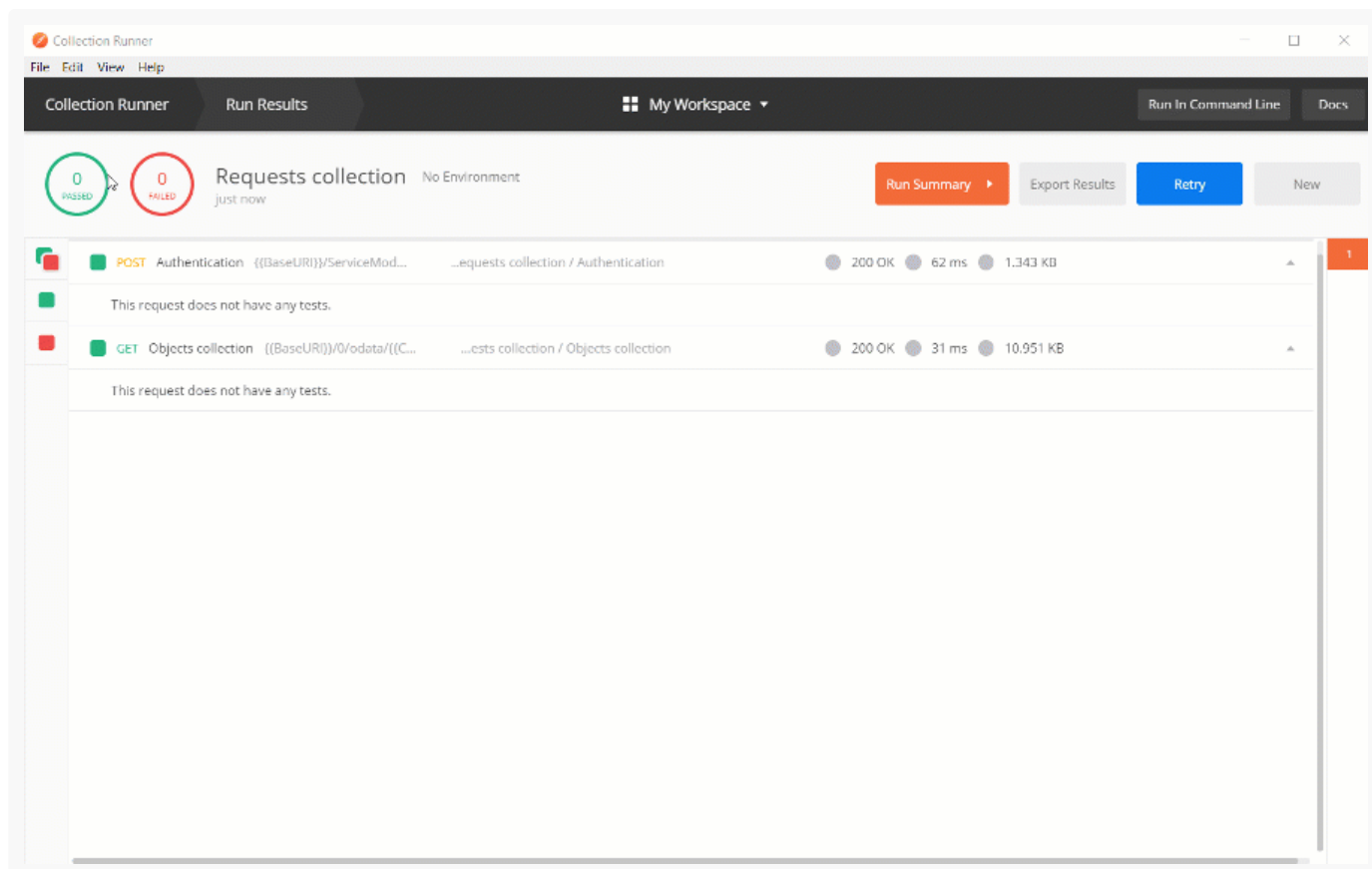
- Строка запроса.
- Заголовки запроса.
- Тело запроса (опционально).
- Заголовки ответа на запрос.
- Тело ответа на запрос (опционально).

Способы отображения информации структурных элементов запроса:

- Кликнуть по имени запроса.



- В меню [View] выбрать [Show Postman Console]. Затем выбрать структурный элемент запроса, информацию по которому необходимо посмотреть.



- Нажать комбинацию клавиш `Alt+Ctrl+C`.

Работа с Postman подробно описана в официальной [документации Postman](#).

OData

 **Сложный**

OData (Open Data Protocol) — это утвержденный ISO/IEC стандарт OASIS, который определяет набор лучших практик для построения и использования REST API. Он позволяет создавать службы на основе REST, которые с помощью простых HTTP-запросов предоставляют возможность публиковать и редактировать ресурсы, идентифицированные с использованием URL и определенные в модели данных.

Назначение протокола OData — выполнение запросов от внешних приложений к серверу баз данных Creatio.

Приложение Creatio поддерживает протоколы OData 4 и OData 3. OData 4 предоставляет больше возможностей, чем OData 3. Основное **отличие** протоколов — ответ на запрос, возвращаемый сервером, имеет разный формат данных. Различия протоколов OData 3 и OData 4 описаны в [официальной документации OData](#). При планировании интеграции с Creatio по протоколу OData необходимо использовать протокол версии 4.

Все внешние запросы к приложению Creatio должны быть аутентифицированы. Рекомендуемым способом для запросов по протоколу OData является аутентификация на основе cookies (**Forms-аутентификация**), которая реализована с помощью веб-сервиса [AuthService.svc](#).

Протокол OData 4

Доступ к объектам Creatio по протоколу OData 4 предоставляет веб-сервис `odata`.

Адрес сервиса odata для .NET Framework

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

Адрес сервиса odata для .NET Core

`https://mycreatio.com/odata`

Протокол OData 3

Доступ к объектам Creatio по протоколу OData 3 предоставляет веб-сервис `EntityDataService.svc`.

Адрес сервиса `EntityDataService.svc`

`https://mycreatio.com/0/ServiceModel/EntityDataService.svc`

Сервис `EntityDataService.svc` предоставляет возможность работы с объектами Creatio в WCF-клиенте.

Windows Communication Foundation (WCF) — программный фреймворк в составе .NET Framework, используемый для обмена данными между приложениями.

Организация **работы WCF-клиента** реализована путем получения метаданных сервиса и создания клиентских прокси-классов. Клиентское приложение будет использовать эти классы-посредники для работы с сервисом `EntityDataService.svc` Creatio.

Чтобы **реализовать клиентское приложение**:

1. Создайте .NET-проект, в котором будет реализована интеграция с Creatio.
2. Сгенерируйте клиентские прокси-классы сервиса `EntityDataService.svc`.
3. Создайте экземпляр контекста среды выполнения сервиса `EntityDataService.svc`.
4. Реализуйте клиентскую бизнес-логику интеграции с использованием методов созданного экземпляра прокси-класса.

Способы генерации прокси-классов сервиса `EntityDataService.svc`:

- С использованием утилиты `DataServiceModel Metadata Utility Tool (DataSvcutil.exe)`.
- Из проекта клиентского приложения Visual Studio.

Генерация прокси-классов с использованием утилиты DataServiceModel Metadata Utility Tool

`DataSvcUtil.exe` представляет собой программу командной строки, предоставляемую сервисами `WCF Data Services`. Утилита использует канал OData и формирует клиентские классы службы данных, необходимые для доступа к службе данных из клиентского приложения .NET Framework. Эта утилита формирует классы данных с использованием следующих **источников метаданных**:

- `WSDL` — документ метаданных службы. Описывает модель данных, предоставленную службой данных.
- `CSDL` — файл модели данных. Используется язык определения концептуальной схемы (`CSDL`), описанный в спецификации [\[MC-CSDL\]: формат файла определения концептуальной схемы](#).
- `EDMX` — *.xml-файл. Создается с помощью программ для работы с моделью `EDM`, входящих в комплект `Entity Framework`. Дополнительные сведения описаны в спецификации [\[MC-EDMX\]: модели EDM для формата упаковки служб данных](#).

Обычно утилита `DataSvcUtil.exe` установлена в каталоге `C:\Windows\Microsoft.NET\Framework\v4.0`. Для 64-разрядных версий систем это каталог `C:\Windows\Microsoft.NET\Framework64\v4.0`.

Формат вызова утилиты `DataSvcutil.exe`

```
datasvcutil /out:file [/in:file | /uri:serviceuri] [/dataservicecollection] [/language:devlang]
```

Детальная информация по утилите `DataSvcutil.exe` приведена в официальной [документации MSDN](#).

Генерация прокси-классов из проекта клиентского приложения Visual Studio

Чтобы сгенерировать прокси-классы из **проекта клиентского приложения Visual Studio**:

1. Щелкните правой клавишей мыши по проекту, в котором планируется реализация интеграции с Creatio, и выберите в контекстном меню пункт [*Add Service Reference...*].
2. В поле [*Address*] введите полный адрес сервиса `EntityDataService.svc`.
3. Нажмите на кнопку [*Go*] и укажите имя и пароль пользователя Creatio. Если аутентификация прошла успешно, то в окне [*Services*] отобразятся поддерживаемые сервисом сущности.
4. В поле [*Namespace*] укажите имя пространства имен, в котором будут расположены сгенерированные прокси-классы. Например, `CreatioServiceReference`. Ссылку на это пространство имен в дальнейшем необходимо добавить в блок `using` кода проекта.
5. Для генерации прокси-классов нажмите кнопку [*OK*]. При этом в проект будет добавлен новый файл кода `Reference.cs`, содержащий описание прокси-классов, которые теперь могут использоваться для обращения и взаимодействия с ресурсами сервиса данных как с объектами.

В Visual Studio также есть возможность генерировать прокси-классы сервиса из сохраненного на диске **файла метаданных сервиса**. Для этого на шаге 2 в поле [*Address*] необходимо ввести полный путь к файлу метаданных с префиксом `file://`.

```
file:///C:/metadata.xml
```

После генерации прокси-классов сервиса в проект добавляется ссылка на сборку `Microsoft.Data.Services.Client.dll`, которая реализует поддержку протокола OData 3. Если в клиентском приложении необходимо использовать протокол более ранней версии, то ссылку на соответствующую сборку необходимо добавить вручную. Данная клиентская библиотека позволяет выполнять запросы к сервису данных `EntityDataService.svc`, используя стандартные шаблоны программирования .NET Framework, включая использование языка запросов LINQ.

Для успешной компиляции в код проекта необходимо **добавить** директивы `using` и объявление переменной адреса сервиса OData.

Директивы `using`

```
using System;
using System.Data.Services.Client;
using System.Net;
using Terrasoft.Sdk.Examples.CreatioServiceReference;
using System.Linq;
```

Объявление переменной адреса сервиса OData

```
private static Uri serverUri = new Uri("http://<имя_сервера>/<имя_приложения>/0/ServiceModel/Ent
```

Ограничения при использовании протокола OData

При использовании протокола OData необходимо учитывать следующие **ограничения**:

- Невозможно создать [системных пользователей](#).
- Невозможно задать культуру возвращаемых данных. Культура определяется культурой пользователя от имени которого выполняется запрос.
- [Тело ответа на запрос](#) может содержать максимум 20 000 строк.
- [Пакетный запрос](#) может содержать максимум 100 подзапросов.
- Максимальный размер файла, который можно загрузить с помощью запроса, задается [системной настройкой](#) [`MaxFileSize`] (по умолчанию — 10 Мб).

Примеры интеграций по OData



Creatio API документация, которая содержит примеры различных CRUD-операций к Creatio с

использованием протоколов OData 3 и OData 4, доступна на [сайте Postman](#).

Примеры запросов с типом данных Stream



Элементы с типом данных Stream:

- Изображения.
- Файлы.
- Двоичные данные.

Для работы с типом данных Stream используются стандартные **методы**:

- GET — получение данных.
- POST — добавление данных.
- PUT — изменение данных.
- DELETE — удаление данных.

Для отображения результата выполнения запросов к Creatio при работе с типом данных Stream необходимо очистить кэш браузера.

Получить данные

Пример. Используя сервис работы с данными OData, получить фото контакта "New user".

Реализация примера

1. Получите идентификатор фото контакта "New user".

Фото контакта содержится в колонке [Data] таблицы [SysImage] базы данных. Чтобы **получить идентификатор фото контакта** "New user", выполните следующий SQL-запрос.

SQL-запрос

```
select Id from SysImage where Id = (select PhotoId from Contact where Name = 'New user')
```

Ответ на SQL-запрос

```
29FE7EDF-4DB9-4E09-92B0-018047BA1F71
```

2. Получите фото контакта "New user".

Чтобы **получить фото контакта** "New user", выполните следующий запрос.

Запрос

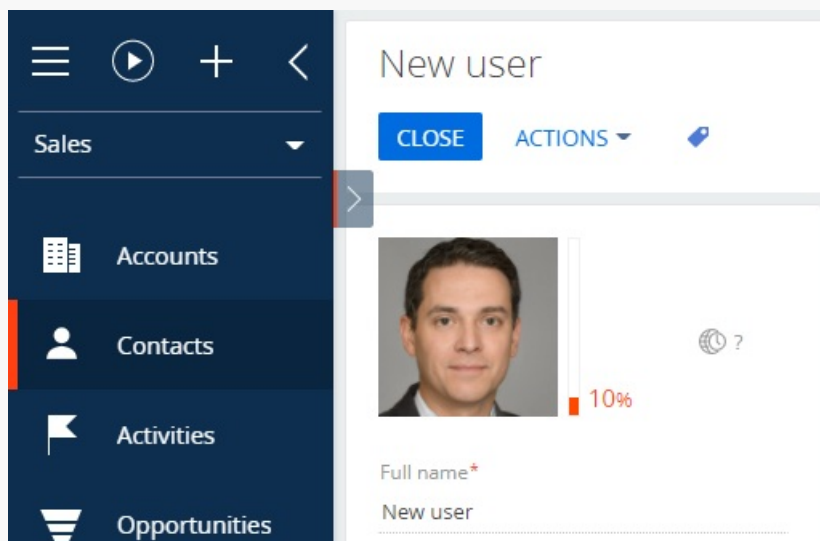
```
// Получить значение поля [Data] экземпляра объекта с [Id] 29FE7EDF-4DB9-4E09-92B0-018047BA1F
GET http://mycreatio.com/0/odata/SysImage(29FE7EDF-4DB9-4E09-92B0-018047BA1F71)/Data
```

Ответ на запрос

Status: ● 200 OK



Результат в приложении Creatio



Добавить данные

Пример. Используя сервис работы с данными OData, добавить контакт "New user". Затем добавить контакту фото.



Реализация примера

1. Добавьте контакт "New user".

Все контакты содержатся в таблице [Contact] базы данных. Чтобы **добавить контакт** "New user", выполните следующий запрос.

Запрос

```
// Добавить экземпляр объекта коллекции [Contact].
POST http://mycreatio.com/0/odata/Contact

Accept: application/json; odata=verbose
Content-Type: application/json; odata=verbose; IEEE754Compatible=true
BPMCSRF: OpK/NuJJ1w/SQxmPvwNvf0

{
  // В поле [Name] записать имя контакта "New user".
  "Name": "New user"
}
```

Ответ на запрос

Status: ● 201 Created

```
{
  "@odata.context": "http://mycreatio.com/0/odata/$metadata#Contact/$entity",
  "Id": "4c63c8fa-467b-48a6-973f-b2069298404f",
  "Name": "New user",
  "OwnerId": "410006e1-ca4e-4502-a9ec-e54d922d2c00",
  "CreatedOn": "2021-01-14T08:33:29.009023Z",
  "CreatedBy": "410006e1-ca4e-4502-a9ec-e54d922d2c00",
  "ModifiedOn": "2021-01-14T08:33:29.009023Z",
  "ModifiedBy": "410006e1-ca4e-4502-a9ec-e54d922d2c00",
  "ProcessListeners": 0,
}
```

```

"Dear": "",
"SalutationTypeId": "00000000-0000-0000-0000-000000000000",
"GenderId": "00000000-0000-0000-0000-000000000000",
"AccountId": "00000000-0000-0000-0000-000000000000",
"DecisionRoleId": "00000000-0000-0000-0000-000000000000",
"TypeId": "00000000-0000-0000-0000-000000000000",
"JobId": "00000000-0000-0000-0000-000000000000",
"JobTitle": "",
"DepartmentId": "00000000-0000-0000-0000-000000000000",
"BirthDate": "0001-01-01T00:00:00Z",
"Phone": "",
"MobilePhone": "",
"HomePhone": "",
"Skype": "",
"Email": "",
"AddressTypeId": "00000000-0000-0000-0000-000000000000",
"Address": "",
"CityId": "00000000-0000-0000-0000-000000000000",
"RegionId": "00000000-0000-0000-0000-000000000000",
"Zip": "",
"CountryId": "00000000-0000-0000-0000-000000000000",
"DoNotUseEmail": false,
"DoNotUseCall": false,
"DoNotUseFax": false,
"DoNotUseSms": false,
"DoNotUseMail": false,
"Notes": "",
"Facebook": "",
"LinkedIn": "",
"Twitter": "",
"FacebookId": "",
"LinkedInId": "",
"TwitterId": "",
"ContactPhoto@odata.mediaEditLink": "Contact(4c63c8fa-467b-48a6-973f-b2069298404f)/ContactPhoto@odata.mediaEditLink",
"ContactPhoto@odata.mediaReadLink": "Contact(4c63c8fa-467b-48a6-973f-b2069298404f)/ContactPhoto@odata.mediaReadLink",
"ContactPhoto@odata.mediaContentType": "application/octet-stream",
"TwitterAFDAId": "00000000-0000-0000-0000-000000000000",
"FacebookAFDAId": "00000000-0000-0000-0000-000000000000",
"LinkedInAFDAId": "00000000-0000-0000-0000-000000000000",
"PhotoId": "00000000-0000-0000-0000-000000000000",
"GPSN": "",
"GPSE": "",
"Surname": "user",
"GivenName": "New",
"MiddleName": "",
"Confirmed": true,
"IsNonActualEmail": false,
"Completeness": 0,

```

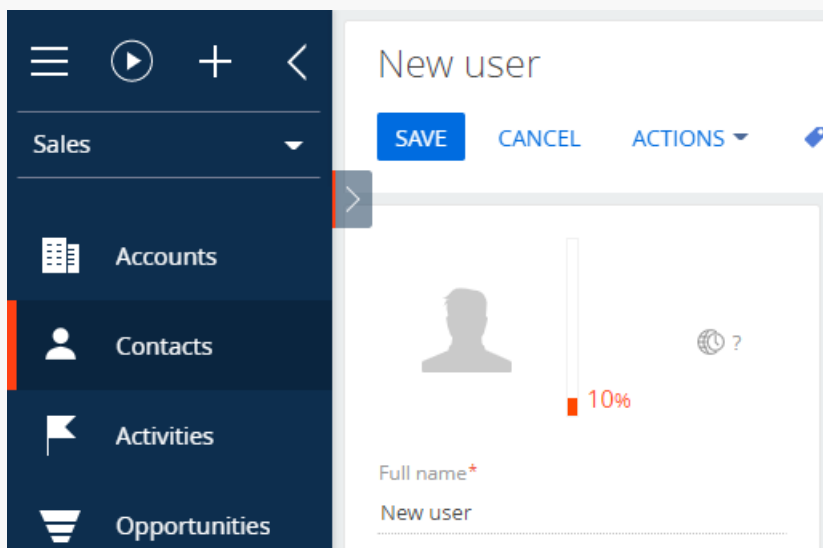


```

    "LanguageId": "6ebc31fa-ee6c-48e9-81bf-8003ac03b019",
    "Age": 0
  }

```

Результат в приложении Creatio



Идентификатор контакта "New user" "4c63c8fa-467b-48a6-973f-b2069298404f".

