

# Namespace CoreRelm

## Classes

### [RelmHelper](#)

Provides a collection of helper methods and properties for interacting with relational databases.

# Class RelmHelper

Namespace: [CoreRelm](#)

Assembly: CoreRelm.dll

Provides a collection of helper methods and properties for interacting with relational databases.

```
public class RelmHelper
```

## Inheritance

[object](#) ← RelmHelper

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

## Remarks

The [RelmHelper](#) class offers a variety of static methods and properties to simplify database operations, including connection management, query execution, and data retrieval. It also provides utilities for handling execution errors, managing database contexts, and working with relational models. This class is designed to abstract common database operations, making it easier to interact with MySQL databases and manage data access layers in a consistent and reusable manner.

## Properties

### CurrentContext

Gets or sets the current context for the application.

```
public static IRelmContext? CurrentContext { get; set; }
```

### Property Value

[IRelmContext](#)

## Remarks

This property holds the active context used throughout the application. Ensure that the context is properly initialized before accessing this property.

## HasError

Convenience function to check if there's an error cached.

```
public static bool HasError { get; }
```

### Property Value

[bool](#)

## LastExecutionError

Convenience function to get the last exception message

```
public static string LastExecutionError { get; }
```

### Property Value

[string](#)

## LastExecutionException

Caches the last execution error encountered

```
public static Exception LastExecutionException { get; }
```

### Property Value

[Exception](#)

## RootLoggingDirectory

Gets the root directory where logging files are stored.

```
public static string RootLoggingDirectory { get; }
```

Property Value

[string](#)

## Methods

**BulkTableWrite<T>(IRelmContext, IEnumerable<T>, string, Type, int, bool, bool, bool)**

Writes a collection of data to a database table in bulk, optimizing for performance.

```
public static int BulkTableWrite<T>(IRelmContext relmContext, IEnumerable<T> sourceData,
string tableName = null, Type forceType = null, int batchSize = 100, bool
allowAutoIncrementColumns = false, bool allowPrimaryKeyColumns = false, bool
allowUniqueColumns = false)
```

Parameters

**relmContext** [IRelmContext](#)

The database context used to perform the bulk write operation. Cannot be [null](#).

**sourceData** [IEnumerable](#)<T>

The collection of data to be written to the table. Cannot be [null](#) or empty.

**tableName** [string](#)

The name of the target database table. If [null](#), the table name is inferred from the type T.

**forceType** [Type](#)

An optional type to enforce for the table schema. If [null](#), the schema is inferred from T.

**batchSize** [int](#)

The number of records to write in each batch. Must be greater than 0. Defaults to 100.

## `allowAutoIncrementColumns` [bool](#)

A value indicating whether auto-increment columns in the target table are allowed to be explicitly written to. If [false](#), auto-increment columns are ignored during the write operation.

## `allowPrimaryKeyColumns` [bool](#)

A value indicating whether primary key columns in the target table are allowed to be explicitly written to. If [false](#), primary key columns are ignored during the write operation.

## `allowUniqueColumns` [bool](#)

A value indicating whether unique columns in the target table are allowed to be explicitly written to. If [false](#), unique columns are ignored during the write operation.

## Returns

### [int](#)

The total number of rows successfully written to the database table.

## Type Parameters

### T

The type of the objects in the `sourceData` collection.

## Remarks

This method is designed for high-performance bulk data insertion and may bypass certain database constraints depending on the provided parameters. Use caution when enabling options such as `allowAutoIncrementColumns`, `allowPrimaryKeyColumns`, or `allowUniqueColumns` to avoid violating database integrity.

## `BulkTableWrite<T>(IRelmContext, T, string, Type, int, bool, bool, bool)`

Writes a collection of data to a database table in bulk, optimizing for performance.

```
public static int BulkTableWrite<T>(IRelmContext relmContext, T sourceData, string tableName = null, Type forceType = null, int batchSize = 100, bool allowAutoIncrementColumns = false, bool allowPrimaryKeyColumns = false, bool allowUniqueColumns = false)
```

## Parameters

### **relmContext** [IRelmContext](#)

The database context used to perform the bulk write operation. Cannot be [null](#).

### **sourceData** T

The data to be written to the table. This can be a collection of objects or a single object. Cannot be [null](#).

### **tableName** [string](#)

The name of the target database table. If [null](#), the table name is inferred from the type T.

### **forceType** [Type](#)

An optional type to enforce for the operation. If [null](#), the type is inferred from T.

### **batchSize** [int](#)

The number of rows to write in each batch. Must be greater than 0. Defaults to 100.

### **allowAutoIncrementColumns** [bool](#)

A value indicating whether auto-increment columns are allowed to be included in the bulk write operation. If [false](#), auto-increment columns are excluded.

### **allowPrimaryKeyColumns** [bool](#)

A value indicating whether primary key columns are allowed to be included in the bulk write operation. If [false](#), primary key columns are excluded.

### **allowUniqueColumns** [bool](#)

A value indicating whether unique columns are allowed to be included in the bulk write operation. If [false](#), unique columns are excluded.

## Returns

### [int](#)

The total number of rows successfully written to the database.

## Type Parameters

The type of the data to be written. This can be a collection or a single object.

## Remarks

This method is designed for high-performance bulk data insertion. It allows fine-grained control over which columns are included in the operation, such as auto-increment, primary key, and unique columns. Use the optional parameters to customize the behavior as needed.

## **BulkTableWrite<T>(IRelmQuickContext, IEnumerable<T>, string, Type, int, bool, bool, bool)**

Writes a collection of data to a database table in bulk, optimizing for performance.

```
public static int BulkTableWrite<T>(IRelmQuickContext relmQuickContext, IEnumerable<T>
sourceData, string tableName = null, Type forceType = null, int batchSize = 100, bool
allowAutoIncrementColumns = false, bool allowPrimaryKeyColumns = false, bool
allowUniqueColumns = false)
```

## Parameters

### **relmQuickContext [IRelmQuickContext](#)**

The database context used to execute the bulk write operation. Cannot be [null](#).

### **sourceData [IEnumerable<T>](#)**

The collection of data to be written to the table. Cannot be [null](#) or empty.

### **tableName [string](#)**

The name of the target database table. If [null](#), the table name is inferred from the type **T**.

### **forceType [Type](#)**

An optional type to enforce for the table schema. If [null](#), the schema is inferred from **T**.

### **batchSize [int](#)**

The number of rows to write in each batch. Must be greater than 0. Defaults to 100.

### **allowAutoIncrementColumns [bool](#)**

A value indicating whether auto-increment columns in the target table are allowed to be written to. If [false](#), auto-increment columns are excluded from the operation.

#### allowPrimaryKeyColumns [bool](#)

A value indicating whether primary key columns in the target table are allowed to be written to. If [false](#), primary key columns are excluded from the operation.

#### allowUniqueColumns [bool](#)

A value indicating whether unique columns in the target table are allowed to be written to. If [false](#), unique columns are excluded from the operation.

### Returns

#### [int](#)

The total number of rows successfully written to the database.

### Type Parameters

#### T

The type of the objects in the [sourceData](#) collection.

### Remarks

This method is designed for high-performance bulk insert operations. It is the caller's responsibility to ensure that the data in [sourceData](#) conforms to the schema of the target table.

## BulkTableWrite<T>(IRelmQuickContext, T, string, Type, int, bool, bool, bool)

Writes data in bulk to a database table, with options to customize the operation.

```
public static int BulkTableWrite<T>(IRelmQuickContext relmQuickContext, T sourceData, string  
tableName = null, Type forceType = null, int batchSize = 100, bool allowAutoIncrementColumns  
= false, bool allowPrimaryKeyColumns = false, bool allowUniqueColumns = false)
```

### Parameters

#### [relmQuickContext](#) [IRelmQuickContext](#)

The database context used to execute the bulk write operation. Cannot be [null](#).

## sourceData [T](#)

The data to be written to the table. This can be a collection or a single object. Cannot be [null](#).

## tableName [string](#)

The name of the target database table. If [null](#), the table name is inferred from the type [T](#).

## forceType [Type](#)

An optional type to enforce for the operation. If [null](#), the type is inferred from [T](#).

## batchSize [int](#)

The number of rows to write in each batch. Must be greater than 0. Defaults to 100.

## allowAutoIncrementColumns [bool](#)

A value indicating whether auto-increment columns in the target table are allowed to be explicitly written. If [false](#), auto-increment columns are ignored during the write operation.

## allowPrimaryKeyColumns [bool](#)

A value indicating whether primary key columns in the target table are allowed to be explicitly written. If [false](#), primary key columns are ignored during the write operation.

## allowUniqueColumns [bool](#)

A value indicating whether unique columns in the target table are allowed to be explicitly written. If [false](#), unique columns are ignored during the write operation.

## Returns

### [int](#)

The total number of rows successfully written to the database table.

## Type Parameters

### [T](#)

The type of the data source to be written to the table.

## Remarks

This method provides a high-performance way to insert or update large amounts of data in a database table. The behavior of the operation can be customized using the optional parameters to control how specific columns (e.g., auto-increment, primary key, or unique columns) are handled.

## BulkTableWrite<T>(MySqlConnection, IEnumerable<T>, string, MySqlTransaction, Type, int, bool, bool, bool)

Writes a collection of data to a database table in bulk, optimizing for performance.

```
public static int BulkTableWrite<T>(MySqlConnection establishedConnection, IEnumerable<T>
sourceData, string tableName = null, MySqlTransaction sqlTransaction = null, Type forceType
= null, int batchSize = 100, bool allowAutoIncrementColumns = false, bool
allowPrimaryKeyColumns = false, bool allowUniqueColumns = false)
```

## Parameters

### establishedConnection MySqlConnection

An open MySql.Data.MySqlClient.MySqlConnection to the database where the data will be written. The connection must remain open for the duration of the operation.

### sourceData [IEnumerable](#)<T>

The collection of data to be written to the database table. Each item in the collection represents a row to be inserted.

### tableName [string](#)

The name of the target database table. If `null`, the table name is inferred from the type `T`.

### sqlTransaction MySqlTransaction

An optional MySql.Data.MySqlClient.MySqlTransaction to associate with the bulk write operation. If `null`, the operation is performed without a transaction.

### forceType [Type](#)

An optional [Type](#) to enforce a specific type mapping for the data being written. If `null`, the type is inferred from `T`.

### batchSize [int](#)

The number of rows to write in each batch. Defaults to 100. Larger batch sizes may improve performance but require more memory.

#### allowAutoIncrementColumns [bool](#)

A value indicating whether columns with auto-increment constraints are included in the bulk write operation. If `true`, auto-increment columns are included; otherwise, they are excluded.

#### allowPrimaryKeyColumns [bool](#)

A value indicating whether primary key columns are included in the bulk write operation. If `true`, primary key columns are included; otherwise, they are excluded.

#### allowUniqueColumns [bool](#)

A value indicating whether unique constraint columns are included in the bulk write operation. If `true`, unique columns are included; otherwise, they are excluded.

### Returns

#### [int](#)

The total number of rows successfully written to the database table.

### Type Parameters

#### T

The type of the objects in the `sourceData` collection.

### Remarks

This method is designed for high-performance bulk data insertion and should be used when inserting large datasets. Ensure that the `establishedConnection` is open and valid before calling this method. If `tableName` is not provided, the method attempts to infer the table name from the type `T`.

## BulkTableWrite<T>(MySqlConnection, T, string, MySqlTransaction, Type, int, bool, bool, bool)

Writes a collection of data to a database table in bulk, optimizing for performance.

```
public static int BulkTableWrite<T>(MySqlConnection establishedConnection, T sourceData,
string tableName = null, MySqlTransaction sqlTransaction = null, Type forceType = null, int
```

```
batchSize = 100, bool allowAutoIncrementColumns = false, bool allowPrimaryKeyColumns =  
false, bool allowUniqueColumns = false)
```

## Parameters

**establishedConnection** MySqlConnection

An open MySql.Data.MySqlClient.MySqlConnection to the database where the data will be written. The connection must remain open for the duration of the operation.

**sourceData** T

The data to be written to the table. This can be a collection of objects or a DataTable.

**tableName** string ↗

The name of the target database table. If null, the table name is inferred from the type of T.

**sqlTransaction** MySqlTransaction

An optional MySql.Data.MySqlClient.MySqlTransaction to associate with the bulk write operation. If null, the operation is performed outside of a transaction.

**forceType** Type ↗

An optional Type ↗ to enforce a specific type for the data being written. If null, the type is inferred from T.

**batchSize** int ↗

The number of rows to write in each batch. Defaults to 100. Larger batch sizes may improve performance but require more memory.

**allowAutoIncrementColumns** bool ↗

Indicates whether auto-increment columns in the target table are allowed to be written. Defaults to false ↗.

**allowPrimaryKeyColumns** bool ↗

Indicates whether primary key columns in the target table are allowed to be written. Defaults to false ↗.

**allowUniqueColumns** bool ↗

Indicates whether unique columns in the target table are allowed to be written. Defaults to [false](#).

## Returns

[int](#)

The number of rows successfully written to the database table.

## Type Parameters

T

The type of the data source. Typically a collection of objects or a DataTable.

## Remarks

This method is designed for high-performance bulk data insertion. It is the caller's responsibility to ensure that the data in `sourceData` matches the schema of the target table.

If `allowAutoIncrementColumns`, `allowPrimaryKeyColumns`, or `allowUniqueColumns` are set to [true](#), the caller must ensure that the data does not violate database constraints.

The method does not automatically handle schema mismatches or data validation. Ensure that the data types and column mappings align with the target table.

## BulkTableWrite<T>(Enum, IEnumerable<T>, string, Type, int, bool, bool, bool)

Writes a collection of data to a database table in bulk, using the specified connection and options.

```
public static int BulkTableWrite<T>(Enum connectionName, IEnumerable<T> sourceData, string  
tableName = null, Type forceType = null, int batchSize = 100, bool allowAutoIncrementColumns  
= false, bool allowPrimaryKeyColumns = false, bool allowUniqueColumns = false)
```

## Parameters

`connectionName` [Enum](#)

The name of the database connection to use. This must be an enumeration value representing a valid connection.

`sourceData` [IEnumerable](#)<T>

The collection of data to write to the database table. Each item in the collection represents a row to be inserted.

#### tableName [string](#)

The name of the target database table. If [null](#), the table name is inferred from the type [T](#).

#### forceType [Type](#)

An optional type to enforce for the operation. If specified, the operation will treat the data as this type.

#### batchSize [int](#)

The number of rows to write in each batch. Defaults to 100. Larger batch sizes may improve performance but require more memory.

#### allowAutoIncrementColumns [bool](#)

A value indicating whether auto-increment columns in the target table are allowed to be explicitly written. Defaults to [false](#).

#### allowPrimaryKeyColumns [bool](#)

A value indicating whether primary key columns in the target table are allowed to be explicitly written. Defaults to [false](#).

#### allowUniqueColumns [bool](#)

A value indicating whether unique columns in the target table are allowed to be explicitly written. Defaults to [false](#).

Returns

#### [int](#)

The total number of rows successfully written to the database.

Type Parameters

#### [T](#)

The type of the objects in the [sourceData](#) collection.

Remarks

This method provides a high-performance way to insert large amounts of data into a database table. It supports optional configuration for handling auto-increment, primary key, and unique columns.

## BulkTableWrite<T>(Enum, T, string, Type, int, bool, bool, bool)

Writes a collection of data to a database table in bulk, optimizing for high performance.

```
public static int BulkTableWrite<T>(Enum connectionName, T sourceData, string tableName = null, Type forceType = null, int batchSize = 100, bool allowAutoIncrementColumns = false, bool allowPrimaryKeyColumns = false, bool allowUniqueColumns = false)
```

### Parameters

#### connectionName [Enum](#)

The name of the database connection to use. This must correspond to a valid connection configuration.

#### sourceData T

The data to be written to the table. This can be a collection or a single object, depending on the implementation.

#### tableName [string](#)

The name of the target database table. If [null](#), the table name is inferred from the type T.

#### forceType [Type](#)

An optional type to enforce when writing the data. If specified, the data will be treated as this type during the operation.

#### batchSize [int](#)

The number of rows to write in each batch. Defaults to 100. Larger batch sizes may improve performance but require more memory.

#### allowAutoIncrementColumns [bool](#)

A value indicating whether columns with auto-increment constraints are allowed to be written. Defaults to [false](#).

#### allowPrimaryKeyColumns [bool](#)

A value indicating whether primary key columns are allowed to be written. Defaults to [false](#).

#### allowUniqueColumns [bool](#)

A value indicating whether unique columns are allowed to be written. Defaults to [false](#).

## Returns

#### [int](#)

The number of rows successfully written to the database.

## Type Parameters

#### T

The type of the data objects to be written to the table.

## Remarks

This method is designed for scenarios where large amounts of data need to be inserted into a database efficiently. It uses bulk operations to minimize database round-trips and improve performance. Ensure that the database schema matches the structure of the data being written to avoid runtime errors.

## DoDatabaseWork(IRelmContext, string, Dictionary<string, object>, bool, bool)

Executes a database operation using the specified query and parameters.

```
public static void DoDatabaseWork(IRelmContext relmContext, string query, Dictionary<string, object> parameters = null, bool throwException = true, bool useTransaction = false)
```

## Parameters

#### relmContext [IRelmContext](#)

The database context used to execute the operation. Cannot be [null](#).

#### query [string](#)

The SQL query to execute. Cannot be [null](#) or empty.

**parameters** [Dictionary](#)<[string](#), [object](#)>

An optional dictionary of parameter names and values to be used in the query. If [null](#), no parameters will be passed.

**throwException** [bool](#)

A value indicating whether an exception should be thrown if the operation fails. The default value is [true](#).

**useTransaction** [bool](#)

A value indicating whether the operation should be executed within a transaction. The default value is [false](#).

## Remarks

This method delegates the database operation to an internal helper. Ensure that the [realmContext](#) is properly configured and that the [query](#) is valid for the target database.

## DoDatabaseWork(IRealmContext, string, Func<MySqlCommand, object>, bool, bool)

Executes a database operation using the specified query and callback function.

```
public static void DoDatabaseWork(IRealmContext realmContext, string query, Func<MySqlCommand, object> actionCallback, bool throwException = true, bool useTransaction = false)
```

## Parameters

**realmContext** [IRealmContext](#)

The database context used to establish the connection. Cannot be [null](#).

**query** [string](#)

The SQL query to be executed. Cannot be [null](#) or empty.

**actionCallback** [Func](#)<MySqlCommand, [object](#)>

A callback function that processes the MySql.Data.MySqlClient.MySqlCommand object and returns a result. The callback cannot be [null](#).

### **throwException** [bool](#)

A value indicating whether exceptions should be thrown if an error occurs during the operation. The default value is [true](#).

### **useTransaction** [bool](#)

A value indicating whether the operation should be executed within a database transaction. The default value is [false](#).

## Remarks

This method delegates the database operation to a helper class. Ensure that the provided `relmContext` is properly configured and that the `query` is valid for the target database.

## DoDatabaseWork(IRelmQuickContext, string, Dictionary<string, object>, bool, bool)

Executes a database operation using the specified query and parameters.

```
public static void DoDatabaseWork(IRelmQuickContext relmQuickContext, string query,
Dictionary<string, object> parameters = null, bool throwException = true, bool
useTransaction = false)
```

## Parameters

### **relmQuickContext** [IRelmQuickContext](#)

The database context used to execute the operation. This cannot be [null](#).

### **query** [string](#)

The SQL query to execute. This cannot be [null](#) or empty.

### **parameters** [Dictionary](#)<[string](#), [object](#)>

An optional dictionary of parameter names and values to be used in the query. If [null](#), no parameters are applied.

### **throwException** [bool](#)

A value indicating whether an exception should be thrown if the operation fails. The default is [true](#).

## **useTransaction** [bool](#)

A value indicating whether the operation should be executed within a transaction. The default is [false](#).

## Remarks

This method delegates the database operation to an internal helper. Ensure that the provided query and parameters are valid for the target database.

## **DoDatabaseWork(IRelmQuickContext, string, Func<MySqlCommand, object>, bool, bool)**

Executes a database operation using the specified query and callback function.

```
public static void DoDatabaseWork(IRelmQuickContext relmQuickContext, string query,  
Func<MySqlCommand, object> actionCallback, bool throwException = true, bool useTransaction  
= false)
```

## Parameters

### **relmQuickContext** [IRelmQuickContext](#)

The database context used to establish the connection and manage the operation.

### **query** [string](#)

The SQL query to execute. Must be a valid SQL statement.

### **actionCallback** [Func](#)<MySqlCommand, [object](#)>

A callback function that processes the MySql.Data.MySqlClient.MySqlCommand object and returns a result.

### **throwException** [bool](#)

A value indicating whether to throw an exception if an error occurs during the operation. [true](#) to throw exceptions; otherwise, [false](#).

### **useTransaction** [bool](#)

A value indicating whether the operation should be executed within a database transaction. [true](#) to use a transaction; otherwise, [false](#).

## Remarks

This method delegates the database operation to [DoDatabaseWork\(Enum, string, Dictionary<string, object>, bool, bool, MySqlTransaction, bool\)](#). Ensure that the `relmQuickContext` is properly configured before calling this method.

## DoDatabaseWork(MySqlConnection, string, Dictionary<string, object>, bool, bool, MySqlTransaction)

Executes a database query using the provided connection and optional parameters.

```
public static void DoDatabaseWork(MySqlConnection establishedConnection, string query,
Dictionary<string, object> parameters = null, bool throwException = true, bool
useTransaction = false, MySqlTransaction sqlTransaction = null)
```

### Parameters

**establishedConnection** MySqlConnection

An open and established MySql.Data.MySqlClient.MySqlConnection to the database. The connection must be valid and open before calling this method.

**query** [string](#)

The SQL query to execute. This must be a valid SQL statement supported by the database.

**parameters** [Dictionary<string, object>](#)

An optional dictionary of parameter names and values to be used in the query. If null, no parameters are applied.

**throwException** [bool](#)

A boolean value indicating whether to throw an exception if an error occurs. If [true](#), exceptions are thrown; otherwise, errors are suppressed.

**useTransaction** [bool](#)

A boolean value indicating whether to execute the query within a transaction. If [true](#), a transaction is used; otherwise, the query is executed without a transaction.

**sqlTransaction** MySqlTransaction

An optional MySql.Data.MySqlClient.MySqlTransaction to use for the query. If provided, this transaction is used instead of creating a new one. Ignored if `useTransaction` is [false](#).

## Remarks

This method delegates the execution to [DoDatabaseWork\(Enum, string, Dictionary<string, object>, bool, bool, MySqlTransaction, bool\)](#). Ensure that the connection is properly managed and disposed of after use.

## DoDatabaseWork(MySqlConnection, string, Func<MySqlCommand, object>, bool, bool, MySqlTransaction)

Executes a database operation using the provided connection, query, and callback function.

```
public static void DoDatabaseWork(MySqlConnection establishedConnection, string query,
Func<MySqlCommand, object> actionCallback, bool throwException = true, bool useTransaction =
false, MySqlTransaction sqlTransaction = null)
```

## Parameters

### establishedConnection MySqlConnection

An open and valid MySql.Data.MySqlClient.MySqlConnection to the database. The connection must be established before calling this method.

### query [string](#)

The SQL query to execute. This query is passed to the callback function for processing.

### actionCallback [Func](#)<MySqlCommand, object>

A callback function that processes the MySql.Data.MySqlClient.MySqlCommand created from the query. The function should return an object representing the result of the operation.

### throwException [bool](#)

A value indicating whether to throw exceptions if an error occurs during the operation. If [true](#), exceptions will be propagated; otherwise, errors will be suppressed. The default value is [true](#).

### useTransaction [bool](#)

A value indicating whether the operation should be executed within a transaction. If [true](#), a transaction will be used unless `sqlTransaction` is provided. The default value is [false](#).

### `sqlTransaction` MySqlTransaction

An optional MySql.Data.MySqlClient.MySqlTransaction to use for the operation. If provided, this transaction will be used instead of creating a new one.

## Remarks

This method delegates the database operation to a helper method and provides flexibility for executing queries with or without transactions. Ensure that the connection is open and valid before calling this method. If `useTransaction` is [true](#) and `sqlTransaction` is not provided, a new transaction will be created and committed automatically.

## DoDatabaseWork(Enum, string, Dictionary<string, object>, bool, bool, MySqlTransaction, bool)

Executes a database operation using the specified connection, query, and parameters.

```
public static void DoDatabaseWork(Enum connectionName, string query, Dictionary<string, object> parameters = null, bool throwException = true, bool useTransaction = false, MySqlTransaction sqlTransaction = null, bool allowUserVariables = false)
```

## Parameters

### `connectionName` [Enum](#)

The name of the database connection to use. This must be a valid value from the specified enumeration.

### `query` [string](#)

The SQL query to execute. The query can include parameter placeholders.

### `parameters` [Dictionary](#)<[string](#), [object](#)>

An optional dictionary of parameter names and values to be used in the query. If null, no parameters will be passed to the query.

### `throwException` [bool](#)

A boolean value indicating whether to throw an exception if an error occurs. If [true](#), exceptions will be thrown; otherwise, errors will be suppressed.

#### **useTransaction** [bool](#)

A boolean value indicating whether to execute the query within a transaction. If [true](#), a transaction will be used; otherwise, the query will execute without a transaction.

