Creatio Academy

Data services

DataService

Version 8.0

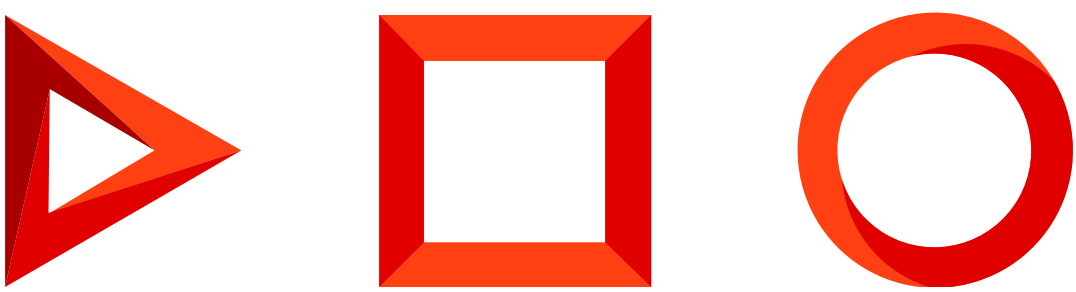


Table of Contents

DataService

Advanced

The Creatio DataService web service is a RESTfull service. RESTful is a quite simple information management interface that doesn't use any additional internal layers, i.e., the data doesn't need to be converted to any third-party format, such as XML. In a simple RESTful service, each record is uniquely identified by a global identifier such as URL. Each URL, in turn, has a strictly specified format. However, this service is not always convenient for transferring large amounts of data.

With the use of the DataService, the data can be automatically configured in various data formats such as XML, JSON, HTML, CSV, and JSV. The data structure is determined by data contracts.

The full list and description of the DataService data contracts is displayed on table.

The Creatio application DataService services

Service	Description
InsertQuery	Add section record query class.
UpdateQuery	Update section record query class.
DeleteQuery	Delete section record query class.
SelectQuery	Select section record query class.
BatchQuery	Package query class.
Filters	Filter class.

Restrictions when using DataService

When using DataService, take into account the following restrictions:

1. Maximum number of records that you can obtain by request is specified in the [*MaxEntityRowCount*] (the default value is 20000). You can change the system setting value in the `.\Terrasoft.WebApp\Web.config` file.

   Attention. We do not recommend changing the [*MaxEntityRowCount*] system setting. This may lead to performance issues. We recommend using the page-oriented display implemented in the "`IsPageable`" and "`RowCount`" properties of the SelectQuery data contract.

2. The number of requests is unlimited.

Create a new record in the section

Advanced

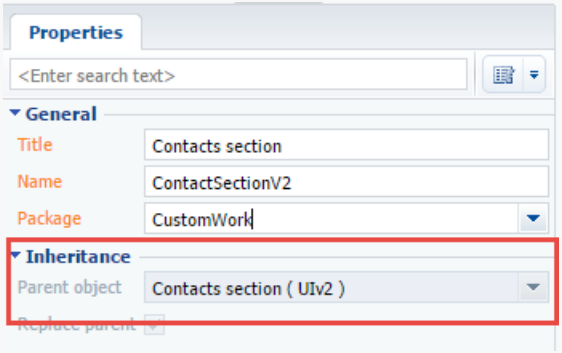
Example. Add to the [*Contacts*] section a button that when clicked invokes a method, using the `InsertQuery` class that adds a record with the following data:

- `Full name` — John Best;
- `Full job title` — Developer;
- `Business phone` — +12 345 678 00 00.

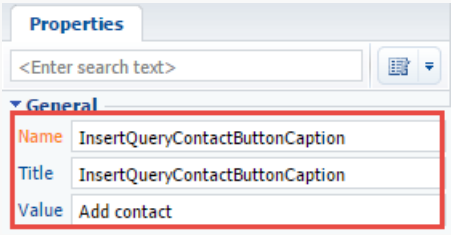
Example implementation algorithm

1. Add button to the [*Contacts*] section

For this particular case, you need to create a replacement client module of the [*Contacts*] section.



In the created client schema, add the `InsertQueryContactButtonCaption` localizable string, and set the "`Add contact`" value to it.



Add a configuration object with the button location settings to the `diff` array.

`diff`

```
diff: /**SCHEMA_DIFF*/[
    // Metadata to be added to the custom button section.
    {
```

```
        // The element is added to the page.
        "operation": "insert",
        // Parent interface element name to which the button is added.
        "parentName": "ActionButtonsContainer",
        // The button is added to the interface element collection
        // of the parent element (its metaname is specified in parentName).
        "propertyName": "items",
        // Added button metaname.
        "name": "InsertQueryContactButton",
        // Additional element properties.
        "values": {
            // Added element type – button.
            itemType: Terrasoft.ViewItemType.BUTTON,
            // Binding button caption to the localizable schema string.
            caption: { bindTo: "Resources.Strings.InsertQueryContactButtonCaption" },
            // Binding method of processing the button click.
            click: { bindTo: "onInsertQueryContactClick" },
            "layout": {
                "column": 1,
                "row": 6,
                "colSpan": 1
            }
        }
    }
]/**SCHEMA_DIFF*/
```

2. Add the processing method for the button click event

In order for a record with the necessary data to be added when a button created in the section is clicked, add the following method to the `methods` section of the replacement client schema:

methods

```
methods: {
    // Method of processing the button click.
    onInsertQueryContactClick: function() {
        // Creating the Terrasoft.InsertQuery class instance.
        var insert = Ext.create("Terrasoft.InsertQuery", {
            // Root schema name.
            rootSchemaName: "Contact"
        });
        // Setting the Terrasoft.ParameterExpression value-parameters.
        // A value-parameter instance is created and added to the column value collection.
        // Creating a value-parameter instance for the [Job title] column.
        insert.setParameterValue("Name", "John Best", Terrasoft.DataValueType.TEXT);
        // Creating a value-parameter instance for the [Business phone] column.
        insert.setParameterValue("Phone", "+12 345 678 00 00", Terrasoft.DataValueType.TEXT);
```

```
        // Creating a value-parameter instance for the [Job title] column.
        insert.setParameterValue("Job", "11D68189-CED6-DF11-9B2A-001D60E938C6", Terrasoft.DataVa
        // Data update query.
        insert.execute(function(response) {
            // Displaying server answer
            window.console.log(response);
        });
        // Updating list data.
        this.reloadGridData();
    }
}
```

Attention. Unlike the previous example, authorization is not required because the code is executed directly in the Creatio application.

The implementation of the `InsertQuery` class for the client part of the application kernel is different from the implementation of the `InsertQuery` in its back end. So, to create the parameters, the `setParameterValue` method is used, and for the query execution — the `execute` method. Learn about all the available properties and methods of the `InsertQuery` class implemented in the kernel client part in the API documentation.

Read records in the section

 Advanced

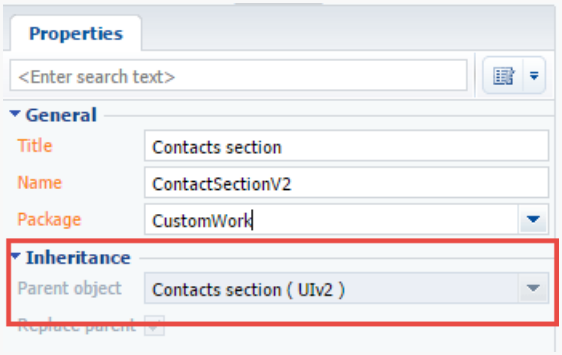
Example. In the [*Contacts*] section, add a button which will open the method that will use DataService to read records in the [*Contacts*] section with the following columns:

- `Id`;
- `Full name`;
- `Number of activities` — aggregate column, which displays the number of activities of this contact.

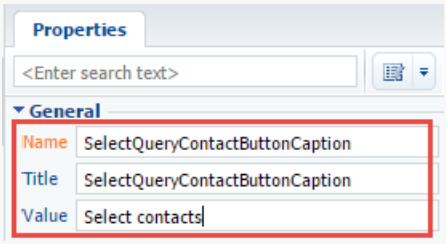
Example implementation algorithm

1. Add a button in the [*Contacts*] section

Create a replacing client module of the [*Contacts*] section.



In the created client schema, add `SelectQueryContactButtonCaption` localizable string and set its value to `Select contacts`.



