Creatio Academy

Data operations (front-end)

Data-operations (front-end)

Version 8.0

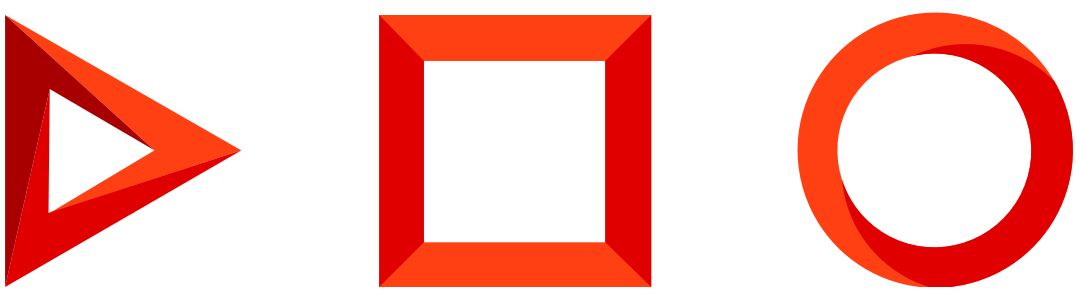

Table of Contents

Data-operations (front-end)

Advanced

Front-end Creatio modules implement data management using the high-level `EntitySchemaQuery` class that builds database selection queries.

The **major features** of the `EntitySchemaQuery` class:

- Data selection queries created using `EntitySchemaQuery` apply the access permissions of the current user.
- The caching mechanism lets you optimize operations by accessing cached query results without direct access to the database.

The data management **procedure** for front-end Creatio modules is as follows:

1. Create an instance of the `EntitySchemaQuery` class.
2. Specify the root schema.
3. Configure a path to the root schema column to add the column to the query.
4. Create filter instances.
5. Add the filters to the query.
6. Filter the query results.

Configure the column paths relative to the root schema

Base an `EntitySchemaQuery` query on the root schema. The **root schema** is the database table relative to which to build paths to the query columns, including the columns of the joined tables. To use a table column in a query, set the path to the column correctly.

Specify the column path using direct connections

The template for configuring the column path using the **direct connections** of `LookupColumnName.LookupSchema'sColumnSchemaName`.

For example, a `[City]` root schema contains a `[Country]` lookup column. The column is connected to the `[Country]` lookup schema via the `Id` column.

The path to the column that contains the name of the country connected to the city, built using the direct connections `Country.Name`. Where:

- `Country` is the name of the lookup column in the `[City]` root schema. Links to the `[Country]` schema.

- `Name` is the name of the column in the `[Country]` lookup schema.

Specify the column path using the reverse connections

The template for configuring the column path using the **reverse connections**

```
[JoinableSchemaName:NameOfTheColumnToLinkTheJoinableSchema:NameOfTheColumnToLinkTheCurrentSchema].
JoinableSchema'sColumnName
```

.

The path to the column that contains the name of the contact who added the city, built using the reverse connections `[Contact:Id:CreatedBy].Name`. Where:

- `Contact` is the name of the joinable schema.
- `Id` is the name of the `[Contact]` schema's column to connect the joinable schema.
- `CreatedBy` is the name of the `[City]` schema's lookup column to connect the current schema.
- `Name` is the value of the `[City]` schema's lookup column.

If the schema's lookup column to connect the current schema is `[Id]`, you do not have to specify it: `[JoinableSchemaName:NameOfTheColumnToConnectTheJoinableSchema].RootSchema'sColumnName`. For example, `[Contact:City].Name`.

Add the columns to the query

The `EntitySchemaQuery` query column is a `Terrasoft.EntityQueryColumn` class instance. Specify the main **features** in the column instance properties:

- header
- value for display
- usage flags
- sorting order and position

Add columns to the query using the `addColumn()` method that returns the query column instance. The `addColumn()` methods configure the name of the column relative to the root schema according to the rules described above: Configure the column paths relative to the root schema. The variations of the `addColumn()` method let you add query columns that contain different parameters. See the variations in the table below.

Variations of the `addColumn()` method

Column type	Method
The column by the specified path relative to the root schema	`addColumn(column, columnAlias)`
The instance of the query column class	
The parameter column	`addParameterColumn(paramValue, paramDataType, columnAlias)`
The function column	`addFunctionColumn(columnPath, functionType, columnAlias)`
The aggregative function column	`addAggregationSchemaColumnFunctionColumn(columnPath, aggregationType, columnAlias)`

Retrieve the query results

The **result** of the `EntitySchemaQuery` query is a collection of Creatio entities. Each collection instance is a dataset string returned by the query.

