Creatio Academy

Data operations (back-end)

Access data directly

Version 7.18

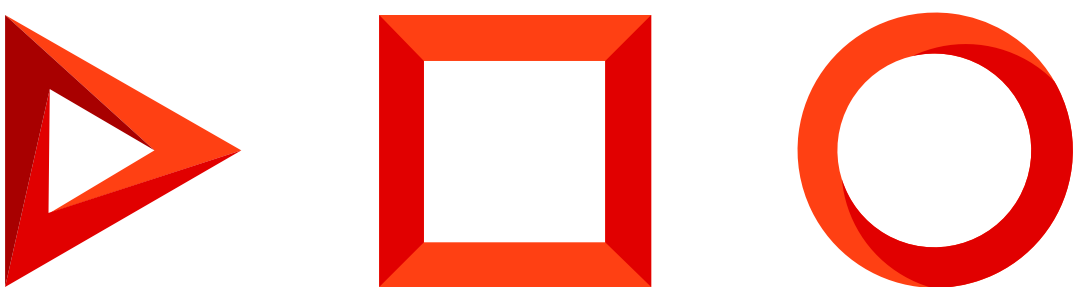

Table of Contents

Access data directly **5**
Retrieve data from the database 5
Add data to the database 6
Modify the database data 8
Delete data from the database 8
Use multithreading for database management 8
Retrieve data from the database **10**
Example 1 11
Example 2 12
Example 3 12
Example 4 13
Example 5 14
Example 6 14
Example 7 15
Add data to the database **16**
Example 1 16
Example 2 16
Add data to the database using subqueries **17**
Example 1 17
Example 2 18
Modify data in the database **18**
Example 1 19
Example 2 19
Delete data from the database **19**
Example 20
Select class **20**
Constructors 21
Properties 21
Methods 23
Insert class **31**
Constructors 32
Properties 32
Methods 33
InsertSelect class **34**
Constructors 35
Properties 35

Methods 36
Update class 38
Constructors 38
Properties 38
Methods 39
UpdateSelect class 41
Constructors 42
Properties 42
Methods 42
Delete class 43
Constructors 43
Properties 43
Methods 44
QueryFunction class 46
QueryFunction class 47
AggregationQueryFunction class 50
IsNullQueryFunction class 53
CreateGuidQueryFunction class 54
CurrentDateTimeQueryFunction class 55
CoalesceQueryFunction class 56
DatePartQueryFunction class 58
DateAddQueryFunction class 59
DateDiffQueryFunction class 61
CastQueryFunction class 63
UpperQueryFunction class 64
CustomQueryFunction class 66
DataLengthQueryFunction class 68
TrimQueryFunction class 70
LengthQueryFunction class 71
SubstringQueryFunction class 72
ConcatQueryFunction class 74
WindowQueryFunction class 75

Access data directly

Medium

The back-end core components provide the following database **access options**:

- access using the ORM model
- direct access

We recommend using the ORM model to access data. That said, direct access to the database is also implemented in the back-end core components. Learn more about running database queries using the ORM model: Access data through ORM.

This article covers direct database queries.

Use the following **classes** to manage data directly:

- `Terrasoft.Core.DB.Select` builds queries to retrieve records from the database table.
- `Terrasoft.Core.DB.Insert` builds queries to add records to the database table.
- `Terrasoft.Core.DB.InsertSelect` builds queries to add records to the database table based on the results of the queries to retrieve records from the database table.
- `Terrasoft.Core.DB.Update` builds queries to modify records in the database table.
- `Terrasoft.Core.DB.UpdateSelect` builds queries to modify records in the database table based on the results of the queries to retrieve records from the database table.
- `Terrasoft.Core.DB.Delete` builds queries to delete records from the database table.
- `Terrasoft.Core.DB.DBExecutor` builds and runs complex database queries. For example, queries that contain several nested filters, various join combinations, etc.

Retrieve data from the database

Use the following **classes** to retrieve data from the database:

- `Terrasoft.Core.DB.Select` retrieves data by accessing the database directly.
- `Terrasoft.Core.Entities.EntitySchemaQuery` retrieves data using the ORM model. Learn more about running database queries using the `Terrasoft.Core.Entities.EntitySchemaQuery` class: Access data through ORM.

The **purpose** of the `Terrasoft.Core.DB.Select` class is to build queries to select records from database tables. After you create and configure the class instance, Creatio will build a `SELECT` query to the Creatio database. You can add icons, filters, and restriction conditions to the query.

The `Terrasoft.Core.DB.Select` class **major features**:

- The resulting query does not apply the access permissions of the current user. The user can access every database table and record.
- The resulting query does not use the cache repository data.

As **result of the query**, is an instance that implements the `System.Data.IDataReader` interface or a scalar value of the corresponding type.

If you need to build queries to select database records that apply the access permissions of the user and use the cache repository data, use the `Terrasoft.Core.Entities.EntitySchemaQuery` class.

Add data to the database

Use the following **classes** to add data to the database:

- `Terrasoft.Core.DB.Insert`
- `Terrasoft.Core.DB.InsertSelect`

Add data in bulk

The **purpose** of the `Terrasoft.Core.DB.Insert` class is to build queries to add records to database tables. After you create and configure the class instance, Creatio will build an `INSERT` query to the Creatio database.

As **result of the query**, is the number of records added using the query.

The class implements multiline insertion via the `Values()` method. After calling the `Values()` method, all subsequent `Set()` method calls fall into the new `ColumnsValues` instance. If the `ColumnsValuesCollection` collection contains more than one dataset, Creatio will build a query that includes several `Values()` blocks.

Multiline insertion example

```
new Insert(UserConnection)
.Into("Table")
.Values()
    .Set("Column1", Column.Parameter(1))
    .Set("Column2", Column.Parameter(1))
    .Set("Column3", Column.Parameter(1))
.Values()
    .Set("Column1", Column.Parameter(2))
    .Set("Column2", Column.Parameter(2))
    .Set("Column3", Column.Parameter(2))
.Values()
    .Set("Column1", Column.Parameter(3))
    .Set("Column2", Column.Parameter(3))
    .Set("Column3", Column.Parameter(3))
.Execute();
```

As a result, Creatio will generate the SQL query.

SQL query

```
-- For Microsoft SQL or PostgreSQL
```

```
INSERT INTO [dbo].[Table] (Column1, Column2, Column3)
VALUES (1, 1, 1),
    (2, 2, 2),
    (3, 3, 3)

-- For Oracle
INSERT ALL
    into Table (column1, column2, column3) values (1, 1, 1)
    into Table (column1, column2, column3) values (2, 2, 2)
    into Table (column1, column2, column3) values (3, 3, 3)
SELECT * FROM dual
```

The **special aspects** of multiline data insertion:

- Microsoft SQL supports up to 2100 parameters when using `Column.Parameter` in the `Set()` expression.
- If your query contains more parameters, the developer must divide the query into subqueries since `Terrasoft.Core.DB.Insert` does not support this option.

Example

```
IEnumerable<IEnumerable<ImportEntity>> GetImportEntitiesChunks(IEnumerable<ImportEntity> enti
    var entitiesList = entities.ToList();
    var columnsList = keyColumns.ToList();
    var maxParamsPerChunk = Math.Abs(MaxParametersCountPerQueryChunk / columnsList.Count + 1)
    var chunksCount = (int)Math.Ceiling(entitiesList.Count / (double)maxParamsPerChunk);
    return entitiesList.SplitOnParts(chunksCount);
}

var entitiesList = GetImportEntitiesChunks(entities, importColumns);
entitiesList.AsParallel().AsOrdered()
    .ForAll(entitiesBatch => {
        try {
            var insertQuery = GetBufferedImportEntityInsertQuery();
            foreach (var importEntity in entitiesBatch) {
                insertQuery.Values();
                SetBufferedImportEntityInsertColumnValues(importEntity, insertQuery,
                        importColumns);
                insertQuery.Set("ImportSessionId", Column.Parameter(importSessionId));
            }
            insertQuery.Execute();
        } catch (Exception e) {
            //...
        }
});
```

- The developer must confirm the column number matches the `Set()` condition number since

`Terrasoft.Core.DB.Insert` does not support this option. The mismatch will create an exception on the database level.

Add data from the selection

The **purpose** of the `Terrasoft.Core.DB.InsertSelect` class is to build queries to add records to the database tables based on the data of the queries to retrieve records from the database table. As such, the class uses the `Terrasoft.Core.DB.Select` class instance as a data source. After you create and configure the class instance, Creatio will build an `INSERT INTO SELECT` query to the Creatio database.

The resulting query built using the `Terrasoft.Core.DB.InsertSelect` class does not apply the current user's access permissions to the corresponding records. The class uses the user connection only to access the database table.

As a **result of the query**, Creatio will add the records retrieved in the `Select` query to the database.

Modify the database data

Use the following **classes** to modify the database data:

- `Terrasoft.Core.DB.Update`
- `Terrasoft.Core.DB.UpdateSelect`

Modify data in bulk

The **purpose** of the `Terrasoft.Core.DB.Update` class is to build queries to modify records in the database tables. After you create and configure the class instance, Creatio will build an `UPDATE` query to the Creatio database.

Modify data based on the selection

The **purpose** of the `Terrasoft.Core.DB.UpdateSelect` class is to build queries to modify records in the database tables based on the data of the queries to retrieve records from the database table. As such, the class uses the `Terrasoft.Core.DB.Select` class instance as a data source. After you create and configure the class instance, Creatio will build an `UPDATE FROM` query to the Creatio database.

As a **result of the query**, Creatio will modify the records retrieved in the `Select` query in the database table.

Delete data from the database

The `Terrasoft.Core.DB.Delete` class deletes data from the database.

The **purpose** of the `Terrasoft.Core.DB.Delete` class is to build queries to delete records from the database tables. After you create and configure the class instance, Creatio will build a `DELETE` query to the Creatio database.

Use multithreading for database management

Multithreading is the use of several parallel threads to send database queries via `UserConnection`.

The `Terrasoft.Core.DB.DBExecutor` class supports multithreading. The class implements building and running several database queries in a single transaction. A single `DBExecutor` instance is available per user. The user cannot create new instances.

Multithreading may lead to synchronization issues related to transaction start and confirmation. The issue can occur even if you use the `DBExecutor` indirectly. For example, through `EntitySchemaQuery`.

If you use the `Terrasoft.Core.DB.DBExecutor` class, you must wrap the creation of the `DBExecutor` instance in the `using` operator. This is required since Creatio uses `unmanaged` resources to manage the database. You can also call the `Dispose()` method to free up resources. Learn more about using the `using` operator in the official Microsoft documentation.