Add a configuration object with the settings determining the button position to the `diff` array.

diff

```
//Setup of section button display.
diff: /**SCHEMA_DIFF*/[
    // Metadata for adding a custom button in a section.
    {
        // Indicates that an elementis added on a page.
        "operation": "insert",
        // Meta name of the parent control element where the button is added.
        "parentName": "ActionButtonsContainer",
        // Indicates that the button is added to the control element collection
        // of parent element (meta-name specified in parentName).
        "propertyName": "items",
        // Meta-name of the added button.
        "name": "SelectQueryContactButton",
        // Additional properties of the element.
        "values": {
            // Type of added element - button.
            itemType: Terrasoft.ViewItemType.BUTTON,
            // Binding button title to a schema localizable string.
            caption: { bindTo: "Resources.Strings.SelectQueryContactButtonCaption" },
            // Binding of the button pressing handler method.
            click: { bindTo: "onSelectQueryContactClick" },
```

```
            "layout": {
                "column": 1,
                "row": 6,
                "colSpan": 1
            }
        }
    }

]/**SCHEMA_DIFF*/
```

2. Add handler method for the button pressing event

To enable reading the records when the button is clicked, add the following method to the `methods` section of the replacing client schema:

methods

```
methods: {
    // Handler method for button click.
    onSelectQueryContactClick: function() {
        // Creating an instance of the Terrasoft.InsertQuery class.
        var select = Ext.create("Terrasoft.EntitySchemaQuery", {
            // Root schema name.
            rootSchemaName: "Contact"
        });
        // Adding the [Full name] column to query.
        select.addColumn("Name");
        // Adding [Number of activities] aggregate column to a query.
        select.addAggregationSchemaColumn(
            // Path to column in relation to the root schema.
            "[Activity:Contact].Id",
            // Aggregation type – quantity.
            Terrasoft.AggregationType.COUNT,
            // Column title.
            "ActivitiesCount",
            // Aggregation function scope - for all elements.
            Terrasoft.AggregationEvalType.ALL);
        // Update query to server
        // Getting whole collection of records and displaying it in the browser console.
        select.getEntityCollection(function(result) {
            if (!result.success) {
                // Processing/logging of error.
                this.showInformationDialog("Data query error");
                return;
            }
            // Displayed message.
            var message = "";
```

```
            // Analyzing resulting collection and generating displayed message.
            result.collection.each(function(item) {
                message += "Full name: " + item.get("Name") +
                ". Number of activities: " + item.get("ActivitiesCount") + "\n";
            });
            // Displaying message in console.
            window.console.log(message);
        }, this);
    }
}
```

Note. Unlike the previous example, authentication is not needed in this case, because the program code is executed by Creatio directly.

In the client of the application core, there is not a class like the server core `SelectQuery` class. To select data from a section, use the `Terrasoft.EntitySchemaQuery` class. For more information on this class methods and properties are described in the API documentation.

Filter records in the section

Advanced

Example. Create a console application that will read the following data from the [*Contacts*] section using the DataService service:

- `Id`;
- `Full name`;
- `Number of activities` is an aggregating column that shows the number of activities of this contact.

It is necessary to filter the data so that only those contacts whose number of activities is in the range of 1 to 3, and the [*Full name*] column value starting with "H" are read.

Example implementation algorithm

1. Create and configure a C# console application project that reads records

To perform this step, you must perform the example of reading records in a third-party application.

The result of the query class implementation instance to read the records with the columns in an abbreviated form:

```
SelectQuery()
```

```
// Query class instance.
var selectQuery = new SelectQuery ()
{
    // Root schema name.
    RootSchemaName = "Contact",
    // Adding columns to query.
    Columns = new SelectQueryColumns ()
    {
        // Column collection.
        Items = new Dictionary <string, SelectQueryColumn> ()
        {
            // Column [Full name].
            {
                // Key.
                "Name",
                // Value.
                new SelectQueryColumn ()
                {
                    // An expression that specifies the column type.
                    Expression = new ColumnExpression ()
                    {
                        // Expression type - schema column.
                        ExpressionType = EntitySchemaQueryExpressionType.SchemaColumn,
                        // Path to the column.
                        ColumnPath = "Name"
                    }
                }
            },
            // Column [Number of activities].
            {
                "ActivitiesCount",
                new SelectQueryColumn ()
                {
                    Expression = new ColumnExpression ()
                    {
                        // Expression - subquery.
                        ExpressionType = EntitySchemaQueryExpressionType.SubQuery,
                        // Path to the column relative to the root schema.
                        ColumnPath = "[Activity: Contact] .Id",
                        // Function type - aggregating.
                        FunctionType = FunctionType.Aggregation,
                        // Aggregation type - number.
                        AggregationType = AggregationType.Count
                    }
                }
            }
        }
```

```
    }
};
```

2. Add filter implementation

In order to filter data, you must create an instance of the `Filters` collection class instance, fill in the necessary properties, and then pass the link to this instance to the `Filters` property of the query class instance that you created in the previous step.

Filter collection class implementation example

```
// Query filters.
var selectFilters = new Filters ()
{
    // Filter Type - group.
    FilterType = Terrasoft.Nui.ServiceModel.DataContract.FilterType.FilterGroup,
    // Filter collection.
    Items = new Dictionary <string, Filter>
    {
// Filter Implementation.
    }
};
// Adding filter to query.
selectQuery.Filters = selectFilters;
// Query class instance serialization to read data from the JSON string.
var json = new JavaScriptSerializer () Serialize (selectQuery).;
```

The `Items` property must contain the key-value type collection. The key is a string containing the filter name, and the value is an instance of the `Filter` class that contains a direct implementation of the filter.

To implement a filter that selects only those contacts that have a number of activities within a range of 1 to 3, you must add the following instance to the collection of filters:

Example of the instance to the collection of filters

```
// Filtration by activity.
{
    // Key.
    "FilterActivities",
    // Value.
    new Filter
    {
        // Filter type - range filter.
        FilterType = Terrasoft.Nui.ServiceModel.DataContract.FilterType.Between,
        // Comparison type - range.
        ComparisonType = FilterComparisonType.Between,
```

```
        // An expression to be tested.
        LeftExpression = new BaseExpression ()
        {
            // Expression type - subquery.
            ExpressionType = EntitySchemaQueryExpressionType.SubQuery,
            // Path to the column relative to the root schema.
            ColumnPath = "[Activity: Contact] .Id",
            // Function type - aggregating.
            FunctionType = FunctionType.Aggregation,
            // Aggregation type - number.
            AggregationType = AggregationType.Count
        }
        // Filter range final expression.
        RightGreaterExpression = new BaseExpression ()
        {
            // Expression type - parameter.
            ExpressionType = EntitySchemaQueryExpressionType.Parameter,
            // Expression parameter.
            Parameter = new Parameter ()
            {
                // Parameter data type - integer.
                DataValueType = DataValueType.Integer,
                // Parameter value.
                Value = 3
            }
        }
        // Filter range initial expression.
        RightLessExpression = new BaseExpression ()
        {
            ExpressionType = EntitySchemaQueryExpressionType.Parameter,
            Parameter = new Parameter ()
            {
                DataValueType = DataValueType.Integer,
                Value = 1
            }
        }
    }
}
```

Add the following instance to the filter collection to filter contact records where the [*Full name*] column value begins with "H":

Example of the instance to the filter collection

```
// Filtering by name.
{
    // Key.
```

```
    "FilterName",
    // Value.
    new Filter
    {
        // Filter type - comparison filter.
        FilterType = Terrasoft.Nui.ServiceModel.DataContract.FilterType.CompareFilter,
        // Comparison type - starts with an expression.
        ComparisonType = FilterComparisonType.StartWith,
        // Expression to be tested.
        LeftExpression = new BaseExpression ()
        {
            // Expression type - schema column.
            ExpressionType = EntitySchemaQueryExpressionType.SchemaColumn,
            // Path to the column.
            ColumnPath = "Name"
        }
        // Filtration expression.
        RightExpression = new BaseExpression ()
        {
            // Expression type - parameter.
            ExpressionType = EntitySchemaQueryExpressionType.Parameter,
             // Expression parameter.
            Parameter = new Parameter ()
            {
                // Parameter data type - text.
                DataValueType = DataValueType.Text,
                // Parameter value.
                Value = "CH"
            }
        }
    }
}
```

Filter records in the section using a macro

Advanced

Example. Create a console application that uses DataService to read records from the [*Contacts*] section with the following columns:

- `Id`;
- `Full name`;
- `Birth date`.

The data must be filtered, so that only contacts who were born in 1992 are shown.

Example implementation algorithm

1. Create and set up a C# application project that reads records

To execute this step, execute the record reading example.

The result of implementing an instance of query class for reading records:

`SelectQuery()`

```
// Instance of query class.
var selectQuery = new SelectQuery()
{
    // Root schema name.
    RootSchemaName = "Contact",
    // Adding columns to query.
    Columns = new SelectQueryColumns()
    {
        // Collection of columns.
        Items = new Dictionary<string, SelectQueryColumn>()
        {
            //Column [Full name].
            {
                // Key.
                "Name",
                // Value.
                new SelectQueryColumn()
                {
                    // Expression that specifies the column type.
                    Expression = new ColumnExpression()
                    {
                        // Type of expression - schema column.
                        ExpressionType = EntitySchemaQueryExpressionType.SchemaColumn,
                        // Pat to column.
                        ColumnPath = "Name"
                    }
                }
            },
            // Column [Number of activities].
            {
                "ActivitiesCount",
                new SelectQueryColumn()
                {
                    Expression = new ColumnExpression()
                    {
                        // Expression type – subquery.
                        ExpressionType = EntitySchemaQueryExpressionType.SubQuery,
                        // Path to column relative to the oot schema.
```

```
                        ColumnPath = "[Activity:Contact].Id",
                        // Function tyoe – aggregation.
                        FunctionType = FunctionType.Aggregation,
                        // Aggregation type – quantity.
                        AggregationType = AggregationType.Count
                    }
                }
            }
        }
    }
};
```

2. Add a filter implementation with macros

To filter the data, create an instance of the `Filters` collection class, fill out the properties with corresponding values, and then pass the instance link to the `Filters` property of the query class created on the previous step.