The **ways to retrieve the query results** are as follows:

- Call the `getEntity()` method to retrieve a particular dataset string by the specified primary key.
- Call the `getEntityCollection()` method to retrieve the entire resulting dataset.

Manage the query filters

Filters are sets of conditions applied when displaying the query data. In SQL terms, a filter is a separate predicate (condition) of the `WHERE` operator.

To create a simple filter in `EntitySchemaQuery`, use the `createFilter()` method that returns the created object of the `Terrasoft.CompareFilter` filter. `EntitySchemaQuery` implements methods that create special kinds of filters.

The `EntitySchemaQuery` instance contains the `filters` property that represents the filter collection of the query (the `Terrasoft.FilterGroup` class instance). The `Terrasoft.FilterGroup` class instance is a collection of `Terrasoft.BaseFilter` elements.

Follow this procedure to add a filter to the query:

- Create a filter instance for the query (the `createFilter()` method, methods that create special kinds of filters).
- Add the filter instance to the query filter collection (the `add()` collection method).

By default, Creatio uses the logical `AND` operation to combine the filters added to the `filters` collection. The `logicalOperation` property of the `filters` collection lets you specify the logical operation for combining filters. The `logicalOperation` accepts the values of the `Terrasoft.core.enums.LogicalOperatorType` enumeration (`AND`, `OR`).

`EntitySchemaQuery` queries let you manage the filters involved in the creation of the resulting dataset. Each element of the `filters` collection includes the `isEnabled` property that determines whether the element takes part in building the resulting query (`true` / `false`). Creatio defines the similar `isEnabled` property for the `filters` collection. If you set this property to `false`, the query filtering will be disabled, yet the query filters collection will remain unchanged. Thus, you can create a query filters collection and use various combinations without modifying the collection directly.

Configure the column paths in the `EntitySchemaQuery` filters according to the general rules for configuring the paths to columns relative to the root schema.

Examples that configure the column paths

Medium

Column path relative to the root schema

- The root schema: `[Contact]`.
- The column that contains the contact address: `Address`.

Example that creates an `EntitySchemaQuery` query to return the values of this column

```
/* Create an instance of the EntitySchemaQuery class with the Contact root schema. */
var esq = this.Ext.create("Terrasoft.EntitySchemaQuery", {
    rootSchemaName: "Contact"
});
/* Add an Address column, set its alias to Address. */
esq.addColumn("Address", "Address");
```

Column path that uses the direct connections

- The root schema: `[Contact]`.
- The column that contains the account name: `Account.Name`.
- The column that contains the name of the account's primary contact: `Account.PrimaryContact.Name`.

Example that creates an `EntitySchemaQuery` query to return the values of these columns

```
/* Create an instance of the EntitySchemaQuery class with the Contact root schema. */
var esq = this.Ext.create("Terrasoft.EntitySchemaQuery", {
    rootSchemaName: "Contact"
});
/* Add an Account lookup column. Then add the Name column from the Account schema linked to the
esq.addColumn("Account.Name", "AccountName");
/* Add an Account lookup column. Then add the PrimaryContact lookup column from the Account sche
esq.addColumn("Account.PrimaryContact.Name", "PrimaryContactName");
```

Column path that uses the reverse connections

- The root schema: `[Contact]`.
- The column that contains the name of the contact who added the city: `[Contact:Id:CreatedBy].Name`.

Example that creates an `EntitySchemaQuery` query to return the values of this column

```
/* Create an EntitySchemaQuery class instance with the Contact root schema. */
var esq = this.Ext.create("Terrasoft.EntitySchemaQuery", {
    rootSchemaName: "Contact"
});
/* Join another Contact schema to the root schema by the Owner column and select the Name columr
esq.addColumn("[Contact:Id:Owner].Name", "OwnerName");
/* Join the Contact schema to the Account lookup column by the PrimaryContact column and select
esq.addColumn("Account.[Contact:Id:PrimaryContact].Name", "PrimaryContactName");
```

Examples that add columns to the query

Medium

Column from the root schema

Example. Add the column from the root schema to the query column collection.

Example that adds the column from the root schema to the query column collection

```
var esq = this.Ext.create(Terrasoft.EntitySchemaQuery, {
    rootSchemaName: "Activity"
});
esq.addColumn("DurationInMinutes", "ActivityDuration");
```

Aggregate column

Example 1. Add the aggregate column to the query column collection. The column must have the `SUM` aggregation type that applies to all table records.