2. Добавьте фото контакта "New user".

Фото контакта должно содержаться в колонке [Data] таблицы [SysImage] базы данных. Для созданного контакта запись в таблице отсутствует, поэтому ее необходимо добавить. Чтобы **добавить запись в таблицу**, выполните следующий запрос.

Запрос

```

// Добавить экземпляр объекта коллекции [SysImage].
POST http://mycreatio.com/0/odata/SysImage

Accept: application/json; odata=verbose
Content-Type: application/json; odata=verbose; IEEE754Compatible=true
BPMCSRF: OpK/NuJJ1w/SQxmPvwNvfO

{
  // В поле [Name] записать название файла с фото контакта.
  "Name": "scr_NewContactPhoto.png",
  // В поле [Id] записать произвольный идентификатор записи в таблице [SysImage].
  "Id": "410006E1-CA4E-4502-A9EC-E54D922D2C01",
  // В поле [MimeType] записать тип файла с фото контакта.
  "MimeType": "image/png"
}

```

Ответ на запрос

Status: ● 201 Created

```
{
  "@odata.context": "http://mycreatio.com/0/odata/$metadata#SysImage/$entity",
  "Id": "410006e1-ca4e-4502-a9ec-e54d922d2c01",
  "CreatedOn": "2021-01-14T08:52:47.7573789Z",
  "CreatedBy": "410006e1-ca4e-4502-a9ec-e54d922d2c00",
  "ModifiedOn": "2021-01-14T08:52:47.7573789Z",
  "ModifiedBy": "410006e1-ca4e-4502-a9ec-e54d922d2c00",
  "ProcessListeners": 0,
  "UploadedOn": "0001-01-01T00:00:00Z",
  "Name": "scr_NewContactPhoto.png",
  "Data@odata.mediaEditLink": "SysImage(410006e1-ca4e-4502-a9ec-e54d922d2c01)/Data",
  "Data@odata.mediaReadLink": "SysImage(410006e1-ca4e-4502-a9ec-e54d922d2c01)/Data",
  "Data@odata.mediaContentType": "application/octet-stream",
  "MimeType": "image/png",
  "HasRef": false,
  "PreviewData@odata.mediaEditLink": "SysImage(410006e1-ca4e-4502-a9ec-e54d922d2c01)/Preview",
  "PreviewData@odata.mediaReadLink": "SysImage(410006e1-ca4e-4502-a9ec-e54d922d2c01)/Preview",
  "PreviewData@odata.mediaContentType": "application/octet-stream"
}
```

В таблицу [SysImage] базы данных была добавлена запись, но колонка [Data] содержит значение "0x".

Изображение необходимо передать в теле запроса, а название изображения должно совпадать из значением поля [Name]. Чтобы **добавить фото контакта** в колонку [Data], выполните следующий запрос.

Запрос

```
// Изменить значение поля [Data] экземпляра объекта с [Id] 410006e1-ca4e-4502-a9ec-e54d922d2c01
PUT http://mycreatio.com/0/odata/SysImage(410006e1-ca4e-4502-a9ec-e54d922d2c01)/Data
```

```
Accept: application/json; text/plain; */*
```

```
Content-Type: application/octet-stream; IEEE754Compatible=true
```

```
BPMCSRF: OpK/NuJJ1w/SQxmPvwNvfO
```



Ответ на запрос

Status: ● 200 OK

3. Выполните привязку добавленного фото к контакту "New user".

Для выполнения привязки фото к контакту "New user" необходимо установить связь между полем [Data] таблицы [SysImage] и полем [PhotoId] таблицы [Contact]. Чтобы **установить привязку**, выполните следующий запрос.

Запрос

```
// Изменить поле [PhotoId] экземпляра объекта с [Id] 4c63c8fa-467b-48a6-973f-b2069298404f кол
PATCH http://mycreatio.com/0/odata/Contact(4c63c8fa-467b-48a6-973f-b2069298404f)
```

Accept: application/json;odata=verbose

Content-Type: application/json; odata=verbose; IEEE754Compatible=true

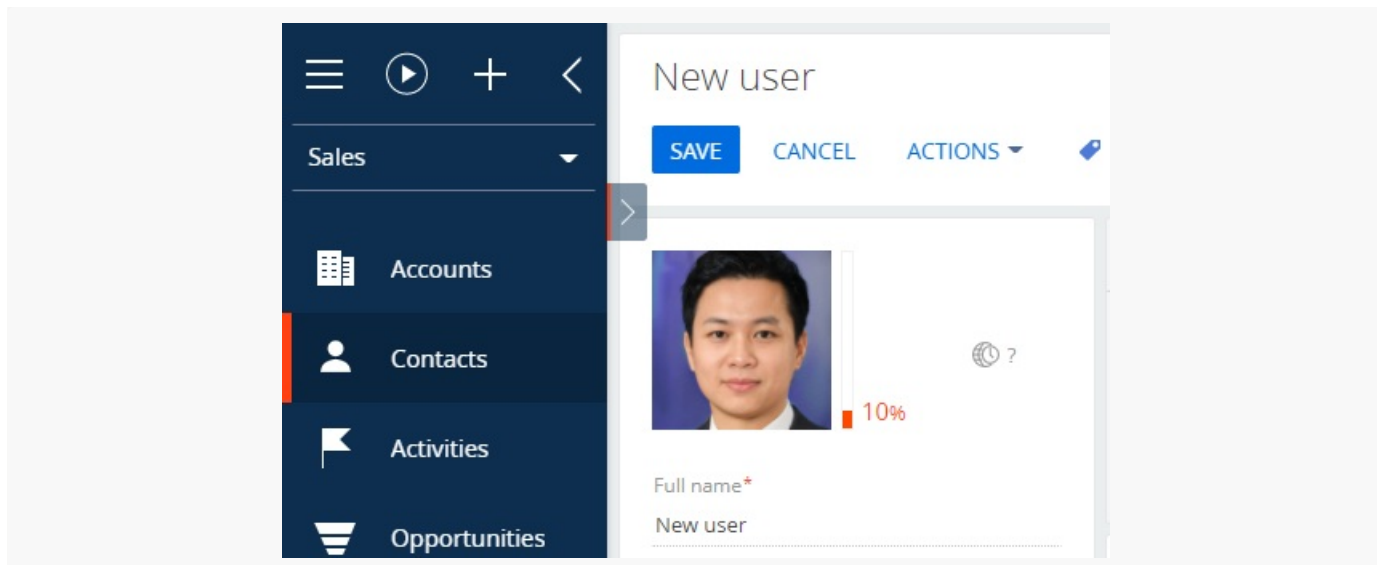
BPMCSRF: OpK/NuJJ1w/SQxmPvwNvf0

```
{
  // В поле [PhotoId] записать идентификатор записи в таблице [SysImage].
  "PhotoId": "410006e1-ca4e-4502-a9ec-e54d922d2c01"
}
```

Ответ на запрос

Status: ● 204 No Content

Результат в приложении Creatio



Чтобы **добавить фото существующему контакту** выполните:

1. `POST` -запрос на добавление экземпляра объекта коллекции `[SysImage]`.
2. `PUT` -запрос на изменение значения поля `[Data]` экземпляра объекта коллекции `[SysImage]`.
3. `PATCH` -запрос на выполнение привязку добавленного фото к контакту "New user".

Изменить данные

Пример. Используя сервис работы с данными OData, изменить фото существующего контакта "New user".



Реализация примера

1. Получите идентификатор фото контакта "New user".

Фото контакта содержится в колонке `[Data]` таблицы `[SysImage]` базы данных. Чтобы **получить идентификатор фото контакта** "New user", выполните следующий SQL-запрос.

SQL-запрос

```
select Id from SysImage where Id = (select PhotoId from Contact where Name = 'New user')
```

Ответ на SQL-запрос

29FE7EDF-4DB9-4E09-92B0-018047BA1F71

2. Измените фото контакта "New user".

Чтобы **изменить фото контакта** "New user", выполните следующий запрос.

Запрос

```
// Изменить поле [Data] экземпляра объекта с [Id] 29FE7EDF-4DB9-4E09-92B0-018047BA1F71 коллек  
PUT http://mycreatio.com/0/odata/SysImage(29FE7EDF-4DB9-4E09-92B0-018047BA1F71)/Data
```

Accept: application/json; text/plain; */*

Content-Type: application/octet-stream; IEEE754Compatible=true

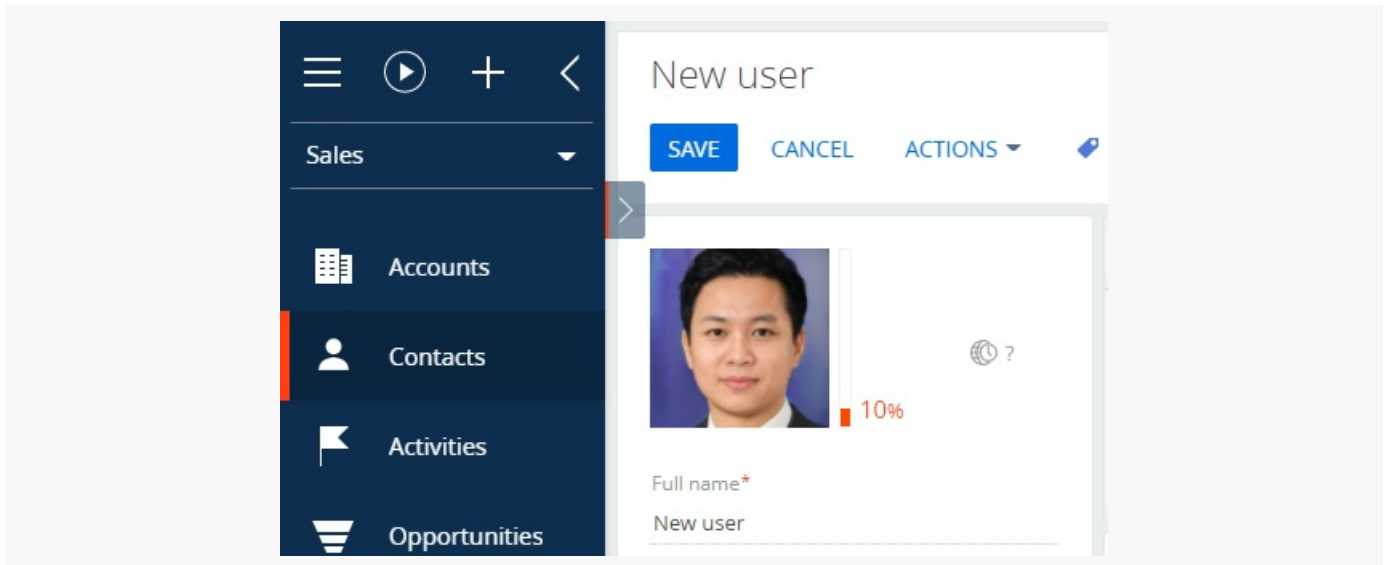
BPMCSRF: OpK/NuJJ1w/SQxmPvwNvf0



Ответ на запрос

Status: ● 200 OK

Результат в приложении Creatio



Удалить данные

Пример. Используя сервис работы с данными OData, удалить фото контакта "New user".

Реализация примера

1. Получите идентификатор фото контакта "New user".

Фото контакта содержится в колонке `[Data]` таблицы `[SysImage]` базы данных. Чтобы **получить идентификатор фото контакта** "New user", выполните следующий SQL-запрос.

SQL-запрос

```
select Id from SysImage where Id = (select PhotoId from Contact where Name = 'New user')
```

Ответ на SQL-запрос

```
29FE7EDF-4DB9-4E09-92B0-018047BA1F71
```

2. Удалить фото контакта "New user".

Чтобы **удалить фото контакта** "New user", выполните следующий запрос.

Запрос

```
// Удалить значение поля [Data] экземпляра объекта с [Id] 29FE7EDF-4DB9-4E09-92B0-018047BA1F7
DELETE http://mycreatio.com/0/odata/SysImage(29FE7EDF-4DB9-4E09-92B0-018047BA1F71)/Data
```

Accept: application/json; text/plain; */*

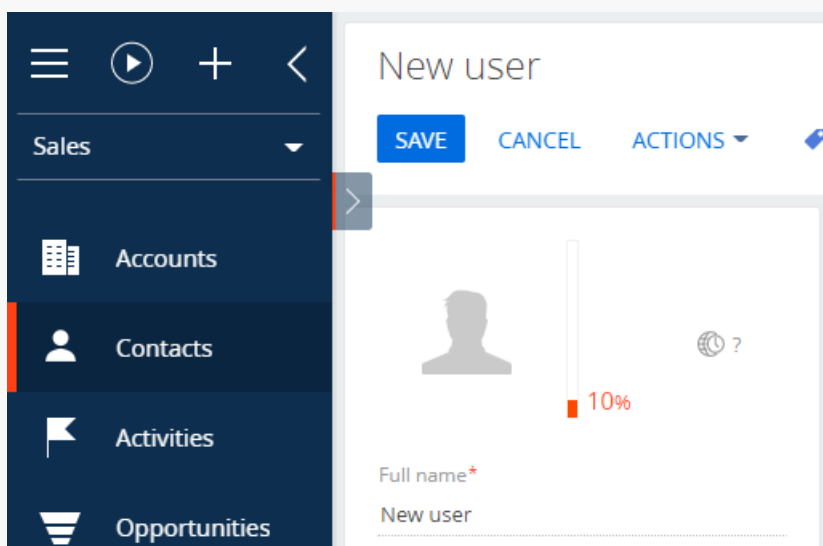
Content-Type: application/json; charset=utf-8; IEEE754Compatible=true

BPMCSRF: OpK/NuJJ1w/SQxmPvwNvf0

Ответ на запрос

Status: ● 204 No Content

Результат в приложении Creatio



Примеры пакетных запросов



Сложный

Пакетный запрос (batch-запрос) позволяет объединить множество HTTP-запросов в один, указав в теле каждый запрос как отдельный объект. Сервер баз данных Creatio возвращает один HTTP-ответ, внутри которого содержатся ответы на каждый переданный запрос. Использование пакетных запросов позволяет повысить производительность приложения.

Для пакетных запросов используются:

- HTTP-метод `POST`.
- Параметр `$batch`.

- Заголовок `Content-Type`.

Значения заголовка `Content-Type`:

- `application/json` — позволяет установить единый тип возвращаемого сервером контента для всех запросов пакетного запроса.
- `multipart/mixed` — позволяет в теле пакетного запроса установить свой заголовок `Content-Type` для каждого запроса.

Если один из запросов завершается с кодом ответа групп 4xx-5xx, то прерывается выполнение последующих запросов. Чтобы после неуспешного выполнения запроса продолжить выполнение следующих запросов, необходимо к основному HTTP-запросу добавить заголовок [Prefer: continue-on-error](#).

Важно. Максимальное количество запросов в пакетном запросе — 100.