#### **sqlTransaction** MySqlTransaction

An optional MySql.Data.MySqlClient.MySqlTransaction object to use for the operation. If provided, the operation will use this transaction; otherwise, a new transaction may be created if [useTransaction](#) is [true](#).

#### **allowUserVariables** [bool](#)

A boolean value indicating whether user-defined variables are allowed in the query. If [true](#), user variables are permitted; otherwise, they are not.

## Remarks

This method delegates the database operation to the [DatabaseWorkHelper.DoDatabaseWork](#) method. Ensure that the [connectionName](#) corresponds to a valid database connection and that the [query](#) is properly formatted for the target database.

## DoDatabaseWork(Enum, string, Func<MySqlCommand, object>, bool, bool, MySqlTransaction, bool)

Executes a database operation using the specified query and callback function.

```
public static void DoDatabaseWork(Enum connectionName, string query, Func<MySqlCommand, object> actionCallback, bool throwException = true, bool useTransaction = false, MySqlTransaction sqlTransaction = null, bool allowUserVariables = false)
```

## Parameters

#### **connectionName** [Enum](#)

The name of the database connection to use. This must be a valid connection identifier.

#### **query** [string](#)

The SQL query to execute. Cannot be null or empty.

**actionCallback** `Func<MySqlCommand, object>`

A callback function that processes the MySql.Data.MySqlClient.MySqlCommand object and returns a result. Cannot be null.

**throwException** `bool`

Specifies whether to throw an exception if an error occurs. If `true`, exceptions will be thrown; otherwise, errors will be suppressed.

**useTransaction** `bool`

Specifies whether the operation should be executed within a transaction. If `true`, a transaction will be used.

**sqlTransaction** `MySqlTransaction`

An optional MySql.Data.MySqlClient.MySqlTransaction object to use for the operation. If provided, this transaction will be used instead of creating a new one.

**allowUserVariables** `bool`

Specifies whether user-defined variables are allowed in the SQL query. If `true`, user variables are permitted.

## Remarks

This method delegates the database operation to an internal helper and provides flexibility for executing queries with or without transactions. Ensure that the `connectionName` corresponds to a valid database connection and that the `query` is properly formatted.

## DoDatabaseWork<T>(IRelmContext, string, Dictionary<string, object>, bool, bool)

Executes a database operation using the specified query and parameters, and optionally within a transaction.

```
public static T DoDatabaseWork<T>(IRelmContext relmContext, string query, Dictionary<string, object> parameters = null, bool throwException = true, bool useTransaction = false)
```

## Parameters

### `realmContext` [IRelmContext](#)

The database context used to execute the operation. Cannot be [null](#).

### `query` [string](#)

The SQL query to execute. Cannot be [null](#) or empty.

### `parameters` [Dictionary](#)<[string](#), [object](#)>

An optional dictionary of parameter names and values to be used in the query. Can be [null](#) if no parameters are required.

### `throwException` [bool](#)

A value indicating whether an exception should be thrown if the operation fails. If [true](#), exceptions will be propagated; otherwise, the method may handle errors internally.

### `useTransaction` [bool](#)

A value indicating whether the operation should be executed within a transaction. If [true](#), the operation will be wrapped in a transaction.

## Returns

### `T`

The result of the database operation, of type `T`.

## Type Parameters

### `T`

The type of the result returned by the database operation.

## Remarks

This method provides a high-level abstraction for executing database operations. The behavior of the operation, including error handling and transaction usage, can be customized using the `throwException` and `useTransaction` parameters.

## DoDatabaseWork<T>(IRelmContext, string, Func<MySqlCommand, object>, bool, bool)

Executes a database operation using the specified query and callback function.

```
public static T DoDatabaseWork<T>(IRelmContext relmContext, string query, Func<MySqlCommand, object> actionCallback, bool throwException = true, bool useTransaction = false)
```

### Parameters

`relmContext` [IRelmContext](#)

The database context used to establish the connection and manage the operation. Cannot be [null](#).

`query` [string](#)

The SQL query to execute. Cannot be [null](#) or empty.

`actionCallback` [Func](#)<MySqlCommand, [object](#)>

A callback function that defines the operation to perform using the MySql.Data.MySqlClient.MySqlCommand object. Cannot be [null](#).

`throwException` [bool](#)

A value indicating whether exceptions encountered during the operation should be thrown. If [true](#), exceptions will be propagated to the caller; otherwise, they will be suppressed.

`useTransaction` [bool](#)

A value indicating whether the operation should be executed within a database transaction. If [true](#), the operation will be wrapped in a transaction.

### Returns

`T`

The result of the database operation, as defined by the `T` type.

### Type Parameters

`T`

The type of the result returned by the database operation.

## Remarks

This method provides a flexible way to execute database operations by allowing the caller to specify a query and a callback function that defines the operation to perform. The operation can optionally be executed within a transaction, and exceptions can be suppressed based on the `throwException` parameter.

## DoDatabaseWork<T>(IRelmQuickContext, string, Dictionary<string, object>, bool, bool)

Executes a database operation using the specified query and parameters, and optionally within a transaction.

```
public static T DoDatabaseWork<T>(IRelmQuickContext relmQuickContext, string query,  
Dictionary<string, object> parameters = null, bool throwException = true, bool  
useTransaction = false)
```

## Parameters

`relmQuickContext` [IRelmQuickContext](#)

The database context used to execute the operation. Cannot be [null](#).

`query` [string](#)

The SQL query to execute. Cannot be [null](#) or empty.

`parameters` [Dictionary](#)<[string](#), [object](#)>

An optional dictionary of parameters to include in the query. Keys represent parameter names, and values represent their corresponding values. Defaults to [null](#).

`throwException` [bool](#)

A value indicating whether an exception should be thrown if the operation fails. Defaults to [true](#).

`useTransaction` [bool](#)

A value indicating whether the operation should be executed within a transaction. Defaults to [false](#).

## Returns

T

The result of the database operation, cast to the specified type T.

## Type Parameters

T

The type of the result expected from the database operation.

## DoDatabaseWork<T>(IRelmQuickContext, string, Func<MySqlCommand, object>, bool, bool)

Executes a database operation using the specified query and callback function.

```
public static T DoDatabaseWork<T>(IRelmQuickContext relmQuickContext, string query,  
Func<MySqlCommand, object> actionCallback, bool throwException = true, bool useTransaction  
= false)
```

## Parameters

relmQuickContext [IRelmQuickContext](#)

The database context used to establish the connection and manage the operation. Cannot be [null](#).

query [string](#)

The SQL query to be executed. Cannot be [null](#) or empty.

actionCallback [Func](#)<MySqlCommand, object>

A callback function that defines the operation to perform using the MySql.Data.MySqlClient.MySqlCommand object. Cannot be [null](#).

throwException [bool](#)

A value indicating whether exceptions should be thrown if an error occurs during the operation. If [true](#), exceptions will be propagated; otherwise, errors will be suppressed.

useTransaction [bool](#)

A value indicating whether the operation should be executed within a database transaction. If [true](#), the operation will be wrapped in a transaction.

Returns

T

The result of the operation, as returned by the `actionCallback`.

Type Parameters

T

The type of the result returned by the callback function.

Remarks

This method provides a flexible way to execute database operations by allowing the caller to specify a query and a custom callback function to process the database command. The caller can optionally enable transaction support and control whether exceptions are thrown or suppressed.

**DoDatabaseWork<T>(MySqlConnection, string, Dictionary<string, object>, bool, bool, MySqlTransaction)**

Executes a database query and returns the result as the specified type.

```
public static T DoDatabaseWork<T>(MySqlConnection establishedConnection, string query,
Dictionary<string, object> parameters = null, bool throwException = true, bool
useTransaction = false, MySqlTransaction sqlTransaction = null)
```

Parameters

**establishedConnection** MySqlConnection

An open and valid MySql.Data.MySqlClient.MySqlConnection to the database.

**query** [string](#)

The SQL query to execute.

**parameters** [Dictionary](#)<[string](#), [object](#)>

An optional dictionary of parameter names and values to be used in the query. If null, no parameters are added.

### `throwException` `bool`

A boolean value indicating whether to throw an exception if an error occurs. If `true`, exceptions are thrown; otherwise, errors are suppressed.

### `useTransaction` `bool`

A boolean value indicating whether to execute the query within a transaction. If `true`, a transaction is used.

### `sqlTransaction` `MySqlTransaction`

An optional `MySQL.Data.MySqlClient.MySqlTransaction` to use for the query. If null, a new transaction is created if `useTransaction` is `true`.

## Returns

### `T`

The result of the query, cast to the specified type `T`.

## Type Parameters

### `T`

The type of the result expected from the query.

## Remarks

This method is a wrapper for executing database queries with optional parameterization, transaction support, and error handling. Ensure the connection is open before calling this method.

## `DoDatabaseWork<T>(MySqlConnection, string, Func<MySqlCommand, object>, bool, bool, MySqlTransaction)`

Executes a database operation using the provided query and callback function.

```
public static T DoDatabaseWork<T>(MySqlConnection establishedConnection, string query,
Func<MySqlCommand, object> actionCallback, bool throwException = true, bool useTransaction =
false, MySqlTransaction sqlTransaction = null)
```

## Parameters

**establishedConnection** MySqlConnection

An open and valid MySql.Data.MySqlClient.MySqlConnection to the database. The connection must already be established.

**query** [string](#)

The SQL query to be executed. This query is passed to the callback function for execution.

**actionCallback** [Func](#)<MySqlCommand, [object](#)>

A callback function that defines the operation to perform using the MySql.Data.MySqlClient.MySqlCommand created from the query.

**throwException** [bool](#)

A value indicating whether exceptions should be thrown if an error occurs during the operation. If [true](#), exceptions are propagated to the caller; otherwise, they are suppressed.

**useTransaction** [bool](#)

A value indicating whether the operation should be executed within a transaction. If [true](#), a transaction is used unless **sqlTransaction** is provided.

**sqlTransaction** MySqlTransaction

An optional MySql.Data.MySqlClient.MySqlTransaction to use for the operation. If provided, this transaction is used regardless of the value of **useTransaction**.

Returns

**T**

The result of type **T** as returned by the **actionCallback**.

Type Parameters

**T**

The type of the result returned by the callback function.

Remarks

This method provides a flexible way to execute database operations by allowing the caller to define the specific action to perform via the **actionCallback**. If **useTransaction** is [true](#) and no **sqlTransaction** is

provided, a new transaction is created for the operation.

## DoDatabaseWork<T>(Enum, string, Dictionary<string, object>, bool, bool, MySqlTransaction, bool)

Executes a database operation and returns the result of the specified type.

```
public static T DoDatabaseWork<T>(Enum connectionName, string query, Dictionary<string, object> parameters = null, bool throwException = true, bool useTransaction = false, MySqlTransaction sqlTransaction = null, bool allowUserVariables = false)
```

### Parameters

#### connectionName [Enum](#)

The name of the database connection to use. Must be a valid enumeration value representing a configured connection.

#### query [string](#)

The SQL query to execute. Cannot be null or empty.

#### parameters [Dictionary](#)<string, object>

An optional dictionary of parameters to include in the query. Keys represent parameter names, and values represent their corresponding values. Can be null if no parameters are needed.

#### throwException [bool](#)

Indicates whether an exception should be thrown if an error occurs during the operation. If [true](#), exceptions will be propagated; otherwise, errors will be suppressed.

#### useTransaction [bool](#)

Specifies whether the operation should be executed within a transaction. If [true](#), a transaction will be used unless [sqlTransaction](#) is provided.

#### sqlTransaction [MySqlTransaction](#)

An optional existing MySql.Data.MySqlClient.MySqlTransaction to use for the operation. If provided, [useTransaction](#) is ignored.

#### allowUserVariables [bool](#)

Indicates whether user-defined variables are allowed in the query. If [true](#), user variables are permitted; otherwise, they are disallowed.

## Returns

T

The result of the database operation, cast to the specified type T.

## Type Parameters

T

The type of the result returned by the database operation.

## Remarks

This method provides a flexible way to execute database operations, supporting parameterized queries, transactions, and user-defined variables. Ensure that the specified type T matches the expected result of the query.

## DoDatabaseWork<T>(Enum, string, Func<MySqlCommand, object>, bool, bool, MySqlTransaction, bool)

Executes a database operation using the specified query and callback function.

```
public static T DoDatabaseWork<T>(Enum connectionName, string query, Func<MySqlCommand, object> actionCallback, bool throwException = true, bool useTransaction = false, MySqlTransaction sqlTransaction = null, bool allowUserVariables = false)
```

## Parameters

connectionName [Enum](#)

The name of the database connection to use. Must be a valid enumeration value representing a configured connection.

query [string](#)

The SQL query to execute. Cannot be null or empty.

actionCallback [Func](#)<MySqlCommand, [object](#)>

A callback function that processes the MySql.Data.MySqlClient.MySqlCommand object and returns a result of type **T**.

#### **throwException** [bool](#)

Specifies whether to throw an exception if an error occurs. If [true](#), exceptions will be thrown; otherwise, errors will be suppressed.

#### **useTransaction** [bool](#)

Specifies whether the operation should be executed within a database transaction. If [true](#), a transaction will be used.

#### **sqlTransaction** MySqlTransaction

An optional MySql.Data.MySqlClient.MySqlTransaction object to use for the operation. If provided, the operation will be executed within this transaction.

#### **allowUserVariables** [bool](#)

Specifies whether user-defined variables are allowed in the SQL query. If [true](#), user variables are permitted.

### Returns

#### **T**

The result of the operation, as returned by the **actionCallback** function.

### Type Parameters

#### **T**

The type of the result returned by the callback function.

### Remarks

This method provides a flexible way to execute database operations by allowing the caller to specify a query and a callback function to process the database command. The operation can optionally be executed within a transaction, and user-defined variables in the query can be enabled if required.

## GetBulkTableWriter<T>(IRelmContext, string, bool, bool, bool,

# bool, bool)

Creates and returns a [BulkTableWriter<T>](#) instance for performing bulk insert operations on a database table.

```
public static BulkTableWriter<T> GetBulkTableWriter<T>(IRelmContext relmContext, string
insertQuery = null, bool useTransaction = false, bool throwException = true, bool
allowAutoIncrementColumns = false, bool allowPrimaryKeyColumns = false, bool
allowUniqueColumns = false)
```

## Parameters

**relmContext** [IRelmContext](#)

The database context used to manage the connection and operations. This parameter cannot be [null](#).

.

**insertQuery** [string](#)

An optional SQL insert query to use for the bulk operation. If [null](#), a default query is generated based on the type **T**.

**useTransaction** [bool](#)

Indicates whether the bulk operation should be performed within a transaction. If [true](#), the operation is transactional; otherwise, it is not.

**throwException** [bool](#)

Determines whether exceptions should be thrown during the operation. If [true](#), exceptions are thrown; otherwise, errors are suppressed.

**allowAutoIncrementColumns** [bool](#)

Specifies whether auto-increment columns are allowed in the bulk operation. If [true](#), auto-increment columns are included; otherwise, they are excluded.

**allowPrimaryKeyColumns** [bool](#)

Specifies whether primary key columns are allowed in the bulk operation. If [true](#), primary key columns are included; otherwise, they are excluded.

**allowUniqueColumns** [bool](#)

Specifies whether unique columns are allowed in the bulk operation. If [true](#), unique columns are included; otherwise, they are excluded.

## Returns

[BulkTableWriter](#)<T>

A [BulkTableWriter](#)<T> instance configured for the specified bulk operation.

## Type Parameters

T

The type representing the table's data model. Each instance of T corresponds to a row in the table.

## Remarks

This method provides a convenient way to configure and execute bulk insert operations on a database table. The behavior of the operation can be customized using the optional parameters.

## GetBulkTableWriter<T>(IRelmQuickContext, string, bool, bool, bool, bool)

Creates and returns a [BulkTableWriter](#)<T> instance for performing bulk insert operations on a database table.

```
public static BulkTableWriter<T> GetBulkTableWriter<T>(IRelmQuickContext relmQuickContext,
    string insertQuery = null, bool useTransaction = false, bool throwException = true, bool
    allowAutoIncrementColumns = false, bool allowPrimaryKeyColumns = false, bool
    allowUniqueColumns = false)
```

## Parameters

**relmQuickContext** [IRelmQuickContext](#)

The database context used to manage the connection and operations. This parameter cannot be [null](#).

**insertQuery** [string](#)

An optional SQL insert query to use for the bulk operation. If [null](#), a default query is generated based on the type [T](#).

#### **useTransaction** [bool](#)

A value indicating whether the bulk operation should be performed within a transaction. The default is [false](#).

#### **throwException** [bool](#)

A value indicating whether exceptions should be thrown if an error occurs during the operation. The default is [true](#).

#### **allowAutoIncrementColumns** [bool](#)

A value indicating whether auto-increment columns are allowed in the bulk operation. The default is [false](#).

#### **allowPrimaryKeyColumns** [bool](#)

A value indicating whether primary key columns are allowed in the bulk operation. The default is [false](#).

#### **allowUniqueColumns** [bool](#)

A value indicating whether unique columns are allowed in the bulk operation. The default is [false](#).

Returns

#### [BulkTableWriter](#)<[T](#)>

A [BulkTableWriter](#)<[T](#)> instance configured for the specified bulk operation.

Type Parameters

#### **T**

The type of the objects to be written to the database table.

Remarks

This method is a wrapper around [GetBulkTableWriter](#)<[T](#)>([Enum](#), [string](#), [bool](#), [bool](#), [bool](#), [bool](#), [bool](#), [bool](#)) and provides additional configuration options for bulk insert operations. Use this method to efficiently insert large amounts of data into a database table.

# GetBulkTableWriter<T>(MySqlConnection, string, bool, bool, MySqlTransaction, bool, bool, bool)

Creates and returns a [BulkTableWriter<T>](#) instance for performing bulk insert operations on a MySQL database table.

```
public static BulkTableWriter<T> GetBulkTableWriter<T>(MySqlConnection  
establishedConnection, string insertQuery = null, bool throwException = true, bool  
useTransaction = true, MySqlTransaction sqlTransaction = null, bool  
allowAutoIncrementColumns = false, bool allowPrimaryKeyColumns = false, bool  
allowUniqueColumns = false)
```

## Parameters

**establishedConnection** MySqlConnection

An open and valid MySql.Data.MySqlClient.MySqlConnection to the target database. The connection must remain open for the duration of the bulk operation.

**insertQuery** [string](#)

An optional custom SQL insert query to use for the bulk operation. If not provided, a default query will be generated based on the type **T** and the target table schema.

**throwException** [bool](#)

A value indicating whether exceptions should be thrown if an error occurs during the bulk operation. If [true](#), exceptions will be thrown; otherwise, errors will be suppressed.

**useTransaction** [bool](#)

A value indicating whether the bulk operation should be performed within a transaction. If [true](#), a transaction will be used unless **sqlTransaction** is provided.

**sqlTransaction** MySqlTransaction

An optional MySql.Data.MySqlClient.MySqlTransaction to use for the bulk operation. If provided, the operation will use this transaction instead of creating a new one. This parameter is ignored if **useTransaction** is [false](#).

**allowAutoIncrementColumns** [bool](#)

A value indicating whether auto-increment columns in the target table are allowed to be explicitly written to. If [false](#), auto-increment columns will be excluded from the operation.

## allowPrimaryKeyColumns [bool](#)

A value indicating whether primary key columns in the target table are allowed to be explicitly written to. If [false](#), primary key columns will be excluded from the operation.

## allowUniqueColumns [bool](#)

A value indicating whether unique columns in the target table are allowed to be explicitly written to. If [false](#), unique columns will be excluded from the operation.

## Returns

### [BulkTableWriter](#)<T>

A [BulkTableWriter](#)<T> instance configured for the specified bulk operation.

## Type Parameters

### T

The type of the objects to be written to the database. Each object represents a row in the target table.

## Remarks

This method provides a flexible way to perform high-performance bulk insert operations on a MySQL database table. The behavior of the operation can be customized using the provided parameters.

## GetBulkTableWriter<T>(Enum, string, bool, bool, bool, bool, bool, bool)

Creates and returns a [BulkTableWriter](#)<T> instance for performing bulk insert operations on a database table.

```
public static BulkTableWriter<T> GetBulkTableWriter<T>(Enum connectionName, string
insertQuery = null, bool useTransaction = false, bool throwException = true, bool
allowUserVariables = false, bool allowAutoIncrementColumns = false, bool
allowPrimaryKeyColumns = false, bool allowUniqueColumns = false)
```

## Parameters

### connectionName [Enum](#)

The name of the database connection, represented as an [Enum](#). This specifies which database connection to use.

#### `insertQuery` [string](#)

An optional SQL insert query to use for the bulk operation. If not provided, a default query is generated based on the type `T`.

#### `useTransaction` [bool](#)

A value indicating whether the bulk operation should be performed within a database transaction. If [true](#), the operation is transactional; otherwise, it is not.

#### `throwException` [bool](#)

A value indicating whether exceptions should be thrown if an error occurs during the operation. If [true](#), exceptions are thrown; otherwise, errors are suppressed.

#### `allowUserVariables` [bool](#)

A value indicating whether user-defined variables are allowed in the SQL query. If [true](#), user variables are permitted; otherwise, they are not.

#### `allowAutoIncrementColumns` [bool](#)

A value indicating whether auto-increment columns are allowed to be included in the bulk operation. If [true](#), auto-increment columns are included; otherwise, they are excluded.

#### `allowPrimaryKeyColumns` [bool](#)

A value indicating whether primary key columns are allowed to be included in the bulk operation. If [true](#), primary key columns are included; otherwise, they are excluded.

#### `allowUniqueColumns` [bool](#)

A value indicating whether unique columns are allowed to be included in the bulk operation. If [true](#), unique columns are included; otherwise, they are excluded.

## Returns

#### `BulkTableWriter`<`T`>

A `BulkTableWriter`<`T`> instance configured for the specified bulk operation.

## Type Parameters

T

The type of the objects to be written to the database. Each object represents a row in the table.

## Remarks

This method is designed for scenarios where large amounts of data need to be inserted into a database table efficiently. The behavior of the bulk operation can be customized using the provided parameters.

## GetColumnName<T>(Expression<Func<T, object>>)

Retrieves the name of the database column associated with the specified property or field.

```
public static string GetColumnName<T>(Expression<Func<T, object>> predicate) where T : IRelmModel
```

## Parameters

predicate [Expression<Func<T, object>>](#)

An expression that specifies the property or field for which to retrieve the column name. The expression should be in the form of a lambda, such as `x => x.PropertyName`.

## Returns

[string](#)

The name of the database column corresponding to the specified property or field.

## Type Parameters

T

The type of the model that implements [IRelmModel](#).

## GetConnectionStringBuilderFromConnectionString(string)

Creates and returns a `MySql.Data.MySqlClient.MySqlConnectionStringBuilder` initialized with the specified connection string.

```
public static MySqlConnectionStringBuilder? GetConnectionStringBuilderFromConnectionString(string  
connectionString)
```

## Parameters

`connectionString` [string](#)

The connection string used to configure the `MySql.Data.MySqlClient.MySqlConnectionStringBuilder`.

## Returns

`MySqlConnectionStringBuilder`

A `MySql.Data.MySqlClient.MySqlConnectionStringBuilder` instance populated with the settings from the provided connection string.

## GetConnectionStringBuilderFromConnectionType(Enum)

Retrieves a `MySql.Data.MySqlClient.MySqlConnectionStringBuilder` instance configured for the specified connection type.

```
public static MySqlConnectionStringBuilder? GetConnectionStringBuilderFromConnectionType(Enum  
connectionName)
```

## Parameters

`connectionName` [Enum](#)

An enumeration value representing the type of connection to configure. This parameter determines the settings applied to the returned connection string builder.

## Returns

`MySqlConnectionStringBuilder`

A `MySql.Data.MySqlClient.MySqlConnectionStringBuilder` instance configured based on the specified `connectionName`.

## Remarks

The method delegates the creation of the connection string builder to the [GetConnectionStringBuilderFromType\(Enum\)](#) method. Ensure that the provided `connectionName` corresponds to a supported connection type.

## GetConnectionStringBuilderFromName(string)

Retrieves a `MySql.Data.MySqlClient.MySqlConnectionStringBuilder` instance based on the specified connection name.

```
public static MySqlConnectionStringBuilder? GetConnectionStringBuilderFromName(string connectionName)
```

### Parameters

`connectionName` [string](#)

The name of the connection to retrieve. This value must correspond to a valid connection name in the application configuration.

### Returns

`MySqlConnectionStringBuilder`

A `MySql.Data.MySqlClient.MySqlConnectionStringBuilder` initialized with the connection string associated with the specified connection name.

## GetConnectionFromConnectionString(string, bool, bool, int)

Creates and returns a new `MySql.Data.MySqlClient.MySqlConnection` instance based on the specified connection string and optional configuration parameters.

```
public static MySqlConnection? GetConnectionFromConnectionString(string connectionString, bool allowUserVariables = false, bool convertZeroDateTime = false, int lockWaitTimeoutSeconds = 0)
```

### Parameters

`connectionString` [string](#)

The connection string used to establish the database connection. This parameter cannot be [null](#) or empty.

#### allowUserVariables [bool](#)

A value indicating whether user-defined variables are allowed in SQL statements. The default is [false](#).

#### convertZeroDateTime [bool](#)

A value indicating whether zero date values (e.g., '0000-00-00') in the database should be converted to [MinValue](#). The default is [false](#).

