Содержание

Создание нового запроса/Редактирование существующего

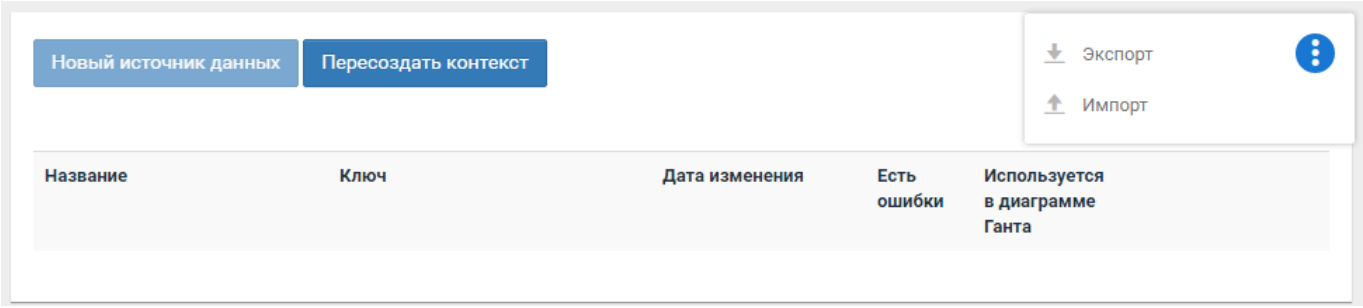
Пересоздание контекста

Первый шаг – **пересоздать контекст**.

Операция должна выполняется для обновления внутреннего словаря объектов и их свойств, для использования в запросах изменений в настройках объектной модели системы. В противном случае новые объекты, реквизиты и изменение ранее созданных не будут доступны для запросов.

Если контекст еще никогда не создавался, то необходимо его пересоздать обязательно, иначе во внутреннем словаре данные об объектах системы будут полностью отсутствовать.
При пересоздании контекста, страница списка запросов закрашивается цветом, означающим недоступность действий и отображаются «шары ожидания».
По окончании перестроения, снимается блокировка недоступности окна, а справа вверху отображается дата и время последнего перестроения, равная текущему времени завершения операции.

Далее вы можете создать новые запросы и/или импортировать уже существующие.



Создание нового запроса

Кнопка «Новый источник данных» и ссылка «Изменить» для имеющихся запросов, открывают одно и то же окно редактирования запроса.

При создании нового запроса по умолчанию подставляется шаблонный запрос. Он полезен для тех, кто редко пишет запросы, в нем определена базовая структура запроса, которую необходимо поменять для конкретного случая.

```
//A query sample
    var projects = dataContext.Projects
          .Where(p => p.Name != null)
          .OrderBy(p => p.CreationDate)
          .Select(p => new { p.Id, p.Name});
    return projects;
```

Каждый запрос имеет свой UID, видимый в окне редактирования. Данный UID используется в методе API и как источник в отчетах "Электронная таблица".

Текст запроса составляется по правилам языка C#. Ввозвращаемое значение должно быть последовательностью объектов типа IQueryable, либо приводиться к нему.

Кнопка «Сделать запрос» выполняет обращение в базу и формирует запрос к СУБД, после чего отображать результат запроса в виде таблицы, представленной страницами данных, а также общее число возвращаемых строк. Справа над таблицей отображается время выполнения запроса.

В каждой колонке отображается значение из запроса.

- Кнопка «Сохранить» – сохраняет запрос, или изменение запроса в системе.
- Чекбокс «Отобразить SQL-запрос» отображает поле с текстами запросов, которые были сгенерированы для получения выборки на сервер СУБД. Текст запроса отображается ниже таблицы с выборкой данных.

Вернуться на страницу со списком запросов можно по ссылке «Вернуться к списку».

Редактирование запроса

Составление запроса требует понимания структуры настроек базы ADVANTA, а также понимание того, как делаются запросы в LINQ.
Больше информации можно найти в открытых источниках и документации по C#.