Пакетный запрос (Content-Type: application/json)

Пакетный запрос

POST `http://mycreatio.com/0/odata/$batch`

`Content-Type: application/json; odata=verbose; IEEE754Compatible=true`

`Accept: application/json`

`BPMCSRF: OpK/NuJJ1w/SQxmPvwNvfO`

`ForceUseSession: true`

```
{
  "requests": [
    {
      // Добавить экземпляр объекта коллекции City.
      "method": "POST",
      "url": "City",
      "id": "t3",
      "body": {
        // Добавить в поле Name значение Burbank.
        "Name": "Burbank"
      },
      "headers": {
        "Content-Type": "application/json;odata=verbose",
        "Accept": "application/json;odata=verbose",
        "Prefer": "continue-on-error"
      }
    },
    {
      // Добавить экземпляр объекта коллекции City.
```



```

    "method": "POST",
    "atomicityGroup": "g1",
    "url": "City",
    "id": "t3",
    "body": {
      // Добавить в поле Id значение 62f9bc01-57cf-4cc7-90bf-8672acc922e3.
      "Id": "62f9bc01-57cf-4cc7-90bf-8672acc922e3",
      // Добавить в поле Name значение Spokane.
      "Name": "Spokane"
    },
    "headers": {
      "Content-Type": "application/json;odata=verbose",
      "Accept": "application/json;odata=verbose",
      "Prefer": "continue-on-error"
    }
  },
  {
    // Изменить экземпляр объекта коллекции City с идентификатором 62f9bc01-57cf-4cc7-90bf-8
    "method": "PATCH",
    "atomicityGroup": "g1",
    "url": "City/62f9bc01-57cf-4cc7-90bf-8672acc922e3",
    "id": "t2",
    "body": {
      // Изменить значение поля Name на Texas.
      "Name": "Texas"
    },
    "headers": {
      "Content-Type": "application/json;odata=verbose",
      "Accept": "application/json;odata=verbose",
      "Prefer": "continue-on-error"
    }
  }
]
}

```

Ответ на пакетный запрос

Status: ● 200 OK

```

{
  "responses": [
    {
      "id": "t3",
      "status": 201,
      "headers": {
        "location": "https://mycreatio.com/0/odata/City(b6a05348-55b1-4314-a228-436ba305

```

```

        "content-type": "application/json; odata.metadata=minimal",
        "odata-version": "4.0"
    },
    "body": {
        "@odata.context": "https://mycreatio.com/0/odata/$metadata#City/$entity",
        "Id": "b6a05348-55b1-4314-a228-436ba305d2f3",
        "CreatedOn": "2020-05-18T17:50:39.2838673Z",
        "CreatedById": "dad159f3-6c2d-446a-98d2-0f4d26662bbe",
        "ModifiedOn": "2020-05-18T17:50:39.2838673Z",
        "ModifiedById": "dad159f3-6c2d-446a-98d2-0f4d26662bbe",
        "Name": "Burbank",
        "Description": "",
        "CountryId": "00000000-0000-0000-0000-000000000000",
        "RegionId": "00000000-0000-0000-0000-000000000000",
        "TimeZoneId": "00000000-0000-0000-0000-000000000000",
        "ProcessListeners": 0
    }
},
{
    "id": "t3",
    "atomicityGroup": "c59e36b2-2aa9-44fa-86d3-e7d68eecbfa0",
    "status": 201,
    "headers": {
        "location": "https://mycreatio.com/0/odata/City(62f9bc01-57cf-4cc7-90bf-8672acc922e3)",
        "content-type": "application/json; odata.metadata=minimal",
        "odata-version": "4.0"
    },
    "body": {
        "@odata.context": "https://mycreatio.com/0/odata/$metadata#City/$entity",
        "Id": "62f9bc01-57cf-4cc7-90bf-8672acc922e3",
        "CreatedOn": "2020-05-18T17:50:39.361994Z",
        "CreatedById": "dad159f3-6c2d-446a-98d2-0f4d26662bbe",
        "ModifiedOn": "2020-05-18T17:50:39.361994Z",
        "ModifiedById": "dad159f3-6c2d-446a-98d2-0f4d26662bbe",
        "Name": "Spokane",
        "Description": "",
        "CountryId": "00000000-0000-0000-0000-000000000000",
        "RegionId": "00000000-0000-0000-0000-000000000000",
        "TimeZoneId": "00000000-0000-0000-0000-000000000000",
        "ProcessListeners": 0
    }
},
{
    "id": "t2",
    "atomicityGroup": "c59e36b2-2aa9-44fa-86d3-e7d68eecbfa0",
    "status": 204,
    "headers": {
        "odata-version": "4.0"
    }
}

```

```

    }
  ]
}

```

Пакетный запрос (Content-Type: application/json и Prefer: continue-on-error)

Пакетный запрос

POST http://mycreatio.com/0/odata/\$batch

Content-Type: application/json; odata=verbose; IEEE754Compatible=true

Accept: application/json

BPMCSRF: OpK/NuJJ1w/SQxmPvwNvf0

ForceUseSession: true

Prefer: continue-on-error

```

{
  "requests": [
    {
      // Добавить экземпляр объекта коллекции City.
      "method": "POST",
      "url": "City",
      "id": "t3",
      "body": {
        // Добавить в поле Name значение Burbank.
        "Name": "Burbank"
      },
      "headers": {
        "Content-Type": "application/json;odata=verbose",
        "Accept": "application/json;odata=verbose",
        "Prefer": "continue-on-error"
      }
    },
    {
      // Изменить экземпляр объекта коллекции City с идентификатором 62f9bc01-57cf-4cc7-90bf-8
      "method": "PATCH",
      "atomicityGroup": "g1",
      "url": "City/62f9bc01-57cf-4cc7-90bf-8672acc922e2",
      "id": "t2",
      "body": {
        // Изменить значение поля Name на Indiana.
        "Name": "Indiana"
      }
    },
  ]
}

```

```

    "headers": {
      "Content-Type": "application/json;odata=verbose",
      "Accept": "application/json;odata=verbose",
      "Prefer": "continue-on-error"
    }
  },
  {
    // Добавить экземпляр объекта коллекции City.
    "method": "POST",
    "atomicityGroup": "g1",
    "url": "City",
    "id": "t3",
    "body": {
      // Добавить в поле Id значение 62f9bc01-57cf-4cc7-90bf-8672acc922a1.
      "Id": "62f9bc01-57cf-4cc7-90bf-8672acc922a1",
      // Добавить в поле Name значение Iowa.
      "Name": "Iowa"
    },
    "headers": {
      "Content-Type": "application/json;odata=verbose",
      "Accept": "application/json;odata=verbose",
      "Prefer": "continue-on-error"
    }
  }
]
}

```

Ответ на пакетный запрос

Status: ● 200 OK

```

{
  "responses": [
    {
      "id": "t3",
      "status": 201,
      "headers": {
        "location": "https://mycreatio.com/0/odata/City(2f5e68f8-38bd-43c1-8e15-a2f13b0aa56a)",
        "content-type": "application/json; odata.metadata=minimal",
        "odata-version": "4.0"
      },
      "body": {
        "@odata.context": "https://mycreatio.com/0/odata/$metadata#City/$entity",
        "Id": "2f5e68f8-38bd-43c1-8e15-a2f13b0aa56a",
        "CreatedOn": "2020-05-18T18:06:50.7329808Z",
        "CreatedById": "dad159f3-6c2d-446a-98d2-0f4d26662bbe",
        "ModifiedOn": "2020-05-18T18:06:50.7329808Z",

```

```

        "ModifiedById": "dad159f3-6c2d-446a-98d2-0f4d26662bbe",
        "Name": "Burbank",
        "Description": "",
        "CountryId": "00000000-0000-0000-0000-000000000000",
        "RegionId": "00000000-0000-0000-0000-000000000000",
        "TimeZoneId": "00000000-0000-0000-0000-000000000000",
        "ProcessListeners": 0
    }
},
{
    "id": "t2",
    "atomicityGroup": "0c1c4019-b9fb-4fb3-8642-2d0660c4551a",
    "status": 204,
    "headers": {
        "odata-version": "4.0"
    }
},
{
    "id": "t3",
    "atomicityGroup": "0c1c4019-b9fb-4fb3-8642-2d0660c4551a",
    "status": 201,
    "headers": {
        "location": "https://mycreatio.com/0/odata/City(62f9bc01-57cf-4cc7-90bf-8672acc922a1)",
        "content-type": "application/json; odata.metadata=minimal",
        "odata-version": "4.0"
    },
    "body": {
        "@odata.context": "https://mycreatio.com/0/odata/$metadata#City/$entity",
        "Id": "62f9bc01-57cf-4cc7-90bf-8672acc922a1",
        "CreatedOn": "2020-05-18T18:06:50.7954775Z",
        "CreatedById": "dad159f3-6c2d-446a-98d2-0f4d26662bbe",
        "ModifiedOn": "2020-05-18T18:06:50.7954775Z",
        "ModifiedById": "dad159f3-6c2d-446a-98d2-0f4d26662bbe",
        "Name": "Iowa",
        "Description": "",
        "CountryId": "00000000-0000-0000-0000-000000000000",
        "RegionId": "00000000-0000-0000-0000-000000000000",
        "TimeZoneId": "00000000-0000-0000-0000-000000000000",
        "ProcessListeners": 0
    }
}
]
}

```

Пакетный запрос (Content-Type: multipart/mixed)

Пакетный запрос

```
POST http://mycreatio.com/0/odata/$batch
```

```
Content-Type: multipart/mixed;boundary=batch_a685-9724-d873; IEEE754Compatible=true
```

```
BPMCSRF: OpK/NuJJ1w/SQxmPvwNvf0
```

```
ForceUseSession: true
```

```
--batch_a685-9724-d873
```

```
Content-Type: multipart/mixed; boundary=changeset_06da-d998-8e7e
```

```
--changeset_06da-d998-8e7e
```

```
Content-Type: application/http
```

```
Content-Transfer-Encoding: binary
```

```
// Добавить экземпляр объекта коллекции City.
```

```
POST City HTTP/1.1
```

```
Content-ID: 1
```

```
Accept: application/atomsvc+xml;q=0.8, application/json;odata=verbose;q=0.5, */*;q=0.1
```

```
Content-Type: application/json;odata=verbose
```

```
// Добавить в поле Name значение Gilbert.
```

```
{"Name": "Gilbert"}
```

```
--changeset_06da-d998-8e7e
```

```
Content-Type: application/http
```

```
Content-Transfer-Encoding: binary
```

```
// Изменить экземпляр объекта коллекции City с идентификатором 62f9bc01-57cf-4cc7-90bf-8672acc922e2
```

```
PATCH City/62f9bc01-57cf-4cc7-90bf-8672acc922e2 HTTP/1.1
```

```
Content-ID: 2
```

```
Accept: application/atomsvc+xml;q=0.8, application/json;odata=verbose;q=0.5, */*;q=0.1
```

```
Content-Type: application/json;odata=verbose
```

```
// Изменить значение поля Name на Lincoln.
```

```
{"Name": "Lincoln"}
```

```
--changeset_06da-d998-8e7e
```

```
Content-Type: application/http
```

```
Content-Transfer-Encoding: binary
```

```
// Удалить экземпляр объекта коллекции City с идентификатором 62f9bc01-57cf-4cc7-90bf-8672acc922e2
```

```
DELETE City/62f9bc01-57cf-4cc7-90bf-8672acc922e2 HTTP/1.1
```

```
Content-ID: 3
```

```
Accept: application/atomsvc+xml;q=0.8, application/json;odata=verbose;q=0.5, */*;q=0.1
```

```
Content-Type: application/json;odata=verbose
```

Ответ на пакетный запрос

Status: ● 200 OK

--batchresponse_e17aace9-5cbb-49bd-b7ad-f1be8cc8c9d8

Content-Type: multipart/mixed; boundary=changesetresponse_a08c1df6-4b82-4a9b-be61-7ef4cc7b23ba

--changesetresponse_a08c1df6-4b82-4a9b-be61-7ef4cc7b23ba

Content-Type: application/http

Content-Transfer-Encoding: binary

Content-ID: 1

HTTP/1.1 201 Created

Location: https://mycreatio.com/0/odata/City(fbd0565f-fa8a-4214-ae89-c976c5f3acb4)

Content-Type: application/json; odata.metadata=minimal

OData-Version: 4.0

```
{
  "@odata.context": "https://mycreatio.com/0/odata/$metadata#City/$entity",
  "Id": "fbd0565f-fa8a-4214-ae89-c976c5f3acb4",
  "CreatedOn": "2020-05-18T18:41:57.0917235Z",
  "CreatedById": "dad159f3-6c2d-446a-98d2-0f4d26662bbe",
  "ModifiedOn": "2020-05-18T18:41:57.0917235Z",
  "ModifiedById": "dad159f3-6c2d-446a-98d2-0f4d26662bbe",
  "Name": "Gilbert",
  "Description": "",
  "CountryId": "00000000-0000-0000-0000-000000000000",
  "RegionId": "00000000-0000-0000-0000-000000000000",
  "TimeZoneId": "00000000-0000-0000-0000-000000000000",
  "ProcessListeners": 0
}
```

--changesetresponse_a08c1df6-4b82-4a9b-be61-7ef4cc7b23ba

Content-Type: application/http

Content-Transfer-Encoding: binary

Content-ID: 2

HTTP/1.1 204 No Content

OData-Version: 4.0

--changesetresponse_a08c1df6-4b82-4a9b-be61-7ef4cc7b23ba

Content-Type: application/http

Content-Transfer-Encoding: binary

Content-ID: 3

HTTP/1.1 204 No Content

```
--changesetresponse_a08c1df6-4b82-4a9b-be61-7ef4cc7b23ba--
--batchresponse_e17aace9-5cbb-49bd-b7ad-f1be8cc8c9d8--
```

Пакетный запрос (Content-Type: multipart/mixed и разными наборами запросов)

Пакетный запрос

```
POST http://mycreatio.com/0/odata/$batch
```

```
Content-Type: multipart/mixed;boundary=batch_a685-9724-d873; IEEE754Compatible=true
Accept: application/json
BPMCSRF: OpK/NuJJ1w/SQxmPvwNvf0
ForceUseSession: true
```

```
--batch_a685-9724-d873
Content-Type: multipart/mixed; boundary=changeset_06da-d998-8e7e
```

```
--changeset_06da-d998-8e7e
Content-Type: application/http
Content-Transfer-Encoding: binary
```

```
// Добавить экземпляр объекта коллекции City.
POST City HTTP/1.1
Content-ID: 1
Accept: application/atomsvc+xml;q=0.8, application/json;odata=verbose;q=0.5, */*;q=0.1
Content-Type: application/json;odata=verbose
```

```
// Добавить в поле Id значение d6bc67b1-9943-4e47-9aaf-91bf83e9c285.
// Добавить в поле Name значение Nebraska.
{"Id": "d6bc67b1-9943-4e47-9aaf-91bf83e9c285", "Name": "Nebraska"}
```

```
--batch_a685-9724-d873
Content-Type: multipart/mixed; boundary=changeset_06da-d998-8e71
```

```
--changeset_06da-d998-8e71
Content-Type: application/http
Content-Transfer-Encoding: binary
```

```
// Добавить экземпляр объекта коллекции City.
POST City HTTP/1.1
Content-ID: 2
Accept: application/atomsvc+xml;q=0.8, application/json;odata=verbose;q=0.5, */*;q=0.1
```



```
Content-Type: application/json;odata=verbose
```

```
// Добавить в поле Id значение d6bc67b1-9943-4e47-9aaf-91bf83e9c286.
// Добавить в поле Name значение Durham.
{"Id": "d6bc67b1-9943-4e47-9aaf-91bf83e9c286", "Name": "Durham"}
```

Ответ на пакетный запрос

Status: ● 200 OK

```
{
  "responses": [
    {
      "id": "1",
      "atomicityGroup": "e9621f72-42bd-47c1-b271-1027e4b68e3b",
      "status": 201,
      "headers": {
        "location": "https://mycreatio.com/0/odata/City(d6bc67b1-9943-4e47-9aaf-91bf83e9c286)",
        "content-type": "application/json; odata.metadata=minimal",
        "odata-version": "4.0"
      },
      "body": {
        "@odata.context": "https://mycreatio.com/0/odata/$metadata#City/$entity",
        "Id": "d6bc67b1-9943-4e47-9aaf-91bf83e9c285",
        "CreatedOn": "2020-05-18T18:49:16.3766324Z",
        "CreatedById": "dad159f3-6c2d-446a-98d2-0f4d26662bbe",
        "ModifiedOn": "2020-05-18T18:49:16.3766324Z",
        "ModifiedById": "dad159f3-6c2d-446a-98d2-0f4d26662bbe",
        "Name": "Nebraska",
        "Description": "",
        "CountryId": "00000000-0000-0000-0000-000000000000",
        "RegionId": "00000000-0000-0000-0000-000000000000",
        "TimeZoneId": "00000000-0000-0000-0000-000000000000",
        "ProcessListeners": 0
      }
    },
    {
      "id": "2",
      "atomicityGroup": "960e2272-d8cb-4b4d-827c-0181485dd71d",
      "status": 201,
      "headers": {
        "location": "https://mycreatio.com/0/odata/City(d6bc67b1-9943-4e47-9aaf-91bf83e9c286)",
        "content-type": "application/json; odata.metadata=minimal",
        "odata-version": "4.0"
      },
      "body": {
        "@odata.context": "https://mycreatio.com/0/odata/$metadata#City/$entity",
```

```

        "Id": "d6bc67b1-9943-4e47-9aaf-91bf83e9c286",
        "CreatedOn": "2020-05-18T18:49:16.4078852Z",
        "CreatedBy": "dad159f3-6c2d-446a-98d2-0f4d26662bbe",
        "ModifiedOn": "2020-05-18T18:49:16.4078852Z",
        "ModifiedBy": "dad159f3-6c2d-446a-98d2-0f4d26662bbe",
        "Name": "Durham",
        "Description": "",
        "CountryId": "00000000-0000-0000-0000-000000000000",
        "RegionId": "00000000-0000-0000-0000-000000000000",
        "TimeZoneId": "00000000-0000-0000-0000-000000000000",
        "ProcessListeners": 0
    }
}
]
}

```

Примеры запросов в WCF-клиенте



Для получения коллекции объектов сервиса используется универсальный класс [DataServiceQuery](#), который представляет собой запрос к сервису, возвращающий коллекцию сущностей конкретного типа.