#### lockWaitTimeoutSeconds [int](#)

The lock wait timeout duration, in seconds, to apply to the connection. A value of [0](#) indicates that the default timeout will be used.

### Returns

#### MySqlConnection

A MySql.Data.MySqlClient.MySqlConnection instance configured with the specified connection string and options.

### Remarks

This method provides a convenient way to create a MySql.Data.MySqlClient.MySqlConnection with additional configuration options such as enabling user-defined variables, converting zero date values, or setting a custom lock wait timeout. Ensure that the connection string is valid and properly formatted for MySQL.

## GetConnectionFromName(string, bool, bool, int)

Retrieves a MySql.Data.MySqlClient.MySqlConnection instance based on the specified connection name.

```
public static MySqlConnection? GetConnectionFromName(string connectionName, bool  
allowUserVariables = false, bool convertZeroDateTime = false, int lockWaitTimeoutSeconds  
= 0)
```

### Parameters

#### connectionName [string](#)

The name of the connection to retrieve. This must correspond to a valid connection string defined in the application's configuration.

#### allowUserVariables [bool](#)

A value indicating whether user-defined variables are allowed in SQL queries. Defaults to [false](#).

#### convertZeroDateTime [bool](#)

A value indicating whether zero date values (e.g., '0000-00-00') in the database should be converted to [MinValue](#). Defaults to [false](#).

#### lockWaitTimeoutSeconds [int](#)

The lock wait timeout, in seconds, to apply to the connection. A value of [0](#) indicates that the default timeout will be used.

Returns

MySqlConnection

A MySql.Data.MySqlClient.MySqlConnection instance configured with the specified options.

## GetConnectionFromType(Enum, bool, bool, int)

Creates and returns a MySql.Data.MySqlClient.MySqlConnection based on the specified connection type and optional configuration settings.

```
public static MySqlConnection? GetConnectionFromType(Enum connectionName, bool  
allowUserVariables = false, bool convertZeroDateTime = false, int lockWaitTimeoutSeconds  
= 0)
```

Parameters

#### connectionName [Enum](#)

An [Enum](#) representing the type of connection to establish. The specific values and their meanings depend on the application's connection configuration.

#### allowUserVariables [bool](#)

A boolean value indicating whether user-defined variables are allowed in SQL statements. [true](#) to allow user variables; otherwise, [false](#). Defaults to [false](#).

#### convertZeroDateTime [bool](#)

A boolean value indicating whether zero date values (e.g., '0000-00-00') should be converted to `DateTime.MinValue`. [true](#) to enable conversion; otherwise, [false](#). Defaults to [false](#).

#### lockWaitTimeoutSeconds [int](#)

An integer specifying the lock wait timeout duration, in seconds, for the connection. A value of 0 indicates that the default timeout will be used. Defaults to 0.

### Returns

#### MySqlConnection

A `MySql.Data.MySqlClient.MySqlConnection` object configured based on the specified parameters.

### Remarks

This method simplifies the creation of MySQL connections. Ensure that the `connectionName` corresponds to a valid configuration in the application's connection settings.

## GetDalTable(Type)

Retrieves the name of the database table associated with the specified Data Access Layer (DAL) object type.

```
public static string GetDalTable(Type DalObjectType)
```

### Parameters

#### DalObjectType [Type](#)

The type of the DAL object for which the table name is being retrieved. This parameter cannot be [null](#).

### Returns

#### [string](#)

The name of the database table associated with the specified DAL object type.

## GetDalTable<T>()

Retrieves the name of the database table associated with the specified data model type.

```
public static string GetDalTable<T>() where T : IRelmModel, new()
```

Returns

[string](#)

A [string](#) representing the name of the database table associated with the specified data model type.

Type Parameters

T

The type of the data model. Must implement [IRelmModel](#) and have a parameterless constructor.

Remarks

This method is typically used to determine the table name for a given data model type in the context of database operations.

## Get dataList<T>(IRelmContext, string, Dictionary<string, object>, bool)

Executes a query against the specified Realm context and retrieves a collection of data items.

```
public static IEnumerable<T> Get dataList<T>(IRelmContext realmContext, string query,  
Dictionary<string, object> parameters = null, bool throwException = true)
```

Parameters

realmContext [IRelmContext](#)

The Realm context used to execute the query. Cannot be [null](#).

### query [string](#)

The query string to execute. Cannot be [null](#) or empty.

### parameters [Dictionary](#)<[string](#), [object](#)>

An optional dictionary of parameters to include in the query. Can be [null](#) if no parameters are needed.

### throwException [bool](#)

A value indicating whether an exception should be thrown if the query fails. If [true](#), an exception is thrown on failure; otherwise, the method returns an empty collection.

## Returns

### [IEnumerable](#)<T>

An [IEnumerable](#)<T> containing the data items retrieved by the query. Returns an empty collection if no items are found or if `throwException` is [false](#) and the query fails.

## Type Parameters

### T

The type of the data items to retrieve.

## Get dataList<T>(IRelmQuickContext, string, Dictionary<string, object>, bool)

Executes a query against the specified context and returns a collection of results of the specified type.

```
public static IEnumerable<T> Get dataList<T>(IRelmQuickContext relmContext, string query, Dictionary<string, object> parameters = null, bool throwException = true)
```

## Parameters

### relmContext [IRelmQuickContext](#)

The context used to execute the query. This parameter cannot be [null](#).

### query [string](#)

The query string to execute. This parameter cannot be [null](#) or empty.

**parameters** [Dictionary](#)<[string](#), [object](#)>

An optional dictionary of parameters to be used in the query. Keys represent parameter names, and values represent their corresponding values. Can be [null](#) if no parameters are needed.

**throwException** [bool](#)

A value indicating whether an exception should be thrown if an error occurs during query execution. If [true](#), an exception will be thrown on error; otherwise, the method will return an empty collection.

Returns

[IEnumerable](#)<[T](#)>

A collection of objects of type [T](#) representing the results of the query. Returns an empty collection if no results are found or if **throwException** is [false](#) and an error occurs.

Type Parameters

[T](#)

The type of objects to be returned in the result set.

**GetDataTable<T>(MySqlConnection, string, Dictionary<string, object>, bool, MySqlTransaction)**

Executes the specified SQL query and retrieves a list of objects of type [T](#).

```
public static IEnumerable<T> GetDataTable<T>(MySqlConnection establishedConnection, string query, Dictionary<string, object> parameters = null, bool throwException = true, MySqlTransaction sqlTransaction = null)
```

Parameters

**establishedConnection** MySqlConnection

An open and valid MySql.Data.MySqlClient.MySqlConnection to use for the query.

**query** [string](#)

The SQL query to execute. Must be a valid SQL statement.

**parameters** [Dictionary](#)<[string](#), [object](#)>

An optional dictionary of parameter names and values to include in the query. Can be [null](#) if no parameters are needed.

**throwException** [bool](#)

A value indicating whether to throw an exception if an error occurs during query execution. If [true](#), exceptions will be thrown; otherwise, errors will be suppressed.

**sqlTransaction** MySqlTransaction

An optional MySql.Data.MySqlClient.MySqlTransaction to associate with the query. Can be [null](#) if no transaction is required.

Returns

[IEnumerable](#)<[T](#)>

An [IEnumerable](#)<[T](#)> containing the results of the query mapped to objects of type [T](#). Returns an empty collection if no results are found.

Type Parameters

[T](#)

The type of objects to map the query results to.

**GetDataTable<T>(Enum, string, Dictionary<string, object>, bool, bool)**

Executes the specified query and retrieves a collection of data mapped to the specified type.

```
public static IEnumerable<T> GetDataTable<T>(Enum connectionName, string query,
Dictionary<string, object> parameters = null, bool throwException = true, bool
allowUserVariables = false)
```

Parameters

**connectionName** [Enum](#)

The name of the connection to use for executing the query.

#### query [string](#)

The SQL query to execute.

#### parameters [Dictionary](#)<[string](#), [object](#)>

An optional dictionary of parameters to include in the query. Keys represent parameter names, and values represent their corresponding values.

#### throwException [bool](#)

A value indicating whether to throw an exception if an error occurs. If [false](#), errors will be suppressed.

#### allowUserVariables [bool](#)

A value indicating whether user-defined variables are allowed in the query.

Returns

#### [IEnumerable](#)<T>

An [IEnumerable](#)<T> containing the results of the query mapped to the specified type. Returns an empty collection if no results are found.

Type Parameters

#### T

The type to which the query results will be mapped.

Remarks

This method uses the specified connection and query to retrieve data from the database. Ensure that the type T has a structure compatible with the query results.

## GetDataObject<T>(IRelmContext, string, Dictionary<string, object>, bool)

Retrieves a single data object of the specified type based on the provided query and parameters.

```
public static T GetDataObject<T>(IRelmContext relmContext, string query, Dictionary<string, object> parameters = null, bool throwException = true) where T : IRelmModel, new()
```

## Parameters

### relmContext [IRelmContext](#)

The context used to execute the query. This provides the connection and configuration for the data source.

### query [string](#)

The query string used to retrieve the data object. The query must be valid for the underlying data source.

### parameters [Dictionary](#)<[string](#), [object](#)>

An optional dictionary of parameters to be used in the query. Keys represent parameter names, and values represent their corresponding values.

### throwException [bool](#)

A boolean value indicating whether an exception should be thrown if the query does not return a result. If [true](#), an exception is thrown when no data is found; otherwise, [null](#) is returned.

## Returns

### T

An instance of the specified type [T](#) populated with the data retrieved by the query. Returns [null](#) if no data is found and [throwException](#) is [false](#).

## Type Parameters

### T

The type of the data object to retrieve. Must implement [IRelmModel](#) and have a parameterless constructor.

## Remarks

This method provides a convenient way to retrieve a single data object. Ensure that the query and parameters are properly constructed to avoid unexpected results.

## GetDataObject<T>(IRelmQuickContext, string, Dictionary<string, object>, bool)

Retrieves a single data object of the specified type based on the provided query and parameters.

```
public static T GetDataObject<T>(IRelmQuickContext relmQuickContext, string query,  
Dictionary<string, object> parameters = null, bool throwException = true) where T :  
IRelmModel, new()
```

### Parameters

**relmQuickContext** [IRelmQuickContext](#)

The context used to execute the query. This provides the necessary connection and configuration for data retrieval.

**query** [string](#)

The query string used to retrieve the data object. This should be a valid query supported by the context.

**parameters** [Dictionary](#)<[string](#), [object](#)>

An optional dictionary of parameters to be used in the query. Keys represent parameter names, and values represent their corresponding values.

**throwException** [bool](#)

A boolean value indicating whether an exception should be thrown if the query does not return a result. If [true](#), an exception is thrown when no result is found; otherwise, [null](#) is returned.

### Returns

**T**

An instance of the specified type **T** representing the retrieved data object. Returns [null](#) if no result is found and **throwException** is [false](#).

### Type Parameters

**T**

The type of the data object to retrieve. Must implement [IRelmModel](#) and have a parameterless constructor.

## Remarks

This method provides a convenient way to retrieve a single data object. Ensure that the query and parameters are properly constructed to avoid unexpected results.

## GetDataObject<T>(MySqlConnection, string, Dictionary<string, object>, bool, MySqlTransaction)

Retrieves a single data object of the specified type from the database based on the provided query and parameters.

```
public static T GetDataObject<T>(MySqlConnection establishedConnection, string query,  
Dictionary<string, object> parameters = null, bool throwException = true, MySqlTransaction  
sqlTransaction = null) where T : IRelmModel, new()
```

## Parameters

**establishedConnection** MySqlConnection

An open MySql.Data.MySqlClient.MySqlConnection to the database. The connection must be valid and already established.

**query** [string](#)

The SQL query to execute. The query should be structured to return a single row of data.

**parameters** [Dictionary](#)<[string](#), [object](#)>

An optional dictionary of parameter names and values to be used in the query. Can be [null](#) if no parameters are required.

**throwException** [bool](#)

A boolean value indicating whether an exception should be thrown if the query does not return a result. If [true](#), an exception is thrown when no data is found; otherwise, [default](#) is returned.

**sqlTransaction** MySqlTransaction

An optional MySql.Data.MySqlClient.MySqlTransaction to associate with the query. Can be [null](#) if no transaction is needed.

## Returns

T

An instance of type T populated with the data retrieved from the database. Returns [default](#) if no data is found and `throwException` is [false](#).

## Type Parameters

T

The type of the data object to retrieve. Must implement [IRelmModel](#) and have a parameterless constructor.

## GetDataObject<T>(Enum, string, Dictionary<string, object>, bool, bool)

Retrieves a single data object of the specified type based on the provided query and parameters.

```
public static T GetDataObject<T>(Enum connectionName, string query, Dictionary<string, object> parameters = null, bool throwException = true, bool allowUserVariables = false)  
where T : IRelmModel, new()
```

## Parameters

`connectionName` [Enum](#)

The connection identifier used to determine the data source.

`query` [string](#)

The query string used to retrieve the data object.

`parameters` [Dictionary](#)<[string](#), [object](#)>

An optional dictionary of parameters to be used in the query. The keys represent parameter names, and the values represent their corresponding values.

`throwException` [bool](#)

A boolean value indicating whether an exception should be thrown if the query fails or no data is found. If [true](#), an exception is thrown; otherwise, [null](#) is returned.

#### allowUserVariables [bool](#)

A boolean value indicating whether user-defined variables are allowed in the query. Set to [true](#) to enable user variables; otherwise, [false](#).

#### Returns

T

An instance of the specified type T populated with the data retrieved from the query, or [null](#) if no data is found and [throwException](#) is [false](#).

#### Type Parameters

T

The type of the data object to retrieve. Must implement [IRelmModel](#) and have a parameterless constructor.

#### Remarks

This method is a convenience wrapper for retrieving a single data object. If multiple objects match the query, only the first one is returned.

## GetDataObjects<T>(IRelmContext, string, Dictionary<string, object>, bool)

Executes a query against the specified Relm context and retrieves a collection of data objects of the specified type.

```
public static IEnumerable<T> GetDataObjects<T>(IRelmContext relmContext, string query, Dictionary<string, object> parameters = null, bool throwException = true) where T : IRelmModel, new()
```

#### Parameters

relmContext [IRelmContext](#)

The Relm context used to execute the query. This context provides the connection and configuration for the database.

#### query [string](#)

The query string to execute. This should be a valid query supported by the Relm context.

#### parameters [Dictionary](#)<[string](#), [object](#)>

An optional dictionary of parameters to include in the query. The keys represent parameter names, and the values represent their corresponding values.

#### throwException [bool](#)

A boolean value indicating whether an exception should be thrown if an error occurs during query execution. If [true](#), an exception will be thrown; otherwise, the method will handle the error silently.

Returns

#### [IEnumerable](#)<T>

An [IEnumerable](#)<T> containing the data objects retrieved by the query. If no data is found, the collection will be empty.

Type Parameters

#### T

The type of data objects to retrieve. Must implement [IRelmModel](#) and have a parameterless constructor.

Remarks

This method provides a convenient way to execute queries and retrieve data objects. Ensure that the query string and parameters are properly formatted to avoid runtime errors.

## GetDataObjects<T>(IRelmQuickContext, string, Dictionary<string, object>, bool)

Executes a query against the specified Relm context and retrieves a collection of data objects of the specified type.

```
public static IEnumerable<T> GetDataObjects<T>(IRelmQuickContext relmContext, string query,  
Dictionary<string, object> parameters = null, bool throwException = true) where T :  
IRelmModel, new()
```

## Parameters

`relmContext` [IRelmQuickContext](#)

The Relm context used to execute the query. This context provides the connection and configuration for the database.

`query` [string](#)

The query string to execute. The query must be valid for the underlying database.

`parameters` [Dictionary](#)<[string](#), [object](#)>

An optional dictionary of parameters to include in the query. Keys represent parameter names, and values represent their corresponding values.

`throwException` [bool](#)

A value indicating whether an exception should be thrown if an error occurs during query execution. If [true](#), exceptions will be thrown; otherwise, errors will be suppressed.

## Returns

[IEnumerable](#)<T>

An [IEnumerable](#)<T> containing the data objects retrieved by the query. The collection will be empty if no matching data is found.

## Type Parameters

T

The type of data objects to retrieve. Must implement [IRelmModel](#) and have a parameterless constructor.

## Remarks

This method provides a convenient way to execute queries and retrieve data objects. Ensure that the query string and parameters are properly formatted to avoid runtime errors.

# GetDataObjects<T>(MySqlConnection, string, Dictionary<string, object>, bool, MySqlTransaction)

Retrieves a collection of data objects of the specified type from the database based on the provided query and parameters.

```
public static IEnumerable<T> GetDataObjects<T>(MySqlConnection establishedConnection, string query, Dictionary<string, object> parameters = null, bool throwException = true, MySqlTransaction sqlTransaction = null) where T : IRelmModel, new()
```

## Parameters

**establishedConnection** MySqlConnection

An open MySql.Data.MySqlClient.MySqlConnection to the database. The connection must be established and valid.

**query** [string](#)

The SQL query to execute for retrieving the data objects.

**parameters** [Dictionary](#)<[string](#), [object](#)>

An optional dictionary of parameter names and values to be used in the query. Can be [null](#) if no parameters are required.

**throwException** [bool](#)

A value indicating whether an exception should be thrown if an error occurs during query execution. If [true](#), exceptions will be propagated; otherwise, errors will be suppressed.

**sqlTransaction** MySqlTransaction

An optional MySql.Data.MySqlClient.MySqlTransaction to associate with the query execution. Can be [null](#) if no transaction is required.

## Returns

[IEnumerable](#)<T>

An [IEnumerable](#)<T> containing the data objects retrieved from the database. The collection will be empty if no matching records are found.

## Type Parameters

T

The type of the data objects to retrieve. Must implement [IRelmModel](#) and have a parameterless constructor.

## Remarks

This method uses the specified SQL query and parameters to retrieve data objects from the database. Ensure that the query matches the structure of the type T.

## GetDataObjects<T>(DataTable)

Converts the rows of the specified [DataTable](#) into a collection of objects of type T.

```
public static IEnumerable<T> GetDataObjects<T>(DataTable existingData) where T : IRelmModel, new()
```

## Parameters

**existingData** [DataTable](#)

The [DataTable](#) containing the data to be converted. Each row in the table is mapped to an object of type T.

## Returns

[IEnumerable](#)<T>

An [IEnumerable](#)<T> containing the objects created from the rows of the existingData table.

## Type Parameters

T

The type of objects to create from the data rows. Must implement [IRelmModel](#) and have a parameterless constructor.

## Remarks

This method uses the [GetDataObjects<T>\(DataTable\)](#) method to perform the conversion. Ensure that the `existingData` table contains columns that match the properties of type `T`.

## GetDataObjects<T>(Enum, string, Dictionary<string, object>, bool, bool)

Retrieves a collection of data objects of the specified type from the database.

```
public static IEnumerable<T> GetDataObjects<T>(Enum connectionName, string query,
Dictionary<string, object> parameters = null, bool throwException = true, bool
allowUserVariables = false) where T : IRelmModel, new()
```

### Parameters

`connectionName` [Enum](#)

The name of the database connection to use. This must be an enumeration value representing a valid connection.

`query` [string](#)

The SQL query to execute for retrieving the data objects.

`parameters` [Dictionary](#)<[string](#), [object](#)>

An optional dictionary of parameter names and values to include in the query. Can be [null](#) if no parameters are needed.

`throwException` [bool](#)

A value indicating whether an exception should be thrown if an error occurs during query execution. If [true](#), exceptions will be thrown; otherwise, errors will be suppressed.

`allowUserVariables` [bool](#)

A value indicating whether user-defined variables are allowed in the query. Set to [true](#) to enable user variables; otherwise, [false](#).

### Returns

[IEnumerable](#)<T>

An [IEnumerable<T>](#) containing the data objects retrieved by the query. If no data is found, an empty collection is returned.

## Type Parameters

T

The type of data objects to retrieve. Must implement [IRelmModel](#) and have a parameterless constructor.

## Remarks

This method provides additional options for query execution. Ensure that the SQL query and parameters are properly constructed to avoid runtime errors.

## GetDataRow(IRelmContext, string, Dictionary<string, object>, bool)

Executes the specified query and retrieves the first row of the result set as a [DataRow](#).

```
public static DataRow GetDataRow(IRelmContext relmContext, string query, Dictionary<string, object> parameters = null, bool throwException = true)
```

## Parameters

**relmContext** [IRelmContext](#)

The database context used to execute the query. Cannot be [null](#).

**query** [string](#)

The SQL query to execute. Cannot be [null](#) or empty.

**parameters** [Dictionary](#)<[string](#), [object](#)>

An optional dictionary of parameter names and values to include in the query. Can be [null](#) if no parameters are required.

**throwException** [bool](#)

A value indicating whether an exception should be thrown if the query does not return any rows. [true](#) to throw an exception when no rows are returned; otherwise, [false](#).

Returns

## [DataRow](#)

A [DataRow](#) representing the first row of the result set. Returns [null](#) if no rows are found and `throwException` is [false](#).

# GetDataRow(IRelmQuickContext, string, Dictionary<string, object>, bool)

Executes the specified query and retrieves the first row of the result set as a [DataRow](#).

```
public static DataRow GetDataRow(IRelmQuickContext relmQuickContext, string query,  
Dictionary<string, object> parameters = null, bool throwException = true)
```

Parameters

### `relmQuickContext` [IRelmQuickContext](#)

The context used to execute the query. This must not be [null](#).

### `query` [string](#)

The SQL query to execute. This must not be [null](#) or empty.

### `parameters` [Dictionary](#)<[string](#), [object](#)>

An optional dictionary of parameter names and values to be used in the query. If [null](#), no parameters are applied.

### `throwException` [bool](#)

A value indicating whether an exception should be thrown if the query does not return any rows. If [true](#), an exception is thrown when no rows are found; otherwise, [null](#) is returned.

Returns

## [DataRow](#)

The first row of the result set as a [DataRow](#), or [null](#) if no rows are found and `throwException` is [false](#).

# GetDataRow(MySqlConnection, string, Dictionary<string, object>, bool, MySqlTransaction)

Executes the specified SQL query and retrieves the first row of the result set as a [DataRow](#).

```
public static DataRow GetDataRow(MySqlConnection establishedConnection, string query,
Dictionary<string, object> parameters = null, bool throwException = true, MySqlTransaction
sqlTransaction = null)
```

## Parameters

**establishedConnection** MySqlConnection

An open MySql.Data.MySqlClient.MySqlConnection to the database. The connection must be in an open state before calling this method.

**query** [string](#)

The SQL query to execute. The query must be a valid SQL statement that returns a result set.

**parameters** [Dictionary](#)<[string](#), [object](#)>

An optional dictionary of parameter names and values to be used in the query. If null, no parameters are applied.

**throwException** [bool](#)

A boolean value indicating whether to throw an exception if the query fails. [true](#) to throw an exception on failure; otherwise, [false](#).

**sqlTransaction** MySqlTransaction

An optional MySql.Data.MySqlClient.MySqlTransaction to associate with the query. If null, the query is executed without a transaction.

## Returns

[DataRow](#)

A [DataRow](#) representing the first row of the result set. Returns [null](#) if the query does not return any rows.

## Remarks

This method is a convenience wrapper for retrieving a single row from the result set of a query. If the query returns multiple rows, only the first row is returned. If no rows are returned, the method returns [null](#).

## GetDataRow(Enum, string, Dictionary<string, object>, bool, bool)

Executes the specified query and retrieves the first row of the result set as a [DataRow](#).

```
public static DataRow GetDataRow(Enum connectionName, string query, Dictionary<string, object> parameters = null, bool throwException = true, bool allowUserVariables = false)
```

### Parameters

`connectionName` [Enum](#)

An [Enum](#) representing the connection name to use for the database operation.

`query` [string](#)

The SQL query to execute. This query must be a valid SQL statement.

`parameters` [Dictionary](#)<[string](#), [object](#)>

An optional dictionary of parameter names and values to be used in the query. If null, no parameters are applied.

`throwException` [bool](#)

A boolean value indicating whether an exception should be thrown if the query fails. [true](#) to throw an exception on failure; otherwise, [false](#).

`allowUserVariables` [bool](#)

A boolean value indicating whether user-defined variables are allowed in the query. [true](#) to allow user variables; otherwise, [false](#).

### Returns

[DataRow](#)

A [DataRow](#) representing the first row of the result set. Returns [null](#) if the query does not return any rows.

## Remarks

This method uses the connection associated with the specified `connectionName` to execute the query. If `parameters` are provided, they are applied to the query as named parameters.

## GetDataTable(IRelmContext, string, Dictionary<string, object>, bool)

Executes the specified query against the provided Relm context and returns the results as a [DataTable](#).

```
public static DataTable GetDataTable(IRelmContext relmContext, string query,  
Dictionary<string, object> parameters = null, bool throwException = true)
```

## Parameters

`relmContext` [IRelmContext](#)

The Relm context used to execute the query. This cannot be [null](#).

`query` [string](#)

The SQL query to execute. This cannot be [null](#) or empty.

`parameters` [Dictionary](#)<[string](#), [object](#)>

An optional dictionary of parameter names and values to be used in the query. Can be [null](#) if no parameters are required.

`throwException` [bool](#)

A value indicating whether an exception should be thrown if an error occurs during query execution. If [true](#), exceptions will be thrown; otherwise, errors will be suppressed.

## Returns

[DataTable](#)

A [DataTable](#) containing the results of the query. If the query returns no results, the [DataTable](#) will be empty.