Call the `dbExecutor.StartTransaction` method to start the transaction. Call `dbExecutor.CommitTransaction` or `dbExecutor.RollbackTransaction` to end the transaction. If the execution goes beyond the scope of the `using` block and you do not call the `dbExecutor.CommitTransaction` method, the transaction will roll back automatically.

Attention. If you run several queries in a single transaction, pass `dbExecutor` to `Execute`, `ExecuteReader`, `ExecuteScalar` methods.

View the source code snippets that use `DBExecutor` below. Do not call the methods of the `DBExecutor` instance in parallel threads.

Correct DBExecutor use example

```
/* Start using the DBExecutor instance in the main thread. */
using (DBExecutor dbExecutor = UserConnection.EnsureDBConnection()) {
    dbExecutor.StartTransaction();
    //...
    dbExecutor.CommitTransaction();
}
//...
var select = (Select)new Select(UserConnection)
    .Column("Id")
    .From("Contact")
    .Where("Name")
    .IsEqual(Column.Parameter("Supervisor"));
/* Continue using the DBExecutor instance in the main thread. */
using (DBExecutor dbExecutor = UserConnection.EnsureDBConnection()) {
    using (IDataReader dataReader = select.ExecuteReader(dbExecutor)) {
        while (dataReader.Read()) {
            //...
        }
    }
}
```

Incorrect DBExecutor use example

```
/* Create a parallel thread. */
var task = new Task(() => {

    // Start using the DBExecutor instance in a parallel thread.
    using (DBExecutor dbExecutor = UserConnection.EnsureDBConnection()) {
        dbExecutor.StartTransaction();
        //...
        dbExecutor.CommitTransaction();
    }
});

/* Run an asynchronous task in the parallel thread. Continue executing the program in the main t
task.Start();
//...
var select = (Select)new Select(UserConnection)
    .Column("Id")
    .From("Contact")
    .Where("Name")
    .IsEqual(Column.Parameter("Supervisor"));

/* If you use the DBExecutor instance in the main thread, this will lead to an error since the p
using (DBExecutor dbExecutor = UserConnection.EnsureDBConnection()) {
    using (IDataReader dataReader = select.ExecuteReader(dbExecutor)) {
        while (dataReader.Read()) {
            //...
        }
    }
}
```

Retrieve data from the database

Advanced

Note. The examples in this article are also implemented in the web service. The package that contains the web service implementation is attached in the Resources block.

The `CreateJson` method processes query results in the examples. View the method below.

`CreateJson` method

```
private string CreateJson(IDataReader dataReader)
{
    var list = new List<dynamic>();
```

```
    var cnt = dataReader.FieldCount;
    var fields = new List<string>();
    for (int i = 0; i < cnt; i++)
    {
        fields.Add(dataReader.GetName(i));
    }
    while (dataReader.Read())
    {
        dynamic exo = new System.Dynamic.ExpandoObject();
        foreach (var field in fields)
        {
            ((IDictionary<String, Object>)exo).Add(field, dataReader.GetColumnValue(field));
        }
        list.Add(exo);
    }
    return JsonConvert.SerializeObject(list);
}
```

Example 1

Example. Select a number of records from the needed table (object schema).

`SelectColumns` **method**

```
public string SelectColumns(string tableName, int top)
{
    top = top > 0 ? top : 1;
    var result = "{}";
    var select = new Select(UserConnection)
        .Top(top)
        .Column(Column.Asterisk())
        .From(tableName);
    using (DBExecutor dbExecutor = UserConnection.EnsureDBConnection())
    {
        using (IDataReader dataReader = select.ExecuteReader(dbExecutor))
        {
            result = CreateJson(dataReader);
        }
    }
    return result;
}
```

Example 2

Example. Select the IDs, names, and birth dates of contacts whose birth dates are later than the required year.

`SelectContactsYoungerThan` **method**

```
public string SelectContactsYoungerThan(string birthYear)
{
    var result = "{}";
    var year = DateTime.ParseExact(birthYear, "yyyy", CultureInfo.InvariantCulture);
    var select = new Select(UserConnection)
            .Column("Id")
            .Column("Name")
            .Column("BirthDate")
        .From("Contact")
        .Where("BirthDate").IsGreater(Column.Parameter(year))
            .Or("BirthDate").IsNull()
        .OrderByDesc("BirthDate")
            as Select;
    using (DBExecutor dbExecutor = UserConnection.EnsureDBConnection())
    {
        using (IDataReader dataReader = select.ExecuteReader(dbExecutor))
        {
            result = CreateJson(dataReader);
        }
    }
    return result;
}
```

Example 3

Example. Select the IDs, names, and birth dates of contacts whose birth dates are later than the specified year and who have the account specified.

`SelectContactsYoungerThanAndHasAccountId` **method**

```
public string SelectContactsYoungerThanAndHasAccountId(string birthYear)
{
    var result = "{}";
    var year = DateTime.ParseExact(birthYear, "yyyy", CultureInfo.InvariantCulture);
```

```
    var select = new Select(UserConnection)
            .Column("Id")
            .Column("Name")
            .Column("BirthDate")
        .From("Contact")
        .Where()
        .OpenBlock("BirthDate").IsGreater(Column.Parameter(year))
            .Or("BirthDate").IsNull()
        .CloseBlock()
        .And("AccountId").Not().IsNull()
        .OrderByDesc("BirthDate")
            as Select;
    using (DBExecutor dbExecutor = UserConnection.EnsureDBConnection())
    {
        using (IDataReader dataReader = select.ExecuteReader(dbExecutor))
        {
            result = CreateJson(dataReader);
        }
    }
    return result;
}
```

Example 4

Example. Select the IDs and names of all contacts by joining them to the IDs and names of the corresponding accounts.

`SelectContactsJoinAccount` method

```
public string SelectContactsJoinAccount()
{
    var result = "{}";
    var select = new Select(UserConnection)
            .Column("Contact", "Id").As("ContactId")
            .Column("Contact", "Name").As("ContactName")
            .Column("Account", "Id").As("AccountId")
            .Column("Account", "Name").As("AccountName")
        .From("Contact")
        .Join(JoinType.Inner, "Account")
            .On("Contact", "Id").IsEqual("Account", "PrimaryContactId")
            as Select;
    using (DBExecutor dbExecutor = UserConnection.EnsureDBConnection())
    {
        using (IDataReader dataReader = select.ExecuteReader(dbExecutor))
```

```
        {
            result = CreateJson(dataReader);
        }
    }
    return result;
}
```

Example 5

Example. Select the IDs and names of the primary account contacts.

`SelectAccountPrimaryContacts` **method**

```
public string SelectAccountPrimaryContacts()
{
    var result = "{}";
    var select = new Select(UserConnection)
            .Column("Id")
            .Column("Name")
        .From("Contact").As("C")
        .Where()
            .Exists(new Select(UserConnection)
                            .Column("A", "PrimaryContactId")
                        .From("Account").As("A")
                        .Where("A", "PrimaryContactId").IsEqual("C", "Id"))
            as Select;
    using (DBExecutor dbExecutor = UserConnection.EnsureDBConnection())
    {
        using (IDataReader dataReader = select.ExecuteReader(dbExecutor))
        {
            result = CreateJson(dataReader);
        }
    }
    return result;
}
```

Example 6

Example. Select the countries and the number of cities in a country if such number is greater than the specified number.

`SelectCountriesWithCitiesCount` **method**

```
public string SelectCountriesWithCitiesCount(int count)
{
    var result = "{}";
    var select = new Select(UserConnection)
            .Column(Func.Count("City", "Id")).As("CitiesCount")
            .Column("Country", "Name").As("CountryName")
        .From("City")
        .Join(JoinType.Inner, "Country")
            .On("City", "CountryId").IsEqual("Country", "Id")
        .GroupBy("Country", "Name")
        .Having(Func.Count("City", "Id")).IsGreater(Column.Parameter(count))
        .OrderByDesc("CitiesCount")
            as Select;
    using (DBExecutor dbExecutor = UserConnection.EnsureDBConnection())
    {
        using (IDataReader dataReader = select.ExecuteReader(dbExecutor))
        {
            result = CreateJson(dataReader);
        }
    }
    return result;
}
```

Example 7

Example. Retrieve the ID of the contact by their name.

`SelectCountryIdByCityName` **method**

```
public string SelectCountryIdByCityName(string CityName)
{
    var result = "";
    var select = new Select(UserConnection)
            .Column("CountryId")
        .From("City")
        .Where("Name").IsEqual(Column.Parameter(CityName)) as Select;
    result = select.ExecuteScalar<Guid>().ToString();
    return result;
}
```

Add data to the database

Advanced

Note. The examples in this article are also implemented in the web service. The package that contains the web service implementation is attached in the Resources block.

Example 1

Example. Add the contact that has the specified name.

`InsertContact` **method**

```
public string InsertContact(string contactName)
{
    contactName = contactName ?? "Unknown contact";
    var ins = new Insert(UserConnection)
        .Into("Contact")
        .Set("Name", Column.Parameter(contactName));
    var affectedRows = ins.Execute();
    var result = $"Inserted new contact with name '{contactName}'. {affectedRows} rows affected"
    return result;
}
```

Example 2

Example. Add the city that has the specified name and connect it to the specified country.

`InsertCity` **method**

```
public string InsertCity(string city, string country)
{
    city = city ?? "unknown city";
    country = country ?? "unknown country";

    var ins = new Insert(UserConnection)
        .Into("City")
        .Set("Name", Column.Parameter(city))
        .Set("CountryId",
```

```
            new Select(UserConnection)
                    .Top(1)
                    .Column("Id")
                .From("Country")
                .Where("Name")
                    .IsEqual(Column.Parameter(country)));
    var affectedRows = ins.Execute();
    var result = $"Inserted new city with name '{city}' located in '{country}'. {affectedRows} r
    return result;
}
```

Add data to the database using subqueries

Advanced

Note. The examples in this article are also implemented in the web service. The package that contains the web service implementation is attached in the Resources block.

Example 1

Example. Add the contact that has the specified name and account.