An example of filter collection class implementation

```
// Query filters.
var selectFilters = new Filters()
{
    // Filter type – group.
    FilterType = Terrasoft.Nui.ServiceModel.DataContract.FilterType.FilterGroup,
    // Filter collection.
    Items = new Dictionary<string, Filter>
    {

        // Filter by year of birth.
        {
            // Key.
            "FilterYear",
            // Value.
            new Filter
            {
                // Filter type – comparison.
                FilterType = Terrasoft.Nui.ServiceModel.DataContract.FilterType.CompareFilter,
                // Comparison type – equal.
                ComparisonType = FilterComparisonType.Equal,
                // Expression to check.
                LeftExpression = new BaseExpression()
                {
                    // Expression type – schema column.
                    ExpressionType = EntitySchemaQueryExpressionType.SchemaColumn,
                    // Path to schema.
                    ColumnPath = "BirthDate"
                },
```

```
                // Expression with which the checked value is compared.
                RightExpression = new BaseExpression
                {
                    // Expression type – function.
                    ExpressionType = EntitySchemaQueryExpressionType.Function,
                    // Function type – macro.
                    FunctionType = FunctionType.Macros,
                    // Macro type – year.
                    MacrosType = EntitySchemaQueryMacrosType.Year,
                    // Function argument.
                    FunctionArgument = new BaseExpression
                    {
                        // Type of expression that determines the argument – parameter.
                        ExpressionType = EntitySchemaQueryExpressionType.Parameter,
                        // Parameter initialization.
                        Parameter = new Parameter
                        {
                            // Parameter type – integer.
                            DataValueType = DataValueType.Integer,
                            // Parameter value.
                            Value = "1992"
                        }
                    }
                }
            }
        }
    }
};
// Adding filters to query.
selectQuery.Filters = selectFilters;
// Serialization of select query class instance in a JSON string.
var json = new JavaScriptSerializer().Serialize(selectQuery);
```

The collection contains a single filter with the `FilterYear` key. Because only those records that have their year of birth equal to 1992 must be selected from the collection, the type of filter is set as a comparison filter. The type of comparison is set as an equality of values. As a verified expression, set the [*Date of birth*] column. Specify the macro function as the expression to compare with.

In this case using a macro is optimal because the birth date is stored in the database in `YYYY-MM-DD` format. The macro automatically determines the year value, so the developer does not need to write additional program code.

Because the `EntitySchemaQueryMacrosType.Year` macro is parametric, the `FunctionArgument` property must be initialized and assigned a link to an instance of the `BaseExpression` class. In it, the integer parameter with value "1992" is defined.

Update a record in the section


Advanced

Example. Create a console application that used DataService to update the "`John Smith`" record added in the example. Add "`j.smith@creatio.com`" as the value in the [*Email*] column of this record.

Example implementation algorithm

1. Create and set up a C# application project

Using the Microsoft Visual Studio development environment (version 2017 and up), create a Visual C# console application project and specify project name, for example, `DataServiceUpdateExample`. Set "`.NET Framework 4.7`" for the project property `Target framework`.

In the `References` section of the project, add dependencies from the following libraries:

- `System.Web.Extensions.dll` – class library included in .NET Framework;
- `Terrasoft.Core.dll` – library of base Creatio server core classes. It can be found using the following path: `[Creatio setup catalog]\Terrasoft.WebApp\bin\Terrasoft.Core.dll`;
- `Terrasoft.Nui.ServiceModel.dll` — application service class library. It can be found using the following path: `[Creatio setup catalog]\Terrasoft.WebApp\bin\Terrasoft.Nui.ServiceModel.dll`;
- `Terrasoft.Common.dll` – library of base Creatio server core classes. It can be found using the following path: `[Creatio setup catalog]\Terrasoft.WebApp\bin\Terrasoft.Common.dll`.

Add the `using` directives to the application source code file:

Adding the `using` directives

```
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. Add fields and constants and field declarations to the source code

To access DataService features, add the fields and constants to the application source code.

Adding the fields and constants

```
// Primary URL of Creatio application. Must be repoaced with a custom one.
private const string baseUri = @"http://example.creatio.com";
```

```
// Request string to the Login methid of the AuthService.svc service.
private const string authServiceUri = baseUri + @"/ServiceModel/AuthService.svc/Login";
// Path string for the UpdateQuery.
private const string updateQueryUri = baseUri + @"/0/DataService/json/reply/UpdateQuery";
// Creatio authentication cookie.
private static CookieContainer AuthCookie = new CookieContainer();
```

Here, three string fields are declared. These fields will be used to form authentication query and read data queries execution paths. Authentication data will be saved in the `AuthCookie` field.

3. Add method that performs Creatio application authentication

Authentication is required to enable access to the DataService for the created application.

4. Add implementation of the record add query

Because the `updateQueryUri` constant declared earlier contains a path for sending data in the JSON format, sent data must be configured beforehand as a string that contains a JSON object that corresponds to the `UpdateQuery` data contract. This can be done directly in a string variable, although a much more secure and convenient way of doing this would be to create an instance of the `UpdateQuery` class, fill out its properties and then serialize it to a string.

Implementation the request for adding a record

```
// Instance of the request class.
 var updateQuery = new UpdateQuery()
 {
     // Root schema name.
     RootSchemaName = "Contact",
     // New column values.
     ColumnValues = new ColumnValues()
     {
         // Key-value collection.
         Items = new Dictionary<string, ColumnExpression>()
         {
             // [Email] column.
             {
                 // key.
                 "Email",
                 // Value – instance of object schema request class.
                 // Configuration of [Email] column.
                 new ColumnExpression()
                 {
                     // Type of expression of obkect schema query – parameter.
                     ExpressionType = EntitySchemaQueryExpressionType.Parameter,
                     // Query expression parameter.
```

```
                    Parameter = new Parameter()
                    {
                        // Parameter value.
                        Value = "j.smith@creatio.com",
                        // Parameter data type – string.
                        DataValueType = DataValueType.Text
                    }
                }
            }
        }
    },
    // Query filters.
    Filters = new Filters()
    {
        // Filter type – group.
        FilterType = Terrasoft.Nui.ServiceModel.DataContract.FilterType.FilterGroup,
        // Filter collection.
        Items = new Dictionary<string, Filter>()
        {
             // Filter by name.
            {
                // Key.
                "FilterByName",
                // Value.
                new Filter
                {
                    // Filter type – comparison filter.
                    FilterType = Terrasoft.Nui.ServiceModel.DataContract.FilterType.CompareFilt
                    // Comparison type – starts with expression.
                    ComparisonType = FilterComparisonType.Equal,
                    // Expression to check.
                    LeftExpression = new BaseExpression()
                    {
                        // Expression type - schema column.
                        ExpressionType = EntitySchemaQueryExpressionType.SchemaColumn,
                        // Path to column.
                        ColumnPath = "Name"
                    },
                    // Filtering expression.
                    RightExpression = new BaseExpression()
                    {
                        // Expression type - parameter.
                        ExpressionType = EntitySchemaQueryExpressionType.Parameter,
                         // Expression parameter.
                        Parameter = new Parameter()
                        {
                            // Parameter data type - text.
                            DataValueType = DataValueType.Text,
                            // Parameter value.
```

```
                            Value = "John Smith"
                        }
                    }
                }
            }
        }
    }
};
// Serialization of update query class instance in a JSON string.
var json = new JavaScriptSerializer().Serialize(updateQuery);
```

Here, an instance of the `UpdateQuery` class is created. In the `ColumnValues` property, the "`j.smith@creatio.com`" value is set for the [*Email*] column. To apply this value to a specific record or group of records, specify a link to a correctly initialized `Filters` class in the `Filters` property. In this case, a single filter is added to the filters collection to select only records that have the "`John Smith`" value in the [*Full name*] column.

The next step is to execute DataService `POST`-query. To do this, create an instance of the < `HttpWebRequest` class, fill its properties and connect the string with the JSON object created earlier then execute the DataService query and process its result. To do this, add the following source code:

Request implementation