# GetDataTable(IRelmQuickContext, string, Dictionary<string, object>, bool)

Executes the specified query against the provided RelmQuick context and returns the results as a [DataTable](#).

```
public static DataTable GetDataTable(IRelmQuickContext relmQuickContext, string query,  
Dictionary<string, object> parameters = null, bool throwException = true)
```

## Parameters

**relmQuickContext** [IRelmQuickContext](#)

The context used to execute the query. This must not be [null](#).

**query** [string](#)

The SQL query to execute. This must not be [null](#) or empty.

**parameters** [Dictionary](#)<[string](#), [object](#)>

An optional dictionary of parameter names and values to be used in the query. If [null](#), no parameters are applied.

**throwException** [bool](#)

A value indicating whether an exception should be thrown if an error occurs during query execution. If [true](#), exceptions are thrown; otherwise, errors are suppressed.

## Returns

[DataTable](#)

A [DataTable](#) containing the results of the query. The table will be empty if the query returns no rows.

## Remarks

This method provides a convenient way to execute queries and retrieve results in a tabular format. Ensure that the [relmQuickContext](#) is properly initialized before calling this method.

# GetDataTable(MySqlConnection, string, Dictionary<string, object>, bool, MySqlTransaction)

Executes the specified SQL query and retrieves the results as a [DataTable](#).

```
public static DataTable GetDataTable(MySqlConnection establishedConnection, string query,
Dictionary<string, object> parameters = null, bool throwException = true, MySqlTransaction
sqlTransaction = null)
```

## Parameters

**establishedConnection** MySqlConnection

An open MySql.Data.MySqlClient.MySqlConnection to the database. The connection must be in an open state before calling this method.

**query** [string](#)

The SQL query to execute. The query can include parameter placeholders.

**parameters** [Dictionary](#)<[string](#), [object](#)>

An optional dictionary of parameter names and their corresponding values to be used in the query. If null, no parameters are applied.

**throwException** [bool](#)

A boolean value indicating whether to throw an exception if an error occurs. If [true](#), exceptions are thrown; otherwise, errors are suppressed.

**sqlTransaction** MySqlTransaction

An optional MySql.Data.MySqlClient.MySqlTransaction to associate with the query. If null, the query is executed without a transaction.

## Returns

[DataTable](#)

A [DataTable](#) containing the results of the query. If the query returns no rows, the [DataTable](#) will be empty.

# GetDataTable(Enum, string, Dictionary<string, object>, bool, bool)

Executes the specified SQL query and returns the results as a [DataTable](#).

```
public static DataTable GetDataTable(Enum connectionName, string query, Dictionary<string, object> parameters = null, bool throwException = true, bool allowUserVariables = false)
```

## Parameters

**connectionName** [Enum](#)

An [Enum](#) representing the name of the database connection to use.

**query** [string](#)

The SQL query to execute. This query must be a valid SQL statement.

**parameters** [Dictionary](#)<[string](#), [object](#)>

An optional dictionary of parameter names and values to be used in the query. If null, no parameters are applied.

**throwException** [bool](#)

A boolean value indicating whether to throw an exception if an error occurs. If [true](#), exceptions are thrown; otherwise, errors are suppressed.

**allowUserVariables** [bool](#)

A boolean value indicating whether user-defined variables are allowed in the query. If [true](#), user-defined variables are permitted; otherwise, they are not.

## Returns

[DataTable](#)

A [DataTable](#) containing the results of the query. The [DataTable](#) will be empty if the query returns no rows.

## Remarks

This method establishes a database connection based on the provided **connectionName** and executes the given query. If **parameters** are provided, they are added to the query to prevent SQL injection.

## GetIdFromInternalId(IRelmContext, string, string)

Retrieves the ID associated with the specified internal ID from the given table.

```
public static string GetIdFromInternalId(IRelmContext relmContext, string tableName,
                                         string InternalId)
```

### Parameters

**relmContext** [IRelmContext](#)

The database context used to execute the query.

**tableName** [string](#)

The name of the table to query. Must not be null or empty.

**InternalId** [string](#)

The internal ID to search for. Must not be null or empty.

### Returns

[string](#)

The ID as a string if a matching record is found; otherwise, [null](#).

### Remarks

This method executes a SQL query to retrieve the ID corresponding to the provided internal ID. Ensure that the table specified by **tableName** contains columns named "ID" and "InternalId".

## GetIdFromInternalId(IRelmQuickContext, string, string)

Retrieves the ID associated with the specified internal ID from the given table.

```
public static string GetIdFromInternalId(IRelmQuickContext relmQuickContext, string
                                         tableName, string InternalId)
```

### Parameters

`relmQuickContext` [IRelmQuickContext](#)

The database quick context used to execute the query.

`tableName` [string](#)

The name of the table to query. Must not be null or empty.

`InternalId` [string](#)

The internal ID to search for. Must not be null or empty.

Returns

[string](#)

The ID corresponding to the specified internal ID, or [null](#) if no match is found.

Remarks

This method executes a SQL query to retrieve the external ID from the specified table. Ensure that the table contains a column named "InternalId" and that the query is executed in a secure and valid context.

## GetIdFromInternalId(Enum, string, string)

Retrieves the ID associated with the specified internal ID from the given table.

```
public static string GetIdFromInternalId(Enum connectionName, string tableName,  
string InternalId)
```

Parameters

`connectionName` [Enum](#)

The database connection identifier, represented as an enumeration value.

`tableName` [string](#)

The name of the database table to query. Must not be null or empty.

`InternalId` [string](#)

The internal ID to search for. Must not be null or empty.

Returns

[string](#)

The ID as a string if a matching record is found; otherwise, [null](#).

Remarks

This method executes a SQL query to retrieve the ID corresponding to the provided internal ID. Ensure that the `connectionName` corresponds to a valid database connection and that the `tableName` exists in the database schema.

## GetLastInsertId(IRelmContext)

Retrieves the identifier of the last inserted row in the current database session.

```
public static string GetLastInsertId(IRelmContext relmContext)
```

Parameters

`relmContext` [IRelmContext](#)

The database context used to execute the query. This context must be properly initialized and connected to the database.

Returns

[string](#)

A string containing the identifier of the last inserted row. The value is determined by the database's `LAST_INSERT_ID()` function.

Remarks

This method relies on the database's `LAST_INSERT_ID()` function, which typically returns the most recent auto-increment value generated during the current session. Ensure that the database supports this function and that the session context is consistent with the operation that generated the ID.

## GetLastInsertId(IRelmQuickContext)

Retrieves the identifier of the last inserted row in the current database session.

```
public static string GetLastInsertId(IRelmQuickContext relmQuickContext)
```

## Parameters

`relmQuickContext` [IRelmQuickContext](#)

The database quick context used to execute the query. This context must be properly initialized and connected to the database.

## Returns

[string](#)

A string representing the identifier of the last inserted row. The value is determined by the database's `LAST_INSERT_ID()` function.

## Remarks

This method relies on the database's `LAST_INSERT_ID()` function, which typically returns the most recent auto-increment value generated during the current session. Ensure that the database supports this function and that the session context is consistent with the operation that generated the ID.

## GetLastInsertId(Enum)

Retrieves the identifier of the last inserted row in the database for the specified connection.

```
public static string GetLastInsertId(Enum connectionName)
```

## Parameters

`connectionName` [Enum](#)

An enumeration value representing the configuration connection string to use for the database query.

## Returns

[string](#)

A string containing the identifier of the last inserted row. The value is determined by the database's LAST\_INSERT\_ID() function.

## Remarks

This method relies on the database's LAST\_INSERT\_ID() function, which typically returns the most recent auto-increment value generated during the current session. Ensure that the database supports this function and that the session context is consistent with the operation that generated the ID.

## GetScalar<T>(IRelmContext, string, Dictionary<string, object>, bool)

Executes a scalar query and retrieves the result as the specified type.

```
public static T GetScalar<T>(IRelmContext relmContext, string query, Dictionary<string, object> parameters = null, bool throwException = true)
```

## Parameters

**relmContext** [IRelmContext](#)

The database context used to execute the query. Cannot be [null](#).

**query** [string](#)

The SQL query string to execute. Cannot be [null](#) or empty.

**parameters** [Dictionary](#)<[string](#), [object](#)>

An optional dictionary of parameter names and values to include in the query. Can be [null](#) if no parameters are required.

**throwException** [bool](#)

Indicates whether an exception should be thrown if the query fails. If [true](#), an exception will be thrown on failure; otherwise, the method will return the default value of [T](#).

## Returns

[T](#)

The scalar result of the query converted to the specified type `T`. Returns the default value of `T` if the query produces no result or fails and `throwException` is [false](#).

## Type Parameters

`T`

The type to which the scalar result will be converted.

## `GetScalar<T>(IRelmQuickContext, string, Dictionary<string, object>, bool)`

Executes a scalar query and retrieves the result as the specified type.

```
public static T GetScalar<T>(IRelmQuickContext relmQuickContext, string query,  
Dictionary<string, object> parameters = null, bool throwException = true)
```

## Parameters

`relmQuickContext` [IRelmQuickContext](#)

The database context used to execute the query. Cannot be [null](#).

`query` [string](#)

The SQL query string to execute. Cannot be [null](#) or empty.

`parameters` [Dictionary](#)<[string](#), [object](#)>

An optional dictionary of parameters to include in the query. Keys represent parameter names, and values represent parameter values. Can be [null](#) if no parameters are needed.

`throwException` [bool](#)

Indicates whether an exception should be thrown if the query fails. If [true](#), an exception will be thrown on failure; otherwise, the method will return the default value of `T`.

## Returns

`T`

The result of the scalar query, cast to the specified type `T`. Returns the default value of `T` if the query fails and `throwException` is [false](#).

## Type Parameters

`T`

The type of the result to be returned.

## `GetScalar<T>(MySqlConnection, string, Dictionary<string, object>, bool, MySqlTransaction)`

Executes a scalar query on the specified MySQL connection and returns the result as the specified type.

```
public static T GetScalar<T>(MySqlConnection establishedConnection, string query,  
Dictionary<string, object> parameters = null, bool throwException = true, MySqlTransaction  
sqlTransaction = null)
```

## Parameters

`establishedConnection` `MySqlConnection`

An open `MySQL.Data.MySqlClient.MySqlConnection` to execute the query on. The connection must be valid and open.

`query` [string](#)

The SQL query string to execute. The query must return a single value.

`parameters` [Dictionary](#)<[string](#), [object](#)>

An optional dictionary of parameter names and values to be used in the query. If null, no parameters are applied.

`throwException` [bool](#)

A boolean value indicating whether to throw an exception if an error occurs. If [true](#), exceptions will be thrown; otherwise, errors will be suppressed.

`sqlTransaction` `MySqlTransaction`

An optional MySql.Data.MySqlClient.MySqlTransaction to associate with the query. If null, the query will not be part of a transaction.

## Returns

T

The scalar result of the query, converted to the specified type T. Returns the default value of T if the query result is null.

## Type Parameters

T

The type to which the scalar result will be converted.

## GetScalar<T>(Enum, string, Dictionary<string, object>, bool, bool)

Executes a scalar query and retrieves the first column of the first row in the result set, cast to the specified type.

```
public static T GetScalar<T>(Enum connectionName, string query, Dictionary<string, object>
parameters = null, bool throwException = true, bool allowUserVariables = false)
```

## Parameters

connectionName [Enum](#)

The name of the database connection to use. Must be a valid enumeration value representing a configured connection.

query [string](#)

The SQL query to execute. The query should return a single value.

parameters [Dictionary](#)<[string](#), [object](#)>

An optional dictionary of parameter names and values to be used in the query. Can be [null](#) if no parameters are required.

throwException [bool](#)

Indicates whether an exception should be thrown if an error occurs during query execution. If [true](#), exceptions will be propagated; otherwise, errors will be suppressed.

#### allowUserVariables [bool](#)

Indicates whether user-defined variables are allowed in the query. Set to [true](#) to enable user variables; otherwise, [false](#).

#### Returns

#### T

The scalar result of the query, cast to the specified type T. Returns the default value of T if the query returns no result.

#### Type Parameters

#### T

The type to which the scalar result will be cast.

#### Remarks

This method is typically used to retrieve single values, such as counts, sums, or other aggregate results, from a database query.

## LoadDataLoaderField<T, R>(IRelmContext, ICollection<T>, Expression<Func<T, R>>)

Loads a specified field for a collection of models using a DataLoader pattern.

```
public static ICollection<T> LoadDataLoaderField<T, R>(IRelmContext relmContext, ICollection<T> target, Expression<Func<T, R>> predicate) where T : IRelmModel, new()
```

#### Parameters

##### relmContext [IRelmContext](#)

The context used to interact with the data source.

##### target [ICollection](#)<T>

The collection of models for which the field will be loaded. Cannot be null.

**predicate** [Expression](#)<[Func](#)<T, R>>

An expression specifying the field to load for each model in the collection. Cannot be null.

Returns

[ICollection](#)<T>

The updated collection of models with the specified field loaded.

Type Parameters

T

The type of the model in the collection. Must implement [IRelmModel](#) and have a parameterless constructor.

R

The type of the field to be loaded.

Remarks

This method uses a DataLoader pattern to efficiently load the specified field for all models in the collection. It is designed to minimize database queries by batching and caching operations.

**LoadDataLoaderField<T, R>(IRelmContext, T, Expression<Func<T, R>>)**

Loads a specified field of a data model using a DataLoader and returns the first result.

```
public static T LoadDataLoaderField<T, R>(IRelmContext relmContext, T target,  
Expression<Func<T, R>> predicate) where T : IRelmModel, new()
```

Parameters

**relmContext** [IRelmContext](#)

The context used to interact with the data source.

## target T

The target data model instance whose field is to be loaded.

## predicate [Expression<Func<T, R>>](#)

An expression specifying the field to load.

## Returns

### T

The first result of the loaded field, or the default value of [R](#) if no results are found.

## Type Parameters

### T

The type of the data model, which must implement [IRelmModel](#) and have a parameterless constructor.

### R

The type of the field to be loaded.

## Remarks

This method uses a DataLoader to load the specified field of the target data model. If multiple results are loaded, only the first result is returned.

## LoadDataLoaderField<T, R>(IRelmQuickContext, T, Expression<Func<T, R>>)

Loads a specified field of a data model using a DataLoader and returns the first result.

```
public static T LoadDataLoaderField<T, R>(IRelmQuickContext relmQuickContext, T target, Expression<Func<T, R>> predicate) where T : IRelmModel, new()
```

## Parameters

### [relmQuickContext](#) [IRelmQuickContext](#)

The context used to interact with the data source.

## target T

The target data model instance whose field is to be loaded.

## predicate [Expression<Func<T, R>>](#)

An expression specifying the field to load from the data model.

## Returns

### T

The first result of the loaded field, or the default value of [R](#) if no results are found.

## Type Parameters

### T

The type of the data model, which must implement [IRelmModel](#) and have a parameterless constructor.

### R

The type of the field to be loaded.

## Remarks

This method uses a DataLoader to load the specified field of the target data model. If multiple results are loaded, only the first result is returned.

## LoadForeignKeyField<T, R>(RelmContextOptionsBuilder, ICollection<T>, Expression<Func<T, ICollection<R>>>)

Loads a foreign key field for a collection of entities, resolving the related entities based on the specified predicate.

```
public static ICollection<T> LoadForeignKeyField<T, R>(RelmContextOptionsBuilder  
relmContextOptionsBuilder, ICollection<T> target, Expression<Func<T, ICollection<R>>>  
predicate) where T : IRelmModel, new() where R : IRelmModel, new()
```

## Parameters

### relmContextOptionsBuilder [RelmContextOptionsBuilder](#)

The options builder used to configure the database context.

#### target [ICollection](#)<T>

The collection of primary entities for which the foreign key field will be loaded.

#### predicate [Expression](#)<[Func](#)<T, [ICollection](#)<R>>>

An expression specifying the foreign key relationship to load.

Returns

#### [ICollection](#)<T>

A collection of the primary entities with the specified foreign key field loaded.

Type Parameters

#### T

The type of the primary entity. Must implement [IRelmModel](#).

#### R

The type of the related entity. Must implement [IRelmModel](#).

Remarks

This method uses the provided `relmContextOptionsBuilder` to configure the database context and resolves the foreign key relationship defined by `predicate` for the given `target` collection.

## LoadForeignKeyField<T, R>(RelmContextOptionsBuilder, ICollection<T>, Expression<Func<T, ICollection<R>>>, Expression<Func<R, object>>)

Loads and populates a foreign key collection field for the specified target entities.

```
public static ICollection<T> LoadForeignKeyField<T, R>(RelmContextOptionsBuilder  
relmContextOptionsBuilder, ICollection<T> target, Expression<Func<T, ICollection<R>>>  
predicate, Expression<Func<R, object>>> additionalConstraints) where T : IRelmModel, new()  
where R : IRelmModel, new()
```

## Parameters

`relmContextOptionsBuilder` [RelmContextOptionsBuilder](#)

The options builder used to configure the database context.

`target` [ICollection<T>](#)

The collection of target entities for which the foreign key field will be loaded.

`predicate` [Expression<Func<T, ICollection<R>>>](#)

An expression specifying the foreign key collection property to load.

`additionalConstraints` [Expression<Func<R, object>>>](#)

An expression defining additional constraints to apply when loading the related entities.

## Returns

[ICollection<T>](#)

A collection of the target entities with the specified foreign key field populated.

## Type Parameters

`T`

The type of the target entities that contain the foreign key collection.

`R`

The type of the related entities in the foreign key collection.

## Remarks

This method uses the provided `relmContextOptionsBuilder` to configure the database context and loads the related entities for the specified foreign key field. The `additionalConstraints` parameter can be used to filter or constrain the related entities being loaded.

**LoadForeignKeyField<T, R>(RelmContextOptionsBuilder, ICollection<T>, Expression<Func<T, R>>)**

Loads and resolves a foreign key field for a collection of entities.

```
public static ICollection<T> LoadForeignKeyField<T, R>(RelmContextOptionsBuilder  
relmContextOptionsBuilder, ICollection<T> target, Expression<Func<T, R>> predicate) where T  
: IRelmModel, new() where R : IRelmModel, new()
```

## Parameters

`relmContextOptionsBuilder` [RelmContextOptionsBuilder](#)

The options builder used to configure the database context.

`target` [ICollection](#)<T>

The collection of entities for which the foreign key field will be loaded.

`predicate` [Expression](#)<[Func](#)<T, R>>

An expression specifying the foreign key field to load.

## Returns

[ICollection](#)<T>

A collection of entities of type `T` with the specified foreign key field resolved.

## Type Parameters

`T`

The type of the entity in the target collection. Must implement [IRelmModel](#).

`R`

The type of the related entity being loaded. Must implement [IRelmModel](#).

## Remarks

This method uses a `CoreRelm.RelmInternal.Helpers.Utilities.ForeignKeyLoader<T>` to load the foreign key field for the specified collection. The `predicate` parameter determines which foreign key field is resolved.

`LoadForeignKeyField<T, R>(RelmContextOptionsBuilder,  
ICollection<T>, Expression<Func<T, R>>, Expression<Func<R,`

## object>>)

Loads and resolves a foreign key field for a collection of entities, applying the specified constraints.

```
public static ICollection<T> LoadForeignKeyField<T, R>(RelmContextOptionsBuilder  
relmContextOptionsBuilder, ICollection<T> target, Expression<Func<T, R>> predicate,  
Expression<Func<R, object>> additionalConstraints) where T : IRelmModel, new() where R :  
IRelmModel, new()
```

## Parameters

`relmContextOptionsBuilder` [RelmContextOptionsBuilder](#)

The options builder used to configure the database context for the operation.

`target` [ICollection](#)<T>

The collection of primary entities for which the foreign key field will be loaded.

`predicate` [Expression](#)<Func<T, R>>

An expression specifying the foreign key relationship between the primary and related entities.

`additionalConstraints` [Expression](#)<Func<R, object>>

An optional expression specifying additional constraints to apply when resolving the foreign key field.

## Returns

[ICollection](#)<T>

A collection of the primary entities with the foreign key field resolved and loaded.

## Type Parameters

`T`

The type of the primary entity in the collection. Must implement [IRelmModel](#).

`R`

The type of the related entity referenced by the foreign key. Must implement [IRelmModel](#).

## Remarks

This method uses a CoreRelm.RelmInternal.Helpers.Utilities.ForeignKeyLoader<T> to resolve and load the foreign key field for the specified collection of entities. The **predicate** defines the relationship between the primary and related entities, while the **additionalConstraints** can be used to further filter or constrain the related entities.

## LoadForeignKeyField<T, R>(RelmContextOptionsBuilder, T, Expression<Func<T, ICollection<R>>>)

Loads a foreign key field for the specified target entity and retrieves the first related entity matching the given predicate.

```
public static T LoadForeignKeyField<T, R>(RelmContextOptionsBuilder  
    relmContextOptionsBuilder, T target, Expression<Func<T, ICollection<R>>> predicate) where T  
    : IRelmModel, new() where R : IRelmModel, new()
```

### Parameters

**relmContextOptionsBuilder** [RelmContextOptionsBuilder](#)

The options builder used to configure the database context.

**target** T

The target entity for which the foreign key field is being loaded.

**predicate** [Expression](#)<[Func](#)<T, ICollection><R>>>

An expression specifying the collection navigation property on the target entity that represents the foreign key relationship.

### Returns

T

The first related entity matching the specified predicate, or [null](#) if no related entities are found.

### Type Parameters

T

The type of the target entity, which must implement [IRelmModel](#).

R

The type of the related entity, which must implement [IRelmModel](#).

## Remarks

This method is typically used to load and access a specific related entity in a foreign key relationship. Ensure that the `relmContextOptionsBuilder` is properly configured to access the database context.

`LoadForeignKeyField<T, R>(RelmContextOptionsBuilder, T, Expression<Func<T, ICollection<R>>>, Expression<Func<R, object>>)`

Loads a foreign key field for the specified target entity, applying the given predicate and additional constraints.

```
public static T LoadForeignKeyField<T, R>(RelmContextOptionsBuilder  
relmContextOptionsBuilder, T target, Expression<Func<T, ICollection<R>>> predicate,  
Expression<Func<R, object>> additionalConstraints) where T : IRelmModel, new() where R :  
IRelmModel, new()
```

## Parameters

`relmContextOptionsBuilder` [RelmContextOptionsBuilder](#)

The options builder used to configure the database context.

`target` T

The target entity for which the foreign key field is being loaded.

`predicate` [Expression](#)<[Func](#)<T, [ICollection](#)<R>>>

An expression specifying the collection navigation property on the target entity.

`additionalConstraints` [Expression](#)<[Func](#)<R, [object](#)>>

An expression specifying additional constraints to apply when loading the related entities.

## Returns

T

The first related entity that matches the specified predicate and constraints, or [null](#) if no match is found.

## Type Parameters

T

The type of the target entity, which must implement [IRelmModel](#).

R

The type of the related entity, which must implement [IRelmModel](#).

## LoadForeignKeyField<T, R>(RelmContextOptionsBuilder, T, Expression<Func<T, R>>)

Loads a foreign key field for the specified target entity using the provided predicate.

```
public static T LoadForeignKeyField<T, R>(RelmContextOptionsBuilder  
relmContextOptionsBuilder, T target, Expression<Func<T, R>> predicate) where T : IRelmModel,  
new() where R : IRelmModel, new()
```

## Parameters

relmContextOptionsBuilder [RelmContextOptionsBuilder](#)

The options builder used to configure the database context.

target T

The target entity for which the foreign key field is to be loaded.

predicate [Expression](#)<[Func](#)<T, R>>

An expression specifying the foreign key field to load.

## Returns

T

The first related entity of type R that matches the foreign key field, or [null](#) if no match is found.

# Type Parameters

T

The type of the target entity. Must implement [IRelmModel](#) and have a parameterless constructor.

R

The type of the related entity. Must implement [IRelmModel](#) and have a parameterless constructor.

## Remarks

This method uses a CoreRelm.RelmInternal.Helpers.Utilities.ForeignKeyLoader<T> to load the foreign key field for the specified target entity.

## LoadForeignKeyField<T, R>(RelmContextOptionsBuilder, T, Expression<Func<T, R>>, Expression<Func<R, object>>)

Loads a foreign key field for the specified target model, applying the given predicate and additional constraints.

```
public static T LoadForeignKeyField<T, R>(RelmContextOptionsBuilder  
    relmContextOptionsBuilder, T target, Expression<Func<T, R>> predicate, Expression<Func<R,  
    object>> additionalConstraints) where T : IRelmModel, new() where R : IRelmModel, new()
```

## Parameters

**relmContextOptionsBuilder** [RelmContextOptionsBuilder](#)

The options builder used to configure the database context.

**target** T

The target model instance for which the foreign key field is being loaded.

**predicate** [Expression](#)<[Func](#)<T, R>>

An expression specifying the foreign key relationship between the target and related models.

**additionalConstraints** [Expression](#)<[Func](#)<R, object>>

An optional expression specifying additional constraints to apply when loading the related model.

## Returns

T

The first related model that matches the specified predicate and additional constraints, or [null](#) if no match is found.

## Type Parameters

T

The type of the target model. Must implement [IRelmModel](#).

R

The type of the related model. Must implement [IRelmModel](#).

## Remarks

This method is typically used to load a related model for a given target model based on a foreign key relationship. The method applies the specified predicate and additional constraints to filter the related models.