`InsertContactWithAccount` **method**

```
public string InsertContactWithAccount(string contactName, string accountName)
{
    contactName = contactName ?? "Unknown contact";
    accountName = accountName ?? "Unknown account";

    var id = Guid.NewGuid();
    var selectQuery = new Select(UserConnection)
            .Column(Column.Parameter(contactName))
            .Column("Id")
        .From("Account")
        .Where("Name").IsEqual(Column.Parameter(accountName)) as Select;
    var insertSelectQuery = new InsertSelect(UserConnection)
        .Into("Contact")
        .Set("Name", "AccountId")
        .FromSelect(selectQuery);

    var affectedRows = insertSelectQuery.Execute();
    var result = $"Inserted new contact with name '{contactName}'" +
```

```
                $" and account '{accountName}'." +
                $" Affected {affectedRows} rows.";
    return result;
}
```

Example 2

Example. Add the contact that has the specified name and bind it to all accounts

`InsertAllAccountsContact` **method**

```
public string InsertAllAccountsContact(string contactName)
{
    contactName = contactName ?? "Unknown contact";

    var id = Guid.NewGuid();
    var insertSelectQuery = new InsertSelect(UserConnection)
        .Into("Contact")
        .Set("Name", "AccountId")
        .FromSelect(
            new Select(UserConnection)
                .Column(Column.Parameter(contactName))
                .Column("Id")
                .From("Account") as Select);

    var affectedRows = insertSelectQuery.Execute();
    var result = $"Inserted {affectedRows} new contacts with name '{contactName}'";
    return result;
}
```

Modify data in the database

Advanced

Note. The examples in this article are also implemented in the web service. The package that contains the web service implementation is attached in the Resources block.

In most cases, a modification query must contain the `Where` condition, which specifies the exact records to modify. If you do not specify the `Where` condition, Creatio will modify all records.

Example 1

Example. Change the contact name.

`ChangeContactName` **method**

```
public string ChangeContactName(string oldName, string newName)
{
    var update = new Update(UserConnection, "Contact")
        .Set("Name", Column.Parameter(newName))
        .Where ("Name").IsEqual(Column.Parameter(oldName));
    var cnt = update.Execute();
    return $"Contacts {oldName} changed to {newName}. {cnt} rows affected.";
}
```

Example 2

Example. Change the user that modified the contact record to the specified user for all existing contacts.

`ChangeAllContactModifiedBy` **method**

```
public string ChangeAllContactModifiedBy(string Name)
{
    var update = new Update(UserConnection, "Contact")
        .Set("ModifiedById",
            new Select(UserConnection).Top(1)
                .Column("Id")
                .From("Contact")
                .Where("Name").IsEqual(Column.Parameter(Name)));
    var cnt = update.Execute();
    return $"All contacts are changed by {Name} now. {cnt} rows affected.";
}
```

The `Where` condition refers to the `Select` query. The `Update` query does not contain the `Where` condition since the goal is to modify all records.

Delete data from the database

Advanced

Note. The examples in this article are also implemented in the web service. The package that contains the web service implementation is attached in the Resources block.

In most cases, a deletion query must contain the `Where` condition, which specifies the exact records to delete. If you do not specify the `Where` condition, Creatio will delete all records.

Example

Example. Delete the contact that has the specified name.

`DeleteContacts` method

```
public string DeleteContacts(string name)
{
    var delete = new Delete(UserConnection)
        .From("Contact")
        .Where("Name").IsEqual(Column.Parameter(name));
    var cnt = delete.Execute();
    return $"Contacts with name {name} were deleted. {cnt} rows affected";
}
```

Select class C#

Advanced

`Terrasoft.Core.DB` namespace.

The `Terrasoft.Core.DB.UpdateSelect` class builds queries to select records in a Creatio database table. As a result of creating and configuring an instance of this class, Creatio will build a `SELECT` SQL query. You can add columns, filters, and restriction conditions to the query. The query results are returned as an instance that implements the `System.Data.IDataReader` interface or as a scalar value of the needed type.

Attention. When working with the `Select` class, Creatio does not apply the current user permissions. All Creatio database records are available. Creatio does not use the data in the cache repository as well. If you need additional Creatio permissions and cache repository management options, use the `EntitySchemaQuery` class.

Note. View the entire list of methods and properties of the `Select` class, its parent classes, as well as the interfaces it implements, in the .NET class library.

Constructors

`Select(UserConnection userConnection)`

Creates a class instance with the specified `UserConnection` parameter.

`Select(UserConnection userConnection, CancellationToken cancellationToken)`

Creates a class instance with the specified `UserConnection` parameter and the cancellation token for managed threads.

`Select(Select source)`

Creates a class instance that is a clone of the instance passed as the argument.

Properties

`UserConnection` `Terrasoft.Core.UserConnection`

The user connection that the query uses.

`RowCount` `int`

The number of records to return after the execution.

`Parameters` `Terrasoft.Core.DB.QueryParameterCollection`

The query parameter collection.

`HasParameters` `bool`

Whether the query has parameters.

`BuildParametersAsValue` `bool`

Whether to add the query parameters to the query text as values.

`Joins` `Terrasoft.Core.DB.JoinCollection`

The collection of `Join` query expressions.

`HasJoins` `bool`

Whether the query has `Join` expressions.

`Condition` `Terrasoft.Core.DB.QueryCondition`

The `Where` condition of the query.

`HasCondition` `bool`

Whether the query has the `Where` expression.

`HavingCondition` `Terrasoft.Core.DB.QueryCondition`

The `Having` condition of the query.

`HasHavingCondition` `bool`

Whether the query has the `Having` expression.

`OrderByItems` `Terrasoft.Core.DB.OrderByItemCollection`

The collection of expressions by which to sort the query results.

`HasOrderByItems` `bool`

Whether the query has order expressions.

`GroupByItems` `Terrasoft.Core.DB.QueryColumnExpressionCollection`

The collection of expressions by which to group the query results.

`HasGroupByItems` `bool`

Whether the query has grouping expressions.

`IsDistinct` `bool`

Whether to return only the distinct records.

`Columns` `Terrasoft.Core.DB.QueryColumnExpressionCollection`

The collection of query column expressions.

`OffsetFetchPaging` `bool`

Whether to paginate the query results.

`RowsOffset` `int`

The number of rows to skip when returning the query result.

`QueryKind` `Terrasoft.Common.QueryKind`

The query type.

Methods

`void ResetCachedSqlText()`

Clears the cached query text.

`QueryParameterCollection GetUsingParameters()`

Returns the parameter collection that the query uses.

`void ResetParameters()`

Clears the query parameter collection.

`QueryParameterCollection InitializeParameters()`

Initializes the query parameter collection.

`IDataReader ExecuteReader(DBExecutor dbExecutor)`

Executes the query using the `DBExecutor` instance. Returns the object that implements the `IDataReader` interface.

Parameters

`dbExecutor`	The `DBExecutor` instance required to run the query.

`IDataReader ExecuteReader(DBExecutor dbExecutor, CommandBehavior behavior)`

Executes the query using the `DBExecutor` instance. Returns the object that implements the `IDataReader` interface.

Parameters

`behavior`	Describes the query results and how they affect the database.
`dbExecutor`	The `DBExecutor` instance required to run the query.

`void ExecuteReader(ExecuteReaderReadMethod readMethod)`

Executes the query by calling the passed `ExecuteReaderReadMethod` delegate method for every record of the resulting set.

Parameters

`readMethod`	The `ExecuteReaderReadMethod` delegate method.

`TResult ExecuteScalar<TResult>()`

Executes the query. Returns the typed first column of the first record in the resulting set.

`TResult ExecuteScalar<TResult>(DBExecutor dbExecutor)`

Executes the query using the `DBExecutor` instance. Returns the typed first column of the first record in the resulting set.

Parameters

`dbExecutor`	The `DBExecutor` instance required to run the query.

`Select Distinct()`

Adds the `DISTINCT` keyword to the SQL query. Excludes record duplicates from the resulting set. Returns the query instance.

`Select Top(int rowCount)`

Sets the number of records to return in the resulting set. Also changes the `RowCount` property value. Returns the query instance.

Parameters

`rowCount`	The number of first records in the resulting set.

```
Select As(string alias)
```

Adds the last query expression alias specified in the argument. Returns the query instance.

Parameters

`alias`	The query expression alias.

```
Select Column(string sourceColumnAlias)
Select Column(string sourceAlias, string sourceColumnAlias)
Select Column(Select subSelect)
Select Column(Query subSelectQuery)
Select Column(QueryCase queryCase)
Select Column(QueryParameter queryParameter)
Select Column(QueryColumnExpression columnExpression)
```

Adds an expression, a subquery, or a parameter to the column expression collection of the query. Returns the query instance.

Parameters

`sourceColumnAlias`	The name of the column for which to add the expression.
`sourceAlias`	The alias of the source from which to add the column expression.
`subSelect`	The added data selection subquery.
`subSelectQuery`	The added subquery.
`queryCase`	The added expression for the `Case` operator.
`queryParameter`	The added query parameter.
`columnExpression`	The expression for whose results to add the condition.

```
Select From(string schemaName)
Select From(Select subSelect)
Select From(Query subSelectQuery)
Select From(QuerySourceExpression sourceExpression)
```

Adds the data source to the query. Returns the query instance.

Parameters

schemaName	The schema name.
subSelect	The selection subquery whose results become the data source for the current query.
subSelectQuery	The subquery whose results become the data source for the current query.
sourceExpression	The expression of the query data source.

```
Join Join(JoinType joinType, string schemaName)
Join Join(JoinType joinType, Select subSelect)
Join Join(JoinType joinType, Query subSelectQuery)
Join Join(JoinType joinType, QuerySourceExpression sourceExpression)
```

Joins a schema, a subquery, or an expression to the current query.

Parameters

joinType	The join type.
schemaName	The joinable schema name.
subSelect	The joinable data selection subquery.
subSelectQuery	The joinable subquery.
sourceExpression	The joinable expression.

Available values (`Terrasoft.Core.DB.JoinType`)

Inner	Inner join.
LeftOuter	Left outer join.
RightOuter	Right outer join.
FullOuter	Full join.
Cross	Cross join.

```
QueryCondition Where()
QueryCondition Where(string sourceColumnAlias)
```

```
QueryCondition Where(string sourceAlias, string sourceColumnAlias)
QueryCondition Where(Select subSelect)
QueryCondition Where(Query subSelectQuery)
QueryCondition Where(QueryColumnExpression columnExpression)
Query Where(QueryCondition condition)
```

Adds the initial condition to the current query.

Parameters

`sourceColumnAlias`	The alias of the schema for which to add the condition.
`sourceAlias`	The alias of the source.
`subSelect`	The data selection subquery for whose results to add the condition.
`subSelectQuery`	The subquery for whose results to add the condition.
`columnExpression`	The expression for whose results to add the condition.
`condition`	The query condition.

```
QueryCondition And()
QueryCondition And(string sourceColumnAlias)
QueryCondition And(string sourceAlias, string sourceColumnAlias)
QueryCondition And(Select subSelect)
QueryCondition And(Query subSelectQuery)
QueryCondition And(QueryParameter parameter)
QueryCondition And(QueryColumnExpression columnExpression)
Query And(QueryCondition condition)
```

Adds the condition (predicate) to the current query condition using the AND logical operation.