Example that adds the aggregate column to the query column collection

```
var esq = this.Ext.create(Terrasoft.EntitySchemaQuery, {
    rootSchemaName: "Activity"
});
esq.addAggregationSchemaColumn("DurationInMinutes", Terrasoft.AggregationType.SUM, "ActivitiesDu
```

Example 2. Add the aggregate column to the query column collection. The column must have the `COUNT` aggregation type that applies to unique table records.

Example that adds the aggregate column to the query column collection

```
var esq = this.Ext.create(Terrasoft.EntitySchemaQuery, {
    rootSchemaName: "Activity"
});
esq.addAggregationSchemaColumn("DurationInMinutes", Terrasoft.AggregationType.COUNT, "UniqueActi
```

Parameter column

Example. Add the parameter column with `TEXT` data type to the query column collection.

Example that adds the parameter column to the query column collection

```
var esq = this.Ext.create(Terrasoft.EntitySchemaQuery, {
    rootSchemaName: "Activity"
});
esq.addParameterColumn("DurationInMinutes", Terrasoft.DataValueType.TEXT, "DurationColumnName");
```

Function column

Example 1. Add the function column with `LENGTH` (value size, in bytes) data type to the query column collection.

Example that adds the function column to the query column collection

```
var esq = this.Ext.create(Terrasoft.EntitySchemaQuery, {
```

```
    rootSchemaName: "Activity"
});
esq.addFunctionColumn("Photo.Data", Terrasoft.FunctionType.LENGTH, "PhotoLength");
```

Example 2. Add the function column with `DATE_PART` (date part) data type to the query column collection. Use the day of the week as the value.

Example that adds the function column to the query column collection.

```
var esq = this.Ext.create(Terrasoft.EntitySchemaQuery, {
    rootSchemaName: "Activity"
});
esq.addDatePartFunctionColumn("StartDate", Terrasoft.DatePartType.WEEK_DAY, "StartDay");
```

Example 3. Add the function column to the query column collection. The column must have the `MACROS` type that does not need to be parameterized: `PRIMARY_DISPLAY_COLUMN` (primary column for display).

Example that adds the function column to the query column collection

```
var esq = this.Ext.create(Terrasoft.EntitySchemaQuery, {
    rootSchemaName: "Activity"
});
esq.addMacrosColumn(Terrasoft.QueryMacrosType.PRIMARY_DISPLAY_COLUMN, "PrimaryDisplayColumnValue
```

Examples that retrieve the query results

Medium

Dataset string by the specified primary key

Example. Retrieve a particular dataset string by the specified primary key

Example that retrieves a particular dataset string

```
/* Retrieve the ID of the mini page object. */
var recordId = this.get("Id");
/* Create an instance of the Terrasoft.EntitySchemaQuery class with the Contact root schema. */
var esq = this.Ext.create("Terrasoft.EntitySchemaQuery", {
```

```
    rootSchemaName: "Contact"
});
/* Add a column that contains the name of the account’s primary contact. */
esq.addColumn("Account.PrimaryContact.Name", "PrimaryContactName");
/* Retrieve one record from the selection by the ID of the mini page object. Display the record
esq.getEntity(recordId, function(result) {
    if (!result.success) {
        // For example, error processing/logging.
        this.showInformationDialog("Data query error");
        return;
    }
    this.showInformationDialog(result.entity.get("PrimaryContactName"));
}, this);
```

Note. If you retrieve lookup columns, the `this.get()` returns the object, not ID of the record in the database. To retrieve the ID, use the `value` property. For example, `this.get('Account').value`.

Resulting dataset

Example. Retrieve the entire resulting dataset.

Example that retrieves the entire dataset

```
var message = "";
/* Create an instance of the Terrasoft.EntitySchemaQuery class with the Contact root schema. */
var esq = Ext.create("Terrasoft.EntitySchemaQuery", {
    rootSchemaName: "Contact"
});
/* Add a column that contains the name of the account connected to the contact. */
esq.addColumn("Account.Name", "AccountName");
/* Add a column that contains the name of the account’s primary contact. */
esq.addColumn("Account.PrimaryContact.Name", "PrimaryContactName");
/* Retrieve the entire record collection and display it in the information box. */
esq.getEntityCollection(function (result) {
    if (!result.success) {
        /* For example, error processing/logging. */
        this.showInformationDialog("Data query error");
        return;
    }
    result.collection.each(function (item) {
        message += "Account name: " + item.get("AccountName") +
        " - primary contact name: " + item.get("PrimaryContactName") + "\n";
    });
```

```
    this.showInformationDialog(message);
}, this);
```