## LoadForeignKeyField<T, R, S>(RelmContextOptionsBuilder, ICollection<T>, Expression<Func<T, ICollection<R>>>, IRelmDataLoader<S>)

Loads a foreign key field for a collection of entities, using the specified predicate and custom data loader.

```
public static ICollection<T> LoadForeignKeyField<T, R, S>(RelmContextOptionsBuilder  
    relmContextOptionsBuilder, ICollection<T> target, Expression<Func<T, ICollection<R>>>  
    predicate, IRelmDataLoader<S> customDataLoader) where T : IRelmModel, new() where R :  
    IRelmModel, new() where S : IRelmModel, new()
```

## Parameters

relmContextOptionsBuilder [RelmContextOptionsBuilder](#)

The options builder used to configure the database context.

**target** [ICollection](#)<T>

The collection of primary entities for which the foreign key field will be loaded.

**predicate** [Expression](#)<[Func](#)<T, [ICollection](#)<R>>>

An expression specifying the foreign key field to load for each entity in the collection.

**customDataLoader** [IRelmDataLoader](#)<S>

A custom data loader used to retrieve the related entities.

Returns

[ICollection](#)<T>

A collection of the primary entities with the specified foreign key field loaded.

Type Parameters

T

The type of the primary entity in the collection. Must implement [IRelmModel](#).

R

The type of the related entity in the foreign key relationship. Must implement [IRelmModel](#).

S

The type of the entity used by the custom data loader. Must implement [IRelmModel](#).

Remarks

This method facilitates the loading of related entities for a collection of primary entities by leveraging a custom data loader. The **predicate** defines the navigation property representing the foreign key relationship, and the **customDataLoader** provides the mechanism for retrieving the related data.

**LoadForeignKeyField<T, R, S>(RelmContextOptionsBuilder,  
ICollection<T>, Expression<Func<T, ICollection<R>>>,  
IRelmDataLoader<S>, Expression<Func<R, object>>)**

Loads and populates a foreign key field for a collection of entities, applying optional constraints and using a custom data loader.

```
public static ICollection<T> LoadForeignKeyField<T, R, S>(RelmContextOptionsBuilder  
relmContextOptionsBuilder, ICollection<T> target, Expression<Func<T, ICollection<R>>>  
predicate, IRelmDataLoader<S> customDataLoader, Expression<Func<R, object>>>  
additionalConstraints) where T : IRelmModel, new() where R : IRelmModel, new() where S :  
IRelmModel, new()
```

## Parameters

`relmContextOptionsBuilder` [RelmContextOptionsBuilder](#)

The options builder used to configure the database context for the operation.

`target` [ICollection](#)<T>

The collection of primary entities for which the foreign key field will be loaded. Cannot be [null](#).

`predicate` [Expression](#)<Func<T, ICollection<R>>>

An expression specifying the foreign key field to load for each entity in the collection.

`customDataLoader` [IRelmDataLoader](#)<S>

A custom data loader used to retrieve the related entities. Cannot be [null](#).

`additionalConstraints` [Expression](#)<Func<R, object>>>

An optional expression specifying additional constraints to apply when loading the related entities.

## Returns

[ICollection](#)<T>

A collection of the primary entities with the specified foreign key field populated.

## Type Parameters

T

The type of the primary entity in the collection. Must implement [IRelmModel](#).

R

The type of the related entity in the foreign key relationship. Must implement [IRelmModel](#).

## S

The type of the entity used by the custom data loader. Must implement [IRelmModel](#).

## Remarks

This method is designed to simplify the process of loading related entities for a collection of primary entities. It supports applying additional constraints to filter the related entities and allows the use of a custom data loader for advanced scenarios. The method ensures that the foreign key field specified by `predicate` is populated for each entity in the `target` collection.

## LoadForeignKeyField<T, R, S>(RelmContextOptionsBuilder, ICollection<T>, Expression<Func<T, R>>, IRelmDataLoader<S>)

Loads and resolves a foreign key field for a collection of entities, using the specified predicate and data loader.

```
public static ICollection<T> LoadForeignKeyField<T, R, S>(RelmContextOptionsBuilder  
relmContextOptionsBuilder, ICollection<T> target, Expression<Func<T, R>> predicate,  
IRelmDataLoader<S> customDataLoader) where T : IRelmModel, new() where R : IRelmModel, new()  
where S : IRelmModel, new()
```

## Parameters

`relmContextOptionsBuilder` [RelmContextOptionsBuilder](#)

The options builder used to configure the database context for loading related data.

`target` [ICollection](#)<T>

The collection of primary entities for which the foreign key field will be loaded.

`predicate` [Expression](#)<Func<T, R>>

An expression specifying the foreign key relationship to be resolved.

`customDataLoader` [IRelmDataLoader](#)<S>

A custom data loader used to fetch the related entities.

## Returns

### [ICollection](#) <T>

A collection of the primary entities with the foreign key field resolved.

## Type Parameters

### T

The type of the primary entity in the collection. Must implement [IRelmModel](#).

### R

The type of the related entity referenced by the foreign key. Must implement [IRelmModel](#).

### S

The type of the entity used by the custom data loader. Must implement [IRelmModel](#).

## Remarks

This method uses a CoreRelm.RelmInternal.Helpers.Utilities.ForeignKeyLoader<T> to load and resolve the foreign key field for the specified collection of entities. The `customDataLoader` allows for custom logic to be applied when fetching the related entities.

## LoadForeignKeyField<T, R, S>(RelmContextOptionsBuilder, ICollection<T>, Expression<Func<T, R>>, IRelmDataLoader<S>, Expression<Func<R, object>>)

Loads and resolves a foreign key field for a collection of entities, applying optional constraints and using a custom data loader.

```
public static ICollection<T> LoadForeignKeyField<T, R, S>(RelmContextOptionsBuilder  
    relmContextOptionsBuilder, ICollection<T> target, Expression<Func<T, R>> predicate,  
    IRelmDataLoader<S> customDataLoader, Expression<Func<R, object>> additionalConstraints)  
    where T : IRelmModel, new() where R : IRelmModel, new() where S : IRelmModel, new()
```

## Parameters

### relmContextOptionsBuilder [RelmContextOptionsBuilder](#)

The options builder used to configure the database context for loading the foreign key field.

#### target [ICollection](#)<T>

The collection of primary entities for which the foreign key field will be loaded.

#### predicate [Expression](#)<[Func](#)<T, R>>

An expression specifying the foreign key relationship between the primary entity and the related entity.

#### customDataLoader [IRelmDataLoader](#)<S>

A custom data loader used to retrieve the related entities.

#### additionalConstraints [Expression](#)<[Func](#)<R, [object](#)>>

An optional expression specifying additional constraints to apply when resolving the foreign key field.

Returns

#### [ICollection](#)<T>

A collection of the primary entities with the foreign key field resolved and loaded.

Type Parameters

#### T

The type of the primary entity in the collection. Must implement [IRelmModel](#).

#### R

The type of the related entity referenced by the foreign key. Must implement [IRelmModel](#).

#### S

The type of the entity used by the custom data loader. Must implement [IRelmModel](#).

Remarks

This method is designed to simplify the process of resolving foreign key relationships for a collection of entities. It allows for the use of a custom data loader and additional constraints to tailor the loading process to specific requirements.

# LoadForeignKeyField<T, R, S>(RelmContextOptionsBuilder, T, Expression<Func<T, ICollection<R>>>, IRelmDataLoader<S>)

Loads a foreign key field for the specified target entity using the provided predicate and custom data loader.

```
public static T LoadForeignKeyField<T, R, S>(RelmContextOptionsBuilder  
relmContextOptionsBuilder, T target, Expression<Func<T, ICollection<R>>> predicate,  
IRelmDataLoader<S> customDataLoader) where T : IRelmModel, new() where R : IRelmModel, new()  
where S : IRelmModel, new()
```

## Parameters

**relmContextOptionsBuilder** [RelmContextOptionsBuilder](#)

The options builder used to configure the database context.

**target** **T**

The target entity for which the foreign key field is being loaded.

**predicate** [Expression](#)<[Func](#)<T, [ICollection](#)<R>>>

An expression specifying the collection navigation property on the target entity.

**customDataLoader** [IRelmDataLoader](#)<S>

A custom data loader used to retrieve the related data.

## Returns

**T**

The first related entity from the loaded foreign key collection, or [null](#) if no related entities are found.

## Type Parameters

**T**

The type of the target entity. Must implement [IRelmModel](#).

**R**

The type of the related entity in the collection. Must implement [IRelmModel](#).

The type of the entity used by the custom data loader. Must implement [IRelmModel](#).

## Remarks

This method uses a custom data loader to retrieve the related entities for the specified foreign key field. The first entity in the resulting collection is returned. If the collection is empty, the method returns [null](#).

**LoadForeignKeyField<T, R, S>(RelmContextOptionsBuilder, T, Expression<Func<T, ICollection<R>>>, IRelmDataLoader<S>, Expression<Func<R, object>>)**

Loads a foreign key field for the specified target model, applying the given predicate, custom data loader, and additional constraints.

```
public static T LoadForeignKeyField<T, R, S>(RelmContextOptionsBuilder
    relmContextOptionsBuilder, T target, Expression<Func<T, ICollection<R>>> predicate,
    IRelmDataLoader<S> customDataLoader, Expression<Func<R, object>> additionalConstraints)
    where T : IRelmModel, new() where R : IRelmModel, new() where S : IRelmModel, new()
```

## Parameters

**relmContextOptionsBuilder** [RelmContextOptionsBuilder](#)

The options builder used to configure the database context for the operation.

**target** T

The target model instance for which the foreign key field will be loaded.

**predicate** [Expression](#)<[Func](#)<T, [ICollection](#)<R>>>

An expression specifying the foreign key field to load as a collection of related entities.

**customDataLoader** [IRelmDataLoader](#)<S>

A custom data loader used to retrieve the related entities.

**additionalConstraints** [Expression](#)<[Func](#)<R, object>>>

An expression specifying additional constraints to apply when loading the related entities.

## Returns

T

The first related entity that matches the specified predicate and constraints, or [null](#) if no match is found.

## Type Parameters

T

The type of the target model that contains the foreign key field. Must implement [IRelmModel](#).

R

The type of the related model in the foreign key relationship. Must implement [IRelmModel](#).

S

The type of the model used by the custom data loader. Must implement [IRelmModel](#).

## Remarks

This method is typically used to load a foreign key field for a model in scenarios where additional constraints or a custom data loader are required.

## LoadForeignKeyField<T, R, S>(RelmContextOptionsBuilder, T, Expression<Func<T, R>>, IRelmDataLoader<S>)

Loads a foreign key field for the specified target model using the provided predicate and data loader.

```
public static T LoadForeignKeyField<T, R, S>(RelmContextOptionsBuilder  
    relmContextOptionsBuilder, T target, Expression<Func<T, R>> predicate, IRelmDataLoader<S>  
    customDataLoader) where T : IRelmModel, new() where R : IRelmModel, new() where S :  
    IRelmModel, new()
```

## Parameters

relmContextOptionsBuilder [RelmContextOptionsBuilder](#)

The options builder used to configure the database context.

## target T

The target model instance for which the foreign key field is being loaded.

## predicate [Expression<Func<T, R>>](#)

An expression specifying the foreign key property to load.

## customDataLoader [IRelmDataLoader<S>](#)

A custom data loader used to retrieve the related data.

## Returns

### T

The first related model of type [R](#) that matches the foreign key, or [null](#) if no match is found.

## Type Parameters

### T

The type of the target model. Must implement [IRelmModel](#) and have a parameterless constructor.

### R

The type of the related model referenced by the foreign key. Must implement [IRelmModel](#) and have a parameterless constructor.

### S

The type of the model used by the custom data loader. Must implement [IRelmModel](#) and have a parameterless constructor.

## Remarks

This method uses a CoreRelm.RelmInternal.Helpers.Utilities.ForeignKeyLoader<T> to load the foreign key field for the specified target model. Ensure that the provided [predicate](#) correctly identifies the foreign key property to be loaded.

## LoadForeignKeyField<T, R, S>(RelmContextOptionsBuilder, T, Expression<Func<T, R>>, IRelmDataLoader<S>, Expression<Func<R, object>>)

Loads a foreign key field for the specified target model using the provided predicate and optional constraints.

```
public static T LoadForeignKeyField<T, R, S>(RelmContextOptionsBuilder<T> relmContextOptionsBuilder, T target, Expression<Func<T, R>> predicate, IRelmDataLoader<S> customDataLoader, Expression<Func<R, object>> additionalConstraints) where T : IRelmModel, new() where R : IRelmModel, new() where S : IRelmModel, new()
```

## Parameters

`relmContextOptionsBuilder` [RelmContextOptionsBuilder](#)

The options builder used to configure the database context.

`target` `T`

The target model instance for which the foreign key field is being loaded.

`predicate` [Expression](#)<[Func](#)<T, R>>

An expression specifying the foreign key property on the target model.

`customDataLoader` [IRelmDataLoader](#)<S>

An optional custom data loader to retrieve the related data.

`additionalConstraints` [Expression](#)<[Func](#)<R, object>>

An optional expression specifying additional constraints to apply when loading the foreign key field.

## Returns

`T`

The first related model that matches the specified predicate and constraints, or [null](#) if no match is found.

## Type Parameters

`T`

The type of the target model. Must implement [IRelmModel](#).

`R`

The type of the related model representing the foreign key. Must implement [IRelModel](#).

## S

The type of the model used by the custom data loader. Must implement [IRelModel](#).

## Remarks

This method is typically used to load a foreign key relationship for a model instance, optionally applying additional constraints or using a custom data loader to retrieve the related data.

## StandardConnectionWrapper(Enum, Action<MySqlConnection, MySqlTransaction>, Action<Exception, string>)

Provides a standardized wrapper for executing database operations within a connection and transaction context.

```
public static void StandardConnectionWrapper(Enum connectionName, Action<MySqlConnection, MySqlTransaction> actionWrapper, Action<Exception, string> exceptionHandler = null)
```

## Parameters

### connectionName [Enum](#)

The type of database connection to use, represented as an enumeration value.

### actionWrapper [Action](#)<MySqlConnection, MySqlTransaction>

A delegate that defines the operations to perform using the provided `MySQL.Data.MySqlClient.MySqlConnection` and `MySQL.Data.MySqlClient.MySqlTransaction`. The connection and transaction are managed by the wrapper.

### exceptionHandler [Action](#)<[Exception](#), [string](#)>

An optional delegate to handle exceptions that occur during the execution of `actionWrapper`. The delegate receives the exception and an error message as parameters. If not provided, exceptions will propagate to the caller.

## Remarks

This method ensures that the database connection and transaction are properly managed, including opening, committing, rolling back, and disposing of resources as necessary. Use this wrapper to simplify

error handling and resource management for database operations.

## StandardConnectionWrapper<T>(Enum, Func<MySqlConnection, MySqlTransaction, T>, Action<Exception, string>)

Executes a database operation within a standard connection and transaction context.

```
public static T StandardConnectionWrapper<T>(Enum connectionName, Func<MySqlConnection, MySqlTransaction, T> actionWrapper, Action<Exception, string> exceptionHandler = null)
```

### Parameters

**connectionName** [Enum](#)

The name of the database connection, represented as an [Enum](#).

**actionWrapper** [Func](#)<MySqlConnection, MySqlTransaction, T>

A delegate that defines the operation to execute. The delegate receives a MySql.Data.MySqlClient.MySqlConnection and a MySql.Data.MySqlClient.MySqlTransaction as parameters and returns a result of type **T**.

**exceptionHandler** [Action](#)<[Exception](#), [string](#)>

An optional delegate to handle exceptions that occur during the operation. The delegate receives the exception and a string message describing the context of the error.

### Returns

**T**

The result of the operation, as defined by the **actionWrapper** delegate.

### Type Parameters

**T**

The type of the result returned by the operation.

### Remarks

This method provides a standardized way to execute database operations, ensuring that the connection and transaction are properly managed. If an exception occurs and an `exceptionHandler` is provided, the exception will be passed to the handler for custom processing.

## UseConfiguration(IConfiguration)

Configures the library to use the specified [IConfiguration](#) instance for application settings and options.

```
public static void UseConfiguration(IConfiguration configuration)
```

### Parameters

`configuration` [IConfiguration](#)

The [IConfiguration](#) instance that provides configuration values for the library. Cannot be null.

### Remarks

Call this method at application startup to ensure the library uses the correct configuration source. This method should be invoked before performing any operations that depend on configuration values.

## WriteToDatabase(IRelmContext, IRelmModel, int, bool, bool, bool, bool)

Writes the specified model to the database using the provided context and configuration options.

```
public static int WriteToDatabase(IRelmContext relmContext, IRelmModel relmModel, int  
batchSize = 100, bool allowAutoIncrementColumns = false, bool allowPrimaryKeyColumns =  
false, bool allowUniqueColumns = false, bool allowAutoDateColumns = false)
```

### Parameters

`relmContext` [IRelmContext](#)

The database context used to perform the write operation.

`relmModel` [IRelmModel](#)

The model to be written to the database.

### batchSize [int](#)

The number of records to process in a single batch. Defaults to 100. Must be a positive integer.

### allowAutoIncrementColumns [bool](#)

A value indicating whether columns with auto-increment constraints are allowed to be written. Defaults to [false](#).

### allowPrimaryKeyColumns [bool](#)

A value indicating whether primary key columns are allowed to be written. Defaults to [false](#).

### allowUniqueColumns [bool](#)

A value indicating whether columns with unique constraints are allowed to be written. Defaults to [false](#).

### allowAutoDateColumns [bool](#)

A value indicating whether columns with automatic date generation are allowed to be written. Defaults to [false](#).

## Returns

### [int](#)

The number of records successfully written to the database.

## Remarks

This method delegates the write operation to the `IRelmModel` implementation, which performs the actual database interaction. The behavior of the write operation is influenced by the provided configuration flags.

## WriteToDatabase(IRelmContext, IEnumerable<IRelmModel>, int, bool, bool, bool, bool)

Writes a collection of models to the specified database context in batches.

```
public static int WriteToDatabase(IRelmContext relmContext, IEnumerable<IRelmModel>
relmModels, int batchSize = 100, bool allowAutoIncrementColumns = false, bool
```

```
allowPrimaryKeyColumns = false, bool allowUniqueColumns = false, bool allowAutoDateColumns  
= false)
```

## Parameters

### realmContext [IRelmContext](#)

The database context to which the models will be written. This parameter cannot be [null](#).

### realmModels [IEnumerable](#)<[IRelmModel](#)>

The collection of models to write to the database. This parameter cannot be [null](#) or empty.

### batchSize [int](#)

The number of models to include in each batch. Must be a positive integer. The default value is 100.

### allowAutoIncrementColumns [bool](#)

A value indicating whether auto-increment columns are allowed to be written. If [true](#), auto-increment columns will be included; otherwise, they will be excluded. The default value is [false](#).

### allowPrimaryKeyColumns [bool](#)

A value indicating whether primary key columns are allowed to be written. If [true](#), primary key columns will be included; otherwise, they will be excluded. The default value is [false](#).

### allowUniqueColumns [bool](#)

A value indicating whether unique columns are allowed to be written. If [true](#), unique columns will be included; otherwise, they will be excluded. The default value is [false](#).

### allowAutoDateColumns [bool](#)

A value indicating whether auto-generated date columns are allowed to be written. If [true](#), auto-generated date columns will be included; otherwise, they will be excluded. The default value is [false](#).

## Returns

### [int](#)

The total number of models successfully written to the database.

## Remarks

This method writes the provided models to the database in batches to optimize performance. The behavior of the write operation can be customized using the optional parameters to control whether specific column types (e.g., auto-increment, primary key, unique, or auto-generated date columns) are included in the operation.

## WriteToDatabase(IRelmQuickContext, IRelmModel, int, bool, bool, bool, bool)

Writes the specified model to the database using the provided context and configuration options.

```
public static int WriteToDatabase(IRelmQuickContext relmQuickContext, IRelmModel relmModel, int batchSize = 100, bool allowAutoIncrementColumns = false, bool allowPrimaryKeyColumns = false, bool allowUniqueColumns = false, bool allowAutoDateColumns = false)
```

### Parameters

`relmQuickContext` [IRelmQuickContext](#)

The database context used to manage the connection and transaction for the operation.

`relmModel` [IRelmModel](#)

The model to be written to the database. This model must conform to the expected schema.

`batchSize` [int](#)

The number of records to process in a single batch. Defaults to 100. Must be a positive integer.

`allowAutoIncrementColumns` [bool](#)

A value indicating whether columns with auto-increment constraints are allowed to be written. If [true](#), auto-increment columns will be included in the operation; otherwise, they will be excluded.

`allowPrimaryKeyColumns` [bool](#)

A value indicating whether primary key columns are allowed to be written. If [true](#), primary key columns will be included in the operation; otherwise, they will be excluded.

`allowUniqueColumns` [bool](#)

A value indicating whether unique columns are allowed to be written. If [true](#), unique columns will be included in the operation; otherwise, they will be excluded.

## allowAutoDateColumns [bool](#)

A value indicating whether columns with automatic date generation constraints are allowed to be written. If [true](#), auto-date columns will be included in the operation; otherwise, they will be excluded.

## Returns

### [int](#)

The number of records successfully written to the database.

## Remarks

This method provides fine-grained control over which types of columns are included in the database write operation. Use the configuration options to tailor the behavior to your specific requirements.

## WriteToDatabase(IRelmQuickContext, IEnumerable<IRelmModel>, int, bool, bool, bool, bool)

Writes a collection of Relm models to the database using the specified context and configuration options.

```
public static int WriteToDatabase(IRelmQuickContext relmQuickContext,  
IEnumerable<IRelmModel> relmModels, int batchSize = 100, bool allowAutoIncrementColumns =  
false, bool allowPrimaryKeyColumns = false, bool allowUniqueColumns = false, bool  
allowAutoDateColumns = false)
```

## Parameters

### [relmQuickContext](#) [IRelmQuickContext](#)

The database context used to manage the connection and transaction for the operation.

### [relmModels](#) [IEnumerable](#)<IRelmModel>

The collection of models to be written to the database. Each model represents a row to be inserted or updated.

### [batchSize](#) [int](#)

The number of models to process in a single batch. Defaults to 100. Larger batch sizes may improve performance but require more memory.

## `allowAutoIncrementColumns bool`

A value indicating whether columns marked as auto-increment are allowed to be written. Defaults to [false](#).

## `allowPrimaryKeyColumns bool`

A value indicating whether primary key columns are allowed to be written. Defaults to [false](#).

## `allowUniqueColumns bool`

A value indicating whether columns with unique constraints are allowed to be written. Defaults to [false](#).

## `allowAutoDateColumns bool`

A value indicating whether columns with automatic date generation (e.g., timestamps) are allowed to be written. Defaults to [false](#).

## Returns

### `int`

The total number of rows successfully written to the database.

## Remarks

This method provides fine-grained control over how data is written to the database by allowing the caller to specify whether certain types of columns (e.g., auto-increment, primary key, unique, or auto-date columns) are included in the operation. Use caution when enabling these options, as they may violate database constraints or lead to unexpected behavior.

# Namespace CoreRelm.Attributes

## Classes

### [RelmColumn](#)

Specifies metadata for a database column, including its name, size, constraints, and other properties.

### [RelmDataLoader](#)

Specifies metadata for associating a data loader with a property, struct, or class.

### [RelmDatabase](#)

Specifies that a class or struct represents a database entity and associates it with a database name.

### [RelmDto](#)

Specifies that the associated property or struct is part of a Data Transfer Object (DTO) used for communication between application layers or services.

### [RelmForeignKey](#)

Specifies a foreign key relationship for a property or struct in a data model.

### [RelmKey](#)

Specifies that the associated property or struct is a key used for identifying or referencing entities.

### [RelmTable](#)

Specifies that a class or struct represents a table in a relational database and provides the name of the table.