В запросах всегда доступны 3 переменные:

- `dataContext` – доступ к наборам объектов
- `parameters` – доступ к параметрам, переданным из вне, для параметризованного запроса
  - В переменной всегда доступен параметр `QueryParameters.Common.BaseUrl` содержащий адрес системы. Доступ к ней возможен например по обращению `parameters.GetValue<string>(QueryParameters.Common.BaseUrl)`
  - `QueryParameters.Common.PersonId`, содержащий GUID текущего пользователя, от которого выполняется запрос. Он может быть полезен, при создании отчетов типа «Я исполнитель», «Я … кто в объекте», для определения кто этот «Я». Доступ к нему возможен, например `parameters.GetValue<Guid>(QueryParameters.Common.PersonId)`
- `workCalendars` – через нее можно получить календарь объекта, или календарь по умолчанию.
  - `workCalendars.GetDefaultCalendar()` – возвращает объект Календарь по умолчанию
  - `workCalendars.GetWorkCalendar(projectId)` – возвращает календарь переданного объекта
    К объекту календарь НЕЛЬЗЯ обращаться в самом запросе, только при дальнейшей обработки возвращенной последовательности, возвращать можно только Id календаря

Через переменную `dataContext` доступны объекты:

- Projects - любые объекты в дереве объектов. Для обращения к объектам конкретного типа, необходимо использовать название объекта после транслитерации типа `Zadacha_c88ec2_List`;

- DirectoryRecords - строки любых справочников. Для обращения к строкам конкретного типа справочника, необходимо использовать название справочника после транслитерации типа `Kontakti_e15d5d_List`;
- реквизиты как свойства объектов и справочников;
- горизонтальные связи, как свойства объектов;
- Persons - пользователи;
- Discussions - дискуссии;
- Documents - документы с их реквизитами и версиями;
- ProjectMembers - участники приглашенные в объекты.

Для действий с реквизитами, в частности типа `DateTime` можно использовать библиотеку `DbFunctions`.

Доступны классификаторы:

- статусов `ProjectStatus`, для вывода и фильтрации объектов по статусу;
- ролей участников объекта `ProjectMemberRole`.

Рабочий календарь. Методы

Рабочий календарь содержит следующие методы:

- `string Id` – ид календаря
- `string Name` – имя календаря
- `bool IsDefault` – является ли календарь по умолчанию
- `double WorkHoursInDay` – количество часов в рабочем дне
- `double WorkHoursInWeek` – количество часов в неделе
- `double WorkDaysInMonth` – количество рабочих дней в месяце
- `double WorkDaysInYear` – количество рабочих дней в году
- `DateTime AddDuration(DateTime dateTime, TimeSpan duration);` – добавить длительность к дате с учетом рабочего календаря
- `DateTime GetDayEnd(DateTime day);` – получить конец рабочего дня
- `DateTime GetDayStart(DateTime day);` – получить начало рабочего дня
- `double GetWorktimeHours(DateTime start, DateTime end);` – получить количество часов между 2 датами с учетом рабочего календаря
- `double GetWorktimeDays(DateTime start, DateTime end);` – получить количество дней между 2 датами с учетом рабочего календаря
- `TimeSpan GetWorktimeSpan(DateTime start, DateTime end);` – получить длительность между 2 датами с учетом рабочего календаря.

> **ВНИМАНИЕ!**
>
> Возвращаемое значение содержит разницу, выраженную в рабочем времени, нельзя получать часы, минуты или другие составляющие напрямую, так как сутки не равны рабочему времени, для получения разницы в рабочих днях необходимо использовать GetWorktimeDays.

- `bool IsWorkDay(DateTime day);` – проверить является ли день рабочим

Кроме 3х переменных, доступны все используемые в системе классификаторы, которые генерируются как статические объекты и имеют наименования типа «Classifier_...», в котором доступны все пункты, в классификаторе, и они могут использоваться в запросах к объектам системы.