Examples that manage the query filters

Medium

Example that manages the query filters

```
/* Create a query instance with the Contact root schema. */
var esq = Ext.create("Terrasoft.EntitySchemaQuery", {
    rootSchemaName: "Contact"
});
esq.addColumn("Name");
esq.addColumn("Country.Name", "CountryName");

/* Create a first filter instance. */
var esqFirstFilter = esq.createColumnFilterWithParameter(Terrasoft.ComparisonType.EQUAL, "Countr

/* Create a second filter instance. */
var esqSecondFilter = esq.createColumnFilterWithParameter(Terrasoft.ComparisonType.EQUAL, "Count

/* Combine the filters in the query filters collection using the OR logical operator. */
esq.filters.logicalOperation = Terrasoft.LogicalOperatorType.OR;

/* Add the filters to the query collection. */
esq.filters.add("esqFirstFilter", esqFirstFilter);
esq.filters.add("esqSecondFilter", esqSecondFilter);

/* Add the objects (query results) filtered by the two filters to the collection. */
esq.getEntityCollection(function (result) {
    if (result.success) {
        result.collection.each(function (item) {
            /* Process the collection elements. */
        });
    }
}, this);

/* Specify that the second filter is not used to build the resulting query. At the same time, do
esqSecondFilter.isEnabled = false;

/* Add the objects (query results) filtered only by the first filter to the collection. */
esq.getEntityCollection(function (result) {
    if (result.success) {
        result.collection.each(function (item) {
            /* Process the collection elements. */
```

```
        });
    }
}, this);
```

Example that uses other filter creation methods

```
/* Create a query instance with the Contact root schema. */
var esq = Ext.create("Terrasoft.EntitySchemaQuery", {
    rootSchemaName: "Contact"
});
esq.addColumn("Name");
esq.addColumn("Country.Name", "CountryName");

/* Select all contacts that do not have a country specified. */
var esqFirstFilter = esq.createColumnIsNullFilter("Country");

/* Select all contacts that have birth dates between 01.1.1970 and 01.1.1980. */
var dateFrom = new Date(1970, 0, 1, 0, 0, 0, 0);
var dateTo = new Date(1980, 0, 1, 0, 0, 0, 0);
var esqSecondFilter = esq.createColumnBetweenFilterWithParameters("BirthDate", dateFrom, dateTo)

/* Add the created filters to the query collection. */
esq.filters.add("esqFirstFilter", esqFirstFilter);
esq.filters.add("esqSecondFilter", esqSecondFilter);

/* Add the objects (query results) filtered by the two filters to the collection. */
esq.getEntityCollection(function (result) {
    if (result.success) {
        result.collection.each(function (item) {
            /* Process the collection elements. */
        });
    }
}, this);
```

EntitySchemaQuery class JS

Medium

The `EntitySchemaQuery` class builds queries to select records in a database.

Methods

`abortQuery()`

Aborts the query.

`addAggregationSchemaColumn(columnPath, aggregationType, [columnAlias], aggregationEvalType)`

Creates an instance of the `Terrasoft.FunctionQueryColumn` function column with the specified `AGGREGATION` type. Adds the instance to the query column collection.

Parameters

`{String} columnPath`	The path to the column to add, relative to `rootSchema`.

`{Terrasoft.AggregationType} aggregationType`	The aggregate function type.
	
	Available values (`Terrasoft.AggregationType`)
	
	`AVG`
	The average of all elements.
	
	`COUNT`
	The number of elements.
	
	`MAX`
	The maximal element.
	
	`MIN`
	The minimal element.
	
	`NONE`
	The aggregate function type is undefined.
	
	`SUM`
	The sum of all elements.

`{String} columnAlias`	The column alias. Optional.
`{Terrasoft.AggregationEvalType} aggregationEvalType`	The aggregate function scope. Available values (`Terrasoft.AggregationEvalType`)

	`NONE`
	The aggregate function scope is undefined.
	
	`ALL`
	Apply to all elements.
	
	`DISTINCT`
	Apply to unique values.

`addColumn(column, [columnAlias], [config])`

Creates an instance of the `Terrasoft.EntityQueryColumn` column. Adds the instance to the query column collection.

Parameters

`{String/Terrasoft.BaseQueryColumn} column`	The path to the column to add (relative to `rootSchema`) or an instance of the `Terrasoft.BaseQueryColumn` query column.
`{String} columnAlias`	The column alias. Optional.
`{Object} config`	The configuration object of the query column.