Parameters

sourceColumnAlias	The name of the schema for which to add the predicate.
sourceAlias	The alias of the source.
subSelect	The data selection subquery that serves as a predicate.
subSelectQuery	The subquery that serves as a predicate
parameter	The parameter for which to add the predicate.
columnExpression	The expression that serves as a predicate.
condition	The query condition.

```
QueryCondition Or()
QueryCondition Or(string sourceColumnAlias)
QueryCondition Or(string sourceAlias, string sourceColumnAlias)
QueryCondition Or(Select subSelect)
QueryCondition Or(Query subSelectQuery)
QueryCondition Or(QueryParameter parameter)
QueryCondition Or(QueryColumnExpression columnExpression)
Query Or(QueryCondition condition)
```

Adds the condition (predicate) to the current query condition using the OR logical operation.

Parameters

sourceColumnAlias	The name of the schema for which to add the predicate.
sourceAlias	The alias of the source.
subSelect	The data select subquery that serves as a predicate.
subSelectQuery	The subquery that serves as a predicate
parameter	The parameter for which to add the predicate.
columnExpression	The expression that serves as a predicate.
condition	The query condition.

```
Query OrderBy(OrderDirectionStrict direction, string sourceColumnAlias)
Query OrderBy(OrderDirectionStrict direction, string sourceAlias, string sourceColumnAlias)
Query OrderBy(OrderDirectionStrict direction, QueryFunction queryFunction)
```

```
Query OrderBy(OrderDirectionStrict direction, Select subSelect)
Query OrderBy(OrderDirectionStrict direction, Query subSelectQuery)
Query OrderBy(OrderDirectionStrict direction, QueryColumnExpression columnExpression)
```

Sorts the query results. Returns the query instance.

Parameters

`direction`	The sorting order.
`sourceColumnAlias`	The alias of the column by which to sort the results.
`sourceAlias`	The alias of the source.
`queryFunction`	The function whose value serves as the sorting key.
`subSelect`	The select subquery whose results serve as the sorting key.
`subSelectQuery`	The subquery whose results serve as the sorting key.
`columnExpression`	The expression whose results serve as the sorting key.

```
Query OrderByAsc(string sourceColumnAlias)
Query OrderByAsc(string sourceAlias, string sourceColumnAlias)
Query OrderByAsc(Select subSelect)
Query OrderByAsc(Query subSelectQuery)
Query OrderByAsc(QueryColumnExpression columnExpression)
```

Sorts query results in the ascending order. Returns the query instance.

Parameters

`sourceColumnAlias`	The alias of the column by which to sort the results.
`sourceAlias`	The alias of the source.
`subSelect`	The select subquery whose results serve as the sorting key.
`subSelectQuery`	The subquery whose results serve as the sorting key.
`columnExpression`	The expression whose results serve as the sorting key.

```
Query OrderByDesc(string sourceColumnAlias)
Query OrderByDesc(string sourceAlias, string sourceColumnAlias)
Query OrderByDesc(Select subSelect)
```

```
Query OrderByDesc(Query subSelectQuery)
Query OrderByDesc(QueryColumnExpression columnExpression)
```

Sorts query results in descending order. Returns the query instance.

Parameters

`sourceColumnAlias`	The alias of the column by which to sort the results.
`sourceAlias`	The alias of the source.
`subSelect`	The select subquery whose results serve as the sorting key.
`subSelectQuery`	The subquery whose results serve as the sorting key.
`columnExpression`	The expression whose results serve as the sorting key.

```
Query GroupBy(string sourceColumnAlias)
Query GroupBy(string sourceAlias, string sourceColumnAlias)
Query GroupBy(QueryColumnExpression columnExpression)
```

Groups the query results. Returns the query instance.

Parameters

`sourceColumnAlias`	The alias of the column by which to group the results.
`sourceAlias`	The alias of the source.
`columnExpression`	The expression whose results serve as the grouping key.

```
QueryCondition Having()
QueryCondition Having(string sourceColumnAlias)
QueryCondition Having(string sourceAlias, string sourceColumnAlias)
QueryCondition Having(Select subSelect)
QueryCondition Having(Query subSelectQuery)
QueryCondition Having(QueryParameter parameter)
QueryCondition Having(QueryColumnExpression columnExpression)
```

Adds the group condition to the current query. Returns the `Terrasoft.Core.DB.QueryCondition` instance that represents the group condition for the query parameter.

Parameters

`sourceColumnAlias`	The alias of the column by which to add the group condition.
`sourceAlias`	The alias of the source.
`subSelect`	The select subquery for whose results to add the group condition.
`subSelectQuery`	The subquery for whose results to add the group condition.
`parameter`	The query parameter to add the group condition.
`columnExpression`	The expression that serves as a predicate.

```
Query Union(Select unionSelect)
Query Union(Query unionSelectQuery)
```

Merges the results of the passed query with the results of the current query. Excludes duplicates from the resulting set.

Parameters

`unionSelect`	The selection subquery.
`unionSelectQuery`	The subquery.

```
Query UnionAll(Select unionSelect)
Query UnionAll(Query unionSelectQuery)
```

Merges the results of the passed query with the results of the current query. Does not exclude duplicates from the resulting set.

Parameters

`unionSelect`	The selection subquery.
`unionSelectQuery`	Subquery.

Insert class C#

Advanced

`Terrasoft.Core.DB` namespace.

The `Terrasoft.Core.DB.Insert` class builds queries to add records to Creatio database tables. As a result of creating and configuring an instance of this class, Creatio will build an `INSERT` SQL query. The query returns the

number of records involved.

Attention. When working with the `Insert` class, Creatio does not apply the default access permissions to the added records. The class uses the user connection only to access the database table.

Note. View the entire list of methods and properties of the `Insert` class, its parent classes, as well as the interfaces it implements, in the .NET class library documentation.

Constructors

`Entity(UserConnection userConnection)`

Creates a new `Entity` class instance for the set `UserConnection` user connection.

`Insert(UserConnection userConnection)`

Creates a class instance with the specified `UserConnection`.

`Insert(Insert source)`

Creates a class instance that is a clone of the instance passed as the argument.

Properties

`UserConnection` `Terrasoft.Core.UserConnection`

The user connection that the query uses.

`Source` `Terrasoft.Core.DB.ModifyQuerySource`

The data source.

`Parameters` `Terrasoft.Core.DB.QueryParameterCollection`

The query parameter collection.

`HasParameters` `bool`

Whether the query has parameters.

`BuildParametersAsValue` `bool`

Whether to add the query parameters to the query text as values.

`ColumnValues` `Terrasoft.Core.DB.ModifyQueryColumnValueCollection`

The collection of query column values.

`ColumnValuesCollection` `List<ModifyQueryColumnValueCollection>`

The collection of column values required to add records in bulk.

Methods

`void ResetCachedSqlText()`

Clears the cached query text.

`QueryParameterCollection GetUsingParameters()`

Returns the parameter collection that the query uses.

`void ResetParameters()`

Clears the query parameter collection.

`void SetParameterValue(string name, object value)`

Sets the value of the query parameter

Parameters

`name`	The parameter name.
`value`	The value.

`void InitializeParameters()`

Initializes the query parameter collection.

`int Execute()`

Executes the query. Returns the number of records that the query involved.

```
int Execute(DBExecutor dbExecutor)
```

Executes the query using the `DBExecutor` instance. Returns the number of records that the query processed.

```
Insert Into(string schemaName)
Insert Into(ModifyQuerySource source)
```

Adds a data source to the current query.

Parameters

`schemaName`	The schema name.
`source`	The data source.

```
Insert Set(string sourceColumnAlias, Select subSelect)
Insert Set(string sourceColumnAlias, Query subSelectQuery)
Insert Set(string sourceColumnAlias, QueryColumnExpression columnExpression)
Insert Set(string sourceColumnAlias, QueryParameter parameter)
```

Adds a `SET` clause to the current query. Required to assign the passed expression or parameter to the column. Returns the current `Insert` instance.

Parameters

`sourceColumnAlias`	The column alias.
`subSelect`	The select subquery.
`subSelectQuery`	The subquery.
`columnExpression`	The column expression.
`parameter`	The query parameter.

```
Insert Values()
```

Initializes the values required to add records in bulk.

InsertSelect class C#

Advanced

`Terrasoft.Core.DB` namespace.

The `Terrasoft.Core.DB.InsertSelect` class builds queries to add records to Creatio database tables. The `Terrasoft.Core.DB.Select` class instance serves as a data source. As a result of creating and configuring an instance of `Terrasoft.Core.DB.InsertSelect`, Creatio will build an `INSERT INTO SELECT` SQL query.

Attention. When working with the `InsertSelect` class, Creatio does not apply the default access permissions to the added records. Moreover, Creatio does not apply any permissions to such records, including object operation permissions, record permissions, column permissions. The class uses the user connection only to access the database table.

Note. As a result of an `InsertSelect` query, Creatio will add all records returned in the `Select` subquery to the database.

Constructors

`InsertSelect(UserConnection userConnection)`

Creates a class instance with the specified `UserConnection`.

`InsertSelect(InsertSelect source)`

Creates a class instance that is a clone of the instance passed as the argument.

Properties

`UserConnection` `Terrasoft.Core.UserConnection`

The user connection that the query uses.

`Source` `Terrasoft.Core.DB.ModifyQuerySource`

The data source.

`Parameters` `Terrasoft.Core.DB.QueryParameterCollection`

The query parameter collection.

`HasParameters` `bool`

Whether the query has parameters.

`BuildParametersAsValue` `bool`

Whether to add the query parameters to the query text as values.

`Columns` `Terrasoft.Core.DB.ModifyQueryColumnValueCollection`

The collection of query column values.

`Select` `Terrasoft.Core.DB.Select`

The `Terrasoft.Core.DB.Select` instance that the query uses.

Methods

`void ResetCachedSqlText()`

Clears the cached query text.

`QueryParameterCollection GetUsingParameters()`

Returns the parameter collection that the query uses.

`void ResetParameters()`

Clears the query parameter collection.

`void SetParameterValue(string name, object value)`

Sets the value of the query parameter

Parameters

`name`	The parameter name.
`value`	The value.

`void InitializeParameters()`

Initializes the query parameter collection.

```
int Execute()
```

Executes the query. Returns the number of records that the query involved.

```
int Execute(DBExecutor dbExecutor)
```

Executes the query using the `DBExecutor` instance. Returns the number of records that the query involved.

```
InsertSelect Into(string schemaName)
InsertSelect Into(ModifyQuerySource source)
```

Adds the data source to the current query.

Parameters

`schemaName`	The schema name.
`source`	The data source.

```
InsertSelect Set(IEnumerable<string> sourceColumnAliases)
InsertSelect Set(params string[] sourceColumnAliases)
InsertSelect Set(IEnumerable<ModifyQueryColumn> columns)
InsertSelect Set(params ModifyQueryColumn[] columns)
```

Adds a set of columns to the current query. The subquery assigns values to the columns. Returns the current `InsertSelect` instance.