Например:

```
var Negotiation = dataContext.Negotiations
    .Where(n => n.Tip_soglasovaniya_6ecd7b.Id ==
Classifier_Tip_soglasovaniya_9a469d.Drugoj_57e6ea78_Id)
```

> Для удобства работы при наборе текста запроса доступна функция Intellisence, выводящая подсказки для выбора объектов (вызов по Ctrl + Space).

Перечень объектов, доступный для запросов, генерируется или обновляется при пересоздании контекста.

Для объекта общего типа Project (а также для частных), реализованы дополнительные методы для получения данных о иерархии.

- `public ICollection<TCustomType> GetChildren<TCustomType>()` – получение детей проекта без учета иерархии
- `public TCustomType GetParent<TCustomType>()` – получение родителя, если родитель не TCustomType, то вернется null
- `public ICollection<TCustomType> GetChildrenHierarchy<TCustomType>(bool includeRoot)` – получение детей с учетом иерархии, если includeRoot, то сам проект тоже попадет в коллекцию, если он соответствующего типа
- `public ICollection<TCustomType> GetParentHierarchy<TCustomType>(bool includeRoot)` – получение родителей с учетом иерархии

LINQ-запрос как источник для "Электронной таблицы"

Результаты LINQ-запроса можно вывести в отчёт "Электронная таблица" как один из источников данных.

При выборе источников для отчёта вы также можете выбрать «linq-источник» – указать его идентификатор.
Можно указать несколько linq-источников. Каждый из них будет выгружаться на отдельный лист таблицы.

После сохранения источников данных, в табличном отчете появляются (по количеству запросов) закладки с данными из запросов. «Query1», «Query2» и т.д. по числу источников-запросов.

Дальнейшая работа отчета «Электронная таблица» никак не отличается от имеющегося функционала.

В случае, если запрос вызывается для отчета из типа объекта, то в него передаются

дополнительные параметры:

- `QueryParameters.SpreadsheetReport.StartDate` – параметр равный началу периода в табличном отчете. Пример его использования может выглядеть примерно так:

```
parameters.GetValueOrDefault<DateTime?>(QueryParameters.SpreadsheetReport.StartDate, DateTime.Now.AddMonths(-1));
```

- `QueryParameters.SpreadsheetReport.EndDate` – параметр равный окончанию периода в табличном отчете.
- `QueryParameters.SpreadsheetReport.ProjectId` – параметр равный идентификатору объекта, из которого вызывается отчет. Пример его использования:

```
parameters.GetValueOrDefault<Guid?>(QueryParameters.SpreadsheetReport.ProjectId, new Guid("ac04eda3-6940-46ae-8c8b-5c43c67a43a8"));
```

WebAPI

Получение Linq-запросов может выполняться от имени любого пользователя с учетом его прав. Нужно использовать CookieContainer для сохранения cookies и уже с ним делать последующие запросы к API.

Linq вызывается через стандартный интерфейс WebApi. Для этого, например в .NET, можно использовать класс HttpClient. Тело POST запросов передается в формате JSON в фигурных скобках.

Первым шагом необходимо авторизоваться, передав запрос с реквизитами подключения Пользователя системы АДВАНТА по ссылке: http://localhost/Master/api/auth/login

```
POST http://localhost/Master/api/auth/login

{
  Login:"admin",
  Password:"123456"
}
```

Альтернативный способ авторизации - используя токен пользователя (для версии системы ADVANTA 3.26 и выше)

```
POST http://localhost/Master/api/auth/login

{
  tokenValue:"kjhfwi327yr92hjrhfbyugs8r384ygrhf8rifh"
}
```

> Здесь и далее `localhost` - адрес системы, а `Master` - приложение системы (может не указываться, если является единственным приложением на домене).