`addDatePartFunctionColumn(columnPath, datePartType, [columnAlias])`

Creates an instance of the `Terrasoft.FunctionQueryColumn` function column with the `DATE_PART` type. Adds the instance to the query column collection.

Parameters

{String} columnPath	The path to the column to add, relative to `rootSchema`.

{Terrasoft.DatePartType} datePartType	The date part to use as the value.
	
	Available values (`Terrasoft.DatePartType`)
	
	`NONE`
	Empty value.
	
	`DAY`
	Day.
	
	`WEEK`
	Week.
	
	`MONTH`
	Month.
	
	`YEAR`
	Year.
	
	`WEEK_DAY`
	Day of the week.
	
	`HOUR`
	Hour.
	
	`HOUR_MINUTE`
	Minute.

{String} columnAlias	The column alias. Optional.

```
addDatePeriodMacrosColumn(macrosType, [macrosValue], [columnAlias])
```

Creates an instance of the `Terrasoft.FunctionQueryColumn` function column with the `MACROS` type that must be parameterized. Adds the instance to the query column collection. For example, the following N days, the third quarter of the year, etc.

Parameters

`{Terrasoft.QueryMacrosType} macrosType`	The column macro type.
`{Number/Date} macrosValue`	The auxiliary variable for the macro. Optional.
`{String} columnAlias`	The column alias. Optional.

`addFunctionColumn(columnPath, functionType, [columnAlias])`

Creates an instance of the `Terrasoft.FunctionQueryColumn` function column. Adds the instance to the query column collection.

Parameters

`{String} columnPath`	The path to the column to add, relative to `rootSchema`.

`{Terrasoft.FunctionType} functionType`	The function type.
	
	Available values (`Terrasoft.FunctionType`)
	
	`NONE`
	The functional expression type is undefined.
	
	`MACROS`
	The insertion that uses a macro.
	
	`AGGREGATION`
	The aggregate function.
	
	`DATE_PART`
	The date part.
	
	`LENGTH`
	The value size, in bytes. Use with binary data.

`{String} columnAlias`	The column alias. Optional.

`addMacrosColumn(macrosType, [columnAlias])`

Creates an instance of the `Terrasoft.FunctionQueryColumn` function column with the `MACROS` type that does not need to be parameterized. For example, current month, current user, primary column, etc. Adds the instance to the query column collection.

Parameters

`{Terrasoft.QueryMacrosType} macrosType`	The column macro type. Available values (`Terrasoft.QueryMacrosType`) `NONE` The macro type is undefined.

`CURRENT_USER`

The current user.

`CURRENT_USER_CONTACT`

The current user contact.

`YESTERDAY`

Yesterday.

`TODAY`

Today.

`TOMORROW`

Tomorrow.

`PREVIOUS_WEEK`

Previous week.

`CURRENT_WEEK`

Current week.

`NEXT_WEEK`

Next week.

`PREVIOUS_MONTH`

Previous month.

`CURRENT_MONTH`

Current month.

`NEXT_MONTH`

Next month.

`PREVIOUS_QUARTER`

Previous quarter.

`CURRENT_QUARTER`

Current quarter.

`NEXT_QUARTER`

Next quarter.

`PREVIOUS_HALF_YEAR`

Previous 6 months.

`CURRENT_HALF_YEAR`

Current 6 months.

`NEXT_HALF_YEAR`

Next 6 months.

`PREVIOUS_YEAR`

Previous year.

`CURRENT_YEAR`

Current year.

`PREVIOUS_HOUR`

Previous hour.

`CURRENT_HOUR`

Current hour.

`NEXT_HOUR`

	NEXT_HOUR Next hour. NEXT_YEAR Next year. NEXT_N_DAYS Next N days. Must be parameterized. PREVIOUS_N_DAYS Previous N days. Must be parameterized. NEXT_N_HOURS Next N hours. Must be parameterized. PREVIOUS_N_HOURS Previous N hours. Must be parameterized. PRIMARY_COLUMN The primary column. PRIMARY_DISPLAY_COLUMN The primary column for display. PRIMARY_IMAGE_COLUMN The primary column for image display.
`{String} columnAlias`	The column alias. Optional.

`addParameterColumn(paramValue, paramDataType, [columnAlias])`

Creates an instance of the `Terrasoft.ParameterQueryColumn` parameter column. Adds the instance to the query column collection.

Parameters

`{Mixed} paramValue`	The parameter value. Must correspond to the data type.
`{Terrasoft.DataValueType} paramDataType`	The parameter data type.
`{String} columnAlias`	The column alias. Optional.