Parameters

`sourceColumnAliases`	A method parameter collection or array, which contains the column aliases.
`Columns`	A method parameter collection or array, which contains the column instances.

```
InsertSelect FromSelect(Select subSelect)
InsertSelect FromSelect(Query subSelectQuery)
```

Adds the `SELECT` clause to the current query.

Parameters

`subSelect`	The select subquery.
`subSelectQuery`	The subquery.

Update class C#

Advanced

`Terrasoft.Core.DB` namespace.

The `Terrasoft.Core.DB.Update` class builds queries to modify records in Creatio database tables. As a result of creating and configuring an instance of this class, Creatio will build an `UPDATE` SQL query.

Note. View the entire list of methods and properties of the `Update` class, its parent classes, as well as the interfaces it implements, in the .NET class library documentation.

Constructors

`Update(UserConnection userConnection)`

Creates a class instance using `UserConnection`.

`Update(UserConnection userConnection, string schemaName)`

Creates a class instance for the specified schema using `UserConnection`.

`Update(UserConnection userConnection, ModifyQuerySource source)`

Creates a class instance for the specified data source using `UserConnection`.

`Update(Insert source)`

Creates a class instance that is a clone of the instance passed as the argument.

Properties

`UserConnection` `Terrasoft.Core.UserConnection`

The user connection that the query uses.

`Condition` `Terrasoft.Core.DB.QueryCondition`

The `Where` condition of the query.

`HasCondition bool`

Whether the query has a `Where` expression.

`Source Terrasoft.Core.DB.ModifyQuerySource`

The query data source.

`ColumnValues Terrasoft.Core.DB.ModifyQueryColumnValueCollection`

The collection of query column values.

Methods

```
void ResetCachedSqlText()
```

Clears the cached query text.

```
QueryParameterCollection GetUsingParameters()
```

Returns the parameter collection that the query uses.

```
int Execute()
```

Executes the query. Returns the number of records that the query involved.

```
int Execute(DBExecutor dbExecutor)
```

Executes the query using the `DBExecutor` instance. Returns the number of records that the query processed.

```
QueryCondition Where()
QueryCondition Where(string sourceColumnAlias)
QueryCondition Where(string sourceAlias, string sourceColumnAlias)
QueryCondition Where(Select subSelect)
QueryCondition Where(Query subSelectQuery)
QueryCondition Where(QueryColumnExpression columnExpression)
Query Where(QueryCondition condition)
```

Adds the initial condition to the current query.

Parameters

`sourceColumnAlias`	The alias of the schema to which to add the condition.
`sourceAlias`	The alias of the source.
`subSelect`	The data selection subquery to whose results to add the condition.
`subSelectQuery`	The subquery to whose results to add the condition.
`columnExpression`	The expression to whose results to add the condition.
`Condition`	The query condition.

```
QueryCondition And()
QueryCondition And(string sourceColumnAlias)
QueryCondition And(string sourceAlias, string sourceColumnAlias)
QueryCondition And(Select subSelect)
QueryCondition And(Query subSelectQuery)
QueryCondition And(QueryParameter parameter)
QueryCondition And(QueryColumnExpression columnExpression)
Query And(QueryCondition condition)
```

Adds the condition (predicate) to the current query condition using the AND logical operation.

Parameters

`sourceColumnAlias`	The name of the schema to which to add the predicate.
`sourceAlias`	The alias of the source.
`subSelect`	The data selection subquery that serves as a predicate.
`subSelectQuery`	The subquery that serves as a predicate
`parameter`	The parameter to which to add the predicate.
`columnExpression`	The expression that serves as a predicate.
`condition`	The query condition.

```
QueryCondition Or()
QueryCondition Or(string sourceColumnAlias)
QueryCondition Or(string sourceAlias, string sourceColumnAlias)
QueryCondition Or(Select subSelect)
QueryCondition Or(Query subSelectQuery)
```

```
QueryCondition Or(QueryParameter parameter)
QueryCondition Or(QueryColumnExpression columnExpression)
Query Or(QueryCondition condition)
```

Adds the condition (predicate) to the current query condition using the OR logical operation.

Parameters

`sourceColumnAlias`	The name of the schema to which to add the predicate.
`sourceAlias`	The alias of the source.
`subSelect`	The data select subquery that serves as a predicate.
`subSelectQuery`	The subquery that serves as a predicate
`parameter`	The parameter to which to add the predicate.
`columnExpression`	The expression that serves as a predicate.
`condition`	The query condition.

```
Update Set(string sourceColumnAlias, Select subSelect)
Update Set(string sourceColumnAlias, Query subSelectQuery)
Update Set(string sourceColumnAlias, QueryColumnExpression columnExpression)
Update Set(string sourceColumnAlias, QueryParameter parameter)
```

Adds a `SET` clause to the current query. Required to assign the passed expression or parameter to the column. Returns the current `Update` instance.

Parameters

`sourceColumnAlias`	The column alias.
`subSelect`	The select subquery.
`subSelectQuery`	The subquery.
`columnExpression`	The column expression.
`parameter`	The query parameter.

UpdateSelect class C#

Advanced

`Terrasoft.Core.DB` namespace.

The `Terrasoft.Core.DB.UpdateSelect` class builds queries to modify records in a Creatio database table. The `Terrasoft.Core.DB.Select` class instance serves as a data source. As a result of creating and configuring the instance of the `UpdateSelect` class, Creatio will build an `UPDATE FROM` SQL query.

Constructors

`public UpdateSelect(UserConnection userConnection, string schemaName, string alias)`

Creates a class instance for the specified schema using `UserConnection`.

Parameters

`userConnection`	User connection that the query uses.
`schemaName`	Schema name.
`alias`	Table alias.

Properties

`SourceAlias string`

Alias of the data table to modify.

`SourceExpression Terrasoft.Core.DB.QuerySourceExpression`

`SELECT` query expression.

Methods

`public UpdateSelect From(string schemaName, string alias)`

Adds the `FROM` expression to the query.

Parameters

`schemaName`	Name of the table to modify.
`alias`	Alias of the table to modify.

`public UpdateSelect Set(string sourceColumnAlias, QueryColumnExpression columnExpression)`

Adds the `SET` column expression to the query.

Parameters

`sourceColumnAlias`	Column alias.
`columnExpression`	Expression that contains the column value.

Delete class C#

Advanced

`Terrasoft.Core.DB` namespace.

The `Terrasoft.Core.DB.Delete` class builds queries to delete records from Creatio database tables. As a result of creating and configuring an instance of this class, Creatio will build a `DELETE` SQL query.

Note. View the entire list of methods and properties of the `Delete` class, its parent classes, as well as the interfaces it implements, in the .NET class library documentation.

Constructors

`Delete(UserConnection userConnection)`

Creates a class instance using `UserConnection`.

`Delete(Delete source)`

Creates a class instance that is a clone of the instance passed as the argument.

Properties

`UserConnection` `Terrasoft.Core.UserConnection`

The user connection that the query uses.

`Condition` `Terrasoft.Core.DB.QueryCondition`

The `Where` condition of the query.

`HasCondition` `bool`

Whether the query has a `Where` expression.

`Source` `Terrasoft.Core.DB.ModifyQuerySource`

The query data source.

Methods

```
void ResetCachedSqlText()
```

Clears the cached query text.

```
QueryParameterCollection GetUsingParameters()
```

Returns the parameter collection that the query uses.

```
int Execute()
```

Executes the query. Returns the number of records that the query involved.

```
int Execute(DBExecutor dbExecutor)
```

Executes the query using the `DBExecutor` instance. Returns the number of records that the query involved.

```
QueryCondition Where()
QueryCondition Where(string sourceColumnAlias)
QueryCondition Where(string sourceAlias, string sourceColumnAlias)
QueryCondition Where(Select subSelect)
QueryCondition Where(Query subSelectQuery)
QueryCondition Where(QueryColumnExpression columnExpression)
Query Where(QueryCondition condition)
```

Adds the initial condition to the current query.

Parameters

`sourceColumnAlias`	The alias of the schema to which to add the condition.
`sourceAlias`	The alias of the source.
`subSelect`	The data selection subquery to whose results to add the condition.
`subSelectQuery`	The subquery to whose results to add the condition.
`columnExpression`	The expression to whose results to add the condition.
`condition`	The query condition.

```
QueryCondition And()
QueryCondition And(string sourceColumnAlias)
QueryCondition And(string sourceAlias, string sourceColumnAlias)
QueryCondition And(Select subSelect)
QueryCondition And(Query subSelectQuery)
QueryCondition And(QueryParameter parameter)
QueryCondition And(QueryColumnExpression columnExpression)
Query And(QueryCondition condition)
```

Adds the condition (predicate) to the current query condition using the AND logical operation.

Parameters

`sourceColumnAlias`	The name of the schema to which to add the predicate.
`sourceAlias`	The alias of the source.
`subSelect`	The data selection subquery that serves as a predicate.
`subSelectQuery`	The subquery that serves as a predicate
`parameter`	The parameter to which to add the predicate.
`columnExpression`	The expression that serves as a predicate.
`condition`	The query condition.

```
QueryCondition Or()
QueryCondition Or(string sourceColumnAlias)
QueryCondition Or(string sourceAlias, string sourceColumnAlias)
QueryCondition Or(Select subSelect)
QueryCondition Or(Query subSelectQuery)
```

```
QueryCondition Or(QueryParameter parameter)
QueryCondition Or(QueryColumnExpression columnExpression)
Query Or(QueryCondition condition)
```

Adds the condition (predicate) to the current query condition using the OR logical operation.

Parameters

`sourceColumnAlias`	The name of the schema to which to add the predicate.
`sourceAlias`	The alias of the source.
`subSelect`	The data select subquery that serves as a predicate.
`subSelectQuery`	The subquery that serves as a predicate
`parameter`	The parameter to which to add the predicate.
`columnExpression`	The expression that serves as a predicate.
`condition`	The query condition.

```
Delete From(string schemaName)
Delete From((ModifyQuerySource source)
```

Adds the data source to the current query. Returns the current `Delete` instance.

Parameters

`schemaName`	The schema name (tables, views).
`source`	The data source.

QueryFunction class C#

Advanced

The `Terrasoft.Core.DB.QueryFunction` class implements the expression function.

The following classes implement the expression function:

- `QueryFunction`. The base class of the expression function.
- `AggregationQueryFunction`. Implements the aggregate expression function.
- `IsNullQueryFunction`. Replaces `null` values with the replacing expression.
- `CreateGuidQueryFunction`. Implements the expression function for a new ID.