Чтобы выполнить запрос к сервису данных `EntityDataService.svc`, предварительно необходимо создать экземпляр объекта контекста среды приложения `Creatio`.

Далее в примерах будет использована forms-аутентификация.

Чтобы **реализовать forms-аутентификацию**:

1. Создайте класс `LoginClass`.
2. Реализуйте поля `authServiceUri` (строка запроса к методу `Login` аутентификационного сервиса `AuthService.svc`) и `AuthCookie` (Cookie аутентификации `Creatio`).
3. Реализуйте метод `TryLogin(string userName, string userPassword)`, который выполняет аутентификацию пользователя и сохраняет ответ сервера в поле `AuthCookie`.
4. Реализуйте метод `OnSendingRequestCookie(object sender, SendingRequestEventArgs e)`, который будет вызван в ответ на событие экземпляра контекста `SendingRequest` (создание нового экземпляра `HttpRequest`).

В методе `OnSendingRequestCookie` выполняется аутентификация пользователя, а полученные в ответ cookies добавляются в запрос на получение данных.

OnSendingRequestCookie

```

static void OnSendingRequestCookie(object sender, SendingRequestEventArgs e)
{

```

```
// Вызов метода класса LoginClass, реализующего аутентификацию переданного в параметрах м
LoginClass.TryLogin("CreatioUserName", "CreatioUserPassword");
var req = e.Request as HttpWebRequest;
// Добавление полученных аутентификационных cookie в запрос на получение данных.
req.CookieContainer = LoginClass.AuthCookie;
e.Request = req;
}
```

Способы выполнения запроса к сервису:

- Выполнение LINQ-запроса к именованному объекту `DataServiceQuery`, который получен из контекста сервиса.
- Неявное перечисление объекта `DataServiceQuery`, который получен из контекста сервиса.
- Явный вызов метода [Execute](#) объекта `DataServiceQuery` или [BeginExecute](#) для асинхронного выполнения.

Получить коллекцию контактов через LINQ-запрос

Пример. Определить и выполнить запрос LINQ, возвращающий все сущности контактов сервиса `EntityDataService.svc`.

Реализация примера

LINQ-запрос

```
public static void GetDataCollectionByLinqWcfExample()
{
    // Создание контекста приложения Creatio.
    var context = new Creatio(serverUri);
    // Определение метода, который добавляет аутентификационные cookie при создании нового запроса
    context.SendingRequest += new EventHandler<SendingRequestEventArgs>(OnSendingRequestCookie);
    try
    {
        // Построение запроса LINQ для получение коллекции контактов.
        var allContacts = from contacts in context.ContactCollection
                           select contacts;
        foreach (Contact contact in allContacts)
        {
            // Выполнение действий с контактами.
        }
    }
    catch (Exception ex)
    {
        // Обработка ошибок.
    }
}
```

```

    }
}

```

Получить коллекцию контактов неявным запросом

Пример. Использовать контекст для неявного выполнения запроса, возвращающего все сущности контактов сервиса `EntityDataService.svc`.

Реализация примера

`GetOdataCollectionByImplicitRequestExample()`

```

public static void GetOdataCollectionByImplicitRequestExample()
{
    // Создание объекта контекста приложения Creatio.
    var context = new Creatio(serverUri);
    // Определение метода, который добавляет аутентификационные cookie при создании нового запроса
    context.SendingRequest += new EventHandler<SendingRequestEventArgs>(OnSendingRequestCookie);
    try
    {
        // Определение неявного запроса к сервису для получения коллекции контактов.
        DataServiceQuery<Contact> allContacts = context.ContactCollection;
        foreach (Contact contact in allContacts)
        {
            // Выполнение действий с контактами.
        }
    }
    catch (Exception ex)
    {
        // Обработка ошибок.
    }
}

```

Получить коллекцию контактов явным запросом

Пример. Использовать контекст [DataServiceContext](#) для явного выполнения запроса к сервису `EntityDataService.svc`, который возвращает все сущности контактов.

Реализация примера

GetOdataCollectionByExplicitRequestExample()

```

public static void GetOdataCollectionByExplicitRequestExample()
{
    // Определение Uri запроса к сервису, который возвращает коллекцию контактов.
    Uri contactUri = new Uri(serverUri, "ContactCollection");
    // Создание объекта контекста приложения Creatio.
    var context = new Creatio(serverUri);
    // Определение метода, который добавляет аутентификационные cookie при создании нового запроса
    context.SendingRequest += new EventHandler<SendingRequestEventArgs>(OnSendingRequestCookie);
    try
    {
        // Выполнение явного запроса к сервису вызовом метода Execute<>().
        foreach (Contact contact in context.Execute<Contact>(contactUri))
        {
            // Выполнение действий с контактами.
        }
    }
    catch (Exception ex)
    {
        // Обработка ошибок.
    }
}

```

Примеры CRUD-операций

Получение объекта

```

public static void GetOdataObjectByWcfExample()
{
    // Создание контекста приложения Creatio.
    var context = new Creatio(serverUri);
    // Определение метода, который добавляет аутентификационные cookie при создании нового запроса
    context.SendingRequest += new EventHandler<SendingRequestEventArgs>(OnSendingRequestCookie);
    //
    var contact = context.ContactCollection.Where(c => c.Name.Contains("User")).First();
    // Выполнение действий над контактом.
}

```

Создание объекта

```

public static void CreateCreatioEntityByOdataWcfExample()
{

```

```

// Создание нового контакта, инициализация свойств.
var contact = new Contact()
{
    Id = Guid.NewGuid(),
    Name = "New Test User"
};
// Создание и инициализация свойств нового контрагента, к которому относится создаваемый кон
var account = new Account()
{
    Id = Guid.NewGuid(),
    Name = "Some Company"
};
contact.Account = account;
// Создание контекста приложения Creatio.
var context = new Creatio(serverUri);
// Определение метода, который добавляет аутентификационные cookie при создании нового запроса
context.SendingRequest += new EventHandler<SendingRequestEventArgs>(OnSendingRequestCookie);
// Добавление созданного контакта в коллекцию контактов модели данных сервиса.
context.AddToAccountCollection(account);
// Добавление созданного контрагента в коллекцию контрагентов модели данных сервиса.
context.AddToContactCollection(contact);
// Установка связи между созданными контактом и контрагентом в модели данных сервиса.
context.SetLink(contact, "Account", account);
// Сохранение изменений данных в Creatio одним запросом.
DataServiceResponse responses = context.SaveChanges(SaveChangesOptions.Batch);
// Обработка ответов от сервера.
}

```

Изменение объекта

```

public static void UpdateCreatioEntityByOdataWcfExample()
{
    // Создание контекста приложения Creatio.
    var context = new Creatio(serverUri);
    // Определение метода, который добавляет аутентификационные cookie при создании нового запроса
    context.SendingRequest += new EventHandler<SendingRequestEventArgs>(OnSendingRequestCookie);
    // Из коллекции контактов выбирается тот, по которому будет изменяться информация.
    var updateContact = context.ContactCollection.Where(c =< c.Name.Contains("Test")).First();
    // Изменение свойств выбранного контакта.
    updateContact.Notes = "New updated description for this contact.";
    updateContact.Phone = "123456789";
    // Сохранение изменений в модели данных сервиса.
    context.UpdateObject(updateContact);
    // Сохранение изменений данных в Creatio одним запросом.
    var responses = context.SaveChanges(SaveChangesOptions.Batch);
}

```

Удаление объекта

```
public static void DeleteCreatioEntityByOdataWcfExample()
{
    // Создание контекста приложения Creatio.
    var context = new Creatio(serverUri);

    context.SendingRequest += new EventHandler<SendingRequestEventArgs>(OnSendingRequestCookie);
    // Из коллекции контактов выбирается тот объект, который будет удален.
    var deleteContact = context.ContactCollection.Where(c => c.Name.Contains("Test")).First();
    // Удаление выбранного объекта из модели данных сервиса.
    context.DeleteObject(deleteContact);
    // Сохранение изменений данных в Creatio одним запросом.
    var responses = context.SaveChanges(SaveChangesOptions.Batch);
    // Обработка ответов от сервера.
}
```

Веб-сервис odata (OData 4) API



Сложный

В зависимости от используемого типа запроса протокол OData 4 может возвращать различные данные. Структура запроса и ответа рассмотрена ниже.

Структура запроса

```
// Строка запроса.
method Creatio_application_address/0/odata/objects_collection(object_id)/object_field?$parameter

// Заголовки запроса.
Accept: application/json
Content-Type: application/json; charset=utf-8; IEEE754Compatible=true
ForceUseSession: true
BPMCSRF: authentication_cookie_value

// Тело запроса (используется в POST и PATCH запросах).
{
    "field1": "value1",
    "field2": "value2",
    ...
}
```

Структура ответа

```
// Код состояния.
Status: code

// Тело ответа (присутствует в GET и POST запросах).
{
  "@odata.context": "http://Creatio_application_address/0/odata/$metadata#data_resource",
  "value": [
    {
      "object1 field1": "object1 field_value1",
      "object1 field2": "object1 field_value2",
      ...
    },
    {
      "object2 field1": "object2 field_value1",
      "object2 field2": "object2 field_value2",
      ...
    },
    ...
  ]
}
```

Строка запроса

method **required**

Приложение Creatio поддерживает следующие методы запроса:

- GET — получение данных.
- POST — добавление данных.
- PATCH — изменение данных.
- DELETE — удаление данных.

Creatio_application_address **required**

Адрес приложения Creatio.

odata **required**

Адрес веб-сервиса протокола OData 4. Неизменяемая часть запроса.

objects_collection **required**

Имя таблицы базы данных (имя коллекции объектов). Получить перечень таблиц базы данных можно выполнив [аутентификацию](#) и один из запросов.

JSON-формат

```
// Результат будет получен в формате JSON.
GET Creatio_application_address/0/odata/

// Заголовки запроса.
ForceUseSession: true
BPMCSRF: authentication_cookie_value
```

XML-формат

```
// Результат будет получен в формате XML.
GET Creatio_application_address/0/odata/$metadata

// Заголовки запроса.
ForceUseSession: true
BPMCSRF: authentication_cookie_value
```

Данные таблицы [SysSchema]

```
// Результат будет получен из таблицы [SysSchema] базы данных в формате JSON.
GET Creatio_application_address/0/odata/SysSchema?$filter=ManagerName eq 'EntitySchemaManager'

// Заголовки запроса.
ForceUseSession: true
BPMCSRF: authentication_cookie_value
```

object_id **optional**

Идентификатор строки записи таблицы базы данных (идентификатор экземпляра объекта коллекции).

object_field **optional**

Поле записи таблицы базы данных (поле экземпляра объекта коллекции).

parameters **optional**

Необязательные параметры OData 4, которые разрешены к использованию в строке `GET`-запроса к Creatio. Для указания параметров необходимо использовать оператор `?`. Имя параметра должно записываться после оператора `$`. Чтобы использовать два и более параметров, необходимо воспользоваться оператором `&`.

Возможные параметры

<code>\$value</code>		Значение поля.
\$count	<code>\$count=true</code>	Количество элементов, которые попали в выборку.
\$skip	<code>\$skip=n</code>	n первых элементов, которые не должны попасть в выборку.
\$top	<code>\$top=n</code>	n первых элементов, которые должны попасть в выборку.
\$select	<code>\$select=field1,field2,...</code>	Набор полей, которые должны попасть в выборку.
\$orderby	<code>\$orderby=field asc</code> или <code>\$orderby=field desc</code>	Сортировка значений поля, которые попали в выборку.
\$expand	<code>\$expand=field1,field2,...</code>	Расширение связанных полей.
\$filter	<code>\$filter=field template 'field_value'</code>	Фильтрация полей, которые должны попасть в выборку.

Заголовки запроса

Accept `application/json` **required**

Тип данных, который можно ожидать в ответе от сервера. Не обязательный к использованию в `GET`-запросах.

Content-Type `application/json; charset=utf-8; IEEE754Compatible=true` **required**

Кодировка и тип ресурса, который передается в теле запроса. Не обязательный к использованию в `GET`-запросах.

ForceUseSession `true` **required**

Заголовок `ForceUseSession` отвечает за принудительное использование уже существующей сессии.

BPMCSRF `authentication_cookie_value` **required**

Аутентификационный cookie.

Тело запроса

`field1, field2, ...` **required**

Имена полей, которые передаются в теле запроса.

`value1, value2, ...` **required**

Значения полей `field1, field2, ...`, которые передаются в теле запроса.

Код состояния ответа

code

Код состояния ответа на запрос.

Возможные коды состояния

● 200 OK

Запрос, который не создает ресурс, успешно завершен и значение ресурса не равно нулю. В этом случае, тело ответа должно содержать значение ресурса, указанного в URL-адресе запроса. Информация, возвращаемая с ответом, зависит от метода, используемого в запросе:

`GET` — запрашиваемый ресурс был найден и передан в теле ответа.

`POST` — ресурс, описывающий результат действия сервера на запрос, передан в теле ответа.

● 201 Created

Запрос, который успешно создает ресурс. Тело ответа должно содержать созданный ресурс. Используется для `POST`-запросов, которые [создают коллекцию](#), [создают объект мультимедиа](#) (например, фотографию) или [вызывают действие через его импорт](#).

● 202 Accepted

Запрос на работу с данными принят в обработку, но еще не завершен. Нет гарантий, что запрос успешно выполнится в процессе обработки данных ([асинхронная обработка запроса](#)).

● 204 No content	Запрос был успешно обработан, но нет необходимости возвращать какие-либо данные. Запрашиваемый ресурс имеет нулевое значение. В ответе были переданы только заголовки, тело ответа должно быть пустым.
● 3xx Redirection	Перенаправление указывает что клиент должен предпринимать дальнейшие действия для выполнения запроса. Ответ должен включать заголовок <code>Location</code> с URL-адресом, по которому можно получить результат. Также ответ может включать заголовок <code>Retry-After</code> , который отображает время (в секундах). Это время характеризует период, в течение которого клиент может подождать прежде чем повторить запрос к ресурсу, который был возвращен в заголовке <code>Location</code> .
● 304 Not modified	Клиент выполняет <code>GET</code> -запрос с заголовком <code>If-None-Match</code> и содержимое не изменилось. Ответ не должен содержать другие заголовки.
● 403 Forbidden	Сервер понял запрос, но отказывается его авторизировать. Это означает, что клиент не уполномочен совершать операции с запрошенным ресурсом. Причиной может быть некорректная кука <code>BPMSRF</code> .
● 404 Not Found	Сервер не может найти ресурс, указанный в URL-адресе. Дополнительная информация может содержаться в теле ответа.
● 405 Method Not Allowed	Ресурс, указанный в URL-адресе, не поддерживает указанный метод запроса. В ответе должен быть получен заголовок <code>Allow</code> , который должен содержать список доступных методов запроса для ресурса.
● 406 Not Acceptable	Ресурс, указанный в URL-адресе запроса, не имеет текущего представления, которое было бы приемлемым для клиента в соответствии с Accept , Accept-Charset и Accept-Language заголовками запроса. Служба не желает предоставлять представление по умолчанию.
● 410 Gone	Запрошенный ресурс больше недоступен. Ресурс раньше был по указанному URL, но был удален и теперь недоступен.
● 412 Prediction Failed	Клиент указал в заголовке запроса условие, которое ресурс не может выполнить.
● 424 Failed Dependency	Текущий запрос к ресурсу невозможно выполнить, потому что запрошенное действие зависит от другого действия, выполнить которое не удалось. Запрос не был выполнен

● 501 Not Implemented

Элементы, которые не удаются, например, не могут зависеть от элементов, через собой зависимости.

Клиент использует метод запроса, который не реализован протоколом OData 4 и не может быть обработан. Тело ответа должно содержать описание нереализованного функционала.

Тело ответа

@odata.context

Информация о типе возвращаемых данных. Кроме пути к источнику данных, элемент `data_resource` может содержать параметр `$entity`, который показывает что в ответе был возвращен единственный экземпляр объекта коллекции. Может присутствовать только для `GET` и `POST` запросов.

value

Содержит коллекцию объектов. Отсутствует если в ответе был возвращен один экземпляр объекта коллекции. Может присутствовать только для `GET` запроса.

[]

Коллекция объектов. Может присутствовать только для `GET` запроса.