# Class RelmColumn

Namespace: [CoreRelm.Attributes](#)

Assembly: CoreRelm.dll

Specifies metadata for a database column, including its name, size, constraints, and other properties.

```
[AttributeUsage(AttributeTargets.Struct | AttributeTargets.Property)]
public sealed class RelmColumn : Attribute
```

## Inheritance

[object](#) ← [Attribute](#) ← RelmColumn

## Inherited Members

[Attribute.Equals\(object\)](#) , [Attribute.GetCustomAttribute\(Assembly, Type\)](#) ,  
[Attribute.GetCustomAttribute\(Assembly, Type, bool\)](#) ,  
[Attribute.GetCustomAttribute\(MemberInfo, Type\)](#) ,  
[Attribute.GetCustomAttribute\(MemberInfo, Type, bool\)](#) ,  
[Attribute.GetCustomAttribute\(Module, Type\)](#) , [Attribute.GetCustomAttribute\(Module, Type, bool\)](#) ,  
[Attribute.GetCustomAttribute\(ParameterInfo, Type\)](#) ,  
[Attribute.GetCustomAttribute\(ParameterInfo, Type, bool\)](#) , [Attribute.GetCustomAttributes\(Assembly\)](#) ,  
[Attribute.GetCustomAttributes\(Assembly, bool\)](#) , [Attribute.GetCustomAttributes\(Assembly, Type\)](#) ,  
[Attribute.GetCustomAttributes\(Assembly, Type, bool\)](#) , [Attribute.GetCustomAttributes\(MemberInfo\)](#) ,  
[Attribute.GetCustomAttributes\(MemberInfo, bool\)](#) ,  
[Attribute.GetCustomAttributes\(MemberInfo, Type\)](#) ,  
[Attribute.GetCustomAttributes\(MemberInfo, Type, bool\)](#) , [Attribute.GetCustomAttributes\(Module\)](#) ,  
[Attribute.GetCustomAttributes\(Module, bool\)](#) , [Attribute.GetCustomAttributes\(Module, Type\)](#) ,  
[Attribute.GetCustomAttributes\(Module, Type, bool\)](#) , [Attribute.GetCustomAttributes\(ParameterInfo\)](#) ,  
[Attribute.GetCustomAttributes\(ParameterInfo, bool\)](#) ,  
[Attribute.GetCustomAttributes\(ParameterInfo, Type\)](#) ,  
[Attribute.GetCustomAttributes\(ParameterInfo, Type, bool\)](#) , [Attribute.GetHashCode\(\)](#) ,  
[Attribute.IsDefaultAttribute\(\)](#) , [Attribute.IsDefined\(Assembly, Type\)](#) ,  
[Attribute.IsDefined\(Assembly, Type, bool\)](#) , [Attribute.IsDefined\(MemberInfo, Type\)](#) ,  
[Attribute.IsDefined\(MemberInfo, Type, bool\)](#) , [Attribute.IsDefined\(Module, Type\)](#) ,  
[Attribute.IsDefined\(Module, Type, bool\)](#) , [Attribute.IsDefined\(ParameterInfo, Type\)](#) ,  
[Attribute.IsDefined\(ParameterInfo, Type, bool\)](#) , [Attribute.Match\(object\)](#) , [Attribute.TypeId](#) ,  
[object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.ReferenceEquals\(object, object\)](#) ,  
[object.ToString\(\)](#)

## Remarks

This attribute is used to annotate properties or structs that represent database columns. It provides detailed configuration options such as column name, size, nullability, uniqueness, default values, and indexing. The attribute can be applied to define primary keys, auto-incrementing columns, and virtual columns, among other features. When applied, the annotated property or struct is treated as a database column with the specified characteristics. This attribute is commonly used in object-relational mapping (ORM) scenarios to define the schema of a database table.

## Constructors

### RelmColumn(string, int, int[], bool, bool, bool, bool, string, string, bool, bool, bool)

Initializes a new instance of the [RelmColumn](#) class, representing a column definition with various configuration options for database schema design.

```
public RelmColumn(string columnName = null, int columnSize = -1, int[] compoundColumnSize = null, bool isNullable = true, bool primaryKey = false, bool autonumber = false, bool unique = false, string defaultValue = null, string index = null, bool indexDescending = false, bool allowDataTruncation = false, bool isVirtual = false)
```

#### Parameters

##### columnName [string](#)

The name of the column. If [null](#) or empty, the column will use the property's underscore name.

##### columnSize [int](#)

The size of the column, typically used for fixed-length data types (e.g., strings). A value of -1 indicates that the size is unspecified.

##### compoundColumnSize [int](#)[]

An array specifying the sizes of compound columns, if applicable. Can be [null](#) if the column is not part of a compound structure.

##### isNullable [bool](#)

A value indicating whether the column allows [null](#) values. The default is [true](#).

## **primaryKey** [bool](#)

A value indicating whether the column is part of the primary key. The default is [false](#).

## **autonumber** [bool](#)

A value indicating whether the column is an auto-incrementing identity column. The default is [false](#).

## **unique** [bool](#)

A value indicating whether the column enforces uniqueness. The default is [false](#).

## **defaultValue** [string](#)

The default value for the column, represented as a string. Can be [null](#) if no default value is specified.

## **index** [string](#)

The name of the index associated with the column, if any. Can be [null](#) if the column is not indexed.

## **indexDescending** [bool](#)

A value indicating whether the index on the column is sorted in descending order. The default is [false](#).

## **allowDataTruncation** [bool](#)

A value indicating whether data truncation is allowed for this column. The default is [false](#).

## **isVirtual** [bool](#)

A value indicating whether the column is a virtual column (i.e., its value is computed rather than stored). The default is [false](#).

# Properties

## AllowDataTruncation

Gets a value indicating whether data truncation is allowed during processing.

```
public bool AllowDataTruncation { get; }
```

## Property Value

[bool](#)

## Autonumber

Gets a value indicating whether the entity is configured to automatically generate unique identifiers.

```
public bool Autonumber { get; }
```

### Property Value

[bool](#)

## ColumnName

Gets the name of the database column associated with this instance. If not specified, the underscore version of the property or struct name is used as the column name.

```
public string ColumnName { get; }
```

### PropertyValue

[string](#)

## ColumnSize

Gets the size of the column, typically representing the maximum number of characters or bytes that the column can hold.

```
public int ColumnSize { get; }
```

### PropertyValue

[int](#)

## CompoundColumnSize

Gets the sizes of the compound columns, such as decimal.

```
public int[] CompoundColumnSize { get; }
```

Property Value

[int](#)[]

## DefaultValue

Gets the default value associated with this column.

```
public string DefaultValue { get; }
```

Property Value

[string](#)

## Index

Gets the value indicating which index this property is attached to.

```
public string Index { get; }
```

Property Value

[string](#)

## IndexDescending

Gets a value indicating whether the index is sorted in descending order.

```
public bool IndexDescending { get; }
```

Property Value

[bool](#)

## IsNullable

Gets a value indicating whether the associated entity or value can be null.

```
public bool IsNullable { get; }
```

## Property Value

[bool](#)

## PrimaryKey

Gets a value indicating whether this property is the primary key of the entity.

```
public bool PrimaryKey { get; }
```

## Property Value

[bool](#)

## Unique

Gets a value indicating whether the column can only hold unique values.

```
public bool Unique { get; }
```

## Property Value

[bool](#)

## Virtual

Gets a value indicating whether the object is virtual.

```
public bool Virtual { get; }
```

Property Value

[bool](#) ↗

# Class RelmDataLoader

Namespace: [CoreRelm.Attributes](#)

Assembly: CoreRelm.dll

Specifies metadata for associating a data loader with a property, struct, or class.

```
[AttributeUsage(AttributeTargets.Class | AttributeTargets.Struct | AttributeTargets.Property,
    AllowMultiple = true)]
public class RelmDataLoader : Attribute
```

## Inheritance

[object](#) ← [Attribute](#) ← RelmDataLoader

## Inherited Members

[Attribute.Equals\(object\)](#), [Attribute.GetCustomAttribute\(Assembly, Type\)](#),  
[Attribute.GetCustomAttribute\(Assembly, Type, bool\)](#),  
[Attribute.GetCustomAttribute\(MemberInfo, Type\)](#),  
[Attribute.GetCustomAttribute\(MemberInfo, Type, bool\)](#),  
[Attribute.GetCustomAttribute\(Module, Type\)](#), [Attribute.GetCustomAttribute\(Module, Type, bool\)](#),  
[Attribute.GetCustomAttribute\(ParameterInfo, Type\)](#),  
[Attribute.GetCustomAttribute\(ParameterInfo, Type, bool\)](#), [Attribute.GetCustomAttributes\(Assembly\)](#),  
[Attribute.GetCustomAttributes\(Assembly, bool\)](#), [Attribute.GetCustomAttributes\(Assembly, Type\)](#),  
[Attribute.GetCustomAttributes\(Assembly, Type, bool\)](#), [Attribute.GetCustomAttributes\(MemberInfo\)](#),  
[Attribute.GetCustomAttributes\(MemberInfo, bool\)](#),  
[Attribute.GetCustomAttributes\(MemberInfo, Type\)](#),  
[Attribute.GetCustomAttributes\(MemberInfo, Type, bool\)](#), [Attribute.GetCustomAttributes\(Module\)](#),  
[Attribute.GetCustomAttributes\(Module, bool\)](#), [Attribute.GetCustomAttributes\(Module, Type\)](#),  
[Attribute.GetCustomAttributes\(Module, Type, bool\)](#), [Attribute.GetCustomAttributes\(ParameterInfo\)](#),  
[Attribute.GetCustomAttributes\(ParameterInfo, bool\)](#),  
[Attribute.GetCustomAttributes\(ParameterInfo, Type\)](#),  
[Attribute.GetCustomAttributes\(ParameterInfo, Type, bool\)](#), [Attribute.GetHashCode\(\)](#),  
[Attribute.IsDefaultAttribute\(\)](#), [Attribute.IsDefined\(Assembly, Type\)](#),  
[Attribute.IsDefined\(Assembly, Type, bool\)](#), [Attribute.IsDefined\(MemberInfo, Type\)](#),  
[Attribute.IsDefined\(MemberInfo, Type, bool\)](#), [Attribute.IsDefined\(Module, Type\)](#),  
[Attribute.IsDefined\(Module, Type, bool\)](#), [Attribute.IsDefined\(ParameterInfo, Type\)](#),  
[Attribute.IsDefined\(ParameterInfo, Type, bool\)](#), [Attribute.Match\(object\)](#), [Attribute.TypeId](#),  
[object.Equals\(object, object\)](#), [object.GetType\(\)](#), [object.MemberwiseClone\(\)](#),  
[object.ReferenceEquals\(object, object\)](#), [object.ToString\(\)](#)

## Remarks

This attribute is used to define the type of a data loader and the key fields that the loader uses to identify or retrieve data. It can be applied to properties, structs, or classes, and supports multiple usages on the same target.

## Constructors

### RelmDataLoader(Type)

Initializes a new instance of the [RelmDataLoader](#) class with the specified loader type.

```
public RelmDataLoader(Type loaderType)
```

#### Parameters

**loaderType** [Type](#)

The type of the loader to be used. This parameter cannot be [null](#).

### RelmDataLoader(Type, string)

Initializes a new instance of the [RelmDataLoader](#) class with the specified loader type and key field.

```
public RelmDataLoader(Type loaderType, string keyField)
```

#### Parameters

**loaderType** [Type](#)

The type of the loader used to process data. This parameter cannot be [null](#).

**keyField** [string](#)

The name of the key field used to identify data. If not [null](#), it will be added as the sole key field.

## Remarks

If **keyField** is [null](#), the [KeyFields](#) property will remain uninitialized.

## RelmDataLoader(Type, string[])

Initializes a new instance of the [RelmDataLoader](#) class with the specified loader type and key fields.

```
public RelmDataLoader(Type loaderType, string[] keyFields)
```

### Parameters

**loaderType** [Type](#)

The type of the loader used to process the data. This must be a valid [Type](#) representing the loader implementation.

**keyFields** [string](#)[]

An array of strings representing the key fields used to identify or process the data. This array cannot be null and must contain at least one element.

## Properties

### KeyFields

Gets or sets the collection of key field names used to uniquely identify an entity.

```
public string[] KeyFields { get; set; }
```

### Property Value

[string](#)[]

### Remarks

Key fields are typically used to identify unique records in a dataset or entity. Ensure that the field names provided correspond to valid and unique identifiers within the context of the entity.

### LoaderType

Gets or sets the type of the loader used to process data or resources.

```
public Type LoaderType { get; set; }
```

## Property Value

[Type](#)

## Remarks

The specified type is expected to define the behavior for loading data or resources. Ensure that the type is compatible with the intended usage context.

# Class RelmDatabase

Namespace: [CoreRelm.Attributes](#)

Assembly: CoreRelm.dll

Specifies that a class or struct represents a database entity and associates it with a database name.

```
[AttributeUsage(AttributeTargets.Class | AttributeTargets.Struct)]
public sealed class RelmDatabase : Attribute
```

## Inheritance

[object](#) ← [Attribute](#) ← RelmDatabase

## Inherited Members

[Attribute.Equals\(object\)](#) , [Attribute.GetCustomAttribute\(Assembly, Type\)](#) ,  
[Attribute.GetCustomAttribute\(Assembly, Type, bool\)](#) ,  
[Attribute.GetCustomAttribute\(MemberInfo, Type\)](#) ,  
[Attribute.GetCustomAttribute\(MemberInfo, Type, bool\)](#) ,  
[Attribute.GetCustomAttribute\(Module, Type\)](#) , [Attribute.GetCustomAttribute\(Module, Type, bool\)](#) ,  
[Attribute.GetCustomAttribute\(ParameterInfo, Type\)](#) ,  
[Attribute.GetCustomAttribute\(ParameterInfo, Type, bool\)](#) , [Attribute.GetCustomAttributes\(Assembly\)](#) ,  
[Attribute.GetCustomAttributes\(Assembly, bool\)](#) , [Attribute.GetCustomAttributes\(Assembly, Type\)](#) ,  
[Attribute.GetCustomAttributes\(Assembly, Type, bool\)](#) , [Attribute.GetCustomAttributes\(MemberInfo\)](#) ,  
[Attribute.GetCustomAttributes\(MemberInfo, bool\)](#) ,  
[Attribute.GetCustomAttributes\(MemberInfo, Type\)](#) ,  
[Attribute.GetCustomAttributes\(MemberInfo, Type, bool\)](#) , [Attribute.GetCustomAttributes\(Module\)](#) ,  
[Attribute.GetCustomAttributes\(Module, bool\)](#) , [Attribute.GetCustomAttributes\(Module, Type\)](#) ,  
[Attribute.GetCustomAttributes\(Module, Type, bool\)](#) , [Attribute.GetCustomAttributes\(ParameterInfo\)](#) ,  
[Attribute.GetCustomAttributes\(ParameterInfo, bool\)](#) ,  
[Attribute.GetCustomAttributes\(ParameterInfo, Type\)](#) ,  
[Attribute.GetCustomAttributes\(ParameterInfo, Type, bool\)](#) , [Attribute.GetHashCode\(\)](#) ,  
[Attribute.IsDefaultAttribute\(\)](#) , [Attribute.IsDefined\(Assembly, Type\)](#) ,  
[Attribute.IsDefined\(Assembly, Type, bool\)](#) , [Attribute.IsDefined\(MemberInfo, Type\)](#) ,  
[Attribute.IsDefined\(MemberInfo, Type, bool\)](#) , [Attribute.IsDefined\(Module, Type\)](#) ,  
[Attribute.IsDefined\(Module, Type, bool\)](#) , [Attribute.IsDefined\(ParameterInfo, Type\)](#) ,  
[Attribute.IsDefined\(ParameterInfo, Type, bool\)](#) , [Attribute.Match\(object\)](#) , [Attribute.TypeId](#) ,  
[object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.ReferenceEquals\(object, object\)](#) ,  
[object.ToString\(\)](#)

## Remarks

This attribute is used to annotate classes or structs that correspond to database entities. The [Database Name](#) property specifies the name of the database associated with the entity.

## Constructors

### RelmDatabase(string)

Initializes a new instance of the [RelmDatabase](#) class with the specified database name.

```
public RelmDatabase(string databaseName)
```

#### Parameters

**databaseName** [string](#)

The name of the database. This value cannot be null, empty, or consist only of white-space characters.

#### Exceptions

[ArgumentNullException](#)

Thrown if **databaseName** is null, empty, or consists only of white-space characters.

## Properties

### DatabaseName

Gets or sets the name of the database.

```
public string DatabaseName { get; }
```

#### Property Value

[string](#)

# Class RelmDto

Namespace: [CoreRelm.Attributes](#)

Assembly: CoreRelm.dll

Specifies that the associated property or struct is part of a Data Transfer Object (DTO) used for communication between application layers or services.

```
[AttributeUsage(AttributeTargets.Struct | AttributeTargets.Property)]
public sealed class RelmDto : Attribute
```

## Inheritance

[object](#) ← [Attribute](#) ← RelmDto

## Inherited Members

[Attribute.Equals\(object\)](#) , [Attribute.GetCustomAttribute\(Assembly, Type\)](#) ,  
[Attribute.GetCustomAttribute\(Assembly, Type, bool\)](#) ,  
[Attribute.GetCustomAttribute\(MemberInfo, Type\)](#) ,  
[Attribute.GetCustomAttribute\(MemberInfo, Type, bool\)](#) ,  
[Attribute.GetCustomAttribute\(Module, Type\)](#) , [Attribute.GetCustomAttribute\(Module, Type, bool\)](#) ,  
[Attribute.GetCustomAttribute\(ParameterInfo, Type\)](#) ,  
[Attribute.GetCustomAttribute\(ParameterInfo, Type, bool\)](#) , [Attribute.GetCustomAttributes\(Assembly\)](#) ,  
[Attribute.GetCustomAttributes\(Assembly, bool\)](#) , [Attribute.GetCustomAttributes\(Assembly, Type\)](#) ,  
[Attribute.GetCustomAttributes\(Assembly, Type, bool\)](#) , [Attribute.GetCustomAttributes\(MemberInfo\)](#) ,  
[Attribute.GetCustomAttributes\(MemberInfo, bool\)](#) ,  
[Attribute.GetCustomAttributes\(MemberInfo, Type\)](#) ,  
[Attribute.GetCustomAttributes\(MemberInfo, Type, bool\)](#) , [Attribute.GetCustomAttributes\(Module\)](#) ,  
[Attribute.GetCustomAttributes\(Module, bool\)](#) , [Attribute.GetCustomAttributes\(Module, Type\)](#) ,  
[Attribute.GetCustomAttributes\(Module, Type, bool\)](#) , [Attribute.GetCustomAttributes\(ParameterInfo\)](#) ,  
[Attribute.GetCustomAttributes\(ParameterInfo, bool\)](#) ,  
[Attribute.GetCustomAttributes\(ParameterInfo, Type\)](#) ,  
[Attribute.GetCustomAttributes\(ParameterInfo, Type, bool\)](#) , [Attribute.GetHashCode\(\)](#) ,  
[Attribute.IsDefaultAttribute\(\)](#) , [Attribute.IsDefined\(Assembly, Type\)](#) ,  
[Attribute.IsDefined\(Assembly, Type, bool\)](#) , [Attribute.IsDefined\(MemberInfo, Type\)](#) ,  
[Attribute.IsDefined\(MemberInfo, Type, bool\)](#) , [Attribute.IsDefined\(Module, Type\)](#) ,  
[Attribute.IsDefined\(Module, Type, bool\)](#) , [Attribute.IsDefined\(ParameterInfo, Type\)](#) ,  
[Attribute.IsDefined\(ParameterInfo, Type, bool\)](#) , [Attribute.Match\(object\)](#) , [Attribute.TypeId](#) ,  
[object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.ReferenceEquals\(object, object\)](#) ,  
[object.ToString\(\)](#)

## Remarks

This attribute is intended to annotate properties or structs that are part of a DTO, providing metadata for tools or frameworks that process or validate DTOs. It can be applied to properties or structs to indicate their role in data transfer scenarios.

# Class RelmForeignKey

Namespace: [CoreRelm.Attributes](#)

Assembly: CoreRelm.dll

Specifies a foreign key relationship for a property or struct in a data model.

```
[AttributeUsage(AttributeTargets.Struct | AttributeTargets.Property)]
public sealed class RelmForeignKey : Attribute
```

## Inheritance

[object](#) ← [Attribute](#) ← RelmForeignKey

## Inherited Members

[Attribute.Equals\(object\)](#) , [Attribute.GetCustomAttribute\(Assembly, Type\)](#) ,  
[Attribute.GetCustomAttribute\(Assembly, Type, bool\)](#) ,  
[Attribute.GetCustomAttribute\(MemberInfo, Type\)](#) ,  
[Attribute.GetCustomAttribute\(MemberInfo, Type, bool\)](#) ,  
[Attribute.GetCustomAttribute\(Module, Type\)](#) , [Attribute.GetCustomAttribute\(Module, Type, bool\)](#) ,  
[Attribute.GetCustomAttribute\(ParameterInfo, Type\)](#) ,  
[Attribute.GetCustomAttribute\(ParameterInfo, Type, bool\)](#) , [Attribute.GetCustomAttributes\(Assembly\)](#) ,  
[Attribute.GetCustomAttributes\(Assembly, bool\)](#) , [Attribute.GetCustomAttributes\(Assembly, Type\)](#) ,  
[Attribute.GetCustomAttributes\(Assembly, Type, bool\)](#) , [Attribute.GetCustomAttributes\(MemberInfo\)](#) ,  
[Attribute.GetCustomAttributes\(MemberInfo, bool\)](#) ,  
[Attribute.GetCustomAttributes\(MemberInfo, Type\)](#) ,  
[Attribute.GetCustomAttributes\(MemberInfo, Type, bool\)](#) , [Attribute.GetCustomAttributes\(Module\)](#) ,  
[Attribute.GetCustomAttributes\(Module, bool\)](#) , [Attribute.GetCustomAttributes\(Module, Type\)](#) ,  
[Attribute.GetCustomAttributes\(Module, Type, bool\)](#) , [Attribute.GetCustomAttributes\(ParameterInfo\)](#) ,  
[Attribute.GetCustomAttributes\(ParameterInfo, bool\)](#) ,  
[Attribute.GetCustomAttributes\(ParameterInfo, Type\)](#) ,  
[Attribute.GetCustomAttributes\(ParameterInfo, Type, bool\)](#) , [Attribute.GetHashCode\(\)](#) ,  
[Attribute.IsDefaultAttribute\(\)](#) , [Attribute.IsDefined\(Assembly, Type\)](#) ,  
[Attribute.IsDefined\(Assembly, Type, bool\)](#) , [Attribute.IsDefined\(MemberInfo, Type\)](#) ,  
[Attribute.IsDefined\(MemberInfo, Type, bool\)](#) , [Attribute.IsDefined\(Module, Type\)](#) ,  
[Attribute.IsDefined\(Module, Type, bool\)](#) , [Attribute.IsDefined\(ParameterInfo, Type\)](#) ,  
[Attribute.IsDefined\(ParameterInfo, Type, bool\)](#) , [Attribute.Match\(object\)](#) , [Attribute.TypeId](#) ,  
[object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.ReferenceEquals\(object, object\)](#) ,  
[object.ToString\(\)](#)

## Remarks

This attribute is used to define the mapping between local keys and foreign keys in a relational context. It can also specify an optional ordering for the related data. The attribute can be applied to properties or structs, and supports both single and multiple key mappings.

## Constructors

### RelmForeignKey(string, string, string)

Initializes a new instance of the [RelmForeignKey](#) class with optional foreign key, local key, and order-by values.

```
public RelmForeignKey(string foreignKey = null, string localKey = null, string orderBy = null)
```

#### Parameters

##### foreignKey [string](#)

An optional foreign key used to establish the relationship. If provided, it will be stored as a single-element array in the [ForeignKeys](#) property.

##### localKey [string](#)

An optional local key used to establish the relationship. If provided, it will be stored as a single-element array in the [LocalKeys](#) property.

##### orderBy [string](#)

An optional order-by clause used to define the sorting of the relationship. If provided, it will be stored as a single-element array in the [OrderBy](#) property.

### RelmForeignKey(string[], string[], string[])

Initializes a new instance of the [RelmForeignKey](#) class, specifying the foreign keys, local keys, and optional ordering criteria.

```
public RelmForeignKey(string[] foreignKeys = null, string[] localKeys = null, string[]
```

```
orderBy = null)
```

## Parameters

### foreignKeys [string\[\]](#)[]

An array of strings representing the foreign key columns in the related table. Can be [null](#) if no foreign keys are specified.

### localKeys [string\[\]](#)[]

An array of strings representing the local key columns in the current table that map to the foreign keys. Can be [null](#) if no local keys are specified.

### orderBy [string\[\]](#)[]

An array of strings specifying the ordering criteria for the related data. Each string represents a column name, optionally followed by a direction (e.g., "ColumnName ASC" or "ColumnName DESC"). Can be [null](#) if no ordering is required.