- `CurrentDateTimeQueryFunction`. Implements the expression function for the current date and time.
- `CoalesceQueryFunction`. Returns the first non-`null` expression from the argument list.
- `DatePartQueryFunction`. Implements the expression function for a part of a `Date/Time` type value.
- `DateAddQueryFunction`. Implements the expression function for the date, calculated by adding the period to the date.
- `DateDiffQueryFunction`. Implements the expression function for the difference between the dates, calculated by subtracting the dates.
- `CastQueryFunction`. Casts the argument expression to the specified data type.
- `UpperQueryFunction`. Converts the argument expression characters to uppercase.
- `CustomQueryFunction`. Implements a custom function.
- `DataLengthQueryFunction`. Calculates bytes used to represent the expression.
- `TrimQueryFunction`. Removes whitespaces from both ends of the expression.
- `LengthQueryFunction`. Returns the expression length.
- `SubstringQueryFunction`. Retrieves a part of the string.
- `ConcatQueryFunction`. Creates a string by merging the string arguments of the function.
- `WindowQueryFunction`. Implements an SQL window function.

QueryFunction class

The `Terrasoft.Core.DB` namespace.

The base class of the expression function.

Note. View the entire list of methods of the `QueryFunction` class, its parent classes, as well as the interfaces it implements, in the .NET class library documentation.

Methods

`static QueryColumnExpression Negate(QueryFunction operand)`

Returns the expression that negates the value of the passed function.

Parameters

`operand`	The expression function.

`static QueryColumnExpression operator -(QueryFunction operand)`

Overloads the operator that negates the passed expression function.

Parameters

`operand`	The expression function.

`static QueryColumnExpression Add(QueryFunction leftOperand, QueryFunction rightOperand)`

Returns the expression that adds the passed expression functions.

Parameters

`leftOperand`	The left operand in the addition operation.
`rightOperand`	The right operand in the addition operation.

`static QueryColumnExpression operator +(QueryFunction leftOperand, QueryFunction rightOperand)`

Overloads the operator that adds two expression functions.

Parameters

`leftOperand`	The left operand in the addition operation.
`rightOperand`	The right operand in the addition operation.

`static QueryColumnExpression Subtract(QueryFunction leftOperand, QueryFunction rightOperand)`

Returns the expression that subtracts the right expression function from the left function.

Parameters

`leftOperand`	The left operand in the subtraction operation.
`rightOperand`	The right operand in the subtraction operation.

`static QueryColumnExpression operator -(QueryFunction leftOperand, QueryFunction rightOperand)`

Overloads the operator that subtracts the right expression function from the left function.

Parameters

`leftOperand`	The left operand in the subtraction operation.
`rightOperand`	The right operand in the subtraction operation.

`static QueryColumnExpression Multiply(QueryFunction leftOperand, QueryFunction rightOperand)`

Returns the expression that multiplies the passed expression functions.

Parameters

`leftOperand`	The left operand in the multiplication operation.
`rightOperand`	The right operand in the multiplication operation.

`static QueryColumnExpression operator *(QueryFunction leftOperand, QueryFunction rightOperand)`

Overloads the operator that multiplies two expression functions.

Parameters

`leftOperand`	The left operand in the multiplication operation.
`rightOperand`	The right operand in the multiplication operation.

`static QueryColumnExpression Divide(QueryFunction leftOperand, QueryFunction rightOperand)`

Returns the expression that divides the left expression function by the right function.

Parameters

`leftOperand`	The left operand in the division operation.
`rightOperand`	The right operand in the division operation.

`static QueryColumnExpression operator /(QueryFunction leftOperand, QueryFunction rightOperand)`

Overloads the operator that divides the expression functions.

Parameters

`leftOperand`	The left operand in the division operation.
`rightOperand`	The right operand in the division operation.

`abstract object Clone()`

Makes a copy of the current `QueryFunction` instance.

`abstract void BuildSqlText(StringBuilder sb, DBEngine dbEngine)`

Generates the query string using the passed `StringBuilder` instance and the `DBEngine` query builder.

Parameters

`sb`	The `StringBuilder` instance required to create the query string.
`dbEngine`	The instance of the database query builder.

`virtual void AddUsingParameters(QueryParameterCollection resultParameters)`

Adds the passed collection of parameters to the function arguments.

Parameters

`resultParameters`	A collection of query parameters to add to the function arguments.

`QueryColumnExpressionCollection GetQueryColumnExpressions()`

Returns the collection of query column expressions for the current query function.

`QueryColumnExpression GetQueryColumnExpression()`

Returns the query column expression for the current query function.

AggregationQueryFunction class

`Terrasoft.Core.DB` namespace.

The class implements the aggregate expression function.

Note. View the entire list of methods and properties of the `AggregationQueryFunction` class, its parent classes, as well as the interfaces it implements, in the .NET class library documentation.

Constructors

`AggregationQueryFunction()`

Initializes a new `AggregationQueryFunction` instance.

`AggregationQueryFunction(AggregationTypeStrict aggregationType, QueryColumnExpression expressior`

Initializes a new `AggregationQueryFunction` instance with the specified type of the aggregate function for the specified column expression.

Parameters

`aggregationType`	The aggregate function type.
`expression`	The column expression to apply the aggregate function.

`AggregationQueryFunction(AggregationTypeStrict aggregationType, IQueryColumnExpressionConvertibl`

Initializes a new `AggregationQueryFunction` instance with the specified type of the aggregate function for the column expression.

Parameters

`aggregationType`	The aggregate function type.
`expression`	The column expression to apply the aggregate function.

`AggregationQueryFunction(AggregationQueryFunction source)`

Initializes a new `AggregationQueryFunction` instance that is a clone of the passed aggregate expression function.

Parameters

`source`	The aggregate function of the `AggregationQueryFunction` expression to clone.

Properties

`AggregationType` `AggregationTypeStrict`

The aggregate function type.

`AggregationEvalType` `AggregationEvalType`

The aggregate function scope.

`Expression` `QueryColumnExpression`

The function argument expression.

Methods

`override void BuildSqlText(StringBuilder sb, DBEngine dbEngine)`

Generates the query string using the specified `StringBuilder` instance and the `DBEngine` query builder.

Parameters

`sb`	The `StringBuilder` instance required to create the query string.
`dbEngine`	The instance of the database query builder.

`override void AddUsingParameters(QueryParameterCollection resultParameters)`

Adds the passed collection of parameters to the function arguments.

Parameters

`resultParameters`	A collection of query parameters to add to the function arguments.

`override object Clone()`

Creates a clone of the current `AggregationQueryFunction` instance.

`AggregationQueryFunction All()`

Sets the [*To All Values*] scope for the current aggregate function.

`AggregationQueryFunction Distinct()`

Sets the [*To Unique Values*] scope for the current aggregate function.

IsNullQueryFunction class

`Terrasoft.Core.DB` namespace.

The class replaces `null` values with the replacing expression.

Note. View the entire list of methods and properties of the `IsNullQueryFunction` class, its parent classes, as well as the interfaces it implements, in the .NET class library documentation.

Constructors

`IsNullQueryFunction()`

Initializes a new `IsNullQueryFunction` instance.

```
IsNullQueryFunction(QueryColumnExpression checkExpression, QueryColumnExpression replacementExpr
IsNullQueryFunction(IQueryColumnExpressionConvertible checkExpression, IQueryColumnExpressionCor
```

Initializes a new `IsNullQueryFunction` instance for the check expression and the replacing expression.

Parameters

`checkExpression`	The expression to compare to `null`.
`replacementExpression`	The expression the function returns if `checkExpression` is `null`.

`IsNullQueryFunction(IsNullQueryFunction source)`

Initializes a new `IsNullQueryFunction` instance that is a clone of the passed expression function.

Parameters

`source`	The aggregate function of the `IsNullQueryFunction` expression to clone.

Properties

`CheckExpression` `QueryColumnExpression`

The function argument expression to compare to `null`.

`ReplacementExpression` `QueryColumnExpression`

The function argument expression to return if the check expression is `null`.

Methods

`override void BuildSqlText(StringBuilder sb, DBEngine dbEngine)`

Generates the query string using the specified `StringBuilder` instance and the `DBEngine` query builder.

Parameters

`sb`	The `StringBuilder` instance required to create the query string.
`dbEngine`	The instance of the database query builder.

`override void AddUsingParameters(QueryParameterCollection resultParameters)`

Adds the passed collection of parameters to the function arguments.

Parameters

`resultParameters`	A collection of query parameters to add to the function arguments.

`override object Clone()`

Creates a clone of the current `IsNullQueryFunction` instance

CreateGuidQueryFunction class

`Terrasoft.Core.DB` namespace.

The class implements the expression function for a new ID.

Note. View the entire list of methods of the `CreateGuidQueryFunction` class, its parent classes, as well as the interfaces it implements, in the .NET class library documentation.

Constructors

`CreateGuidQueryFunction()`

Initializes a new `CreateGuidQueryFunction` instance.

`CreateGuidQueryFunction(CreateGuidQueryFunction source)`

Initializes a new `CreateGuidQueryFunction instance` that is a clone of the passed function.

Parameters

`source`	The `CreateGuidQueryFunction` function to clone.

Methods

`override void BuildSqlText(StringBuilder sb, DBEngine dbEngine)`

Generates the query string using the specified `StringBuilder` instance and the `DBEngine` query builder.

Parameters

`sb`	The `StringBuilder` instance required to create the query string.
`dbEngine`	The instance of the database query builder.

`override object Clone()`

Clones the current `CreateGuidQueryFunction` instance.

CurrentDateTimeQueryFunction class

The `Terrasoft.Core.DB` namespace.

The class implements the expression function for the current date and time.

Note. View the entire list of methods of the `CurrentDateTimeQueryFunction` class, its parent classes, as well as the interfaces it implements, in the .NET class library documentation.

Constructors

`CurrentDateTimeQueryFunction()`

Initializes a new `CurrentDateTimeQueryFunction` instance.

`CurrentDateTimeQueryFunction(CurrentDateTimeQueryFunction source)`

Initializes a new `CurrentDateTimeQueryFunction` instance that is a clone of the passed function.

Parameters

source	The `CurrentDateTimeQueryFunction` function to clone.

Methods

`override void BuildSqlText(StringBuilder sb, DBEngine dbEngine)`

Generates the query string using the specified `StringBuilder` instance and the `DBEngine` query builder.

Parameters

sb	The `StringBuilder` instance required to create the query string.
dbEngine	The instance of the database query builder.

`override object Clone()`

Creates a clone of the current `CurrentDateTimeQueryFunction` instance.

CoalesceQueryFunction class

`Terrasoft.Core.DB` namespace.