{}

Экземпляры объектов коллекции. Может присутствовать только для `GET` и `POST` запросов.

object1 field1, object1 field2, ..., object2 field1, object2 field2, ...

Имена полей `field1, field2, ...` экземпляров объектов `object1, object2, ...` коллекции. Может присутствовать только для `GET` и `POST` запросов.

object1 field_value1, object1 field_value2, ..., object2 field_value1, object2 field_value2, ...

Значения полей `field1, field2, ...` экземпляров объектов `object1, object2, ...` коллекции. Может присутствовать только для `GET` и `POST` запросов.

Веб-сервис EntityDataService.svc (OData 3) API



В зависимости от используемого типа запроса протокол OData 3 может возвращать различные данные. Структура запроса и ответа рассмотрена ниже.

Структура запроса

```
// Строка запроса.
method Creatio_application_address/0/ServiceModel/EntityDataService.svc/objects_collectionColle

// Заголовки запроса.
Accept: application/atom+xml; type=entry
Content-Type: application/json; odata=verbose
ForceUseSession: true
BPMCSRF: authentication_cookie_value

// Тело запроса (используется в POST и PATCH запросах).
{
  "field1": "value1",
  "field2": "value2",
  ...
}
```

Структура ответа

```
// Код состояния.
Status: code

// Тело ответа (присутствует для GET и POST запросов).
<?xml version="1.0" encoding="utf-8"?>
<feed xml:base="http://mycreatio.com/0/ServiceModel/EntityDataService.svc/" xmlns="http://www.w3
  <id>http://mycreatio.com/0/ServiceModel/EntityDataService.svc/data_resource</id>
  <title type="text">data_resource</title>
  <updated>date and time of request</updated>
  <link rel="self" title="data_resource" href="data_resource" />
  <entry>
    metadata_data
    <content type="application/xml">
      <m:properties>
        <d:object1 field1>object1 field_value1</d:object1 field1>
        <d:object1 field2>object1 field_value2</d:object1 field2>
        ...
      </m:properties>
    </content>
  </entry>
  <entry>
```

```

    metadata_data
    <content type="application/xml">
      <m:properties>
        <d:object2 field1>object2 field_value1</d:object2 field1>
        <d:object2 field2>object2 field_value2</d:object2 field2>
        ...
      </m:properties>
    </content>
  </entry>
  ...
</feed>

```

Строка запроса

method **required**

Приложение Creatio поддерживает следующие методы запроса:

- GET — получение данных.
- POST — добавление данных.
- PATCH — изменение данных.
- DELETE — удаление данных.

Creatio_application_address **required**

Адрес приложения Creatio.

ServiceModel **required**

Путь к веб-сервису протокола OData 3. Неизменяемая часть запроса.

EntityDataService.svc **required**

Адрес веб-сервиса протокола OData 3. Неизменяемая часть запроса.

objects_collectionCollection **required**

Имя таблицы базы данных (имя коллекции объектов). При использовании протокола OData 3 к первому имени коллекции объектов в строке запроса необходимо добавлять слово `Collection` (например, `ContactCollection`). Получить перечень таблиц базы данных можно выполнив запрос к базе данных.

MySQL

```
SELECT * FROM INFORMATION_SCHEMA.TABLES
```

Oracle

```
SELECT * FROM ALL_TABLES
```

PostgreSQL

```
SELECT table_name FROM information_schema.tables
```

`guid'object_id'` **optional**

Идентификатор строки записи таблицы базы данных (идентификатор экземпляра объекта коллекции). Например, `guid'00000000-0000-0000-0000-000000000000'`).

`object_field` **optional**

Поле записи таблицы базы данных (поле экземпляра объекта коллекции).

`parameters` **optional**

Необязательные параметры OData 3, которые разрешены к использованию в строке `GET` -запроса к Creatio. Для указания параметров необходимо использовать оператор `?` . Имя параметра должно записываться после оператора `$` . Чтобы использовать два и более параметров, необходимо воспользоваться оператором `&` .

Возможные параметры

\$value		Значение поля.
\$count	<code>\$count=true</code>	Количество элементов, которые попали в выборку.
\$skip	<code>\$skip=n</code>	n первых элементов, которые не должны попасть в выборку.
\$top	<code>\$top=n</code>	n первых элементов, которые должны попасть в выборку.
\$select	<code>\$select=field1,field2,...</code>	Набор полей, которые должны попасть в выборку.
\$orderby	<code>\$orderby=field asc</code> или <code>\$orderby=field desc</code>	Сортировка значений поля, которые попали в выборку.
\$expand	<code>\$expand=field1,field2,...</code>	Расширение связанных полей.
\$filter	<code>\$filter=field template 'field_value'</code>	Фильтрация полей, которые должны попасть в выборку.

Заголовки запроса

Accept application/atom+xml; type=entry **required**

Тип данных, который можно ожидать в ответе от сервера. Сервер возвращает ответ в формате XML. Не обязательный к использованию в GET-запросах.

Content-Type application/json; odata=verbose **required**

Кодировка и тип ресурса, который передается в теле запроса. Не обязательный к использованию в GET-запросах.

ForceUseSession true **required**

Заголовок ForceUseSession отвечает за принудительное использование уже существующей сессии. Отсутствует необходимость использования в запросе к сервису аутентификации AuthService.svc.

BPMCSRF authentication_cookie_value **required**

Аутентификационный cookie.

Тело запроса

field1, field2, ... **required**

Имена полей, которые передаются в теле запроса.

value1, value2, ... **required**

Значения полей `field1, field2, ...`, которые передаются в теле запроса.

Код состояния ответа

code

Код состояния ответа на запрос.

[Возможные коды состояния](#)

● 200 OK	Запрос <code>GET</code> , <code>PUT</code> , <code>MERGE</code> или <code>PATCH</code> успешно завершен. Тело ответа должно содержать значение объекта или свойства, указанного в URL-адресе запроса.
● 201 Created	Запрос <code>POST</code> успешно создал объект или ссылку. Тело ответа должно содержать обновленный объект.
● 202 Accepted	Запрос на изменение данных был принят в обработку, но еще не завершен. Тело ответа должно содержать <code>заголовок Location</code> в дополнение в <code>заголовке Retry-After</code> . Тело ответа должно быть пустым. Сервер должен вернуть код ответа 303 с <code>заголовком Location</code> , который содержит окончательный URL-адрес для получения результата запроса. Тело и заголовки окончательного URL-адреса должны быть отформатированы также, как и выполнение первоначального запроса на изменение данных.
● 204 No content	Запрос на изменение данных. Запрашиваемый ресурс имеет нулевое значение. Тело ответа должно быть пустым.
● 3xx Redirection	Запрос на изменение данных. Перенаправление указывает что клиент должен предпринимать дальнейшие действия для выполнения запроса. Ответ должен включать <code>заголовок Location</code> с URL-адресом, по которому можно получить результат.
● 4xx Client Error	Некорректные запросы. Сервер возвращает код в ответ на клиентские ошибки в дополнение к запросам на несуществующие ресурсы, такие как сущности, коллекции сущностей или свойства. Если тело ответа определено для кода ошибки, тело ошибки является таким, как определено для соответствующего <code>формата</code> .
● 404 Not Found	Объект или коллекция, указанные в URL-адресе, не существуют. Тело ответа должно быть пустым.

Тело ответа

entry

Экземпляр объекта коллекции.

metadata_data

Метаданные экземпляра объекта коллекции.

object1 field1, object1 field2, ..., object2 field1, object2 field2, ...

Имена полей `field1, field2, ...` экземпляров объектов `object1, object2, ...` коллекции.

`object1 field_value1, object1 field_value2, ..., object2 field_value1, object2 field_value2, ...`

Значения полей `field1, field2, ...` экземпляров объектов `object1, object2, ...` коллекции.

Аутентификация



Средний

Аутентификация — проверка подлинности предъявленного пользователем идентификатора. Положительным **результатом аутентификации** является авторизация пользователя, то есть предоставление ему прав доступа к ресурсам, определенным для выполнения его задач. Подробнее об аутентификации читайте в [Википедии](#).

Все внешние запросы к веб-сервисам Creatio должны быть аутентифицированы.

Виды аутентификации

Виды аутентификации, которые поддерживаются в Creatio:

- Анонимная аутентификация (Anonymous).
- Базовая аутентификация (Basic-аутентификация).
- Аутентификация на основе cookies (Forms-аутентификация).
- Аутентификация на основе открытого протокола авторизации OAuth 2.0. Настройка подробно описана в статье [Настроить авторизацию интегрированных приложений по протоколу OAuth 2.0](#).

Рекомендуемым способом аутентификации для интеграции с приложением является **Forms-аутентификация**, которая реализована с помощью веб-сервиса `AuthService.svc`. Cookie, полученные в ответ от веб-сервиса `AuthService.svc`, необходимо использовать в следующих запросах к Creatio.

Пример использования аутентификационного cookie приведен в описании сервисов работы с данными [OData](#) и [DataService](#).

Защита от CSRF-атак

CSRF (англ. Cross Site Request Forgery — "межсайтовая подделка запроса") — вид атак на посетителей веб-сайтов, использующий недостатки протокола HTTP. По умолчанию защита включена, но ее можно отключить.

Важно. Отключение защиты от CSRF-атак рекомендуется использовать только в случае применения **basic-аутентификации**.

Защиту от CSRF-атак можно **отключить** для:

- Всех сервисов приложения.

- Одного сервиса приложения.
- Нескольких методов разных сервисов.

Чтобы отключить защиту от CSRF-атак для **всех сервисов приложения**, отключите настройку

`UseCsrfToken` в файлах `Web.Config` и `...\Terrasoft.WebApp\Web.Config`.

```
<add value="false" key="UseCsrfToken" />
```

Чтобы отключить защиту от CSRF-атак для **одного сервиса приложения**, задайте имя сервиса в настройке `DisableCsrfTokenValidationForPaths` файла `Web.Config`.

```
<add key="DisableCsrfTokenValidationForPaths" value="/ServiceModel/ MsgUtilService.svc" />
```

Чтобы отключить защиту от CSRF-атак для **нескольких методов разных сервисов**, задайте методы в настройке `DisableCsrfTokenValidationForPaths` файла `Web.Config`.

```
<add key="DisableCsrfTokenValidationForPaths" value="/MsgUtilService.svc/Ping,/AuthService.svc/L
```

Реализовать аутентификацию на C#

 Средний

Пример. Реализовать аутентификацию на C#.

Реализовать аутентификацию

1. Создайте обычное консольное приложение C#, назвав его, например, `RequestAuthentication`.
2. Реализуйте аутентификацию.

Пример программной реализации аутентификации

```
// Отправляет запрос сервису аутентификации и обрабатывает ответ.
public void TryLogin() {
    var authData = @"{
        ""UserName"":"" + _userName + @""",
        ""UserPassword"":"" + _userPassword + @""
    }";
    var request = CreateRequest(_authServiceUrl, authData);
    _authCookie = new CookieContainer();
    request.CookieContainer = _authCookie;
```

```

// При успешной аутентификации сохраняем аутентификационные куки для
// дальнейшего использования в запросах к Creatio. В случае неудачной
// аутентификации в консоль приложения выводится сообщение о причине
// ошибки.
using (var response = (HttpWebResponse)request.GetResponse())
{
    if (response.StatusCode == HttpStatusCode.OK)
    {
        using (var reader = new StreamReader(response.GetResponseStream()))
        {
            var responseMessage = reader.ReadToEnd();
            Console.WriteLine(responseMessage);
            if (responseMessage.Contains("\"Code\":1"))
            {
                throw new UnauthorizedAccessException($"Unauthorized {_userName} for {_ap
            }
        }
        string authName = ".ASPXAUTH";
        string authCookieValue = response.Cookies[authName].Value;
        _authCookie.Add(new Uri(_appUrl), new Cookie(authName, authCookieValue));
    }
}
}

// Создает запрос к сервису аутентификации.
private HttpWebRequest CreateRequest(string url, string requestData = null)
{
    HttpWebRequest request = (HttpWebRequest)WebRequest.Create(url);
    request.ContentType = "application/json";
    request.Method = "POST";
    request.KeepAlive = true;
    if (!string.IsNullOrEmpty(requestData))
    {
        using (var requestStream = request.GetRequestStream())
        {
            using (var writer = new StreamWriter(requestStream))
            {
                writer.Write(requestData);
            }
        }
    }
    return request;
}

// Метод реализует защиту от CSRF-атак: копирует cookie, содержащий CSRF-токен,
// и помещает его в заголовок следующего запроса.
private void AddCsrfToken(HttpWebRequest request) {
    var cookie = request.CookieContainer.GetCookies(new Uri(_appUrl))["BPMCSRF"];
    if (cookie != null) {
        request.Headers.Add("BPMCSRF", cookie.Value);
    }
}

```

```
}
}
```

Веб-сервис AuthService.svc API

 Средний

Структура запроса

```
// Строка запроса.
POST Creatio_application_address/ServiceModel/AuthService.svc/Login

// Заголовки запроса.
Accept: application/json
ForceUseSession: true

// Тело запроса.
{
  "UserName": "Name of user",
  "UserPassword": "Password of user"
}
```

Структура ответа

```
// Код состояния.
Status: code

// Заголовки ответа.
Set-Cookie: BPMLoader=cookie_value; path=/Creatio_application_address; HttpOnly
Set-Cookie: .ASPXAUTH=cookie_value; path=/Creatio_application_address; HttpOnly
Set-Cookie: BPMCSRF=cookie_value; path=/
Set-Cookie: UserName=cookie_value; expires=date_expire_to; path=/; HttpOnly

// Тело ответа.
{
  "Code": 0,
  "Message": "",
  "Exception": null,
  "PasswordChangeUrl": null,
  "RedirectUrl": null
}
```

Строка запроса

POST **required**

Метод запроса на аутентификацию. Неизменяемая часть запроса.

Creatio_application_address **required**

Адрес приложения Creatio.

ServiceModel/AuthService.svc/Login **required**

Метод веб-сервиса аутентификации, который необходимо вызвать для выполнения аутентификации. Неизменяемая часть запроса.

Заголовки запроса

Content-Type application/json **required**

Кодировка и тип ресурса, который передается в теле запроса.

ForceUseSession true **required**

Заголовок `ForceUseSession` отвечает за принудительное использование уже существующей сессии.

Тело запроса

UserName string **required**

Имя пользователя Creatio.

UserPassword string **required**

Пароль пользователя Creatio.

Код состояния ответа

code

Код состояния ответа на запрос.

[Возможные коды состояния](#)

● 200 OK	Запрос успешно завершен и значение ресурса не равно нулю. В этом случае тело ответа должно содержать код состояния аутентификации. Если содержит значение 0, то аутентификация успешна. В случае неудачной аутентификации код состояния будет равен 1 и тело запроса будет содержать сообщение о причине неудачной аутентификации.
● 403 Forbidden	Сервер не может предоставить доступ к ресурсу, указанному в запросе (например, в случае ошибки в имени метода). Дополнительная информация может содержаться в теле ответа.

Заголовки ответа

Ответ на POST запрос содержит аутентификационные cookie, которые необходимо сохранить на стороне клиента или на клиентском компьютере, чтобы использовать их в дальнейших запросах веб-службы Creatio.

Тело ответа

Code

Код состояния аутентификации. Если содержит значение 0, то аутентификация успешна. Иначе — неуспешна.

Message

Сообщение о причине неуспешной аутентификации.

Exception

Объект, содержащий детальное описание исключения, связанного с неуспешной аутентификацией.

Fiddler



Fiddler — это приложение прокси-сервера для регистрации, проверки и изменения трафика HTTP и HTTPS между компьютером и веб-сервером или серверами. **Назначение** Fiddler — предоставлять разработчикам возможность проверки запросов и ответов, включая HTTP-заголовки и метаданные (например, файлы cookie, кэширование и кодирование информации).

Fiddler рекомендуется использовать для тестирования запросов при интеграции с Creatio по протоколу [OData 4](#) и [OData 3](#).

Этапы работы с запросами в Fiddler:

- Формирование запроса.

- Выполнение запроса.

Коллекция запросов позволяет выполнять несколько запросов один за другим. Использование коллекций значительно сокращает время тестирования запросов.

Коллекция запросов к Creatio должна включать:

1. `post` -запрос к сервису аутентификации `AuthService.svc`. Работа с сервисом аутентификации подробно описана в статье [Аутентификация](#).
2. Пользовательский запрос для работы с данными.