```
// Converting a JSON object string to a byte array.
byte[] jsonArray = Encoding.UTF8.GetBytes(json);
// Creating an insrance of HTTP request.
var updateRequest = HttpWebRequest.Create(updateQueryUri) as HttpWebRequest;
// Defining a request method.
updateRequest.Method = "POST";
// Determining type of request content.
updateRequest.ContentType = "application/json";
// Adding authentication cookie received earlier to a request.
updateRequest.CookieContainer = AuthCookie;
// Set length for request content.
updateRequest.ContentLength = jsonArray.Length;

// Putting BPMCSRF token to the request header.
CookieCollection cookieCollection = AuthCookie.GetCookies(new Uri(authServiceUri));
string csrfToken = cookieCollection["BPMCSRF"].Value;
updateRequest.Headers.Add("BPMCSRF", csrfToken);

// Plase a JSON-object to request content.
using (var requestStream = updateRequest.GetRequestStream())
{
    requestStream.Write(jsonArray, 0, jsonArray.Length);
}
// Executing HTTP request and getting a response from server.
using (var response = (HttpWebResponse)updateRequest.GetResponse())
```

```
{
    // Displaying response in console.
    using (StreamReader reader = new StreamReader(response.GetResponseStream()))
    {
        Console.WriteLine(reader.ReadToEnd());
    }
}
```

Delete a record in the section

Advanced

Example. Create a console application that, using the DataService service, will delete the "`John Best`" contact record added in the example.

Example implementation algorithm

1. Create and configure a C# console application project

Using the Microsoft Visual Studio (version 2017 and up) development environment, create a Visual C# console application project and name it *DataServiceDeleteExample*. The `Target framework` project property must be set to `.NET Framework 4.7`.

In the `References` section of the project you need to add dependencies of the following libraries:

- `System.Web.Extensions.dll` is a class library included in the .NET Framework
- `Terrasoft.Core.dll` is a main class library of the application server kernel. Can be found by the following path: `[Directory with the installed application]\Terrasoft.WebApp\bin\Terrasoft.Core.dll`
- `Terrasoft.Nui.ServiceModel.dll` class library the application services. Can be found by the following path: `[Directory with the application installed]\Terrasoft.WebApp\bin\Terrasoft.Nui.ServiceModel.dll`.
- `Terrasoft.Common.dll` is a main class library of the application server kernel. Can be found by the following path: `[Directory with the installed application]\Terrasoft.WebApp\bin\Terrasoft.Common.dll`.

Add `using` directives to the application source code file:

Adding `using` directives

```
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. Add field declarations and constants to the application source code

To access the DataService features, you must add the fields and constants to the application source code.

Adding the fields and constants

```
 // Main Creatio URL. Has to be changed to a custom one.
 private const string baseUri = @"http://example.creatio.com";
// Query string to the Login method of the AuthService.svc service.
 private const string authServiceUri = baseUri + @"/ServiceModel/AuthService.svc/Login";
 // DeleteQuery query path string.
 private const string deleteQueryUri = baseUri + @"/0/DataService/json/reply/DeleteQuery";
 // Creatio cookie authentication.
 private static CookieContainer AuthCookie = new CookieContainer();
```

Three string constant fields that are used to carry out the authentication requests and requests to read data are declared here. The authentication data will be stored in the `AuthCookie` field.

3. Add a method that performs authentication in the Creatio application

You need to authenticate the newly created application to access the DataService web service.

4. Implement a query to add a record

As the previously declared `updateQueryUri` constant contains the path for sending data in JSON format, the data sent must be pre-configured in the form of a string containing a description of the JSON object corresponding to the `UpdateQuery` data contract. This can be done directly in a lowercase variable but it is much easier and safer to create an instance of the `UpdateQuery` class, fill its properties, and then serialize it to a string.

Implementation the request to adding a record

```
// Query class instance.
var deleteQuery = new DeleteQuery()
{
    // Root schema name.
    RootSchemaName = "Contact",
    // Query filters.
    Filters = new Filters()
    {
        // Filter type – group.
        FilterType = Terrasoft.Nui.ServiceModel.DataContract.FilterType.FilterGroup,
```

```
        // Filter collection.
        Items = new Dictionary<string, Filter>()
        {
             // Filtration by name.
            {
                // Key.
                "FilterByName",
                // Value.
                new Filter
                {
                    // Filter type – comparison filter.
                    FilterType = Terrasoft.Nui.ServiceModel.DataContract.FilterType.CompareFilte
                    // Comparison type – starts with an expression.
                    ComparisonType = FilterComparisonType.Equal,
                    // Expression to be checked.
                    LeftExpression = new BaseExpression()
                    {
                        // Expression type – schema column.
                        ExpressionType = EntitySchemaQueryExpressionType.SchemaColumn,
                        // Column path.
                        ColumnPath = "Name"
                    },
                    // Filtration expression.
                    RightExpression = new BaseExpression()
                    {
                        // Expression type – parameter.
                        ExpressionType = EntitySchemaQueryExpressionType.Parameter,
                         // Expression parameter.
                        Parameter = new Parameter()
                        {
                            // Parameter data type – text.
                            DataValueType = DataValueType.Text,
                            // Parameter value.
                            Value = "John Best"
                        }
                    }
                }
            }
        }
    }
};
// Class instance serialization of the JSON string adding query.
var json = new JavaScriptSerializer().Serialize(updateQuery);
```

This creates an instance of the `DeleteQuery` class. The `Contact` value is set in the `RootSchemaName` property. To delete a particular record or group of records, you need to set a link to the correctly initialized `Filters` class instance to the `Filters` property. In this case, a single filter that selects only records with the `John Best` value in

the [*Full name*] column is added to the filter collection.

In the final step you must perform `POST` query to the DataService service. To do this, create an instance of the `HttpWebRequest` class, fill in its properties, attach a previously created string with the JSON object to a request, and then execute and process the result of the query to the DataService service. To do this, add the following source code:

Request implementation

```
// Converting a JSON object string to a byte array.
byte[] jsonArray = Encoding.UTF8.GetBytes(json);
// Creating an insrance of HTTP request.
var updateRequest = HttpWebRequest.Create(updateQueryUri) as HttpWebRequest;
// Defining a request method.
updateRequest.Method = "POST";
// Determining type of request content.
updateRequest.ContentType = "application/json";
// Adding authentication cookie received earlier to a request.
updateRequest.CookieContainer = AuthCookie;
// Set length for request content.
updateRequest.ContentLength = jsonArray.Length;

// Putting BPMCSRF token to the request header.
CookieCollection cookieCollection = AuthCookie.GetCookies(new Uri(authServiceUri));
string csrfToken = cookieCollection["BPMCSRF"].Value;
updateRequest.Headers.Add("BPMCSRF", csrfToken);

// Plase a JSON-object to request content.
using (var requestStream = updateRequest.GetRequestStream())
{
    requestStream.Write(jsonArray, 0, jsonArray.Length);
}
// Executing HTTP request and getting a response from server.
using (var response = (HttpWebResponse)updateRequest.GetResponse())
{
    // Displaying response in console.
    using (StreamReader reader = new StreamReader(response.GetResponseStream()))
    {
        Console.WriteLine(reader.ReadToEnd());
    }
}
```

Add and update a record in the section

Advanced

Example. Create a console application that will use DataService to:

- add a contact record with the value "John Smith" in the [*Full name*] column;
- change the value of the [*Business phone*] column to `012 345 67 89` for all contact records that have "`John Smith`" as the value in the [*Full name*].

Records must be added and modified via a batch query.

Example implementation algorithm

1. Create and set up a C# application project

Using the Microsoft Visual Studio development environment (version 2017 and up), create a Visual C# console application project and specify the project name, for example, `DataServiceBatchExample`. Set "`.NET Framework 4.7`" for the project property `Target framework`.

In the `References` section of the project, add dependencies from the following libraries:

- `System.Web.Extensions.dll` – class library included in .NET Framework;
- `Terrasoft.Core.dll` – library of base Creatio server core classes. It can be found using the following path: `[Creatio setup catalog]\Terrasoft.WebApp\bin\Terrasoft.Core.dll`;
- `Terrasoft.Nui.ServiceModel.dll` — application service class library. It can be found using the following path: `[Creatio setup catalog]\Terrasoft.WebApp\bin\Terrasoft.Nui.ServiceModel.dll`;
- `Terrasoft.Common.dll` – library of base Creatio server core classes. It can be found using the following path: `[Creatio setup catalog]\Terrasoft.WebApp\bin\Terrasoft.Common.dll`.

Add the `using` directives to the application source code file.