The class returns the first non-`null` expression from the argument list.

Note. View the entire list of methods and properties of the `CoalesceQueryFunction` class, its parent classes, as well as the interfaces it implements, in the .NET class library documentation.

Constructors

`CoalesceQueryFunction()`

Initializes a new `CoalesceQueryFunction` instance.

`CoalesceQueryFunction(CoalesceQueryFunction source)`

Initializes a new `CoalesceQueryFunction` instance that is a clone of the passed function.

Parameters

source	The `CoalesceQueryFunction` function to clone.

`CoalesceQueryFunction(QueryColumnExpressionCollection expressions)`

Initializes a new `CoalesceQueryFunction` instance for the passed collection of column expressions.

Parameters

`expressions`	The collection of query column expressions.

`CoalesceQueryFunction(QueryColumnExpression[] expressions)`
`CoalesceQueryFunction(IQueryColumnExpressionConvertible[] expressions)`

Initializes a new `CoalesceQueryFunction` instance for the passed array of column expressions.

Parameters

`expressions`	The array of query column expressions.

Properties

`Expressions QueryColumnExpressionCollection`

The collection of function argument expressions.

Methods

`override void BuildSqlText(StringBuilder sb, DBEngine dbEngine)`

Generates the query string using the specified `StringBuilder` instance and the `DBEngine` query builder.

Parameters

`sb`	The `StringBuilder` instance required to create the query string.
`dbEngine`	The instance of the database query builder.

`override object Clone()`

Creates a clone of the current `CoalesceQueryFunction` instance.

`override void AddUsingParameters(QueryParameterCollection resultParameters)`

Adds the specified parameters to the collection.

Parameters

`resultParameters`	A collection of query parameters to add to the function arguments.

DatePartQueryFunction class

`Terrasoft.Core.DB` namespace.

The class implements the expression function for a part of a `Date/Time` type value.

Note. View the entire list of methods and properties of the `DatePartQueryFunction` class, its parent classes, as well as the interfaces it implements, in the .NET class library documentation.

Constructors

`DatePartQueryFunction()`

Initializes a new instance of the `DatePartQueryFunction` class.

`DatePartQueryFunction(DatePartQueryFunctionInterval interval, QueryColumnExpression expression)`
`DatePartQueryFunction(DatePartQueryFunctionInterval interval, IQueryColumnExpressionConvertible`

Initializes a new `DatePartQueryFunction` instance with the set expression of the `Date/Time` type column and the specified date part.

Parameters

`interval`	The date part.
`expression`	The expression of the `Date/Time` type column.

`DatePartQueryFunction(DatePartQueryFunction source)`

Initializes a new `DatePartQueryFunction` instance that is a clone of the passed function.

Parameters

`source`	The `DatePartQueryFunction` function to clone.

Properties

`Expression` `QueryColumnExpression`

The expression of the function argument.

`Interval` `DatePartQueryFunctionInterval`

The date part that the function returns.

`UseUtcOffset` `bool`

Whether to use the Universal Coordinated Time (UTC) offset relative to the set local time.

`UtcOffset` `int?`

The UTC offset.

Methods

`override void BuildSqlText(StringBuilder sb, DBEngine dbEngine)`

Generates the query string using the specified `StringBuilder` instance and the `DBEngine` query builder.

Parameters

`sb`	The `StringBuilder` instance required to create the query string.
`dbEngine`	The instance of the database query builder.

`override void AddUsingParameters(QueryParameterCollection resultParameters)`

Adds the specified parameters to the collection.

Parameters

`resultParameters`	A collection of query parameters to add to the function arguments.

`override object Clone()`

Creates a clone of the current `DatePartQueryFunction` instance.

DateAddQueryFunction class

`Terrasoft.Core.DB` namespace.

The class implements the expression function for the date, calculated by adding the period to the date.

Note. View the entire list of methods and properties of the `DateAddQueryFunction` class, its parent classes, as well as the interfaces it implements, in the .NET class library documentation.

Constructors

DateAddQueryFunction()

Initializes a new instance of the `DateAddQueryFunction` class.

DateAddQueryFunction(DatePartQueryFunctionInterval interval, int number, QueryColumnExpression e
DateAddQueryFunction(DatePartQueryFunctionInterval interval, IQueryColumnExpressionConvertible r
DateAddQueryFunction(DatePartQueryFunctionInterval interval, int number, IQueryColumnExpressionC

Initializes a `DateAddQueryFunction` instance with the specified parameters.

Parameters

`interval`	The date part to add the period.
`number`	The value to add to `interval`.
`expression`	The expression of the column that contains the original date.

DateAddQueryFunction(DateAddQueryFunction source)

Initializes a `DateAddQueryFunction` instance that is a clone of the passed function.

Parameters

`source`	The instance of the `DateAddQueryFunction` function to clone.

Properties

Expression QueryColumnExpression

The expression of the column that contains the original date.

Interval DatePartQueryFunctionInterval

The date part to add the period.

`Number` `int`

The period to add.

`NumberExpression` `QueryColumnExpression`

The expression that contains the period to add.

Methods

`override void BuildSqlText(StringBuilder sb, DBEngine dbEngine)`

Generates the query string using the specified `StringBuilder` instance and the `DBEngine` query builder.

Parameters

`sb`	The `StringBuilder` instance required to create the query string.
`dbEngine`	The instance of the database query builder.

`override void AddUsingParameters(QueryParameterCollection resultParameters)`

Adds the specified parameters to the collection.

Parameters

`resultParameters`	A collection of query parameters to add to the function arguments.

`override object Clone()`

Creates a clone of the current `DateAddQueryFunction` instance.

DateDiffQueryFunction class

`Terrasoft.Core.DB` namespace.

The class implements the expression function for the difference between the dates, calculated by subtraction.

Note. View the entire list of methods and properties of the `DateDiffQueryFunction` class, its parent classes, as well as the interfaces it implements, in the .NET class library documentation.

Constructors

`DateDiffQueryFunction(DateDiffQueryFunctionInterval interval, QueryColumnExpression startDateExp`
`DateDiffQueryFunction(DateDiffQueryFunctionInterval interval, IQueryColumnExpressionConvertible`

Initializes a `DateDiffQueryFunction` instance with the specified parameters.

Parameters

`interval`	The date interval unit.
`startDateExpression`	The expression of the column that contains the start date.
`endDateExpression`	The expression of the column that contains the end date.

`DateDiffQueryFunction(DateDiffQueryFunction source)`

Initializes a `DateDiffQueryFunction` instance that is a clone of the passed function.

Parameters

`source`	The instance of the `DateDiffQueryFunction` to clone.

Properties

`StartDateExpression` `QueryColumnExpression`

The expression of the column that contains the start date.

`EndDateExpression` `QueryColumnExpression`

The expression of the column that contains the end date.

`Interval` `DateDiffQueryFunctionInterval`

The date difference's measurement unit that the function returns.

Methods

`override void BuildSqlText(StringBuilder sb, DBEngine dbEngine)`

Generates the query string using the specified `StringBuilder` instance and the `DBEngine` query builder.

Parameters

`sb`	The `StringBuilder` instance required to create the query string.
`dbEngine`	The instance of the database query builder.

```
override void AddUsingParameters(QueryParameterCollection resultParameters)
```

Adds the specified parameters to the collection.

Parameters

`resultParameters`	A collection of query parameters to add to the function arguments.

```
override object Clone()
```

Creates a clone of the current `DateDiffQueryFunction` instance.

CastQueryFunction class

`Terrasoft.Core.DB` namespace.

The class casts the argument expression to the specified data type.

Note. View the entire list of methods and properties of the `CastQueryFunction` class, its parent classes, as well as the interfaces it implements, in the .NET class library documentation.

Constructors

```
CastQueryFunction(QueryColumnExpression expression, DBDataValueType castType)
CastQueryFunction(IQueryColumnExpressionConvertible expression, DBDataValueType castType)
```

Initializes a new `CastQueryFunction` instance with the specified column expression and target data type.

Parameters

`expression`	The query column expression.
`castType`	The target data type.

```
CastQueryFunction(CastQueryFunction source)
```

Initializes a new `CastQueryFunction` instance that is a clone of the passed function.

Parameters

`source`	The `CastQueryFunction` function to clone.

Properties

`Expression` `QueryColumnExpression`

The expression of the function argument.

`CastType` `DBDataValueType`

The target data type.

Methods

`override void BuildSqlText(StringBuilder sb, DBEngine dbEngine)`

Generates the query string using the specified `StringBuilder` instance and the `DBEngine` query builder.

Parameters

`sb`	The `StringBuilder` instance required to create the query string.
`dbEngine`	The instance of the database query builder.

`override void AddUsingParameters(QueryParameterCollection resultParameters)`

Adds the specified parameters to the collection.

Parameters

`resultParameters`	A collection of query parameters to add to the function arguments.

`override object Clone()`

Creates a clone of the current `CastQueryFunction` instance.

UpperQueryFunction class

`Terrasoft.Core.DB` namespace.

The class converts the argument expression characters to uppercase.

Note. View the entire list of methods and properties of the `UpperQueryFunction` class, its parent classes, as well as the interfaces it implements, in the .NET class library documentation.

Constructors

`UpperQueryFunction()`

Initializes a new instance of the `UpperQueryFunction` class.

`UpperQueryFunction(QueryColumnExpression expression)`
`UpperQueryFunction(IQueryColumnExpressionConvertible expression)`

Initializes a new `UpperQueryFunction` instance for the specified column expression.

Parameters

`expression`	The query column expression.

`UpperQueryFunction(UpperQueryFunction source)`

Initializes a new `UpperQueryFunction` instance that is a clone of the passed function.

Parameters

`source`	The `UpperQueryFunction` function to clone.

Properties

`Expression`

The expression of the function argument.

Methods

`override void BuildSqlText(StringBuilder sb, DBEngine dbEngine)`

Generates the query string using the specified `StringBuilder` instance and the `DBEngine` query builder.

Parameters

`sb`	The `StringBuilder` instance required to create the query string.
`dbEngine`	The instance of the database query builder.

`override void AddUsingParameters(QueryParameterCollection resultParameters)`

Adds the specified parameters to the collection.

Parameters

`resultParameters`	A collection of query parameters to add to the function arguments.

`override object Clone()`

Creates a clone of the current `UpperQueryFunction` instance.

CustomQueryFunction class

`Terrasoft.Core.DB` namespace.

The class implements a custom function.

Note. View the entire list of methods and properties of the `CustomQueryFunction` class, its parent classes, as well as the interfaces it implements, in the .NET class library documentation.

Constructors

`CustomQueryFunction()`

Initializes a new instance of the `CustomQueryFunction` class.