## Properties

### ForeignKeys

Gets the collection of foreign key names associated with the entity.

```
public string[] ForeignKeys { get; }
```

### Property Value

[string\[\]](#)[]

### LocalKeys

Gets the collection of local keys associated with the entity.

```
public string[] LocalKeys { get; }
```

Property Value

[string](#)[]

## OrderBy

Gets the list of fields used to determine the order of items.

```
public string[] OrderBy { get; }
```

Property Value

[string](#)[]

# Class RelmKey

Namespace: [CoreRelm.Attributes](#)

Assembly: CoreRelm.dll

Specifies that the associated property or struct is a key used for identifying or referencing entities.

```
[AttributeUsage(AttributeTargets.Struct | AttributeTargets.Property)]
public sealed class RelmKey : Attribute
```

## Inheritance

[object](#) ← [Attribute](#) ← RelmKey

## Inherited Members

[Attribute.Equals\(object\)](#) , [Attribute.GetCustomAttribute\(Assembly, Type\)](#) ,  
[Attribute.GetCustomAttribute\(Assembly, Type, bool\)](#) ,  
[Attribute.GetCustomAttribute\(MemberInfo, Type\)](#) ,  
[Attribute.GetCustomAttribute\(MemberInfo, Type, bool\)](#) ,  
[Attribute.GetCustomAttribute\(Module, Type\)](#) , [Attribute.GetCustomAttribute\(Module, Type, bool\)](#) ,  
[Attribute.GetCustomAttribute\(ParameterInfo, Type\)](#) ,  
[Attribute.GetCustomAttribute\(ParameterInfo, Type, bool\)](#) , [Attribute.GetCustomAttributes\(Assembly\)](#) ,  
[Attribute.GetCustomAttributes\(Assembly, bool\)](#) , [Attribute.GetCustomAttributes\(Assembly, Type\)](#) ,  
[Attribute.GetCustomAttributes\(Assembly, Type, bool\)](#) , [Attribute.GetCustomAttributes\(MemberInfo\)](#) ,  
[Attribute.GetCustomAttributes\(MemberInfo, bool\)](#) ,  
[Attribute.GetCustomAttributes\(MemberInfo, Type\)](#) ,  
[Attribute.GetCustomAttributes\(MemberInfo, Type, bool\)](#) , [Attribute.GetCustomAttributes\(Module\)](#) ,  
[Attribute.GetCustomAttributes\(Module, bool\)](#) , [Attribute.GetCustomAttributes\(Module, Type\)](#) ,  
[Attribute.GetCustomAttributes\(Module, Type, bool\)](#) , [Attribute.GetCustomAttributes\(ParameterInfo\)](#) ,  
[Attribute.GetCustomAttributes\(ParameterInfo, bool\)](#) ,  
[Attribute.GetCustomAttributes\(ParameterInfo, Type\)](#) ,  
[Attribute.GetCustomAttributes\(ParameterInfo, Type, bool\)](#) , [Attribute.GetHashCode\(\)](#) ,  
[Attribute.IsDefaultAttribute\(\)](#) , [Attribute.IsDefined\(Assembly, Type\)](#) ,  
[Attribute.IsDefined\(Assembly, Type, bool\)](#) , [Attribute.IsDefined\(MemberInfo, Type\)](#) ,  
[Attribute.IsDefined\(MemberInfo, Type, bool\)](#) , [Attribute.IsDefined\(Module, Type\)](#) ,  
[Attribute.IsDefined\(Module, Type, bool\)](#) , [Attribute.IsDefined\(ParameterInfo, Type\)](#) ,  
[Attribute.IsDefined\(ParameterInfo, Type, bool\)](#) , [Attribute.Match\(object\)](#) , [Attribute.TypeId](#) ,  
[object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.ReferenceEquals\(object, object\)](#) ,  
[object.ToString\(\)](#)

## Remarks

This attribute can be applied to properties or structs to indicate their role as a key in a relational or entity-based context. It is primarily used for metadata purposes and does not enforce any behavior at runtime.

# Class RelmTable

Namespace: [CoreRelm.Attributes](#)

Assembly: CoreRelm.dll

Specifies that a class or struct represents a table in a relational database and provides the name of the table.

```
[AttributeUsage(AttributeTargets.Class | AttributeTargets.Struct)]
public class RelmTable : Attribute
```

## Inheritance

[object](#) ← [Attribute](#) ← RelmTable

## Inherited Members

[Attribute.Equals\(object\)](#) , [Attribute.GetCustomAttribute\(Assembly, Type\)](#) ,  
[Attribute.GetCustomAttribute\(Assembly, Type, bool\)](#) ,  
[Attribute.GetCustomAttribute\(MemberInfo, Type\)](#) ,  
[Attribute.GetCustomAttribute\(MemberInfo, Type, bool\)](#) ,  
[Attribute.GetCustomAttribute\(Module, Type\)](#) , [Attribute.GetCustomAttribute\(Module, Type, bool\)](#) ,  
[Attribute.GetCustomAttribute\(ParameterInfo, Type\)](#) ,  
[Attribute.GetCustomAttribute\(ParameterInfo, Type, bool\)](#) , [Attribute.GetCustomAttributes\(Assembly\)](#) ,  
[Attribute.GetCustomAttributes\(Assembly, bool\)](#) , [Attribute.GetCustomAttributes\(Assembly, Type\)](#) ,  
[Attribute.GetCustomAttributes\(Assembly, Type, bool\)](#) , [Attribute.GetCustomAttributes\(MemberInfo\)](#) ,  
[Attribute.GetCustomAttributes\(MemberInfo, bool\)](#) ,  
[Attribute.GetCustomAttributes\(MemberInfo, Type\)](#) ,  
[Attribute.GetCustomAttributes\(MemberInfo, Type, bool\)](#) , [Attribute.GetCustomAttributes\(Module\)](#) ,  
[Attribute.GetCustomAttributes\(Module, bool\)](#) , [Attribute.GetCustomAttributes\(Module, Type\)](#) ,  
[Attribute.GetCustomAttributes\(Module, Type, bool\)](#) , [Attribute.GetCustomAttributes\(ParameterInfo\)](#) ,  
[Attribute.GetCustomAttributes\(ParameterInfo, bool\)](#) ,  
[Attribute.GetCustomAttributes\(ParameterInfo, Type\)](#) ,  
[Attribute.GetCustomAttributes\(ParameterInfo, Type, bool\)](#) , [Attribute.GetHashCode\(\)](#) ,  
[Attribute.IsDefaultAttribute\(\)](#) , [Attribute.IsDefined\(Assembly, Type\)](#) ,  
[Attribute.IsDefined\(Assembly, Type, bool\)](#) , [Attribute.IsDefined\(MemberInfo, Type\)](#) ,  
[Attribute.IsDefined\(MemberInfo, Type, bool\)](#) , [Attribute.IsDefined\(Module, Type\)](#) ,  
[Attribute.IsDefined\(Module, Type, bool\)](#) , [Attribute.IsDefined\(ParameterInfo, Type\)](#) ,  
[Attribute.IsDefined\(ParameterInfo, Type, bool\)](#) , [Attribute.Match\(object\)](#) , [Attribute.TypeId](#) ,  
[object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) ,  
[object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

## Remarks

This attribute is used to associate a class or struct with a specific table name in a relational database. The table name is specified via the [TableName](#) property.

## Constructors

### RelmTable(string)

Initializes a new instance of the [RelmTable](#) class with the specified table name.

```
public RelmTable(string tableName)
```

#### Parameters

**tableName** [string](#)

The name of the table. This value cannot be null, empty, or consist only of white-space characters.

#### Exceptions

[ArgumentNullException](#)

Thrown if **tableName** is null, empty, or consists only of white-space characters.

## Properties

### TableName

Gets or sets the name of the database table associated with this instance.

```
public string TableName { get; }
```

#### Property Value

[string](#)

# Namespace CoreRelm.Enums

## Classes

### [Commands](#)

Represents a collection of commands that can be used to define and manipulate query operations.

### [Triggers](#)

Represents a collection of database trigger types that define actions to be executed before or after specific data modification operations.

## Enums

### [Commands.Command](#)

Represents the set of commands that can be used to define and manipulate query operations.

### [Triggers.TriggerTypes](#)

Specifies the types of triggers that can be executed in response to database operations.

# Class Commands

Namespace: [CoreRelm.Enums](#)

Assembly: CoreRelm.dll

Represents a collection of commands that can be used to define and manipulate query operations.

```
public class Commands
```

## Inheritance

[object](#) ← Commands

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

## Remarks

The [Commands.Command](#) enumeration provides a set of predefined commands commonly used in query construction, such as filtering, ordering, grouping, and limiting results. These commands can be used to specify the behavior of a query in a structured and consistent manner.

# Enum Commands.Command

Namespace: [CoreRelm.Enums](#)

Assembly: CoreRelm.dll

Represents the set of commands that can be used to define and manipulate query operations.

```
public enum Commands.Command
```

## Fields

**Count** = 10

Gets the number of elements contained in the collection.

**DistinctBy** = 9

Returns a collection of distinct elements from the input sequence based on a specified key selector.

This method uses deferred execution. The distinctness of elements is determined by comparing the keys returned by **keySelector**. The order of the elements in the returned collection is preserved based on their first occurrence in the source sequence.

**GroupBy** = 6

Groups the elements of a sequence according to a specified key selector function.

This method uses deferred execution. The query is not executed until the resulting collection is enumerated.

**Limit** = 7

Gets or sets the maximum allowable limit result for the operation.

**Offset** = 8

Gets or sets the offset value used for positioning or alignment.

**OrderBy** = 2

Sorts the elements of a sequence in ascending order according to a specified key.

This method performs a stable sort, meaning that if two elements have the same key, their original order in the sequence is preserved. To perform a descending sort, use the [OrderByDescending<TSource, TKey>\(IEnumerable<TSource>, Func<TSource, TKey>\)](#) method.

#### OrderByDescending = 3

Sorts the elements of a sequence in descending order according to a specified key.

This method performs a stable sort; that is, if two elements have the same key, their original order is preserved in the returned sequence.

#### Reference = 1

Represents a reference to an object or entity.

This class is typically used to store and manage references to other objects or entities. It may include additional metadata or functionality depending on the specific implementation.

#### Set = 4

Sets the value of the specified property or field.

This method allows dynamic assignment of values to properties or fields by name. Ensure that the property or field exists and is accessible in the current context.

#### SetPostfix = 5

Sets the postfix string that will be appended to the output of the operation.

#### Where = 0

Filters a sequence of values based on a predicate.

This method uses deferred execution. The query represented by this method is not executed until the resulting sequence is enumerated.

## Remarks

This enumeration provides a collection of commands commonly used in query-building scenarios, such as filtering, sorting, grouping, and limiting results. Each command corresponds to a specific operation that can be applied to a query.

# Class Triggers

Namespace: [CoreRelm.Enums](#)

Assembly: CoreRelm.dll

Represents a collection of database trigger types that define actions to be executed before or after specific data modification operations.

```
public class Triggers
```

## Inheritance

[object](#) ← Triggers

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

## Remarks

The [Triggers.TriggerTypes](#) enumeration provides values for common database triggers, including those that occur before or after insert, update, or delete operations. These trigger types can be used to specify the timing and context of database actions.

# Enum Triggers.TriggerTypes

Namespace: [CoreRelm.Enums](#)

Assembly: CoreRelm.dll

Specifies the types of triggers that can be executed in response to database operations.

```
public enum Triggers.TriggerTypes
```

## Fields

**AfterDelete = 5**

Occurs after an item has been deleted.

This event is triggered once the deletion process is completed successfully. Subscribers can use this event to perform any post-deletion operations, such as cleanup or logging.

**AfterInsert = 1**

Occurs after an item has been inserted into the collection or data store.

This event allows subscribers to perform additional actions or processing after an item has been successfully inserted.

**AfterUpdate = 3**

Occurs after an update operation has been completed.

This event is triggered once the update process finishes successfully. Subscribers can use this event to perform any post-update actions, such as refreshing data or logging.

**BeforeDelete = 4**

Occurs before an item is deleted.

This event allows subscribers to perform any necessary actions or validations prior to the deletion of the item. If the operation should be canceled, subscribers can throw an exception or use a specific cancellation mechanism provided by the implementation.

**BeforeInsert = 0**

Occurs before an item is inserted into the collection.

This event allows subscribers to perform custom logic or validation before an item is added to the collection. If the operation should be canceled, the event handler can throw an exception or modify the state as needed.

**BeforeUpdate = 2**

Occurs before an update operation is performed.

This event allows subscribers to execute custom logic or validate data prior to the update operation. If the operation should be canceled, subscribers can throw an exception or modify the state as needed.

## Remarks

This enumeration defines trigger types that correspond to specific stages of database operations, such as inserts, updates, and deletes. Triggers can be executed either before or after the operation.

# Namespace CoreRelm.Exceptions

## Classes

[MissingRelmDatabaseAttributeException](#)

[MissingRelmTableAttributeException](#)

# Class MissingRelmDatabaseAttributeException

Namespace: [CoreRelm.Exceptions](#)

Assembly: CoreRelm.dll

```
public sealed class MissingRelmDatabaseAttributeException : Exception, ISerializable
```

## Inheritance

[object](#) ← [Exception](#) ← MissingRelmDatabaseAttributeException

## Implements

[ISerializable](#)

## Inherited Members

[Exception.GetBaseException\(\)](#) , [Exception.GetType\(\)](#) , [Exception.ToString\(\)](#) , [Exception.Data](#) ,  
[Exception.HelpLink](#) , [Exception.HResult](#) , [Exception.InnerException](#) , [Exception.Message](#) ,  
[Exception.Source](#) , [Exception.StackTrace](#) , [Exception.TargetSite](#) , [object.Equals\(object\)](#) ,  
[object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.ReferenceEquals\(object, object\)](#)

# Constructors

## MissingRelmDatabaseAttributeException(Type)

```
public MissingRelmDatabaseAttributeException(Type t)
```

## Parameters

t [Type](#)

# Class MissingRelmTableAttributeException

Namespace: [CoreRelm.Exceptions](#)

Assembly: CoreRelm.dll

```
public sealed class MissingRelmTableAttributeException : Exception, ISerializable
```

## Inheritance

[object](#) ← [Exception](#) ← MissingRelmTableAttributeException

## Implements

[ISerializable](#)

## Inherited Members

[Exception.GetBaseException\(\)](#) , [Exception.GetType\(\)](#) , [Exception.ToString\(\)](#) , [Exception.Data](#) ,  
[Exception.HelpLink](#) , [Exception.HResult](#) , [Exception.InnerException](#) , [Exception.Message](#) ,  
[Exception.Source](#) , [Exception.StackTrace](#) , [Exception.TargetSite](#) , [object.Equals\(object\)](#) ,  
[object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.ReferenceEquals\(object, object\)](#)

# Constructors

## MissingRelmTableAttributeException(Type)

```
public MissingRelmTableAttributeException(Type t)
```

## Parameters

t [Type](#)

# Namespace CoreRelm.Extensions

## Classes

[ChecksumExtensions](#)

[CoreRelmServiceCollectionExtensions](#)

[DbContextExtensions](#)

Provides extension methods for configuring and interacting with database contexts.

[ListExtensions](#)

Provides a set of extension methods for performing database operations, loading related data, and manipulating collections in the context of relational models.

[ModelExtensions](#)

# Class ChecksumExtensions

Namespace: [CoreRelm.Extensions](#)

Assembly: CoreRelm.dll

```
public static class ChecksumExtensions
```

## Inheritance

[object](#) ← ChecksumExtensions

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

# Methods

## Sha256Hex(string)

```
public static string Sha256Hex(this string text)
```

### Parameters

**text** [string](#)

### Returns

[string](#)

# Class CoreRelmServiceCollectionExtensions

Namespace: [CoreRelm.Extensions](#)

Assembly: CoreRelm.dll

```
public static class CoreRelmServiceCollectionExtensions
```

## Inheritance

[object](#) ← CoreRelmServiceCollectionExtensions

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

## Methods

### AddCoreRelm(IServiceCollection, IConfiguration)

```
public static IServiceCollection AddCoreRelm(this IServiceCollection services,  
IConfiguration configuration)
```

#### Parameters

services [IServiceCollection](#)

configuration [IConfiguration](#)

#### Returns

[IServiceCollection](#)

# Class DbContextExtensions

Namespace: [CoreRelm.Extensions](#)

Assembly: CoreRelm.dll

Provides extension methods for configuring and interacting with database contexts.

```
public static class DbContextExtensions
```

## Inheritance

[object](#) ← DbContextExtensions

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

## Remarks

This class contains utility methods that extend the functionality of database contexts, such as setting session-specific database parameters. These methods are designed to simplify common database operations and enhance developer productivity when working with database contexts.

## Methods

### SetLockWaitTimeout(IRelmContext, int)

Sets the InnoDB lock wait timeout for the current database session.

```
public static void SetLockWaitTimeout(this IRelmContext context, int seconds = 300)
```

#### Parameters

**context** [IRelmContext](#)

The database context. This parameter cannot be [null](#).

**seconds** [int](#)

The lock wait timeout, in seconds. The default value is 300 seconds (5 minutes).

## Remarks

This method executes a SQL command to set the InnoDB lock wait timeout for the current session. The timeout determines how long a transaction will wait for a lock before timing out.

## Exceptions

### [ArgumentNullException](#)

Thrown if `context` is [null](#).

# Class ListExtensions

Namespace: [CoreRelm.Extensions](#)

Assembly: CoreRelm.dll

Provides a set of extension methods for performing database operations, loading related data, and manipulating collections in the context of relational models.

```
public static class ListExtensions
```

## Inheritance

[object](#) ← ListExtensions

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

## Remarks

The [ListExtensions](#) class includes methods for bulk writing data to a database, loading foreign key fields, flattening hierarchical data structures, generating DTOs, and retrieving dictionary entries. These methods are designed to work with relational models that implement the [IRelmModel](#) interface and are optimized for scenarios involving database interactions and data transformation. Many methods in this class support advanced customization through optional parameters, such as specifying table names, batch sizes, and constraints for foreign key loading. Additionally, the methods are designed to be used with dependency injection and context-based database operations.

## Methods

### FlattenTreeObject<T>(IEnumerable<T>, Func<T, ICollection<T>>)

Flattens a hierarchical structure of objects into a single collection.

```
public static ICollection<T> FlattenTreeObject<T>(this IEnumerable<T> enumerableList,  
Func<T, ICollection<T>> getChildrenFunction)
```

## Parameters

## `enumerableList` [IEnumerable<T>](#)

The top-level collection of objects to flatten.

## `getChildrenFunction` [Func<T, ICollection<T>>](#)

A function that retrieves the child objects of a given object. The function should return an [ICollection<T>](#) of child objects, or [null](#) if there are no children.

Returns

## [ICollection<T>](#)

A flattened [ICollection<T>](#) containing all objects in the hierarchy, including the top-level objects and their descendants.

Type Parameters

### T

The type of the objects in the hierarchy.

Remarks

This method recursively traverses the hierarchy defined by the `getChildrenFunction` and combines all objects into a single collection. If an object has no children, it is included as-is.

## `GenerateDTO<T>(IEnumerable<T>, ICollection<string>, ICollection<string>, string, Func<IRelmModel, Dictionary<string, object>>)`

Generates a collection of dynamic objects (DTOs) from the specified collection of base objects, including or excluding specific properties as needed.

```
public static ICollection<dynamic> GenerateDTO<T>(this IEnumerable<T> baseObjects,  
    ICollection<string> includeProperties = null, ICollection<string> excludeProperties = null,  
    string sourceObjectName = null, Func<IRelmModel, Dictionary<string, object>>  
    getAdditionalObjectProperties = null) where T : IRelmModel
```

Parameters

## baseObjects [IEnumerable](#)<T>

The collection of base objects to generate DTOs from. This parameter cannot be [null](#).

## includeProperties [ICollection](#)<string>

An optional collection of property names to include in the generated DTOs. If [null](#), all properties are included by default unless explicitly excluded.

## excludeProperties [ICollection](#)<string>

An optional collection of property names to exclude from the generated DTOs. If [null](#), no properties are excluded unless explicitly specified in `includeProperties`.

## sourceObjectName [string](#)

An optional name of the source object to include in the DTOs for identification purposes. If [null](#), the source object name is not included.

## getAdditionalObjectProperties [Func](#)<[IRelmModel](#), [Dictionary](#)<string, object>>

An optional function that takes an [IRelmModel](#) object and returns a dictionary of additional properties to include in the generated DTOs. If [null](#), no additional properties are added.

## Returns

## [ICollection](#)<dynamic>

A collection of dynamic objects (DTOs) generated from the specified base objects. Each DTO includes the properties specified in `includeProperties` and excludes those in `excludeProperties`, along with any additional properties provided by `getAdditionalObjectProperties`.

## Type Parameters

### T

The type of the base objects, which must implement the [IRelmModel](#) interface.

## Remarks

This method is an extension method for collections of objects implementing the [IRelmModel](#) interface. It allows for flexible generation of DTOs by specifying properties to include or exclude, and by adding custom properties through the `getAdditionalObjectProperties` function.

## GetEntry<TKey, TValue>(IDictionary<TKey, TValue>, TKey)

Retrieves the key-value pair associated with the specified key in the dictionary.

```
public static KeyValuePair<TKey, TValue> GetEntry<TKey, TValue>(this IDictionary<TKey, TValue> dictionary, TKey key)
```

### Parameters

**dictionary** [IDictionary](#)<TKey, TValue>

The dictionary from which to retrieve the key-value pair. Cannot be [null](#).

**key** TKey

The key whose associated key-value pair is to be retrieved. Must exist in the dictionary.

### Returns

[KeyValuePair](#)<TKey, TValue>

A [KeyValuePair](#)<TKey, TValue> containing the specified key and its associated value.

### Type Parameters

**TKey**

The type of the keys in the dictionary.

**TValue**

The type of the values in the dictionary.

## LoadDataLoaderField<T, R>(ICollection<T>, IRelmContext, Expression<Func<T, R>>)

Loads and initializes a specific field for a collection of models using a data loader.

```
public static ICollection<T> LoadDataLoaderField<T, R>(this ICollection<T> modelData, IRelmContext relmContext, Expression<Func<T, R>> predicate) where T : IRelmModel, new()
```

## Parameters

**modelData** [ICollection<T>](#)

The collection of models for which the field will be loaded. Cannot be [null](#).

**relmContext** [IRelmContext](#)

The context used to access the data loader. Cannot be [null](#).

**predicate** [Expression<Func<T, R>>](#)

An expression specifying the field to be loaded. Cannot be [null](#).

## Returns

[ICollection<T>](#)

The original collection of models with the specified field loaded.

## Type Parameters

**T**

The type of the model in the collection. Must implement [IRelmModel](#).

**R**

The type of the field to be loaded.

## Remarks

This method uses a data loader to populate the specified field for each model in the collection. The field to be loaded is determined by the provided **predicate** expression.

**LoadDataLoaderField<T, R>(ICollection<T>, IRelmQuickContext, Expression<Func<T, R>>)**

Loads and populates a specified field for a collection of models using a data loader.

```
public static ICollection<T> LoadDataLoaderField<T, R>(this ICollection<T> modelData, IRelmQuickContext relmQuickContext, Expression<Func<T, R>> predicate) where T : IRelmModel, new()
```

## Parameters

`modelData` [ICollection<T>](#)

The collection of models for which the field will be loaded.

`relmQuickContext` [IRelmQuickContext](#)

The context used to facilitate data loading operations.

`predicate` [Expression<Func<T, R>>](#)

An expression specifying the field to be loaded for each model in the collection.

## Returns

[ICollection<T>](#)

The original collection of models with the specified field loaded.

## Type Parameters

`T`

The type of the model in the collection. Must implement [IRelmModel](#).

`R`

The type of the field to be loaded.

## Remarks

This method uses a data loader to populate the specified field for each model in the collection. The `predicate` parameter determines which field is loaded.

**LoadDataLoaderField<T, R>(ICollection<T>, RelmContextOptionsBuilder, Expression<Func<T, R>>)**

Loads and populates a specified field for a collection of models using a data loader.

```
public static ICollection<T> LoadDataLoaderField<T, R>(this ICollection<T> modelData, RelmContextOptionsBuilder relmContextOptionsBuilder, Expression<Func<T, R>> predicate) where T : IRelmModel, new()
```

## Parameters

`modelData` [ICollection<T>](#)

The collection of models to be updated with the loaded field data.

`realmContextOptionsBuilder` [RealmContextOptionsBuilder](#)

The options builder used to configure the database context for loading the field.

`predicate` [Expression<Func<T, R>>](#)

An expression specifying the field to be loaded for each model in the collection.

## Returns

[ICollection<T>](#)

The updated collection of models with the specified field populated.

## Type Parameters

`T`

The type of the model in the collection. Must implement [IRelmModel](#) and have a parameterless constructor.

`R`

The type of the field to be loaded, as specified by the predicate.

## Remarks

This method uses a data loader to populate the specified field in the provided collection of models. Ensure that the `realmContextOptionsBuilder` is properly configured to access the required data source.

**LoadForeignKeyField<T, R>(ICollection<T>, IRealmContext, Expression<Func<T, ICollection<R>>>)**

Loads and populates a foreign key collection field for a collection of models.

```
public static ICollection<T> LoadForeignKeyField<T, R>(this ICollection<T> modelData,  
IRelmContext realmContext, Expression<Func<T, ICollection<R>>> predicate) where T :
```

`IReImModel, new() where R : IReImModel, new()`

## Parameters

`modelData ICollection<T>`

The collection of primary models for which the foreign key field will be loaded.

`reImContext IReImContext`

The context used to access the data source and resolve the foreign key relationships.

`predicate Expression<Func<T, ICollection<R>>>`

An expression specifying the foreign key collection property to load for each model in `modelData`.

## Returns

`ICollection<T>`

The original collection of primary models with the specified foreign key field populated for each model.

## Type Parameters

`T`

The type of the primary model in the collection. Must implement [IReImModel](#).

`R`

The type of the related model in the foreign key collection. Must implement [IReImModel](#).

## Remarks

This method uses the provided `reImContext` to resolve and populate the foreign key collection specified by the `predicate` for each model in the `modelData` collection.

`LoadForeignKeyField<T, R>(ICollection<T>, IReImContext, Expression<Func<T, ICollection<R>>>, Expression<Func<R, object>>)`

Loads and populates a foreign key collection field for a collection of models.

```
public static ICollection<T> LoadForeignKeyField<T, R>(this ICollection<T> modelData,  
IRelmContext relmContext, Expression<Func<T, ICollection<R>>> predicate, Expression<Func<R,  
object>> additionalConstraints) where T : IRelmModel, new() where R : IRelmModel, new()
```

## Parameters

**modelData** [ICollection](#)<T>

The collection of primary models for which the foreign key field will be loaded.

**relmContext** [IRelmContext](#)

The context used to access the data source and resolve the foreign key relationships.

**predicate** [Expression](#)<[Func](#)<T, ICollection><R>>>

An expression specifying the foreign key field to populate in the primary model.

**additionalConstraints** [Expression](#)<[Func](#)<R, object>>>

An optional expression specifying additional constraints to apply when loading the related models.