`createBetweenFilter(leftExpression, rightLessExpression, rightGreaterExpression)`

Creates a `Between` filter instance.

Parameters

`{Terrasoft.BaseExpression} leftExpression`	The expression to check in the filter.
`{Terrasoft.BaseExpression} rightLessExpression`	The initial expression of the filtering scope.
`{Terrasoft.BaseExpression} rightGreaterExpression`	The final expression of the filtering scope.

`createColumnBetweenFilterWithParameters(columnPath, lessParamValue, greaterParamValue, paramData`

Creates a `Between` filter instance to check if the column is within the specified scope.

Parameters

`{String} columnPath`	The path to the check column, relative to the `rootSchema` root schema.
`{Mixed} lessParamValue`	The initial value of the filter.
`{Mixed} greaterParamValue`	The final value of the filter.
`{Terrasoft.DataValueType} paramDataType`	The parameter data type.

`createColumnFilterWithParameter(comparisonType, columnPath, paramValue, paramDataType)`

Creates a `Compare` filter to instance to compare the column to a specified value.

Parameters

`{Terrasoft.ComparisonType} comparisonType`	The comparison operation type.
`{String} columnPath`	The path to the check column, relative to the `rootSchema` root schema.
`{Mixed} paramValue`	The parameter value.
`{Terrasoft.DataValueType} paramDataType`	The parameter data type.

`createColumnInFilterWithParameters(columnPath, paramValues, paramDataType)`

Creates an `In` filter instance to compare the value of a specified column to a parameter.

Parameters

`{String} columnPath`	The path to the check column, relative to the `rootSchema` root schema.
`{Array} paramValues`	The parameter value array.
`{Terrasoft.DataValueType} paramDataType`	The parameter data type.

`createColumnIsNotNullFilter(columnPath)`

Creates an `IsNull` filter instance to check the specified column.

Parameters

`{String} columnPath`	The path to the check column, relative to the `rootSchema` root schema.

`createColumnIsNullFilter(columnPath)`

Creates an `IsNull` filter instance to check the specified column.

Parameters

`{String} columnPath`	The path to the check column, relative to the `rootSchema` root schema.

`createCompareFilter(comparisonType, leftExpression, rightExpression)`

Creates a `Compare` filter instance.

Parameters

`{Terrasoft.ComparisonType} comparisonType`	The comparison operation type.
`{Terrasoft.BaseExpression} leftExpression`	The expression to check in the filter.
`{Terrasoft.BaseExpression} rightExpression`	The filtering expression.

`createExistsFilter(columnPath)`

Creates an `Exists` filter instance for the `[Exists by the specified condition]` type comparison. Sets the expression of the column at the specified path as the check value.

Parameters

`{String} columnPath`	The path to the column for whose expression to configure the filter.

`createFilter(comparisonType, leftColumnPath, rightColumnPath)`

Creates an instance of the `Terrasoft.CompareFilter` class filter to compare the values of two columns.

Parameters

`{Terrasoft.ComparisonType} comparisonType`	The comparison operation type.
`{String} leftColumnPath`	The path to the check column, relative to the `rootSchema` root schema.
`{String} rightColumnPath`	The path to the filter column, relative to the `rootSchema` root schema.

`createFilterGroup()`

Creates a filter group instance.

`createInFilter(leftExpression, rightExpressions)`

Creates an `In` filter instance.

Parameters

`{Terrasoft.Base Expression} left Expression`	The expression to check in the filter.
`{Terrasoft.Base Expression[]} right Expressions`	The array of expressions to compare to `leftExpression`.

`createIsNotNullFilter(leftExpression)`

Creates an `IsNull` filter instance.

Parameters

`{Terrasoft.Base Expression} left Expression`	The expression to check by the `IS NOT NULL` condition.

`createIsNullFilter(leftExpression)`

Creates an `IsNull` filter instance.

Parameters

`{Terrasoft.Base Expression} left Expression`	The expression to check by the `IS NULL` condition.

`createNotExistsFilter(columnPath)`

Creates an `Exists` filter instance for the `[Does not exist by the specified condition]` type comparison. Sets the expression of the column located at the specified path as the check value.

Parameters

`{String} columnPath`	The path to the column for whose expression to configure the filter.

`createPrimaryDisplayColumnFilterWithParameter(comparisonType, paramValue, paramDataType)`

Creates a filter object to compare the primary column to a parameter.

Parameters

`{Terrasoft.ComparisonType} comparisonType`	The comparison type.
`{Mixed} paramValue`	The parameter value.
`{Terrasoft.DataValueType} paramDataType`	The parameter data type.