Adding the `using` directives

```
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. Add fields and constants and field declarations to the source code

To access DataService features, add the fields and constants to the application source code.

Adding the fields and constants

```
// Primary URL of Creatio application. Must be replaced with a custom one.
private const string baseUri = @"http://example.creatio.com";
// Request string to the Login methid of the AuthService.svc service.
private const string authServiceUri = baseUri + @"/ServiceModel/AuthService.svc/Login";
// Path string for the BatchQuery.
private const string batchQueryUri = baseUri + @"/0/DataService/json/reply/BatchQuery";
// Creatio authentication cookie.
private static CookieContainer AuthCookie = new CookieContainer();
```

Here, three string fields are declared. These fields will be used to form authentication query and read data queries execution paths. Authentication data will be saved in the `AuthCookie` field.

3. Add method that performs Creatio application authentication

Authentication is required to enable access of the created application to the DataService.

4. Implement query adding request

Because the `batchQueryUri` constant declared previously earlier contains a path for sending data in the JSON format, sent data must be configured beforehand as a string that contains a JSON object description. Use data contract classes to create separate queries then serialize them in a string.

For a query to add a contact record with the name "`John Smith`", add the following program code:

Implementation the request for adding a record

```
// Insert query.
var insertQuery = new InsertQuery()
{
    RootSchemaName = "Contact",
    ColumnValues = new ColumnValues()
    {
        Items = new Dictionary<string, ColumnExpression>()
        {
            {
                "Name",
                new ColumnExpression()
                {
                    ExpressionType = EntitySchemaQueryExpressionType.Parameter,
                    Parameter = new Parameter
                    {
                        Value = "John Smith",
                        DataValueType = DataValueType.Text
                    }
                }
```

```
            }
        }
    }
};
```

To change the value of the [*Business phone*] column to `012 345 67 89` for all contact records that have "John Smith" value in the [*Full name*] column, add the code.

Implementation the request for updating a record

```
 // Update query.
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.CompareFilte
                    ComparisonType = FilterComparisonType.Equal,
                    LeftExpression = new BaseExpression()
                    {
                        ExpressionType = EntitySchemaQueryExpressionType.SchemaColumn,
                        ColumnPath = "Name"
```

```
                    },
                    RightExpression = new BaseExpression()
                    {
                        ExpressionType = EntitySchemaQueryExpressionType.Parameter,
                        Parameter = new Parameter()
                        {
                            DataValueType = DataValueType.Text,
                            Value = "John Smith"
                        }
                    }
                }
            }
        }
    }
};
```

After serializing the created instances of the query class, add information about the qualified name of the corresponding data contract to the strings with JSON objects. Compose the string with batch query.

Batch query

```
// Serialization of update query class instance in a JSON string.
var jsonInsert = new JavaScriptSerializer().Serialize(insertQuery);
// Inserting query type in a JSON string.
jsonInsert = jsonInsert.Insert(1, @"""__type"": ""Terrasoft.Nui.ServiceModel.DataContract.Insert
// Serialization of instance of the update query class in a JSON string.
var jsonUpdate = new JavaScriptSerializer().Serialize(updateQuery);
// Inserting query type in a JSON string.
jsonUpdate = jsonUpdate.Insert(1, @"""__type"": ""Terrasoft.Nui.ServiceModel.DataContract.Update
// Creating batch query.
var json = @"{""items"": [" + jsonInsert + "," + jsonUpdate + "]}";
```

The next step is to execute the `POST` DataService query. To do this, create an instance of the `HttpWebRequest` class, fill its properties and connect the string with JSON object, created earlier, then execute the DataService query and process its result. To do this, add the source code.

Request implementation

```
// Converting a JSON object string in a byte array.
byte[] jsonArray = Encoding.UTF8.GetBytes(json);
// Creating an instance of HTTP request.
var batchRequest = HttpWebRequest.Create(deleteQueryUri) as HttpWebRequest;
// Defining request method.
batchRequest.Method = "POST";
// Determining request content.
batchRequest.ContentType = "application/json";
```

```
// Adding authentication cookie received earlier to a query.
batchRequest.CookieContainer = AuthCookie;
// Set the ContentLength property of the WebRequest.
batchRequest.ContentLength = jsonArray.Length;

// Adding CSRF token to the request header.
CookieCollection cookieCollection = AuthCookie.GetCookies(new Uri(authServiceUri));
string csrfToken = cookieCollection["BPMCSRF"].Value;
batchRequest.Headers.Add("BPMCSRF", csrfToken);

// Adding JSON object to the query contents.
using (var requestStream = batchRequest.GetRequestStream())
{
    requestStream.Write(jsonArray, 0, jsonArray.Length);
}
// Executing HTTP request and getting reply from server.
using (var response = (HttpWebResponse)batchRequest.GetResponse())
{
    // Displaying response in console.
    using (StreamReader reader = new StreamReader(response.GetResponseStream()))
    {
        Console.WriteLine(reader.ReadToEnd());
    }
}
// Application execution delay.
Console.ReadKey();
```

InsertQuery class C#

Advanced

The Namespace `Terrasoft.Nui.ServiceModel.DataContract`.

The `InsertQuery` data contract is used to add records to sections. The data is transferred to the DataService via HTTP by using the `POST` request with the URL.

The structure of request for adding data

```
// URL format of the POST query to add data to DataService.
http(s)://[Creatio application address]/[Configuration number]/dataservice/[Data fromat]/reply/I
```

An example the request for adding data

```
// URL example for the POST query to add data to DataService.
```

```
http(s)://example.creatio.com/0/dataservice/json/reply/InsertQuery
```

The `InsertQuery` data contract has a hierarchical structure with multiple nesting levels. In the Creatio application server part, the InsertQuery data contract is represented by the `InsertQuery` class of the `Terrasoft.Nui.ServiceModel.DataContract` namespace of the `Terrasoft.Nui.ServiceModel.dll` class library. However, for simplicity, the hierarchical structure of the `InsertQuery` data contract is conveniently presented as a JSON format object:

Structure of the `InsertQuery` data contract

```
{
    "RootSchemaName":"[Root object schema name]",
    "OperationType":[Record operation type],
    "ColumnValues":{
        "Items":{
            "Added column name":{
                "ExpressionType":[Expression type],
                "Parameter":{
                    "DataValueType":[Data type],
                    "Value":"[Column value]"
                }
            }...
        }
    }
}
```

Properties

`RootSchemaName`

A string containing the name of the root object schema of the added record.

`OperationType`

Operation type is set by the `QueryOperationType` namespace `Terrasoft.Nui.ServiceModel.DataContract` namespace enumeration value. For the `InsertQuery` the `QueryOperationType.Insert` value is set.

Possible values (`QueryOperationType`)

Select	0
Insert	1
Update	2
Delete	3
Batch	4

`ColumnValues`

Contains a collection of column values of the added record. Its `ColumnValues` type is defined in the `Terrasoft.Nui.ServiceModel.DataContract` namespace.<

The ColumnValues class C#

The Namespace `Terrasoft.Nui.ServiceModel.DataContract`.

The `ColumnValues` class has a single `Items` property that is defined as a collection of the `Dictionary<string, ColumnExpression>` keys and values. The key is a string with the added column title, and the value is the object with the `ColumnExpression` type defined in the `Terrasoft.Nui.ServiceModel.DataContract` namespace. The basic properties of the `ColumnExpression` class used when adding records, are given in table.

Properties

`ExpressionType`

The expression type that defines the value that will be contained in the added column. Set by the `EntitySchemaQueryExpressionType` enumeration of the `Terrasoft.Core.Entities` namespace defined in the `Terrasoft.Core` class library. For the `InsertQuery` the `EntitySchemaQueryExpressionType.Parameter` value is set.

Possible values (`EntitySchemaQueryExpressionType`)

SchemaColumn	0	Schema column.
Function	1	Function.
Parameter	2	Parameter.
SubQuery	3	Subquery.
ArithmeticOperation	4	Arithmetic operation.

`Parameter`

Defines the value that will be contained in the added column. Its `Parameter` type is defined in the `Terrasoft.Nui.ServiceModel.DataContract` namespace.

The Parameter class C#

The Namespace `Terrasoft.Nui.ServiceModel.DataContract`.

The `Parameter` class has multiple properties, two of which are used to add records.

Properties

`DataValueType`

The data value type that defines the value that will be contained in the added column. Set by the `DataValueType` enumeration value of the `Terrasoft.Nui.ServiceModel.DataContract` namespace.

Possible values (`DataValueType`)

Guid	0
Text	1
Integer	4
Float	5
Money	6
DateTime	7
Date	8
Time	9
Lookup	10
Enum	11
Boolean	12
Blob	13
Image	14
ImageLookup	16
Mapping	18

`Value`

The object that contains the added column value.

SelectQuery class C#

Advanced

The Namespace `Terrasoft.Nui.ServiceModel.DataContract`.

The `SelectQuery` data contract is used for reading section records. The query data is transferred to DataService via HTTP, with the help of `POST` by the URL.

The structure of request for reading data

```
// URL format of the POST query to read data from DataService.
http(s):///[Creatio application address]/[Configuration number]/dataservice/[Data fromat]/reply/S
```

An example the request for reading data