Выполнить запрос к Creatio в Fiddler


 Средний

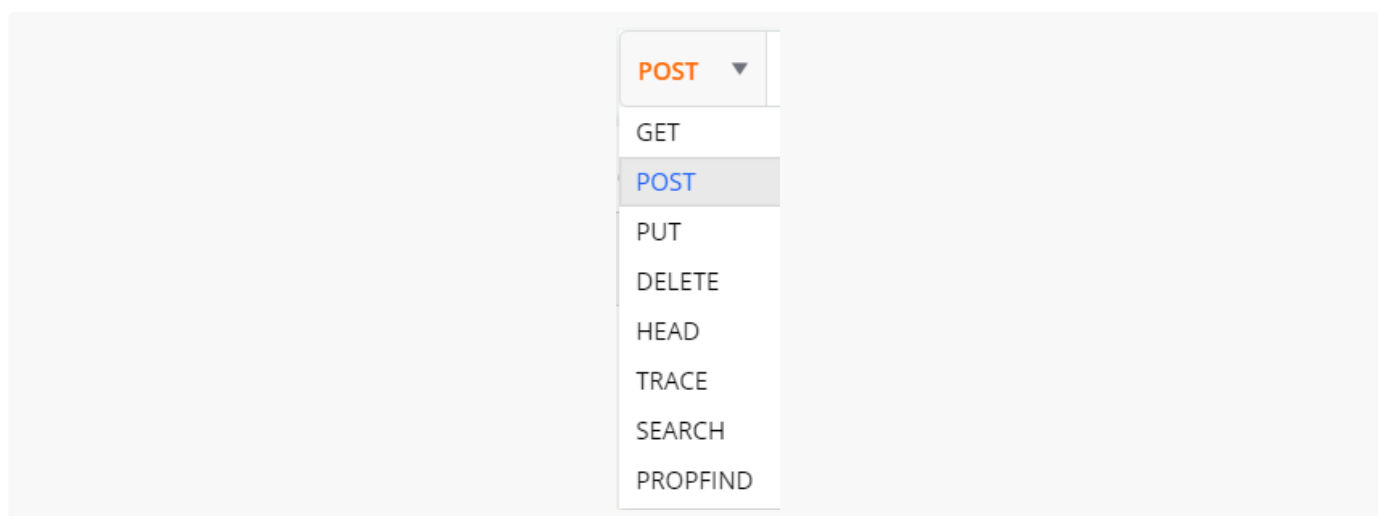
Пример. Используя сервис odata, добавить новую активность с колонками:

- [*Заголовок*] ([*Title*]) — "Process the incoming website form request".
- [*Ответственный*] ([*Owner*]) — [`Id`] текущего пользователя.
- [*Заметки*] ([*Notes*])—
"Please, email to client@gmail.com and process the following request: clients request".

1. Сформировать запрос

Для формирования **запроса на добавление данных** в Fiddler:

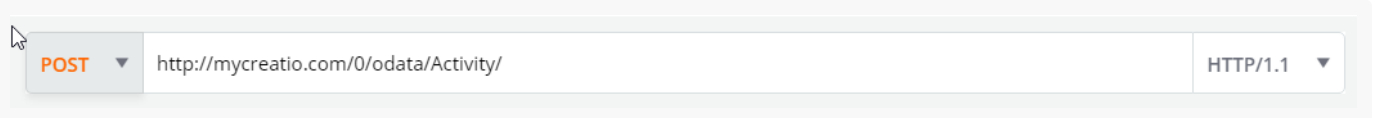
1. Создайте новый запрос, нажав кнопку  [*New Request*].
2. В выпадающем списке панели инструментов рабочей области Fiddler выберите метод запроса `POST`.



3. В поле запроса панели инструментов рабочей области Fiddler введите строку запроса:

Пример строки запроса

```
http(s)://mycreatio.com/0/odata/Activity/
```



4. Выберите версию 1.1 протокола HTTP.

5. Перейдите на вкладку [*Headers*] и заполните заголовки запроса:

- **Content-Type** : application/json .
- **Accept** : application/json .
- **Cookie** : BPMLOADER , .ASPXAUTH , BPMSESSIONID и BPMCSRF — cookie получены из запроса к сервису аутентификации.
- **BPMCSRF** : "CSRF-токен BPMCSRF" — содержит значение одноименной cookie.

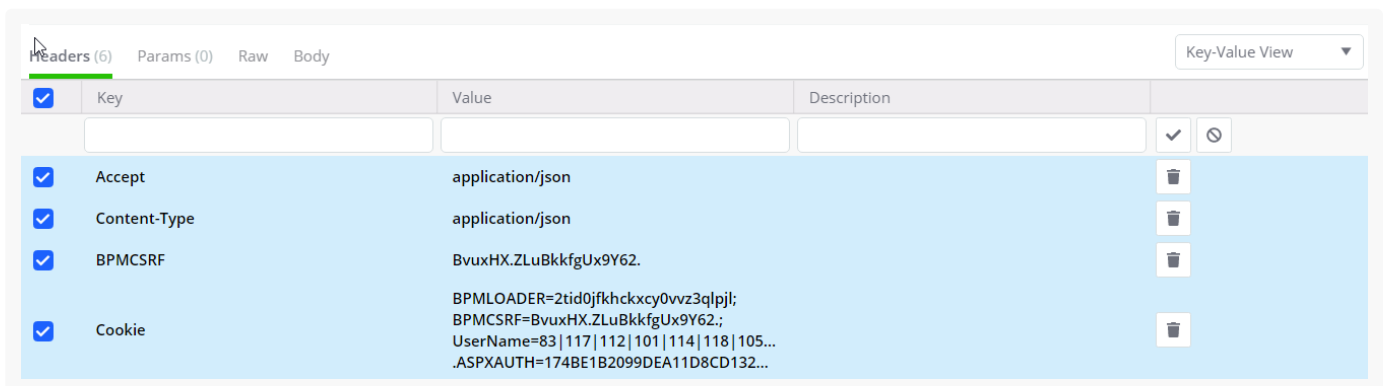
Пример HTTP-заголовка запроса

Accept:application/json

Content-Type:application/json

Cookie: BPMSESSIONID=cxa54p2dsb4wnqbbzvgyxcoo; BPMCSRF=6yCmyILS1IE8/toyQm9Ca.; BPMLOADER=rqqj

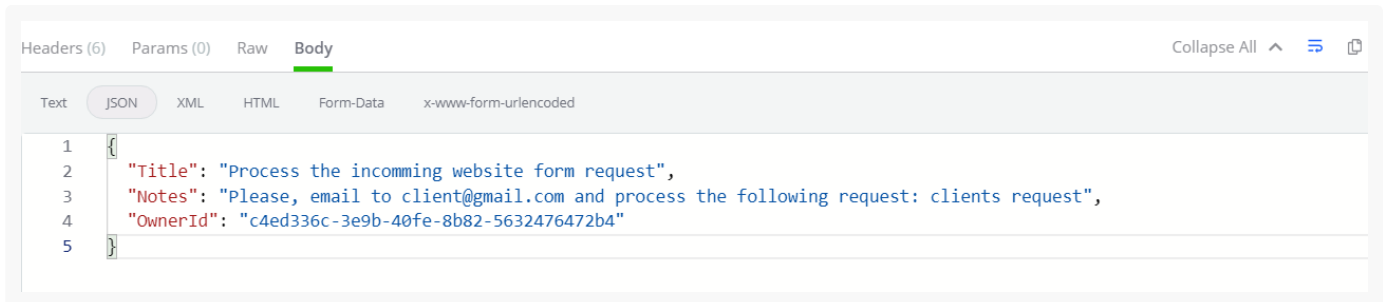
BPMCSRF: 6yCmyILS1IE8/toyQm9Ca.



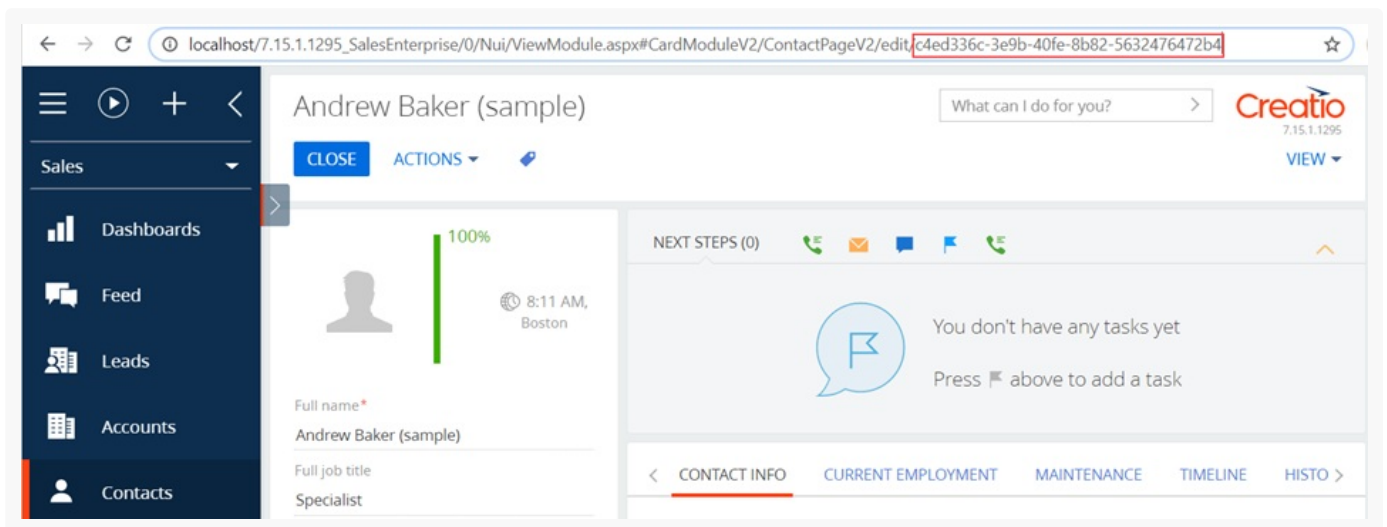
6. Перейдите на вкладку [*Body*] и заполните тело запроса.

Пример тела запроса

```
{
  "Title": "Process the incoming website form request",
  "Notes": "Please, email to client@gmail.com and process the following request: clients requ",
  "OwnerId": "c4ed336c-3e9b-40fe-8b82-5632476472b4"
}
```

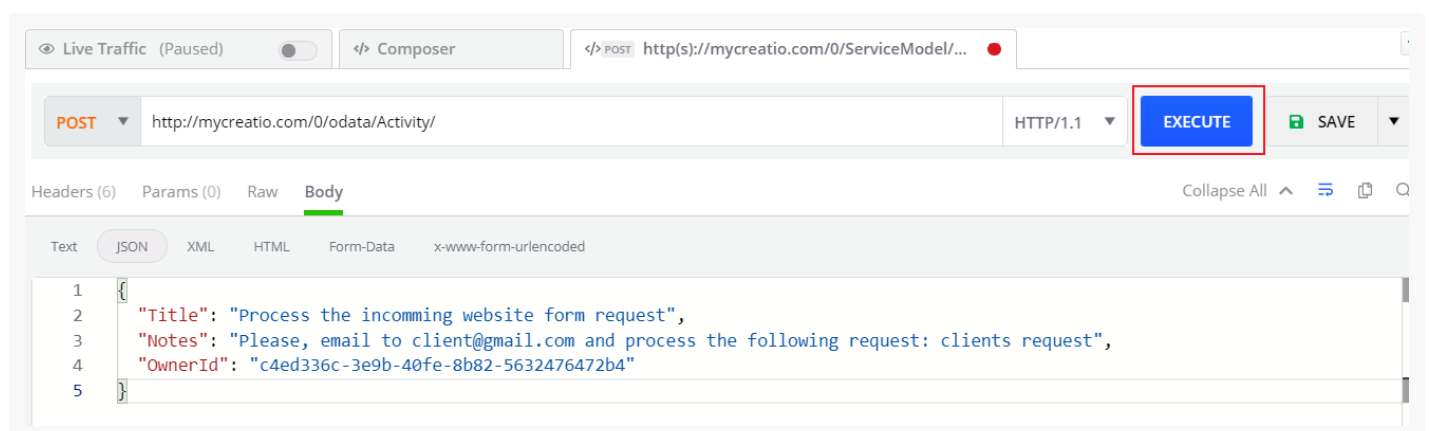


Для справочной колонки в запросе необходимо к названию колонки добавить суффикс Id. В качестве значения указать идентификатор из таблицы базы данных. В приведенном примере используется справочная колонка [Ответственный] ([Owner]), для которой в запросе указывается идентификатор `OwnerId`. Значение идентификатора ответственного можно посмотреть в браузере, открыв соответствующую страницу контакта.



2. Выполнить запрос

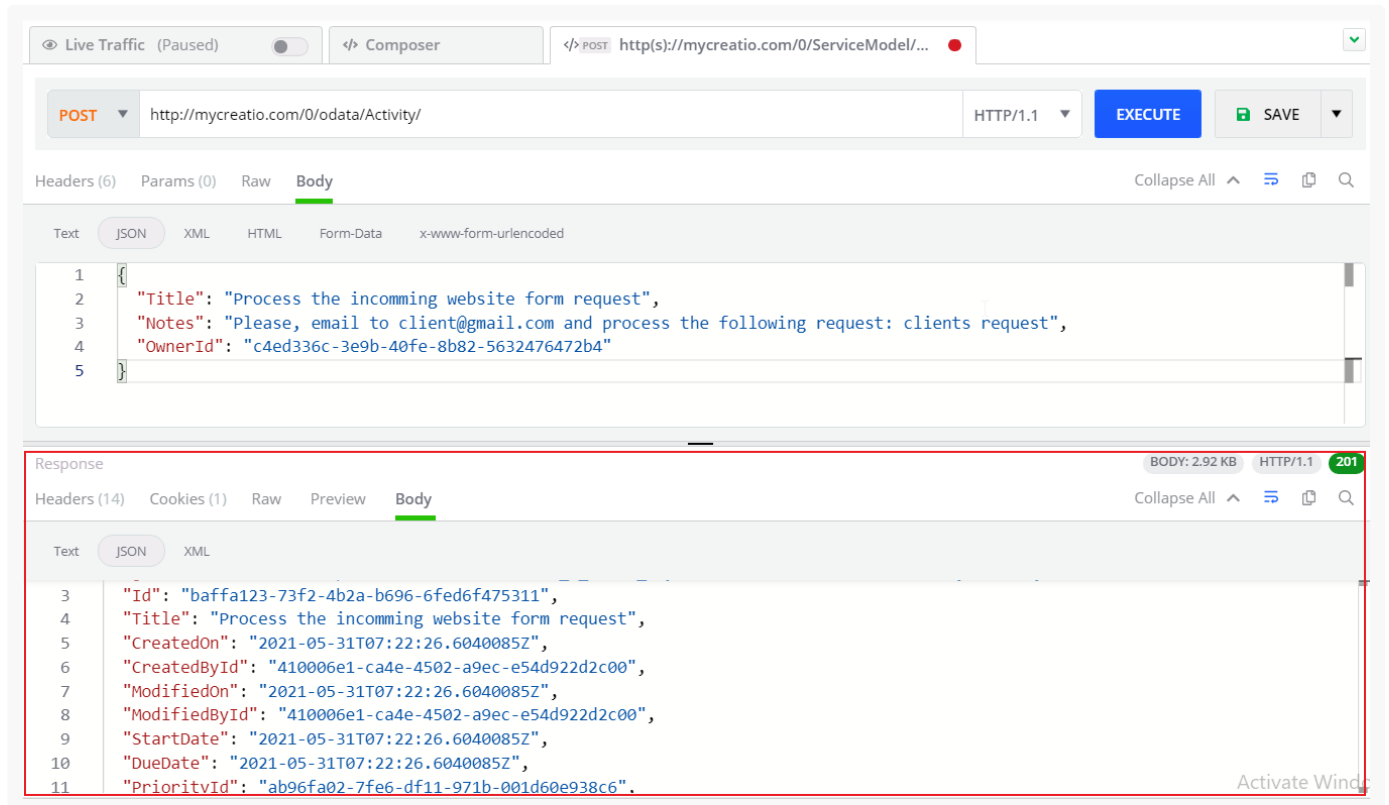
Чтобы выполнить запрос в Fiddler, на панели инструментов рабочей области нажмите кнопку [Execute].



Результат выполнения

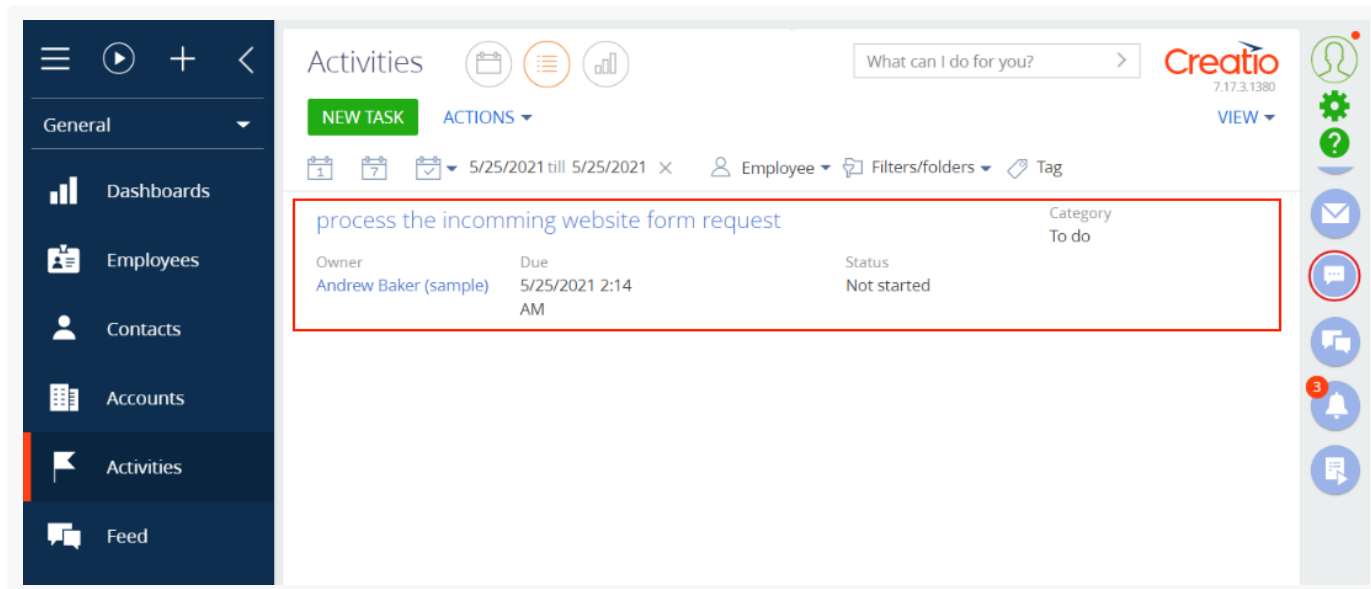
В результате выполнения запроса:

- В блоке [*Response*] отобразится код состояния ответа. На вкладке [*Body*] отобразится ответ от сервиса `odata`.