`destroy()`

Deletes the object instance. If the object has already been deleted, writes an error message to the console. Calls the `onDestroy` virtual method to redefine in subclasses.

`enablePrimaryColumnFilter(primaryColumnValue)`

Enables filters by the primary key.

Parameters

`{String/Number} primaryColumnValue`	The value of the primary key.

`error(message)`

Writes an error message to the message log.

Parameters

`{String} message`	The error message to write to the message log.

`execute(callback, scope)`

The request to execute the query on the server.

Parameters

`{Function} callback`	The function to call after receiving the server response.
`{Object} scope`	The scope within which to call the `callback` function.

`{Object} getDefSerializationInfo()`

Returns the object that contains additional information for serialization.

`getEntity(primaryColumnValue, callback, scope)`

Returns the entity instance by the specified `primaryColumnValue` primary key. Calls the `callback` function within the `scope` scope after retrieving the data.

Parameters

`{String/Number} primaryColumnValue`	The value of the primary key.
`{Function} callback`	The function to call after receiving the server response.
`{Object} scope`	The scope within which to call the `callback` function.

`getEntityCollection(callback, scope)`

Returns the collection of entity instances that represent the query outcome. Calls the `callback` function within the `scope` scope after retrieving the data.

Parameters

`{Function} callback`	The function to call after receiving the server response.
`{Object} scope`	The scope within which to call the `callback` function.

`{Object} getTypeInfo()`

Returns the information about the element type.

`log(message, [type])`

Writes a message to the message log.

Parameters

`{String|Object} message`	The message to write to the message log.
`{Terrasoft.LogMessageType} type`	The type of the `callback` message log. Optional. By default, `console.log`. The `Terrasoft.core.enums.LogMessageType` enumeration specifies the available values..

`onDestroy()`

Deletes the event subscriptions and destroys the object.

`serialize(serializationInfo)`

Serializes the object in JSON.

Parameters

`{String} serializationInfo`	The results in JSON.

`setSerializableProperty(serializableObject, propertyName)`

Assigns the property name to the object if the object is not empty or not a function.

Parameters

`{Object} serializableObject`	The serializable object.
`{String} propertyName`	The property name.

`warning(message)`

Writes a warning message to the message log.

Parameters

`{String} message`	The message to write to the message log.

DataManager class JS

Medium

DataManager class JS

`DataManager` is a singleton class available via the `Terrasoft` global object. The class provides the `dataStore` repository. You can upload the contents of one or more database tables to the repository.

```
dataStore: {
    /* The DataManagerItem type data collection of the SysModule schema. */
    SysModule: sysModuleCollection,
 /* The DataManagerItem type data collection of the SysModuleEntity schema. */
    SysModuleEntity: sysModuleEntityCollection
}
```

Each record of the collection represents the record of the corresponding database table.

Properties

`{Object} dataStore`

The data collection repository.

`{String} itemClassName`

The record class name. Has the `Terrasoft.DataManagerItem` value.

Methods

`{Terrasoft.Collection} select(config, callback, scope)`

If `dataStore` does not contain a data collection that has the `config.entitySchemaName` name, the method builds and executes a database query, then returns the retrieved data. Otherwise, the method returns the data collection from `dataStore`.

Parameters

`{Object} config`	The configuration object.
	
	Configuration object properties
	
	`{String} entitySchemaName`
	The schema name.
	
	`{Terrasoft.FilterGroup} filters`
	The conditions.

`{Function} callback`	The callback function.
`{Object} scope`	The scope of the callback function.

`{Terrasoft.DataManagerItem} createItem(config, callback, scope)`

Creates a new record of the `config.entitySchemaName` type. The record columns have the `config.columnValues` values.

Parameters

`{Object} config`	The configuration object.
	
	Configuration object properties
	
	`{String} entitySchemaName`
	The schema name.
	
	`{Object} columnValues`
	The record column values.

`{Function} callback`	The callback function.
`{Object} scope`	The scope of the callback function.

`{Terrasoft.DataManagerItem} addItem(item)`

Adds the `item` record to the schema data collection.

Parameters

`{Terrasoft.DataManagerItem} item`	The record to add.

`{Terrasoft.DataManagerItem} findItem(entitySchemaName, id)`

Returns the record from the data collection of the schema that has the `entitySchemaName` name and `id` ID.

Parameters

`{String} entitySchemaName`	The data collection name.
`{String} id`	The record ID.

`{Terrasoft.DataManagerItem} remove(item)`

Selects the `isDeleted` flag for the `item` record. Once the changes are recorded, the record will be deleted from the database.