```
// URL example of the POST query to read data from DataService.
http(s)://example.creatio.com/0/dataservice/json/reply/SelectQuery
```

The `SelectQuery` data contract has a complex hierarchical structure with a number of nesting levels. In the Creatio server core, it is represented by a `SelectQuery` class of the `Terrasoft.Nui.ServiceModel.DataContract` namespace of the `Terrasoft.Nui.ServiceModel.dll` library of classes. The hierarchical data structure of the `SelectQuery` data contract can be conveniently viewed in JSON format:

Structure of the `SelectQuery` data contract

```
{
    "RootSchemaName":"[Object root schema name]",
    "OperationType":[Type of record operation],
    "Columns":{
        "Items":{
            "Name":{
                "OrderDirection":[Sorting order],
                "OrderPosition":[Column position],
                "Caption":"[Title]",
                "Expression":{
                    "ExpressionType":[Expression type],
                    "ColumnPath":"[Path to column]",
                    "Parameter":[Parameter],
                    "FunctionType":[Function type],
                    "MacrosType":[Macro type],
                    "FunctionArgument":[Function argument],
                    "DatePartType":[Type of date part],
                    "AggregationType":[Aggregation type],
                    "AggregationEvalType":[Aggregation scope],
                    "SubFilters":[Buit-in filters]
                }
            }
        }
    },
    "AllColumns":[Indicates that all columns are selected],
    "ServerESQCacheParameters":{
        "CacheLevel":[Caching level],
        "CacheGroup":[Caching group],
        "CacheItemName":[Record key in repository]
    },
```

```
    "IsPageable":[Indicates page-by-page],
    "IsDistinct":[Indicates uniqueness],
    "RowCount":[Number of selected records],
    "ConditionalValues":[Conditions for building a pageable query],
    "IsHierarchical":[Indicates hierarchical data selection],
    "HierarchicalMaxDepth":[Maximum nesting level of the hierarchical query],
    "HierarchicalColumnName":[Column name used to create hierarchical query],
    "HierarchicalColumnValue":[Initial value of hierarchical column],
    "Filters":[Filters]
    }
}
```

Properties

`RootSchemaName` `string`

String that contains root schema name of the added record object.

`OperationType` `QueryOperationType`

Type of write operation. Specified as a `QueryOperationType` enumeration value of the `Terrasoft.Nui.ServiceModel.DataContract` namespace. The `QueryOperationType.Select` value is set for `SelectQuery`.

Possible values (`QueryOperationType`)

Select	0
Insert	1
Update	2
Delete	3
Batch	4

`Columns` `SelectQueryColumns`

Contains a collection of the record columns being read. It has the `SelectQueryColumns` type defined in the `Terrasoft.Nui.ServiceModel.DataContract` namespace. It must be configured if the `AllColumns` checkbox is set to `false` and a set of specific root schema columns, which does not include the [*Id*] column, is required.

`AllColumns` `bool`

Indicates if all columns are selected. If the value is set to `true`, all columns of the root schema will be selected by the query.

`ServerESQCache` `Parameters`

Parameters of `EntitySchemaQuery` caching on server. The `ServerESQCacheParameters` type is defined in the `Terrasoft.Nui.ServiceModel.DataContract` namespace.

`IsPageable` `bool`

Indicates whether the data is selected page-by-page.<

`IsDistinct` `bool`

Indicates whether duplicates must be eliminated in the resulting data set.

`RowCount` `int`

Number of selected strings. By default, the value is -1, i.e. all strings are selected.

`ConditionalValues` `ColumnValues`

Conditions of creating a page-by-page query. The `ColumnValues` type is defined in the `Terrasoft.Nui.ServiceModel.DataContract` namespace.

`IsHierarchical` `bool`

Indicates whether the data is selected hierarchically.

`HierarchicalMaxDepth` `int`

Maximum nesting level of a hierarchical query.

`hierarchicalColumnName` `string`

Name of the column used for creating a hierarchical query.

`hierarchicalColumnValue` `string`

Initial value of hierarchical column from which the hierarchy will be built.

`Filters` `Filters`

Collection of query filters. The `Filters` type is defined in the `Terrasoft.Nui.ServiceModel.DataContract` namespace.

`ColumnValues` `ColumnValues`

Contains collection of column values for the added record. The `ColumnValues` type is defined in the `Terrasoft.Nui.ServiceModel.DataContract` namespace.

The SelectQueryColumns class C#

The Namespace `Terrasoft.Nui.ServiceModel.DataContract`.

The `SelectQueryColumns` class has a single `Items` property, defined as a collection of keys and values `Dictionary<string, SelectQueryColumn>`. The key is the string with the name of the added column. The value is an instance of the `SelectQueryColumn` class, defined in the `Terrasoft.Nui.ServiceModel.DataContract` namespace. The properties of the `SelectQueryColumn` are available in the table.

Properties

`OrderDirection` `OrderDirection`

Sorting order. Specified with a value from the `OrderDirection` enumeration of the `Terrasoft.Common` name space defined in the `Terrasoft.Common` class library.

`OrderPosition` `int`

Sets position number in the collection of the query columns, by which the sorting is done.

`Caption` `string`

Column title.

`Expression` `ColumnExpression`

Property that defines expression of the type of selected column.

The ColumnExpression class C#

The Namespace `Terrasoft.Nui.ServiceModel.DataContract`.

The `ColumnExpression` class defines expression that sets the type of the schema column. The class is defined in the `Terrasoft.Nui.ServiceModel.DataContract` namespace of the `Terrasoft.Nui.ServiceModel` library. The properties of an instance of this class are filled in depending on the `ExpressionType` property, which sets the expression type. The full list of the `ColumnExpression` class properties is available in table.

Properties

`ExpressionType` `EntitySchemaQuery ExpressionType`

Type of expression that determines the value that the added column will contain. Specified with a value from the `ntitySchemaQueryExpressionType` enumeration of the `Terrasoft.Core.Entities` namespace defined in the `Terrasoft.Core` class library. The `EntitySchemaQueryExpressionType.Parameter` value is set for `InsertQuery`.

Possible values (`EntitySchemaQueryExpressionType`)

SchemaColumn	0	Schema column.
Function	1	Function.
Parameter	2	Parameter.
SubQuery	3	Subquery.
ArithmeticOperation	4	Arithmetic operation.

`ColumnPath` `string`

Path to the column in relation to the root schema. Rules for building paths are available in the Build path to columns article.

`Parameter` `Parameter`

Determines the value that the added column will contain. The `Parameter` type is defined in the `Terrasoft.Nui.ServiceModel.DataContract` namespace.

`FunctionType` `FunctionType`

Function type. Specified with a value from the `FunctionType` enumeration, which is defined in the `Terrasoft.Nui.ServiceModel.DataContract` namespace.

Possible values (`FunctionType`)

None	0	Not defined.
Macros	1	Macro.
Aggregation	2	Aggregate function.
DatePart	3	Part of date value.
Length	4	Length.

`MacrosType` `EntitySchemaQuery MacrosType`

Macro type. Specified with a value of the `EntitySchemaQueryMacrosType` enumeration, which is defined in the `Terrasoft.Core.Entities` namespace.

`FunctionArgument` `BaseExpression`

Function argument. Accepts a value if the function is defined with a parameter. The `BaseExpression` class is defined in the `Terrasoft.Nui.ServiceModel.DataContract` namespace, is an ancestor for the `ColumnExpresion` class and has the same set of properties.

`DatePartType` `DatePart`

Part of date value Specified with a value from the `DatePart` enumeration, which is defined in the `Terrasoft.Nui.ServiceModel.DataContract` namespace.

Possible values (`DatePart`)

None	0	Not defined.
Day	1	Day.
Week	2	Week.
Month	3	Month
Year	4	Year.
Weekday	5	Week day.
Hour	6	Hour.
HourMinute	7	Minute.

`AggregationType` `AggregationType`

Aggregate function type. Specified with a value from the `AggregationType` enumeration defined in the `Terrasoft.Common` namespace defined in the `Terrasoft.Common` class library.

`AggregationEvalType` `AggregationEvalType`

Aggregate function scope. Specified with a value from the `AggregationEvalType` enumeration defined in the `Terrasoft.Common` namespace defined in the `Terrasoft.Common` class library.

`SubFilters` `Filters`

Collection of subquery filters. The `Filters` type is defined in the `Terrasoft.Nui.ServiceModel.DataContract` namespace.

The Parameter class C#

The Namespace `Terrasoft.Nui.ServiceModel.DataContract`.

The `Parameter` class is defined in the `Terrasoft.Nui.ServiceModel.DataContract` namespace. Its properties are available in table.

Properties

`DataValueType` `DataValueType`

Type of data for the value that the added column will contain. Specified as a `DataValueType` enumeration value of the `Terrasoft.Nui.ServiceModel.DataContract` namespace.

Possible values (`DataValueType`)

Guid	0
Text	1
Integer	4
Float	5
Money	6
DateTime	7
Date	8
Time	9
Lookup	10
Enum	11
Boolean	12
Blob	13
Image	14
ImageLookup	16
Mapping	18

`Value` `object`

The object that contains the value of the added column.

`ArrayValue` `string[]`

Array of the added column values. Used when serializing arrays and BLOBs.

`ShouldSkipConvertion` `bool`

Indicates the need to skip the process of providing the type for the `Value` property.

The ServerESQCacheParameters class C#

The Namespace `Terrasoft.Nui.ServiceModel.DataContract`.

The `ServerESQCacheParameters` class is defined in the `Terrasoft.Nui.ServiceModel.DataContract` namespace. Its properties are available in table.

Properties

`CacheLevel` `int`

Data allocation level in the `EntitySchemaQuery` cache.

`CacheGroup` `string`

Caching group.

`CacheItemName` `string`

Repository record key.

The `Filters` class is defined in the `Terrasoft.Nui.ServiceModel.DataContract` namespace. For details on the properties of this class and its use, please see the Filters class article.

Filters class C#

Advanced

During the execution of DataService operations, it is often necessary to filter data. For example, when reading section records, you need to fetch only those records that meet certain criteria. Creatio provides the `Filters` class to form these criteria.

The `Filters` class C#

The Namespace `Terrasoft.Nui.ServiceModel.DataContract`.

The `Filters` class is defined in the `Terrasoft.Nui.ServiceModel.DataContract` namespace of the `Terrasoft.Nui.ServiceModel.dll` class library. For simplicity, the hierarchical structure of the `Filters` data filter is conveniently presented as a JSON format object:

Structure of the `Filters` data filter