В теле ответа содержится добавленная запись в формате JSON. Элемент `"Id"` содержит идентификатор добавленной активности, который можно использовать при составлении других запросов, например, на редактирование.

- В раздел [*Активности*] ([*Activities*]) будет добавлена новая запись.



Сервис запуска бизнес-процессов

 **Сложный**

Одной из задач интеграции внешнего приложения с Creatio является запуск бизнес-процессов. С этой целью в сервисной модели Creatio реализован веб-сервис `ProcessEngineService.svc`. **Назначение** `ProcessEngineService.svc` — запуск бизнес-процессов.

Запуск бизнес-процессов из внешнего приложения

Адрес сервиса `ProcessEngineService.svc`

`https://mycreatio.com/0/ServiceModel/ProcessEngineService.svc`

Основные **методы** сервиса `ProcessEngineService.svc`:

- `Execute()` — запускает бизнес-процесс. Позволяет передать набор входящих параметров бизнес-процесса и вернуть результат его выполнения.
- `ExecProcElByUid()` — запускает отдельный элемент бизнес-процесса. Запускать на выполнение можно только элемент выполняющегося процесса.

На заметку. Полный перечень методов веб-сервиса доступен в [Библиотеке .NET классов](#).

Запуск бизнес-процесса из front-end части

Для запуска бизнес-процесса из клиентской схемы используется модуль `ProcessModuleUtilities` пакета

`NUI`. Этот модуль предоставляет удобный интерфейс для выполнения запросов к сервису `ProcessEngineService.svc`.

Чтобы **запустить бизнес-процесс из клиентской схемы**:

1. Подключите в качестве зависимости модуль `ProcessModuleUtilities` в модуль страницы, из которой вызывается сервис.
2. Вызовите метод `executeProcess(args)` модуля `ProcessModuleUtilities`, передав ему в качестве параметра объект `args` с такими свойствами:

Свойства объекта `args`

Свойство	Описание
<code>sysProcessName</code>	Имя вызываемого процесса (необязательное свойство в случае, если определено свойство <code>sysProcessId</code>).
<code>sysProcessId</code>	Уникальный идентификатор вызываемого процесса (необязательное свойство, если определено свойство <code>sysProcessName</code>).
<code>parameters</code>	Объект, свойствами которого являются входящие параметры вызываемого процесса.

Запустить бизнес-процесс через веб-сервис



Пример. Используя веб-сервис `ProcessEngineService.svc`, из строки навигации браузера или из консольного приложения запустить демонстрационные бизнес-процессы создания и считывания контактов Creatio, .

1. Создать процесс добавления контакта

На заметку. Особенности и лучшие практики создания бизнес-процессов в Creatio подробно описаны в [документации по настройке процессов](#).

1. Создайте пользовательский пакет и установите его в качестве текущего.
2. Перейдите в [дизайнер процессов](#).
3. Заполните значения **свойств бизнес-процесса**:
 - [*Название*] ([*Name*]) — "Add New External Contact".

- [Код] ([Code]) — "UsrAddNewExternalContact".

Для остальных свойств оставьте значения по умолчанию.

4. Добавьте параметры бизнес-процесса.

С помощью параметров в процесс передаются реквизиты добавляемого контакта — имя и телефон.

Значения свойств параметра `ContactName` :

- [Заголовок] ([Title]) — "Имя контакта" ("Contact Name").
- [Код] ([Code]) — "ContactName".
- [Тип данных] ([Data type]) — "Текст (50 символов)" ("Text (50 characters)").

Значения свойств параметра `ContactPhone` :

- [Заголовок] ([Title]) — "Телефон контакта" ("Contact Phone").
- [Код] ([Code]) — "ContactPhone".
- [Тип данных] ([Data type]) — "Текст (50 символов)" ("Text (50 characters)").

SETTINGS PARAMETERS METHODS

ADD PARAMETER ▾

Title *

Contact Name

Code *

ContactName

Data type *

Text (50 characters)

Value

Select value

SAVE CANCEL

T Contact Phone

Select value

5. Добавьте элемент [Задание-сценарий] ([*ScriptTask*]).

Значения **свойств элемента**:

- [*Название*] ([*Name*]) — "Add contact".
- [*Код*] ([*Code*]) — "ScriptTaskAddContact".

6. Реализуйте логику добавления нового контакта.

Чтобы редактировать код сценария, дважды щелкните по элементу на диаграмме. На панели настройки элемента откроется окно для ввода и редактирования программного кода.

ScriptTaskAddContact

```
/* Создание экземпляра схемы объекта [Контакт]. */
var schema = UserConnection.EntitySchemaManager.GetInstanceByName("Contact");
/* Создание экземпляра нового объекта. */
var entity = schema.CreateEntity(UserConnection);
/* Установка значений по умолчанию для колонок объекта. */
entity.SetDefColumnValues();
string contactName = Get<string>("ContactName");
string contactPhone = Get<string>("ContactPhone");
/* Установка значения колонки [Name] из параметра процесса. */
entity.SetColumnValue("Name", contactName);
/* Установка значения колонки [Phone] из параметра процесса. */
entity.SetColumnValue("Phone", contactPhone);
/* Сохранение нового контакта. */
entity.Save();
return true;
```

7. После внесения изменений сохраните бизнес-процесс, нажав на кнопку [Сохранить] ([Save]) на панели инструментов дизайнера процессов.

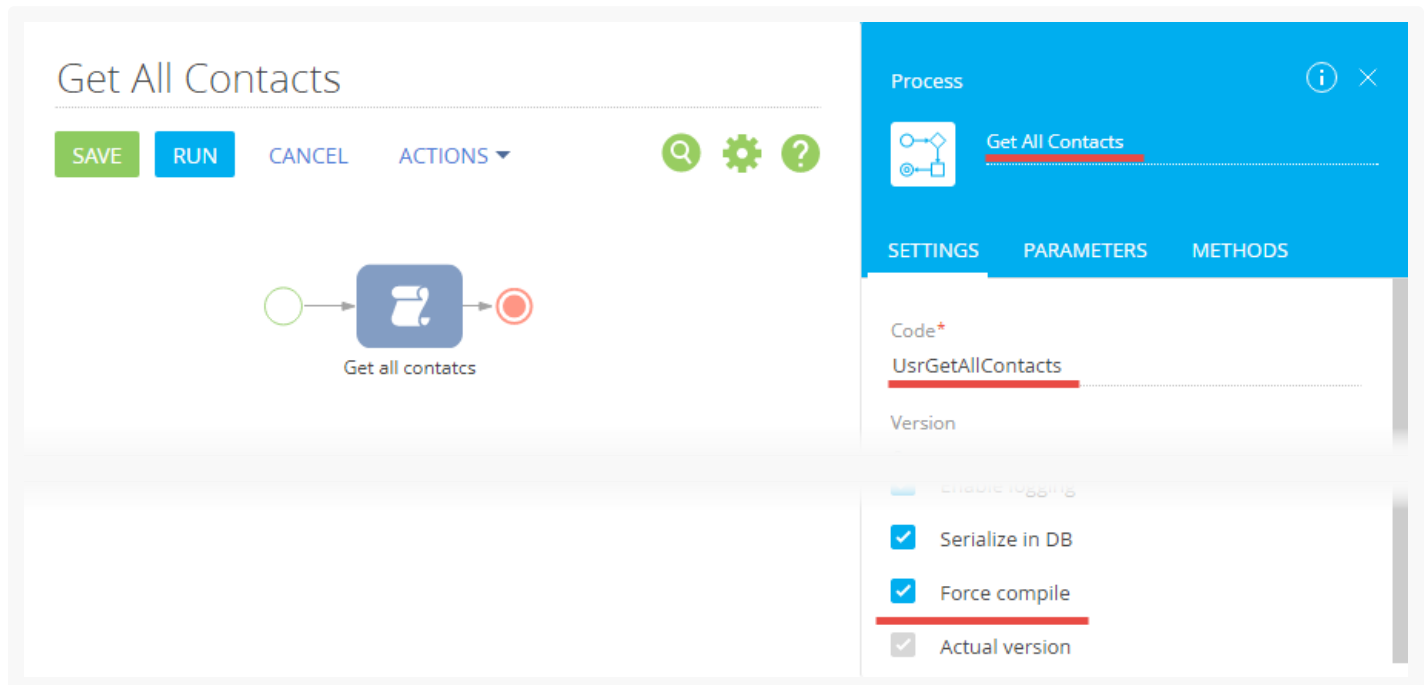
2. Создать процесс чтения контактов

Бизнес-процесс, формирующий список всех контактов, также содержит один элемент [Задание-сценарий] ([ScriptTask]), в котором реализуется необходимая логика.

Значения **свойств бизнес-процесса**:

- [Название] ([Name]) — "Get All Contacts".
- [Код] ([Code]) — "UsrGetAllContacts".

Для остальных свойств оставьте значения по умолчанию.



Процесс `UsrGetAllContacts` содержит единственный параметр `ContactList`, через который процесс будет возвращать список всех контактов системы в виде JSON-объекта. Тип параметра — **строка неограниченной длины**.

SETTINGS
PARAMETERS
METHODS

ADD PARAMETER ▼

Title *

Contact List

Code *

ContactList

Data type *

Unlimited length text

Value

Select value

SAVE CANCEL

Логику выборки контактов реализуйте в элементе процесса [*Задание-сценарий*] ([*ScriptTask*]).

Значения **свойств элемента**:

- [*Название*] ([*Name*]) — "Get all contacts".
- [*Код*] ([*Code*]) — "ScriptTaskGetAllContacts".

ScriptTaskGetAllContacts

```

/* Создание экземпляра EntitySchemaQuery. */
EntitySchemaQuery query = new EntitySchemaQuery(UserConnection.EntitySchemaManager, "Contact");
/* Признак для обязательного выбора первичной колонки [Id]. */
query.PrimaryQueryColumn.IsAlwaysSelect = true;
/* Добавление в запрос колонок. */
query.AddColumn("Name");
query.AddColumn("Phone");
/* Получение результирующей коллекции. */
var entities = query.GetEntityCollection(UserConnection);
/* Формирование списка контактов для сериализации в JSON. */
List<object> contacts = new List<object>();
foreach (var item in entities)
{
    var contact = new
    {
        Id = item.GetTypedColumnValue<Guid>("Id"),
        Name = item.GetTypedColumnValue<string>("Name"),
        Phone = item.GetTypedColumnValue<string>("Phone")
    };
    contacts.Add(contact);
}

```

```
/* Сохранение сериализованной в JSON коллекции контактов в параметр ContactList. */
string contactList = JsonConvert.SerializeObject(contacts);
Set<string>("ContactList", contactList);
return true;
```

После внесения изменений сохраните и опубликуйте бизнес-процесс.

3. Запустить выполнение бизнес-процессов из строки навигации браузера

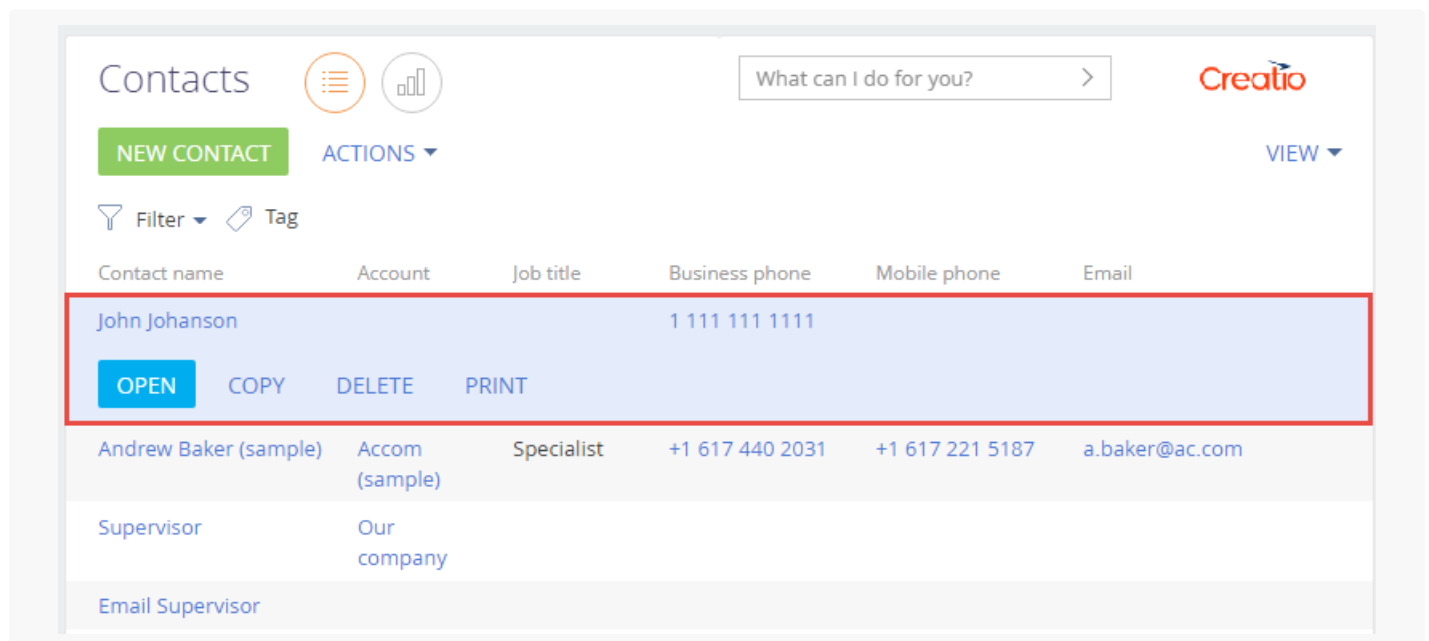
Поскольку вызов метода сервиса возможен с помощью `GET`-запроса, то для выполнения запуска бизнес-процесса можно использовать браузер.

Для запуска процесса создания нового контакта в строку навигации браузера введите URL:

URL для запуска процесса создания нового контакта

`http[s]://[Адрес приложения Creatio]/0/ServiceModel/ProcessEngineService.svc/UsrAddNewExternalCc`

После перехода по указанному URL, в приложении будет добавлен новый контакт.



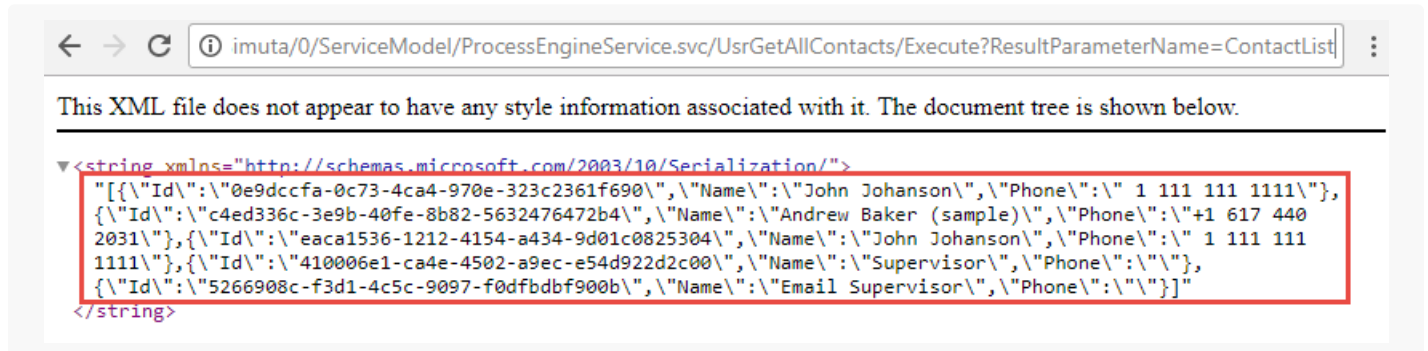
Важно. Новый контакт будет создан при каждом успешном запросе к сервису. Если выполнить несколько запросов с одинаковыми параметрами, будет создано несколько контактов-дублей.

Для запуска процесса чтения всех контактов в строку навигации браузера введите URL:

URL для запуска процесса чтения всех контактов

```
http[s]://[Адрес приложения Creatio]/0/ServiceModel/ProcessEngineService.svc/UsrGetAllContacts/E
```

После выполнения перехода по указанному URL, в окне браузера будет отображен JSON-объект, содержащий коллекцию контактов.



4. Запустить выполнение бизнес-процессов из консольного приложения

Полностью исходный код консольного пользовательского приложения, предназначенного для запуска бизнес-процессов с помощью сервиса `ProcessEngineService.svc`, доступен на [GitHub](#).