Parameters

`{Terrasoft.DataManagerItem} item`	The record to delete.

`removeItem(item)`

Deletes the record from the schema data collection.

Parameters

`{Terrasoft.DataManagerItem} item`	The record to delete.

`{Terrasoft.DataManagerItem} update(config, callback, scope)`

Updates the record that has the `config.primaryColumnValue` primary column value with the values from `config.columnValues`.

Parameters

`{Object} config`	The configuration object.
	
	Configuration object properties
	
	`{String} entitySchemaName`
	The schema name.
	
	`{String} primaryColumnValue`
	The primary column value.
	
	`{Mixed} columnValues`
	The column values.

`{Function} callback`	The callback function.
`{Object} scope`	The scope of the callback function.

`{Terrasoft.DataManagerItem} discardItem(item)`

Discards the changes to the `item` record made as part of the current `DataManager` session.

Parameters

`{Terrasoft.DataManagerItem} item`	The record, changes to which to discard.

`{Object} save(config, callback, scope)`

Saves the data collections of the schemas specified in `config.entitySchemaNames` to the database.

Parameters

`{Object} config`	The configuration object. Configuration object properties `{String[]} entitySchemaName` The name of the schema to save. Leave empty to save the data collections of all schemas.
`{Function} callback`	The callback function.
`{Object} scope`	The scope of the callback function.

DataManagerItem class JS

Properties

`{Terrasoft.BaseViewMode} viewModel`

The object projection of the database record.

Methods

`setColumnValue(columnName, columnValue)`

Assigns the new `columnValue` value to the column that has the `columnName` name.

Parameters

`{String} columnName`	The column name.
`{String} columnValue`	The column value.

`{Mixed} getColumnValue(columnName)`

Returns the value of the column that has the `columnName` name.

Parameters

`{String} columnName`	The column name.

`{Object} getValues()`

Returns the values of all record columns.

`remove()`

Selects the `isDeleted` flag for the record.

`discard()`

Discards the changes to the record made as part of the current `DataManager` session.

`{Object} save(callback, scope)`

Records the changes in the database.

Parameters

`{Function} callback`	The callback function.
`{Object} scope`	The scope of the callback function.

`{Boolean} getIsNew()`

Returns the flag that marks the record as new.

`{Boolean} getIsChanged()`

Returns the flag that marks the record as changed.

Use examples

Retrieve the records from the [*Contact*] table

```
/* Define the configuration object. */
var config = {
    /* The entity schema name. */
    entitySchemaName: "Contact",
    /* Exclude duplicates from the resulting dataset. */
    isDistinct: true
};
/* Retrieve the data. */
Terrasoft.DataManager.select(config, function (collection) {
    /* Save the retrieved records to the local repository. */
    collection.each(function (item) {
```

```
        Terrasoft.DataManager.addItem(item);
    });
}, this);
```

Add a new record to the `DataManager` object

```
/* Define the configuration object. */
var config = {
    /* The entity schema name. */
    entitySchemaName: "Contact",
    /* The column values. */
    columnValues: {
        Id: "00000000-0000-0000-0000-000000000001",
        Name: "Name1"
    }
};
/* Create a new record. */
Terrasoft.DataManager.createItem(config, function (item) {
    Terrasoft.DataManager.addItem(item);
}, this);
```

Retrieve the record and change the column value

```
/* Retrieve the record. */
var item = Terrasoft.DataManager.findItem("Contact",
     "00000000-0000-0000-0000-000000000001");
/* Assign the new "Name2" value to the Name column. */
item.setColumnValue("Name", "Name2");
```

Delete the record from `DataManager`

```
/* Define the configuration object. */
var config = {
    /* The entity schema name. */
    entitySchemaName: "Contact",
    /* The primary column value. */
    primaryColumnValue: "00000000-0000-0000-0000-000000000001"
};
/* Select the isDeleted flag for the item record. */
Terrasoft.DataManager.remove(config, function () {
}, this);
```

Discard the changes made as part of the current `DataManager` session.

```
/* Retrieve the record. */
var item = Terrasoft.DataManager.findItem("Contact",
     "00000000-0000-0000-0000-000000000001");
/* Discard the changes to the record. */
Terrasoft.DataManager.discardItem(item);
```

Record the changes in the database

```
/* Define the configuration object. */
var config = {
    /* The entity schema name. */
    entitySchemaNames: ["Contact"]
};
/* Save the changes in the database. */
Terrasoft.DataManager.save(config, function () {
}, this);
```