```
"Filters":{
    "RootSchemaName":["Root schema name"],
    "FilterType":[Filter type],
    "ComparisonType":[Comparison type],
    "LogicalOperation":[Logical operation],
    "IsNull":[Completeness checkbox],
```

```
    "IsEnabled":[Activation checkbox],
    "IsNot":[Negation operator checkbox],
    "SubFilters":[Subquery filters],
    "Items":[Filter group collection],
    "LeftExpression":[Expression to be checked],
    "RightExpression":[Filtration expression],
    "RightExpressions":[Filtration expressions array],
    "RightLessExpression":[Initial filtration range expression],
    "RightGreaterExpression":[Final filtration range expression],
    "TrimDateTimeParameterToDate":[Cutting time for date/time parameters checkbox],
    "Key":["Filter key in the filter collection"],
    "IsAggregative":[Aggregating filter checkbox],
    "LeftExpressionCaption":["Expression title to be checked"],
    "ReferenceSchemaName":["Reference schema name"]
}
```

Properties

`RootSchemaName` `string`

A string containing the name of the root object schema of the added record.

`FilterType` `FilterType`

Filter type. Set by the `FilterType` enumeration value of the `Terrasoft.Nui.ServiceModel.DataContract` namespace.

Possible values (`FilterType`)

`None`	0	Filter type not defined.
`CompareFilter`	1	Comparison filter. Used to compare expression results.
`IsNullFilter`	2	The filter that defines whether an expression is empty.
`Between`	3	The filter that defines whether an expression is one of the expressions.
`InFilter`	4	The filter that defines whether an expression equals one of the expressions.
`Exists`	5	Existence filter.
`FilterGroup`	6	Filter group. Filter groups can be nested in one another, i.e., the collection itself can be an element of another collection.

`ComparisonType` `FilterComparisonType`

Comparison operation type. Set by the `FilterComparisonType` enumeration value of the `Terrasoft.Core.Entities` namespace.

`LogicalOperation` `LogicalOperationStrict`

Logical operation. This type does not allow the `None` value specified in the `LogicalOperationStrict` enumeration of the `Terrasoft.Common` namespace.

`IsNull` `bool`

Expression completion checkbox.

`IsEnabled` `bool`

Checkbox that defines whether the filter is active and will be taken into account when building a request.

`IsNot` `bool`

Specifies whether to use the negation logical operator.

`SubFilters` `Filters`

Subrequest filters. Cannot contain filters with other subrequests.

`Items` `Dictionary<string, Filter>`

Collection containing a filter group.

`LeftExpression` `BaseExpression`

The expression in the left part of the comparison, i.e. the expression to be tested. The `BaseExpression` class is defined in the `Terrasoft.Nui.ServiceModel.DataContract` namespace.

`RightExpression` `BaseExpression`

The filter expression that will be compared to the expression contained in the `LeftExpression` property.

`RightExpressions` `BaseExpression[ ]`

The expression array that will be compared to the expression contained in the `LeftExpression` property.

`RightLessExpression` `BaseExpression`

Initial filtration range expression.

`RightGreaterExpression` `BaseExpression`

Final filtration range expression.

`TrimDateTime ParameterToDate` `bool`

Checkbox indicating whether to cut time from the date-time parameters.

`Key` `string`

Filter key in the collection of `Items` filters.

`IsAggregative` `bool`

Aggregating filter checkbox.

`LeftExpressionCaption` `string`

Left comparison part title.

`ReferenceSchemaName` `string`

The object schema name referenced by the left part of the filter if the column type is lookup.

The BaseExpression class C#

The Namespace `Terrasoft.Nui.ServiceModel.DataContract`.

The `BaseExpression` class is the base expression class. It is defined in the `Terrasoft.Nui.ServiceModel.DataContract` namespace of the `Terrasoft.Nui.ServiceModel` library. The properties of this class instance are populated depending on the `ExpressionType` property that specifies the expression type. A complete list of the `BaseExpression` class properties is given in table.

Properties

`ExpressionType` `EntitySchemaQuery ExpressionType`

The expression type that defines the value that will be contained in the added column. Set by the `EntitySchemaQueryExpressionType` enumeration of the `Terrasoft.Core.Entities` namespace defined in the

`Terrasoft.Core` class library. For the `InsertQuery` the `EntitySchemaQueryExpressionType.Parameter` value is set.

Possible values (`EntitySchemaQuery ExpressionType`)

SchemaColumn	0	Schema column.
Function	1	Function.
Parameter	2	Parameter.
SubQuery	3	Subquery.
ArithmeticOperation	4	Arithmetic operation.

`ColumnPath` `string`

The path to a column relative to the root schema. The rules for building the paths can be found in the "The use of EntitySchemaQuery for creation of queries in database" article.

`Parameter` `Parameter`

Defines the value that will be contained in the added column. Its `Parameter` type is defined in the `Terrasoft.Nui.ServiceModel.DataContract` namespace.

`FunctionType` `FunctionType`

Function type. Set by the value from the `FunctionType` enumeration defined in the `Terrasoft.Nui.ServiceModel.DataContract` namespace.

Possible values (`FunctionType`)

None	0	Not defined.
Macros	1	Macro.
Aggregation	2	Aggregating function.
DatePart	3	Date part.
Length	4	Length.

`MacrosType` `EntitySchemaQuery MacrosType`

Macro type. Set by the value from the `EntitySchemaQueryMacrosType` enumeration defined in the `Terrasoft.Core.Entities` namespace.

`FunctionArgument` `BaseExpression`

Function argument. Takes the value if the function is defined with a parameter. The `BaseExpression` class is defined in the `Terrasoft.Nui.ServiceModel.DataContract` namespace and is the ancestor of the `ColumnExpresion` class and has the same set of properties.

`DatePartType` `DatePart`

Date part. Set by the value from the `DatePart` enumeration defined in the `Terrasoft.Nui.ServiceModel.DataContract` namespace.

Possible values (`DatePart`)

None	0	Not defined.
Day	1	Day.
Week	2	Week.
Month	3	Month.
Year	4	Year.
Weekday	5	Day of the week.
Hour	6	Hour.
HourMinute	7	Minute.

`AggregationType` `AggregationType`

Aggregating function type. Sets the value of `AggregationType` enumeration defined in the namespace `Terrasoft.Common` defined in the class library `Terrasoft.Common`.

`AggregationEvalType` `AggregationEvalType`

Aggregating function Set by the value from the `AggregationEvalType` enumeration defined in the `Terrasoft.Core.DB` namespace defined in the `Terrasoft.Core` class library.

`SubFilters` `Filters`

Subquery filter collection. Its `Filters` type is defined in the `Terrasoft.Nui.ServiceModel.DataContract` namespace.

EntitySchemaQueryMacrosType enumeration C#

Advanced

When creating queries to DataService, both parameterized (ie requiring an argument) and non-parameterized macros can be used. Macro types that must be used in the macro expressions are defined in the `EntitySchemaQueryMacrosType` enumeration in the `Terrasoft.Core.Entities` namespace. Enumeration values of macro types and their descriptions are available in table.

Possible values (`EntitySchemaQueryMacrosType`)

`CurrentHalfYear`	16	Current half-year (January-June or July-December).
CurrentHour	21	Current hour.
CurrentMonth	10	Current month.
CurrentQuarter	13	Current quarter.
CurrentUser	1	Current user.
CurrentUserContact	2	Contact record of the current user.
CurrentWeek	7	Current week.
CurrentYear	19	Current year.
DayOfMonth	28	Day of month. Requires parameterization.
DayOfWeek	29	Week day. Requires parameterization.
Hour	30	Hour. Requires parameterization.
HourMinute	31	Time. Requires parameterization.
Month	32	Month. Requires parameterization.
NextHalfYear	17	Next half-year (January-June or July-December).
NextHour	22	Next hour.
NextMonth	11	Next month.

NextNDays	24	Next N days. Requires parameterization.
NextNHours	26	Next N hours. Requires parameterization.
NextQuarter	14	Next quarter.
NextWeek	8	Next week.
NextYear	23	Next year.
None	0	Type of macro not defined.
PreviousHalfYear	15	Previous half-year (January-June or July-December).
PreviousHour	20	Previous hour.
PreviousMonth	9	Previous month.
PreviousNDays	25	Previous N days. Requires parameterization.
PreviousNHours	27	Previous N hours. Requires parameterization.
PreviousQuarter	12	Previous quarter.
PreviousWeek	6	Previous week.
PreviousYear	18	Previous year.
Today	4	Today.
Tomorrow	5	Tomorrow.
Year	33	Year. Requires parameterization.
Yesterday	3	Yesterday.

UpdateQuery class C#

Advanced

The Namespace `Terrasoft.Nui.ServiceModel.DataContract`.

The `UpdateQuery` data contract is used for updating section records. The query data is transferred to DataService via HTTP, with the help of `POST` by the URL.

The structure of request for updating data