`CustomQueryFunction(string functionName, QueryColumnExpressionCollection expressions)`

Initializes a new `CustomQueryFunction` instance for the specified function and passed collection of column expressions.

Parameters

`functionName`	The name of the function.
`expressions`	The collection of query column expressions.

```
CustomQueryFunction(string functionName, QueryColumnExpression[] expressions)
CustomQueryFunction(string functionName, IQueryColumnExpressionConvertible[] expressions)
```

Initializes a new `CustomQueryFunction` instance for the specified function and passed array of column expressions.

Parameters

`functionName`	The name of the function.
`expressions`	The array of query column expressions.

```
CustomQueryFunction(CustomQueryFunction source)
```

Initializes a new `CustomQueryFunction` instance that is a clone of the passed function.

Parameters

`source`	The `CustomQueryFunction` function to clone.

Properties

```
Expressions QueryColumnExpressionCollection
```

The collection of function argument expressions.

```
FunctionName string
```

The name of the function.

Methods

```
override void BuildSqlText(StringBuilder sb, DBEngine dbEngine)
```

Generates the query string using the specified `StringBuilder` instance and the `DBEngine` query builder.

Parameters

`sb`	The `StringBuilder` instance required to create the query string.
`dbEngine`	The instance of the database query builder.

`override void AddUsingParameters(QueryParameterCollection resultParameters)`

Adds the specified parameters to the collection.

Parameters

`resultParameters`	A collection of query parameters to add to the function arguments.

`override object Clone()`

Creates a clone of the current `CustomQueryFunction` instance.

DataLengthQueryFunction class

`Terrasoft.Core.DB` namespace.

The class calculates bytes used to represent the expression.

Note. View the entire list of methods and properties of the `DataLengthQueryFunction` class, its parent classes, as well as the interfaces it implements, in the .NET class library documentation.

Constructors

`DataLengthQueryFunction()`

Initializes a new instance of the `DataLengthQueryFunction` class.

`DataLengthQueryFunction(QueryColumnExpression expression)`

Initializes a new `DataLengthQueryFunction` instance for the specified column expression.

Parameters

`expression`	The query column expression.

`DataLengthQueryFunction(IQueryColumnExpressionConvertible columnNameExpression)`

Initializes a new `DataLengthQueryFunction` instance for the specified column expression.

Parameters

`columnNameExpression`	The query column expression.

`DataLengthQueryFunction(DataLengthQueryFunction source)`

Initializes a new `DataLengthQueryFunction` instance that is a clone of the passed function.

Parameters

`source`	The `DataLengthQueryFunction` function to clone.

Properties

`Expression QueryColumnExpression`

The function argument expression.

Methods

`override void BuildSqlText(StringBuilder sb, DBEngine dbEngine)`

Generates the query string using the specified `StringBuilder` instance and the `DBEngine` query builder.

Parameters

`sb`	The `StringBuilder` instance required to create the query string.
`dbEngine`	The instance of the database query builder.

`override void AddUsingParameters(QueryParameterCollection resultParameters)`

Adds the passed collection of parameters to the function arguments.

Parameters

`resultParameters`	A collection of query parameters to add to the function arguments.

`override object Clone()`

Creates a clone of the current `DataLengthQueryFunction` instance.

TrimQueryFunction class

`Terrasoft.Core.DB` namespace.

The class removes whitespaces from both ends of the expression.

Note. View the entire list of methods and properties of the `TrimQueryFunction` class, its parent classes, as well as the interfaces it implements, in the .NET class library documentation.

Constructors

```
TrimQueryFunction(QueryColumnExpression expression)
TrimQueryFunction(IQueryColumnExpressionConvertible expression)
```

Initializes a new `TrimQueryFunction` instance for the specified column expression.

Parameters

`expression`	The query column expression.

```
TrimQueryFunction(TrimQueryFunction source)
```

Initializes a new `TrimQueryFunction` instance that is a clone of the passed function.

Parameters

`source`	The `TrimQueryFunction` function to clone.

Properties

```
Expression QueryColumnExpression
```

The expression of the function argument.

Methods

```
override void BuildSqlText(StringBuilder sb, DBEngine dbEngine)
```

Generates the query string using the specified `StringBuilder` instance and the `DBEngine` query builder.

Parameters

`sb`	The `StringBuilder` instance required to create the query string.
`dbEngine`	The instance of the database query builder.

`override void AddUsingParameters(QueryParameterCollection resultParameters)`

Adds the passed collection of parameters to the function arguments.

Parameters

`resultParameters`	A collection of query parameters to add to the function arguments.

`override object Clone()`

Creates a clone of the current `TrimQueryFunction` instance.

LengthQueryFunction class

`Terrasoft.Core.DB` namespace.

The class returns the length of the expression.

> **Note.** View the entire list of methods and properties of the `LengthQueryFunction` class, its parent classes, as well as the interfaces it implements, in the .NET class library documentation.

Constructors

`LengthQueryFunction()`

Initializes a new instance of the `LengthQueryFunction` class.

`LengthQueryFunction(QueryColumnExpression expression)`
`LengthQueryFunction(IQueryColumnExpressionConvertible expression)`

Initializes a new `LengthQueryFunction` instance for the specified column expression.

Parameters

`expression`	The query column expression.

`LengthQueryFunction(LengthQueryFunction source)`

Initializes a new `LengthQueryFunction` instance that is a clone of the passed function.

Parameters

`source`	The `LengthQueryFunction` function to clone.

Properties

`Expression QueryColumnExpression`

The function argument expression.

Methods

`override void BuildSqlText(StringBuilder sb, DBEngine dbEngine)`

Generates the query string using the specified `StringBuilder` instance and the `DBEngine` query builder.

Parameters

`sb`	The `StringBuilder` instance required to create the query string.
`dbEngine`	The instance of the database query builder.

`override void AddUsingParameters(QueryParameterCollection resultParameters)`

Adds the passed collection of parameters to the function arguments.

Parameters

`resultParameters`	A collection of query parameters to add to the function arguments.

`override object Clone()`

Creates a clone of the current `LengthQueryFunction` instance.

SubstringQueryFunction class

`Terrasoft.Core.DB` namespace.

The class retrieves a part of the string.

Note. View the entire list of methods and properties of the `SubstringQueryFunction` class, its parent classes, as well as the interfaces it implements, in the .NET class library documentation.

Constructors

`SubstringQueryFunction(QueryColumnExpression expression, int start, int length)`
`SubstringQueryFunction(IQueryColumnExpressionConvertible expression, int start, int length)`

Initializes a new `SubstringQueryFunction` instance for the specified column expression, starting position and the substring length.

Parameters

`expression`	The query column expression.
`start`	The starting position of the substring.
`length`	The length of the substring.

`SubstringQueryFunction(SubstringQueryFunction source)`

Initializes a new `SubstringQueryFunction` instance that is a clone of the passed function.

Parameters

`source`	The `SubstringQueryFunction` function to clone.

Properties

`Expression` `QueryColumnExpression`

The expression of the function argument.

`StartExpression` `QueryColumnExpression`

The starting position of the substring.

`LengthExpression` `QueryColumnExpression`

The length of the substring.

Methods

`override void BuildSqlText(StringBuilder sb, DBEngine dbEngine)`

Generates the query string using the specified `StringBuilder` instance and the `DBEngine` query builder.

Parameters

`sb`	The `StringBuilder` instance required to create the query string.
`dbEngine`	The instance of the database query builder.

`override void AddUsingParameters(QueryParameterCollection resultParameters)`

Adds the passed collection of parameters to the function arguments.

Parameters

`resultParameters`	A collection of query parameters to add to the function arguments.

`override object Clone()`

Creates a clone of the current `SubstringQueryFunction` instance.

ConcatQueryFunction class

`Terrasoft.Core.DB` namespace.

The class creates a string by merging the string arguments of the function.

Note. View the entire list of methods and properties of the `ConcatQueryFunction` class, its parent classes, as well as the interfaces it implements, in the .NET class library documentation.

Constructors

`ConcatQueryFunction(QueryColumnExpressionCollection expressions)`

Initializes a new `ConcatQueryFunction` instance for the passed expression collection.

Parameters

`expressions`	The collection of query column expressions.

`ConcatQueryFunction(ConcatQueryFunction source)`

Initializes a new `ConcatQueryFunction` instance that is a clone of the passed function.

Parameters

`source`	The `ConcatQueryFunction` function to clone.

Properties

`Expressions QueryColumnExpressionCollection`

The collection of function argument expressions.

Methods

`override void BuildSqlText(StringBuilder sb, DBEngine dbEngine)`

Generates the query string using the specified `StringBuilder` instance and the `DBEngine` query builder.

Parameters

`sb`	The `StringBuilder` instance required to create the query string.
`dbEngine`	The instance of the database query builder.

`override void AddUsingParameters(QueryParameterCollection resultParameters)`

Adds the passed collection of parameters to the function arguments.

Parameters

`resultParameters`	A collection of query parameters to add to the function arguments.

`override object Clone()`

Creates a clone of the current `ConcatQueryFunction` instance.

WindowQueryFunction class

The `Terrasoft.Core.DB` namespace.

The class implements an SQL window function.

Note. View the entire list of methods and properties of the `WindowQueryFunction` class, its parent classes, as well as the interfaces it implements, in the .NET class library documentation.

Constructors

`WindowQueryFunction(QueryFunction innerFunction)`

Implements an SQL window function.

Parameters

`innerFunction`	The nested function.

`WindowQueryFunction(QueryFunction innerFunction, QueryColumnExpression partitionByExpression = r`

Implements an SQL window function.

Parameters

`innerFunction`	The nested function.
`partitionByExpression`	The expression that separates the query.
`orderByExpression`	The expression that sorts the query.

`WindowQueryFunction(WindowQueryFunction source) : this( source.InnerFunction, source.PartitionBy`

Initializes a new `WindowQueryFunction` instance that is a clone of the passed function.

Parameters

`source`	The `WindowQueryFunction` function to clone.

Properties

`InnerFunction` `QueryFunction`

The function to apply.

`PartitionByExpression` `QueryColumnExpression`

Partition by items.

`OrderByExpression` `QueryColumnExpression`

Sorts by items.

Methods

```
override void BuildSqlText(StringBuilder sb, DBEngine dbEngine)
```

Generates the query string using the specified `StringBuilder` instance and the `DBEngine` query builder.

Parameters

`sb`	The `StringBuilder` instance required to create the query string.
`dbEngine`	The instance of the database query builder.

```
override void AddUsingParameters(QueryParameterCollection resultParameters)
```

Adds the passed collection of parameters to the function arguments.

Parameters

`resultParameters`	A collection of query parameters to add to the function arguments.

```
override object Clone()
```

Creates a clone of the current `WindowQueryFunction` instance.