После авторизации отправляется запрос непосредственно к Linq, используя ключ источника (DataSourceKey)

```
POST http://localhost/Master/api/queries/get
{
    DataSourceKey: "Key_of_Query",
    Parameters: { A: 123, B: "321" },
    PageSize: 5
}
```

или Id LINQ-запроса (DataSourceId)

```
POST http://localhost/Master/api/queries/get
{
    DataSourceId: "54d198c3-bbd9-4812-9143-76c632a80e44",
    Parameters: { A: 123, B: "321" },
    PageSize: 20
}
```

> Здесь Parameters - передаваемые в LINQ-запрос параметры (необязательное), а PageSize - максимальное количество первых по порядку возвращаемых из запроса записей (необязательное, но если не указано - всегда возвращаются только первые 10 записей из результатов выполнения запроса).

Обращение к данным системы с текстом LINQ-запроса (Text) непосредственно в вызове метода

```
POST http://localhost/Master/api/queries/get
{
    Text: 'var a = (long)parameters["A"];
    var b = parameters.GetValue<string>("B");
    return dataContext.Zadacha_2b33fa_List
   .Select(z => new { z.Name });',
    Parameters: { A: 123, B: "321" },
    PageSize: 100
}
```

Пример вызова LINQ-запроса в PowerShell

```
$baseAddress = New-Object -TypeName Uri -ArgumentList $urlA2
$cookieContainer = New-Object -TypeName System.Net.CookieContainer

$handler = New-Object -TypeName System.Net.Http.HttpClientHandler
$handler.CookieContainer = $cookieContainer

$client = New-Object -TypeName System.Net.Http.HttpClient -ArgumentList
$handler
```

```
$client.BaseAddress = $baseAddress
$encoding = [System.Text.Encoding]::GetEncoding("utf-8")

# Авторизация
$auth = @{ 'Login' = $login; 'Password' = $password) } | ConvertTo-Json
$authResponse = $client.PostAsync(
            [string]::Join('/', $urlA2, "api/auth/login"),
            (New-Object -TypeName System.Net.Http.StringContent -
ArgumentList $auth, $encoding,"application/json"))

# Запрос
$query = @{
  'DataSourceId' = $DataSourceId
  'PageSize' = 1000
} | ConvertTo-Json
$queryResponse = $client.PostAsync(
             [string]::Join('/', $urlA2, "api/queries/get"),
                          (New-Object -TypeName System.Net.Http.StringContent
-ArgumentList $query, $encoding,"application/json"))

$queryResponseResult = $queryResponse.Result.Content.ReadAsStringAsync()

# Результат Linq запроса
$result = $queryResponseResult.Result | ConvertFrom-Json
```

Пример вызова LINQ-запроса в Python

в файле Python-скрипта обязательно используйте utf-8 кодировку!

```
import requests

# укажите свои параметры авторизации и адрес размещения системы
LOGIN = 'xxxxx'
PASSWORD = 'xxxxx'
DOMAIN = 'http://xxxxx' # при необходимости, указывайте протокол https в строке с
реальным адресом домена системы

session = requests.Session()

# авторизация
response = session.post(
    url=DOMAIN+'/api/auth/login',
    json={
        'Login': LOGIN,
        'Password': PASSWORD,
    },
)

# сохранение полученных после авторизации cookies для последующих запросов
cookies = session.cookies.get_dict()
```

```
# параметры для обращения к LINQ-запросу (может быть больше, см. выше)
LINQ = {
    'DataSourceId': "6b7c0388-a249-4d39-9cb6-xxxxxxxxxxx", # id LINQ-запроса в системе ADVANTA
    'PageSize': 100, # максимальное количество возвращаемых записей
}

# получение данных из LINQ-запроса
response = session.post(
    url=DOMAIN+'/api/queries/get',
    cookies=cookies,
    json=LINQ,
)

data = response.json()
```

Примеры LINQ-запросов