```
// URL format of the POST query to DataService to update data.
http(s):///[Creatio application address]/[Configuration number]/dataservice/[Data fromat]/reply/U
// URL example of the POST query to DataService to update data.
```

An example the request for updating data

```
http(s)://example.creatio.com/0/dataservice/json/reply/UpdateQuery
```

The `UpdateQuery` data contract has a hierarchical structure with a number of nesting levels. In the Creatio server core, it is represented by a UpdateQuery class of the `Terrasoft.Nui.ServiceModel.DataContract` namespace of the `Terrasoft.Nui.ServiceModel.dll` library of classes. For the hierarchical data structure of the `UpdateQuery` data contract can be conveniently viewed in JSON format:

Structure of the `UpdateQuery` data contract

```
{
    "RootSchemaName":"[Root schema]",
    "OperationType":[Type of operation with record],
    "IsForceUpdate":[Force update],
    "ColumnValues":{
        "Items":{
            "Name of the added column":{
                "ExpressionType":[Expression type],
                "Parameter":{
                    "DataValueType":[Data type],
                    "Value":"[Column value]"
                }
            }...
        }
    },
    "Filters":[Request filters]
}
```

Properties

`RootSchemaName` `string`

String that contains root schema name of added record object.

`OperationType` `QueryOperationType`

Type of write operation. Specified as a `QueryOperationType` enumeration value of the `Terrasoft.Nui.ServiceModel.DataContract` namespace. The `QueryOperationType.Select` value is set for `SelectQuery`.

Possible values (`QueryOperationType`)

Select	0
Insert	1
Update	2
Delete	3
Batch	4

`IsForceUpdate` `bool`

Indicates force update. If the value is `true`, the entity will be saved on the server even if column values have been modified. Default value: `false`.

`ColumnValues` `ColumnValues`

Contains collection of column values for the added record. The `ColumnValues` type is defined in the `Terrasoft.Nui.ServiceModel.DataContract` namespace.

`Filters` `Filters`

Collection of query filters. The < `Filters` type is defined in the `Terrasoft.Nui.ServiceModel.DataContract` namespace.

The ColumnValues class C#

The Namespace `Terrasoft.Nui.ServiceModel.DataContract`.

The `ColumnValues` class has a single `Items` property, defined as a collection of keys and values `Dictionary<string, ColumnExpression>`. The key is the string with the name of the added column. The value is an object of the `ColumnExpression` type, defined in the `Terrasoft.Nui.ServiceModel.DataContract` namespace. General properties of the `ColumnExpression` class used when adding records are available in table.

Properties

`ExpressionType` `EntitySchemaQuery ExpressionType`

Type of expression that determines the value that the added column will contain. Specified with a value from the `EntitySchemaQueryExpressionType` enumeration of the `Terrasoft.Core.Entities` namespace defined in the `Terrasoft.Core` class library. The `EntitySchemaQueryExpressionType.Parameter` value is set for `InsertQuery`.

Possible values (`EntitySchemaQueryExpressionType`)

SchemaColumn	0	Schema column.
Function	1	Function.
Parameter	2	Parameter.
SubQuery	3	Subquery.
ArithmeticOperation	4	Arithmetic operation.

`Parameter` `Parameter`

Determines the value that the added column will contain. The `Parameter` type is defined in the `Terrasoft.Nui.ServiceModel.DataContract` namespace.

The Parameter class C#

The Namespace `Terrasoft.Nui.ServiceModel.DataContract`.

The `Parameter` class has a number of properties, only two of which are used for adding records.

Properties

`DataValueType` `DataValueType`

Type of data for the value that the added column will contain. Specified as a `DataValueType` enumeration value of the `Terrasoft.Nui.ServiceModel.DataContract` namespace.

Possible values (`DataValueType`)

Guid	0
Text	1
Integer	4
Float	5
Money	6
DateTime	7
Date	8
Time	9
Lookup	10
Enum	11
Boolean	12
Blob	13
Image	14
ImageLookup	16
Mapping	18

`Value` `object`

The object that contains the value of the added column. Has the `Object` type.

The Filters class C#

The Namespace `Terrasoft.Nui.ServiceModel.DataContract`.

An instance of the `UpdateQuery` class must contain a link to a correctly initialized instance of the `Filters` class in the `Filters` property. Otherwise, new column values from the `ColumnValues` property will be set for ALL section records.

DeleteQuery class C#

Advanced

The Namespace `Terrasoft.Nui.ServiceModel.DataContract`.

The `DeleteQuery` contract is used to delete sections. The data is transferred to the DataService via HTTP by using the `POST` request with the following URL:

The structure of request for deleting data

```
// URL format of the POST query to DataService to delete data.
http(s):///[Creatio application address]/[Configuration number]/dataservice/[Data fromat]/reply/D
```

An example the request for deleting data

```
// URL example of the POST query to DataService to delete data.
http(s)://example.creatio.com/0/dataservice/json/reply/DeleteQuery
```

The `DeleteQuery` data contract has a hierarchical structure with multiple nesting levels. In the Creatio application server part, the `DeleteQuery` data contract is represented by the `DeleteQuery` class of the `Terrasoft.Nui.ServiceModel.DataContract` namespace of the `Terrasoft.Nui.ServiceModel.dll` class library. However, for simplicity, the hierarchical structure of the `DeleteQuery` data contract is conveniently presented as a JSON format object:

Structure of the `DeleteQuery` data contract

```
{
    "RootSchemaName":"[Root schema]",
    "OperationType":[Record operation type],
    "ColumnValues":[Column values. Not used.],
    "Filters":[Query filters]
}
```

Properties

`RootSchemaName` `string`

A string containing the name of the root object schema of the added record.

`OperationType` `QueryOperationType`

Operation type is set by the `QueryOperationType` namespace `Terrasoft.Nui.ServiceModel.DataContract` namespace enumeration value. For the `SelectQuery` the `QueryOperationType.Select` value is set.

Possible values (`QueryOperationType`)

Select	0
Insert	1
Update	2
Delete	3
Batch	4

`ColumnValues` `ColumnValues`

Contains a collection of column values of the added record. Inherited from the `BaseQuery` parent class. Not used in this type of queries.

`Filters` `Filters`

Query filter collection. Its `Filters` type is defined in the `Terrasoft.Nui.ServiceModel.DataContract` namespace.

The Filters class C#

The Namespace `Terrasoft.Nui.ServiceModel.DataContract`.

The `DeleteQuery` query class instance must contain a link to the correctly initialized `Filters` class instance in the `Filters` property. Otherwise ALL section records will be deleted.

BatchQuery class C#

Advanced

Batch queries are used to minimize requests to DataServise, which improves application performance. Packet query is a collection that contains a custom set of DataService requests. The query data is transferred to DataService via HTTP, with the help of `POST` by the URL.

The structure of batch query

```
// URL format of the batch POST query to DataService.
http(s):///[Creatio application address]/[Configuration number]/dataservice/[Data fromat]/reply/B
```

An example the batch query

```
// URL example of the batch POST query to DataService.
```

```
http(s)://example.creatio.com/0/dataservice/json/reply/BatchQuery
```

The data that comprises a batch query can be passed in different formats. One of the more convenient formats is JSON.

The structure of a batch query in JSON format

```
{
    "items": [
        {
            "__type": "[Full qualified name of the query type]",
            //One-time query contents.
            ...
        },
        // Other one-time queries.
        ...
    ]
}
```

To generate the contents of one-time queries that comprise a batch query, use the following data constants: `InsertQuery`, `SelectQuery`, `UpdateQuery` and `DeleteQuery`.