Чтобы **запустить выполнение** бизнес-процессов из консольного приложения:

1. Выполните **аутентификацию**. Для этого предназначен сервис аутентификации `AuthService.svc`. Консольное приложение для [выполнения аутентификации](#) можно взять за основу для примера, приведенного ниже.
2. Для формирования запросов к сервису `ProcessEngineService.svc` в исходный код класса `Program` добавьте строковое поле, содержащее базовый URL сервиса.

URL для формирования запросов к сервису `ProcessEngineService.svc`

```
private const string processServiceUri = baseUri + @"/0/ServiceModel/ProcessEngineService.svc
```

3. Для выполнения запуска бизнес-процесса добавления контакта в исходный код класса `Program` добавьте метод `GET`.

Метод `GET` для выполнения запуска бизнес-процесса

```
public static void AddContact(string contactName, string contactPhone)
{
    /* Формирование URL запроса. */
    string requestString = string.Format(processServiceUri +
        "UsrAddNewExternalContact/Execute?ContactName={0}&ContactPhone={1}",
```

```

        contactName, contactPhone);

    /* Создание Http-запроса. */
    HttpWebRequest request = HttpWebRequest.Create(requestString) as HttpWebRequest;
    request.Method = "GET";
    request.CookieContainer = AuthCookie;
    /* Выполнение запроса и анализ Http-ответа. */
    using (var response = request.GetResponse())
    {
        /* Поскольку сервис вернет пустую строку, то можно вывести свойства http-ответа. */
        Console.WriteLine(response.ContentLength);
        Console.WriteLine(response.Headers.Count);
    }
}

```

4. Добавьте метод запуска процесса чтения контактов.

Метод GET для выполнения запуска бизнес-процесса

```

public static void GetAllContacts()
{
    /* Формирование URL запроса. */
    string requestString = processServiceUri +
        "UsrGetAllContacts/Execute?ResultParameterName=ContactList";
    HttpWebRequest request = HttpWebRequest.Create(requestString) as HttpWebRequest;
    request.Method = "GET";
    request.CookieContainer = AuthCookie;
    /* Создание Http-запроса. */
    using (var response = request.GetResponse())
    {
        /* Выполнение запроса и вывод результата. */
        using (var reader = new StreamReader(response.GetResponseStream()))
        {
            string responseText = reader.ReadToEnd();
            Console.WriteLine(responseText);
        }
    }
}

```

5. Вызов добавленных методов выполните в главном методе программы после успешной аутентификации.

Вызов методов

```

static void Main(string[] args)
{
    if (!TryLogin("Supervisor", "Supervisor"))
    {

```

```

        Console.WriteLine("Wrong login or password. Application will be terminated.");
    }
    else
    {
        try
        {
            /* Вызов методов запуска бизнес-процессов. */
            AddContact("John Johanson", "+1 111 111 1111");
            GetAllContacts();
        }
        catch (Exception)
        {
            /* Обработка исключений. */
            throw;
        }
    }
};
Console.WriteLine("Press ENTER to exit...");
Console.ReadLine();
}

```

Результат выполнения пользовательского приложения

```

0
8
<string xmlns="http://schemas.microsoft.com/2003/10/Serialization/">[{\"Id\": \"0e9dccfa-0c73-4ca4-970e-323c2361f690\", \
\"Name\": \"John Johanson\", \"Phone\": \" 1 111 111 1111\"}, {\"Id\": \"a0c4b7fe-8060-4611-8c91-35e339f524f1\", \"Name\": \"Joh
n Johanson\", \"Phone\": \" 1 111 111 1111\"}, {\"Id\": \"c4ed336c-3e9b-40fe-8b82-5632476472b4\", \"Name\": \"Andrew Baker (sa
mple)\", \"Phone\": \"+1 617 440 2031\"}, {\"Id\": \"d6799742-ae56-4149-8f55-cbddc7387d0b\", \"Name\": \"John Johanson\", \"Pho
ne\": \" 1 111 111 1111\"}, {\"Id\": \"410006e1-ca4e-4502-a9ec-e54d922d2c00\", \"Name\": \"Supervisor\", \"Phone\": \"\", {\"Id
\": \"5266908c-f3d1-4c5c-9097-f0dfbdbf900b\", \"Name\": \"Email Supervisor\", \"Phone\": \"\"}]}</string>
Press ENTER to exit...
-

```

Запустить процесс из клиентского модуля



Средний

Чтобы запустить бизнес-процесс из JavaScript-кода клиентской схемы, необходимо:

1. В модуль страницы, из которой вызывается сервис, подключить в качестве зависимости модуль `ProcessModuleUtilities`. Этот модуль предоставляет удобный интерфейс для выполнения запросов к сервису `ProcessEngineService.svc`.
2. Вызвать метод `executeProcess(args)` модуля `ProcessModuleUtilities`, передав ему в качестве параметра объект `args` с такими свойствами:

Свойства объекта args

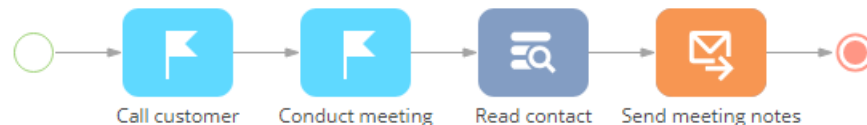
Свойство	Описание
sysProcessName	Имя вызываемого процесса (необязательное свойство в случае, если определено свойство sysProcessId).
sysProcessId	Уникальный идентификатор вызываемого процесса (необязательное свойство в случае, если определено свойство sysProcessName).
parameters	Объект, свойствами которого являются входящие параметры вызываемого процесса.

Пример. На страницу контрагента добавить действие, по которому будет запускаться бизнес-процесс "Проведение встречи". В бизнес-процесс передать в качестве параметра основной контакт контрагента.

1. Создать и настроить пользовательский бизнес-процесс "Проведение встречи"

1. Создайте бизнес-процесс.

В примере используется бизнес-процесс "Проведение встречи", создание которого детально описано в разделе "[Как работать с email](#)" документации по настройке процессов .



1. Добавьте в бизнес-процесс параметр.

Заполните **свойства параметра**:

- [Заголовок] ([Title]) — "Meeting contact".
- [Код] ([Code]) — "ProcessSchemaContactParameter".
- [Тип данных] ([Data type]) — "Уникальный идентификатор" ("Unique identifier").

The screenshot shows a 'Process' configuration window titled 'Holding a meeting'. It has three tabs: 'SETTINGS', 'PARAMETERS', and 'METHODS'. The 'PARAMETERS' tab is active. Below the tabs is a blue button labeled 'ADD PARAMETER'. A form is displayed with the following fields:

- Title***: Meeting contact
- Code***: ProcessSchemaContactParameter
- Data type***: Unique identifier
- Value**: Select value

At the bottom right of the form are two buttons: 'SAVE' (blue) and 'CANCEL' (grey).

1. В свойствах первого действия процесса [Позвонить клиенту] ([Call customer]) заполните поле [Контакт] ([Contact]) входящим параметром процесса.

The screenshot shows a 'Perform task' configuration window titled 'Call customer'. It has a blue header bar with a flag icon and the title. Below the header, there is a checkbox labeled 'Show page automatically' which is checked. The section 'Who performs the task?' contains a text field with the value '[#System variable.Current user contact#]'. Below this is a 'Hint for user' field. The 'Remind in' section shows '10' in a text field and 'minutes' as a label. At the bottom, there is a 'Connected to' section with a plus icon. A red box highlights the 'Contact' field, which contains the value '[#Meeting contact#]'.

2. Создать замещающую страницу контрагента и добавить на нее действие

Добавление действия на страницу записи подробно описано в статье [Добавить действие на страницу записи](#).

В схему замещающего модуля страницы записи и схему раздела контрагента добавьте локализуемую строку `CallProcessCaption` с заголовком действия, например, "Назначить встречу" ("Schedule a meeting").

Дополнительно в объявлении модуля страницы записи подключите в качестве зависимости модуль `ProcessModuleUtilities`.

Исходные коды схемы раздела и страницы записи раздела [*Контрагенты*] ([*Accounts*]) приведены ниже.

3. Добавить необходимые методы в схемы

Для запуска процесса воспользуйтесь методом `executeProcess()` модуля `ProcessModuleUtilities`, в который в качестве параметра необходимо передать объект со следующими свойствами:

- имя созданного бизнес-процесса,
- объект с проинициализированными входящими параметрами для процесса.

В исходном коде, приведенном ниже, это реализовано в методе `callCustomProcess()`. Также реализованы методы проверки существования основного контакта `isAccountPrimaryContactSet()` и добавления элементов меню действий `getActions()`.

Исходный код замещающего модуля страницы записи:

```
define("AccountPageV2", ["ProcessModuleUtilities"], function(ProcessModuleUtilities) {
    return {
        // Название схемы объекта страницы записи.
        entitySchemaName: "Account",
        // Методы модели представления страницы записи.
        methods: {
            // Проверяет, заполнено ли поле [Основной контакт] страницы.
            isAccountPrimaryContactSet: function() {
                return this.get("PrimaryContact") ? true : false;
            },
            // Переопределение базового виртуального метода, возвращающего коллекцию действий ст
            getActions: function() {
                // Вызывается родительская реализация метода для получения
                // коллекции проинициализированных действий базовой страницы.
                var actionMenuItems = this.callParent(arguments);
                // Добавление линии-разделителя.
                actionMenuItems.addItem(this.getActionsMenuItem({
                    Type: "Terrasoft.MenuSeparator",
                    Caption: ""
                }));
                // Добавление пункта меню [Назначить встречу] в список действий страницы записи.
```

```

        actionMenuItems.addItem(this.getActionsMenuItem({
            // Привязка заголовка пункта меню к локализуемой строке схемы.
            "Caption": { bindTo: "Resources.Strings.CallProcessCaption" },
            // Привязка метода-обработчика действия.
            "Tag": "callCustomProcess",
            // Привязка свойства видимости пункта меню к значению, которое возвращает ме
            "Visible": { bindTo: "isAccountPrimaryContactSet" }
        }));
        return actionMenuItems;
    },
    // Метод-обработчик действия.
    callCustomProcess: function() {
        // Получение идентификатора основного контакта контрагента.
        var contactParameter = this.get("PrimaryContact");
        // Объект, который будет передан в качестве аргумента в метод executeProcess().
        var args = {
            // Имя процесса, который необходимо запустить.
            sysProcessName: "UsrCustomProcess",
            // Объект со значением входящего параметра ContactParameter для процесса Cus
            parameters: {
                ProcessSchemaContactParameter: contactParameter.value
            }
        };
        // Запуск пользовательского бизнес-процесса.
        ProcessModuleUtilities.executeProcess(args);
    }
};
});

```

Для корректного отображения действия в меню действий в совмещенном режиме отображения страницы с вертикальным реестром добавьте реализацию метода `isAccountPrimaryContactSet()` в схему раздела .

Исходный код замещающего модуля схемы раздела:

```

define("AccountSectionV2", [], function() {
    return {
        // Название схемы раздела.
        entitySchemaName: "Account",
        methods: {
            // Проверяет, заполнено ли поле [Основной контакт] выбранной записи.
            isAccountPrimaryContactSet: function() {
                // Определение активной записи.
                var activeRowId = this.get("ActiveRow");
                if (!activeRowId) {
                    return false;
                }
            }
        }
    };
});

```

```

    }
    // Получение коллекции данных списочного представления реестра раздела.
    // Получение модели выбранного контрагента по заданному значению первичной колонки
    var selectedAccount = this.get("GridData").get(activeRowId);
    if (selectedAccount) {
        // Получение свойства модели – наличие основного контакта.
        var selectedPrimaryContact = selectedAccount.get("PrimaryContact");
        // Метод возвращает true, если основной контакт установлен. Иначе возвращает
        return selectedPrimaryContact ? true : false;
    }
    return false;
}
}
};
});

```

Результат выполнения примера

После сохранения схем и обновления страницы приложения в меню действий страницы контрагента появится новое действие [Назначить встречу] ([Schedule a meeting]). Это действие будет доступным только в случае наличия основного контакта для активной записи реестра. При выполнении действия будет запущен пользовательский бизнес-процесс "Проведение встречи" ("Holding a meeting"). При этом в параметр бизнес-процесса будет передан основной контакт контрагента.

Вызов бизнес-процесса по действию на странице записи

The screenshot displays the 'Feature IT' contact record in a CRM application. The 'ACTIONS' menu is open, showing several options: 'Set up access rights', 'Follow the feed', 'Update with social networks data', and 'Schedule a meeting'. The 'Schedule a meeting' option is highlighted with a red box. The contact details for Caleb Jones are visible, including his name, web address (www.feature-it.com), primary phone number (+1 212 735 2537), and category (IT companies). A 'Primary contact' pop-up shows Tony Campbell, with his full job title 'Development Department Man...'. The contact also has an alternate phone number (+1 212 735 2538) and an address in New York. The 'Banking details' and 'Noteworthy events' sections are currently empty.

Результат запуска бизнес-процесса. Передача параметра со страницы контрагента в бизнес-процесс

"Call customer, offer presentation"

What can I do for you?

SAVE CANCEL ACTIONS

VIEW

Due* 6/18/2018 3:31 AM Reporter* Supervisor

Status* Not started Priority* Medium

Category* Call

Call direction Incoming

Show in calendar ☒

GENERAL INFORMATION PARTICIPANTS ATTACHMENTS AND NOTES EMAIL CALLS FEED

Result

Result details

Connected to

Account Contract Invoice Opportunity Contact Tony Campbell Document Lead Order

"Call customer, offer presentation" Holding a meeting localhost

Successfully started Holding a meeting localhost

Task reminder Tony Campbell: "Call customer, offer presentation" localhost

Веб-сервис ProcessEngineService.svc API

Сложный

Структура запроса

```
/* Строка запроса. */
GET Creatio_application_address/0/ServiceModel/ProcessEngineService.svc/[processSchemaName/]metr

/* Заголовки запроса. */
ForceUseSession: true
BPMCSRF: authentication_cookie_value
```

Структура ответа для метода Execute()

```
/* Код состояния. */
Status: code

/* Тело ответа. */

"["
```

```

{"Id\\":\\"object1_id\\",\\"object1 field1\\":\\"object1 field_value1\\",\\"object1 field2\\":\\"
{"Id\\":\\"object2_id\\",\\"object2 field1\\":\\"object2 field_value1\\",\\"object2 field2\\":\\"
...
{}},
]"

```

Строка запроса

GET **required**

Метод запроса на запуск бизнес-процессов.

Creatio_application_address **required**

Адрес приложения Creatio.

ServiceModel **required**

Путь к сервису запуска бизнес-процессов. Неизменяемая часть запроса.

ProcessEngineService.svc **required**

Адрес веб-сервиса запуска бизнес-процессов. Неизменяемая часть запроса.

methodName **required**

Метод веб-сервиса запуска бизнес-процессов.

Основные **методы**:

- `Execute()` — запуск бизнес-процесса. Позволяет передавать набор входящих параметров и возвращать результат выполнения веб-сервиса.
- `ExecProcElById()` — запуск отдельного элемента бизнес-процесса. Запускать на выполнение можно только элемент выполняющегося процесса. Если элемент процесса, запускаемый методом `ExecProcElById()`, уже выполнен на момент вызова метода, то повторно такой элемент выполняться не будет.

ProcessSchemaName **optional**

Используется для метода `Execute()`.

Название схемы бизнес-процесса. Название схемы бизнес-процесса можно узнать в разделе [*Конфигурация*] ([*Configuration*]).

ResultParameterName **optional**

Используется для метода `Execute()`.

Переменная для кода параметра процесса. Неизменяемая часть запроса.

resultParameterName **optional**

Используется для метода `Execute()`.

Код параметра процесса, который хранит результат выполнения процесса. Если этот параметр не задан, то веб-сервис запустит указанный бизнес-процесс без ожидания результата его выполнения. Если в вызываемом процессе отсутствует данный код параметра, веб-сервис вернет значение `null`.

inputParameter **optional**

Используется для метода `Execute()`.

Переменная для кода входящих параметров бизнес-процесса. Если передается несколько входящих параметров, то они должны быть объединены символом `&`.

inputParameterValue **optional**

Используется для метода `Execute()`.

Значения входящих параметров бизнес-процесса.

ProcessElementUID **optional**

Используется для метода `ExecProcElByUID()`.

Идентификатор запускаемого элемента процесса.

Заголовки запроса

ForceUseSession true **required**

Заголовок `ForceUseSession` отвечает за принудительное использование уже существующей сессии. Отсутствует необходимость использования в запросе к сервису аутентификации [AuthService.svc](#).

BPMCSRF authentication_cookie_value **required**

Аутентификационный cookie.

Тело ответа

[]

Коллекция объектов.

{}

Экземпляры объектов коллекции.

Id

Название поля [*Id*].

object1_id, object2_id, ...

Идентификатор экземпляра объекта коллекции.

object1 field1, object1 field2, ..., object2 field1, object2 field2, ...

Имена полей field1, field2, ... экземпляров объектов object1, object2, ... коллекции.

object1 field_value1, object1 field_value2, ..., object2 field_value1, object2 field_value2, ...

Значения полей field1, field2, ... экземпляров объектов object1, object2, ... коллекции. Может присутствовать только для GET и POST запросов.