## Returns

[ICollection](#)<T>

A collection of the primary models with the specified foreign key field populated.

## Type Parameters

**T**

The type of the primary model in the collection. Must implement [IRelmModel](#).

**R**

The type of the related model representing the foreign key. Must implement [IRelmModel](#).

## Remarks

This method is an extension method that simplifies the process of loading related data for a collection of models. It uses the provided **relmContext** to resolve the foreign key relationships and applies any

additional constraints specified by `additionalConstraints`.

## LoadForeignKeyField<T, R>(ICollection<T>, IRelmContext, Expression<Func<T, R>>)

Loads and resolves a foreign key field for a collection of models.

```
public static ICollection<T> LoadForeignKeyField<T, R>(this ICollection<T> modelData,  
IRelmContext relmContext, Expression<Func<T, R>> predicate) where T : IRelmModel, new()  
where R : IRelmModel, new()
```

### Parameters

`modelData` [ICollection](#)<T>

The collection of models for which the foreign key field will be loaded.

`relmContext` [IRelmContext](#)

The context used to resolve the foreign key relationships.

`predicate` [Expression](#)<[Func](#)<T, R>>

An expression specifying the foreign key field to load.

### Returns

[ICollection](#)<T>

A collection of models with the specified foreign key field resolved.

### Type Parameters

T

The type of the model in the collection. Must implement [IRelmModel](#).

R

The type of the related model to be loaded. Must implement [IRelmModel](#).

### Remarks

This method uses the provided `relmContext` to resolve the foreign key relationships for the specified field in the collection of models. The `predicate` determines which foreign key field to load.

## LoadForeignKeyField<T, R>(ICollection<T>, IRelmContext, Expression<Func<T, R>>, Expression<Func<R, object>>)

Loads and resolves a foreign key field for a collection of models, applying the specified predicate and additional constraints.

```
public static ICollection<T> LoadForeignKeyField<T, R>(this ICollection<T> modelData,  
IRelmContext relmContext, Expression<Func<T, R>> predicate, Expression<Func<R, object>>  
additionalConstraints) where T : IRelmModel, new() where R : IRelmModel, new()
```

### Parameters

`modelData` [ICollection](#)<T>

The collection of primary models for which the foreign key field will be loaded.

`relmContext` [IRelmContext](#)

The context used to resolve the foreign key relationships.

`predicate` [Expression](#)<Func<T, R>>

An expression specifying the foreign key relationship between the primary and related models.

`additionalConstraints` [Expression](#)<Func<R, object>>

An optional expression specifying additional constraints to apply when resolving the foreign key field.

### Returns

[ICollection](#)<T>

A collection of the primary models with the foreign key field resolved based on the specified predicate and constraints.

### Type Parameters

T

The type of the primary model in the collection. Must implement [IRelmModel](#).

## R

The type of the related model to be loaded. Must implement [IRelmModel](#).

## Remarks

This method is an extension method that simplifies the process of resolving foreign key relationships for a collection of models. It uses the provided `relmContext` to perform the resolution and applies the specified `predicate` to define the relationship. Additional constraints can be applied using the `additionalConstraints` parameter.

## LoadForeignKeyField<T, R>(ICollection<T>, IRelmQuickContext, Expression<Func<T, ICollection<R>>>)

Loads and populates a foreign key collection field for each entity in the specified model data collection.

```
public static ICollection<T> LoadForeignKeyField<T, R>(this ICollection<T> modelData,  
IRelmQuickContext relmQuickContext, Expression<Func<T, ICollection<R>>> predicate) where T :  
IRelmModel, new() where R : IRelmModel, new()
```

## Parameters

`modelData` [ICollection](#)<T>

The collection of primary model entities for which the foreign key field will be loaded.

`relmQuickContext` [IRelmQuickContext](#)

The database context used to query and load the related entities.

`predicate` [Expression](#)<[Func](#)<T, ICollection<R>>>

An expression specifying the foreign key collection property to populate for each entity in `modelData`.

## Returns

[ICollection](#)<T>

The original `modelData` collection with the specified foreign key field populated for each entity.

## Type Parameters

T

The type of the primary model entities in the collection. Must implement [IRelmModel](#).

R

The type of the related entities in the foreign key collection. Must implement [IRelmModel](#).

## Remarks

This method uses the provided `relmQuickContext` to query and load the related entities for the specified foreign key collection property. The foreign key field is populated for each entity in the `modelData` collection based on the provided `predicate`.

**LoadForeignKeyField<T, R>(ICollection<T>, IRelmQuickContext, Expression<Func<T, ICollection<R>>>, Expression<Func<R, object>>)**

Loads and populates a foreign key collection field for a collection of model entities.

```
public static ICollection<T> LoadForeignKeyField<T, R>(this ICollection<T> modelData,  
IRelmQuickContext relmQuickContext, Expression<Func<T, ICollection<R>>> predicate,  
Expression<Func<R, object>> additionalConstraints) where T : IRelmModel, new() where R :  
IRelmModel, new()
```

## Parameters

`modelData` [ICollection](#)<T>

The collection of primary model entities for which the foreign key field will be loaded.

`relmQuickContext` [IRelmQuickContext](#)

The database context used to query and load the related entities.

`predicate` [Expression](#)<[Func](#)<T, ICollection<R>>>

An expression specifying the foreign key field to populate in the primary model entities.

`additionalConstraints` [Expression](#)<[Func](#)<R, object>>>

An expression specifying additional constraints to apply when querying the related entities.

Returns

### [ICollection](#)<T>

A collection of the primary model entities with the specified foreign key field populated.

Type Parameters

T

The type of the primary model entities in the collection. Must implement [IRelmModel](#).

R

The type of the related entities referenced by the foreign key. Must implement [IRelmModel](#).

Remarks

This method is an extension method that facilitates the loading of related entities for a collection of primary model entities. It uses the specified database context and constraints to query and populate the foreign key field.

## LoadForeignKeyField<T, R>(ICollection<T>, IRelmQuickContext, Expression<Func<T, R>>)

Loads and resolves a foreign key field for the specified collection of models.

```
public static ICollection<T> LoadForeignKeyField<T, R>(this ICollection<T> modelData,  
IRelmQuickContext relmQuickContext, Expression<Func<T, R>> predicate) where T : IRelmModel,  
new() where R : IRelmModel, new()
```

Parameters

### modelData [ICollection](#)<T>

The collection of models for which the foreign key field will be loaded.

### relmQuickContext [IRelmQuickContext](#)

The database context used to resolve the foreign key relationships.

**predicate** [Expression](#)<Func<T, R>>

An expression specifying the foreign key field to load.

Returns

[ICollection](#)<T>

The original collection of models with the specified foreign key field resolved and loaded.

Type Parameters

T

The type of the model in the collection. Must implement [IRelmModel](#).

R

The type of the related model to be loaded. Must implement [IRelmModel](#).

Remarks

This method uses the provided `relmQuickContext` to resolve the foreign key relationships for the specified field in the collection of models. The foreign key field is loaded based on the `predicate` expression.

**LoadForeignKeyField<T, R>(ICollection<T>, IRelmQuickContext, Expression<Func<T, R>>, Expression<Func<R, object>>)**

Loads and resolves a foreign key field for a collection of models, applying the specified constraints.

```
public static ICollection<T> LoadForeignKeyField<T, R>(this ICollection<T> modelData,  
IRelmQuickContext relmQuickContext, Expression<Func<T, R>> predicate, Expression<Func<R,  
object>> additionalConstraints) where T : IRelmModel, new() where R : IRelmModel, new()
```

Parameters

`modelData` [ICollection](#)<T>

The collection of primary models for which the foreign key field will be loaded.

`relmQuickContext` [IRelmQuickContext](#)

The database context used to resolve the foreign key relationships.

**predicate** [Expression](#) <Func<T, R>>

An expression specifying the foreign key relationship between the primary model and the related model.

**additionalConstraints** [Expression](#) <Func<R, object>>

An optional expression specifying additional constraints to apply when resolving the foreign key field.

Returns

[ICollection](#) <T>

A collection of the primary models with the foreign key field resolved and populated.

Type Parameters

T

The type of the primary model in the collection. Must implement [IRelmModel](#).

R

The type of the related model representing the foreign key. Must implement [IRelmModel](#).

Remarks

This method is an extension method designed to simplify the process of resolving foreign key relationships for a collection of models. It uses the provided database context and expressions to load the related data efficiently. The additional constraints can be used to filter or customize the resolution process.

**LoadForeignKeyField<T, R>(ICollection<T>, RelmContextOptionsBuilder, Expression<Func<T, ICollection<R>>>)**

Loads and populates a foreign key collection field for the specified collection of model data.

```
public static ICollection<T> LoadForeignKeyField<T, R>(this ICollection<T> modelData, RelmContextOptionsBuilder relmContextOptionsBuilder, Expression<Func<T, ICollection<R>>>)
```

```
predicate) where T : IRelmModel, new() where R : IRelmModel, new()
```

## Parameters

**modelData** [ICollection<T>](#)

The collection of primary model data for which the foreign key field will be loaded.

**relmContextOptionsBuilder** [RelmContextOptionsBuilder](#)

The context options builder used to configure the data loading process.

**predicate** [Expression<Func<T, ICollection<R>>>](#)

An expression specifying the foreign key field to load, represented as a navigation property.

## Returns

[ICollection<T>](#)

A collection of the primary model data with the specified foreign key field populated.

## Type Parameters

**T**

The type of the primary model in the collection. Must implement [IRelmModel](#).

**R**

The type of the related model representing the foreign key. Must implement [IRelmModel](#).

## Remarks

This method uses the provided **relmContextOptionsBuilder** to configure the loading process and populates the foreign key field specified by the **predicate** for each item in the **modelData** collection.

**LoadForeignKeyField<T, R>(ICollection<T>,  
RelmContextOptionsBuilder, Expression<Func<T,  
ICollection<R>>>, Expression<Func<R, object>>)**

Loads and populates a foreign key collection field for the specified model data.

```
public static ICollection<T> LoadForeignKeyField<T, R>(this ICollection<T> modelData,  
RelmContextOptionsBuilder relmContextOptionsBuilder, Expression<Func<T, ICollection<R>>>  
predicate, Expression<Func<R, object>> additionalConstraints) where T : IRelmModel, new()  
where R : IRelmModel, new()
```

## Parameters

**modelData** [ICollection](#)<T>

The collection of primary model instances for which the foreign key field will be loaded.

**relmContextOptionsBuilder** [RelmContextOptionsBuilder](#)

The options builder used to configure the database context for the operation.

**predicate** [Expression](#)<Func<T, ICollection><R>>>

An expression specifying the foreign key collection property to populate.

**additionalConstraints** [Expression](#)<Func<R, object>>>

An optional expression defining additional constraints to apply when loading the related data.

## Returns

[ICollection](#)<T>

A collection of the primary model instances with the specified foreign key field populated.

## Type Parameters

T

The type of the primary model, which must implement [IRelmModel](#).

R

The type of the related model, which must implement [IRelmModel](#).

## Remarks

This method uses the provided **relmContextOptionsBuilder** to configure the database context and loads the related data for the specified foreign key field. The **additionalConstraints** parameter can be used to filter the related data further, if needed.

# LoadForeignKeyField<T, R>(ICollection<T>, RelmContextOptionsBuilder, Expression<Func<T, R>>)

Loads and resolves a foreign key field for the specified model data collection.

```
public static ICollection<T> LoadForeignKeyField<T, R>(this ICollection<T> modelData,  
RelmContextOptionsBuilder relmContextOptionsBuilder, Expression<Func<T, R>> predicate) where  
T : IRelmModel, new() where R : IRelmModel, new()
```

## Parameters

**modelData** [ICollection](#)<T>

The collection of models for which the foreign key field will be loaded.

**relmContextOptionsBuilder** [RelmContextOptionsBuilder](#)

The context options builder used to configure the loading process.

**predicate** [Expression](#)<[Func](#)<T, R>>

An expression specifying the foreign key field to load.

## Returns

[ICollection](#)<T>

A collection of models with the specified foreign key field resolved.

## Type Parameters

**T**

The type of the model in the collection. Must implement [IRelmModel](#).

**R**

The type of the related model to be loaded. Must implement [IRelmModel](#).

## Remarks

This method uses a CoreRelm.RelmInternal.Helpers.Utilities.ForeignKeyLoader<T> to load the foreign key field specified by the **predicate**. The foreign key field is resolved based on the provided context options

and the relationship defined in the model.

## LoadForeignKeyField<T, R>(ICollection<T>, RelmContextOptionsBuilder, Expression<Func<T, R>>, Expression<Func<R, object>>)

Loads and populates a foreign key field for a collection of models based on the specified predicate and additional constraints.

```
public static ICollection<T> LoadForeignKeyField<T, R>(this ICollection<T> modelData, RelmContextOptionsBuilder relmContextOptionsBuilder, Expression<Func<T, R>> predicate, Expression<Func<R, object>> additionalConstraints) where T : IRelmModel, new() where R : IRelmModel, new()
```

### Parameters

**modelData** [ICollection](#)<T>

The collection of primary models for which the foreign key field will be loaded.

**relmContextOptionsBuilder** [RelmContextOptionsBuilder](#)

The options builder used to configure the database context for loading the foreign key data.

**predicate** [Expression](#)<Func<T, R>>

An expression specifying the foreign key relationship between the primary model and the related model.

**additionalConstraints** [Expression](#)<Func<R, object>>

An optional expression specifying additional constraints to apply when loading the foreign key data.

### Returns

[ICollection](#)<T>

A collection of primary models with the specified foreign key field populated.

### Type Parameters

T

The type of the primary model in the collection. Must implement [IRelmModel](#).

R

The type of the related model representing the foreign key. Must implement [IRelmModel](#).

## Remarks

This method uses the provided `relmContextOptionsBuilder` to configure the database context and load the related data. The `predicate` defines the relationship between the primary and related models, while the `additionalConstraints` can be used to further filter the related data being loaded.

## LoadForeignKeyField<T, R, S>(ICollection<T>, IRelmContext, Expression<Func<T, ICollection<R>>>, IRelmDataLoader<S>)

Loads and populates a foreign key collection field for a collection of models using the specified predicate and data loader.

```
public static ICollection<T> LoadForeignKeyField<T, R, S>(this ICollection<T> modelData,
IRelmContext relmContext, Expression<Func<T, ICollection<R>>> predicate, IRelmDataLoader<S>
customDataLoader) where T : IRelmModel, new() where R : IRelmModel, new() where S :
IRelmModel, new()
```

## Parameters

`modelData` [ICollection](#)<T>

The collection of primary models for which the foreign key field will be loaded.

`relmContext` [IRelmContext](#)

The context used to interact with the data source.

`predicate` [Expression](#)<[Func](#)<T, [ICollection](#)<R>>>

An expression specifying the foreign key field to be loaded.

`customDataLoader` [IRelmDataLoader](#)<S>

A custom data loader used to retrieve the related data.

## Returns

## [ICollection](#)<T>

A collection of primary models with the specified foreign key field populated.

## Type Parameters

### T

The type of the primary model in the collection.

### R

The type of the related model representing the foreign key field.

### S

The type of the model used by the custom data loader.

## Remarks

This method uses the provided `customDataLoader` to load the related data for the foreign key field specified by `predicate`. The method is an extension method and operates on the collection of primary models (`modelData`).

## LoadForeignKeyField<T, R, S>(ICollection<T>, IRelmContext, Expression<Func<T, ICollection<R>>>, IRelmDataLoader<S>, Expression<Func<R, object>>)

Loads and populates a foreign key collection field for a collection of models using the specified data loader and constraints.

```
public static ICollection<T> LoadForeignKeyField<T, R, S>(this ICollection<T> modelData, IRelmContext relmContext, Expression<Func<T, ICollection<R>>> predicate, IRelmDataLoader<S> customDataLoader, Expression<Func<R, object>> additionalConstraints) where T : IRelmModel, new() where R : IRelmModel, new() where S : IRelmModel, new()
```

## Parameters

### modelData [ICollection](#)<T>

The collection of primary models for which the foreign key field will be loaded.

`relmContext` [IRelmContext](#)

The context used to manage the data loading operation.

`predicate` [Expression](#)<[Func](#)<T, [ICollection](#)<R>>>

An expression specifying the foreign key field to populate in the primary model.

`customDataLoader` [IRelmDataLoader](#)<S>

A custom data loader used to retrieve the related data.

`additionalConstraints` [Expression](#)<[Func](#)<R, [object](#)>>>

An expression specifying additional constraints to apply when loading the related data.

Returns

[ICollection](#)<T>

A collection of the primary models with the specified foreign key field populated.

Type Parameters

T

The type of the primary model in the collection.

R

The type of the related model representing the foreign key field.

S

The type of the model used by the custom data loader.

Remarks

This method uses a custom data loader and optional constraints to populate a foreign key field in the primary models. It is designed to work with models that implement the [IRelmModel](#) interface.

**LoadForeignKeyField<T, R, S>(ICollection<T>, IRelmContext, Expression<Func<T, R>>, IRelmDataLoader<S>)**

Loads and resolves a foreign key field for a collection of models, using the specified predicate and custom data loader.

```
public static ICollection<T> LoadForeignKeyField<T, R, S>(this ICollection<T> modelData,  
IRelmContext relmContext, Expression<Func<T, R>> predicate, IRelmDataLoader<S>  
customDataLoader) where T : IRelmModel, new() where R : IRelmModel, new() where S :  
IRelmModel, new()
```

## Parameters

**modelData** [ICollection](#)<T>

The collection of models for which the foreign key field will be loaded.

**relmContext** [IRelmContext](#)

The context used to interact with the data source.

**predicate** [Expression](#)<[Func](#)<T, R>>

An expression specifying the foreign key field to load.

**customDataLoader** [IRelmDataLoader](#)<S>

A custom data loader used to retrieve the related data.

## Returns

[ICollection](#)<T>

The original collection of models with the specified foreign key field resolved.

## Type Parameters

**T**

The type of the model in the collection. Must implement [IRelmModel](#).

**R**

The type of the related model being loaded. Must implement [IRelmModel](#).

**S**

The type of the model used by the custom data loader. Must implement [IRelmModel](#).

## Remarks

This method is an extension method that facilitates the resolution of foreign key relationships for a collection of models. The `customDataLoader` allows for custom logic to be applied when loading the related data.

## LoadForeignKeyField<T, R, S>(ICollection<T>, IRelmContext, Expression<Func<T, R>>, IRelmDataLoader<S>, Expression<Func<R, object>>)

Loads and resolves a foreign key field for a collection of models, applying optional constraints and using a custom data loader.

```
public static ICollection<T> LoadForeignKeyField<T, R, S>(this ICollection<T> modelData, IRelmContext relmContext, Expression<Func<T, R>> predicate, IRelmDataLoader<S> customDataLoader, Expression<Func<R, object>> additionalConstraints) where T : IRelmModel, new() where R : IRelmModel, new() where S : IRelmModel, new()
```

## Parameters

`modelData` [ICollection](#)<T>

The collection of primary models for which the foreign key field will be loaded.

`relmContext` [IRelmContext](#)

The context used to interact with the data source.

`predicate` [Expression](#)<[Func](#)<T, R>>

An expression specifying the foreign key field to load in the primary model.

`customDataLoader` [IRelmDataLoader](#)<S>

A custom data loader used to retrieve the related data for the foreign key field.

`additionalConstraints` [Expression](#)<[Func](#)<R, object>>

An optional expression specifying additional constraints to apply when resolving the foreign key field.

## Returns

[ICollection](#)<T>

A collection of the primary models with the specified foreign key field resolved and populated.

## Type Parameters

T

The type of the primary model in the collection. Must implement [IRelmModel](#).

R

The type of the related model representing the foreign key. Must implement [IRelmModel](#).

S

The type of the model used by the custom data loader. Must implement [IRelmModel](#).

## Remarks

This method is an extension method designed to simplify the process of resolving foreign key relationships in a collection of models. It allows for the use of a custom data loader and additional constraints to tailor the loading process to specific requirements.

## LoadForeignKeyField<T, R, S>(ICollection<T>, IRelmQuickContext, Expression<Func<T, ICollection<R>>>, IRelmDataLoader<S>)

Loads and populates a foreign key collection field for a collection of models.

```
public static ICollection<T> LoadForeignKeyField<T, R, S>(this ICollection<T> modelData,  
IRelmQuickContext relmQuickContext, Expression<Func<T, ICollection<R>>> predicate,  
IRelmDataLoader<S> customDataLoader) where T : IRelmModel, new() where R : IRelmModel, new()  
where S : IRelmModel, new()
```

## Parameters

modelData [ICollection](#)<T>

The collection of primary models for which the foreign key field will be loaded.

## `relmQuickContext` [IRelmQuickContext](#)

The context used to manage data loading operations.

## `predicate` [Expression](#)<[Func](#)<T, [ICollection](#)<R>>>

An expression specifying the foreign key collection property to populate.

## `customDataLoader` [IRelmDataLoader](#)<S>

A custom data loader used to retrieve the related data.

Returns

## [ICollection](#)<T>

The original collection of primary models with the specified foreign key field populated.

Type Parameters

### T

The type of the primary model in the collection.

### R

The type of the related model in the foreign key collection.

### S

The type of the model used by the custom data loader.

Remarks

This method uses a CoreRelm.RelmInternal.Helpers.Utilities.ForeignKeyLoader<T> to load the foreign key collection field for each model in the provided collection. The `customDataLoader` allows for custom logic to retrieve the related data, which is then used to populate the specified foreign key field.

`LoadForeignKeyField<T, R, S>(ICollection<T>,  
IRelmQuickContext, Expression<Func<T, ICollection<R>>>,  
IRelmDataLoader<S>, Expression<Func<R, object>>)`

Loads and populates a foreign key collection field for a collection of models using the specified data loader and constraints.

```
public static ICollection<T> LoadForeignKeyField<T, R, S>(this ICollection<T> modelData,  
IRelmQuickContext relmQuickContext, Expression<Func<T, ICollection<R>>> predicate,  
IRelmDataLoader<S> customDataLoader, Expression<Func<R, object>>> additionalConstraints)  
where T : IRelmModel, new() where R : IRelmModel, new() where S : IRelmModel, new()
```

## Parameters

**modelData** [ICollection](#)<T>

The collection of primary models for which the foreign key field will be loaded.

**relmQuickContext** [IRelmQuickContext](#)

The context used to manage data loading operations.

**predicate** [Expression](#)<[Func](#)<T, ICollection><R>>>

An expression specifying the foreign key field to be loaded for each primary model.

**customDataLoader** [IRelmDataLoader](#)<S>

A custom data loader used to retrieve the related data.

**additionalConstraints** [Expression](#)<[Func](#)<R, object>>>

An expression specifying additional constraints to apply when loading the related data.

## Returns

[ICollection](#)<T>

A collection of primary models with the specified foreign key field populated.

## Type Parameters

**T**

The type of the primary model in the collection.

**R**

The type of the related model representing the foreign key field.

## S

The type of the model used by the custom data loader.

## Remarks

This method uses a CoreRelm.RelmInternal.Helpers.Utilities.ForeignKeyLoader<T> to load the foreign key field for the provided collection of models. The `customDataLoader` allows for custom logic to retrieve the related data, and the `additionalConstraints` parameter can be used to further refine the data loading process.

## LoadForeignKeyField<T, R, S>(ICollection<T>, IRelmQuickContext, Expression<Func<T, R>>, IRelmDataLoader<S>)

Loads and resolves a foreign key field for a collection of models, using the specified predicate and custom data loader.

```
public static ICollection<T> LoadForeignKeyField<T, R, S>(this ICollection<T> modelData, IRelmQuickContext relmQuickContext, Expression<Func<T, R>> predicate, IRelmDataLoader<S> customDataLoader) where T : IRelmModel, new() where R : IRelmModel, new() where S : IRelmModel, new()
```

## Parameters

`modelData` [ICollection](#)<T>

The collection of primary models for which the foreign key field will be loaded.

`relmQuickContext` [IRelmQuickContext](#)

The context used to manage data loading operations.

`predicate` [Expression](#)<[Func](#)<T, R>>

An expression that specifies the foreign key field to be loaded.

`customDataLoader` [IRelmDataLoader](#)<S>

A custom data loader used to retrieve the related data.

## Returns

[ICollection](#)<T>

A collection of the primary models with the specified foreign key field resolved.

## Type Parameters

T

The type of the primary model in the collection. Must implement [IRelmModel](#).

R

The type of the related model being loaded. Must implement [IRelmModel](#).

S

The type of the model used by the custom data loader. Must implement [IRelmModel](#).

## Remarks

This method is an extension method that simplifies the process of resolving foreign key relationships for a collection of models. The `customDataLoader` allows for custom logic to be applied when retrieving the related data.

**LoadForeignKeyField<T, R, S>(ICollection<T>,  
IRelmQuickContext, Expression<Func<T, R>>,  
IRelmDataLoader<S>, Expression<Func<R, object>>)**

Loads and resolves a foreign key field for a collection of models, applying additional constraints if specified.

```
public static ICollection<T> LoadForeignKeyField<T, R, S>(this ICollection<T> modelData,  
IRelmQuickContext relmQuickContext, Expression<Func<T, R>> predicate, IRelmDataLoader<S>  
customDataLoader, Expression<Func<R, object>> additionalConstraints) where T : IRelmModel,  
new() where R : IRelmModel, new() where S : IRelmModel, new()
```

## Parameters

modelData [ICollection](#)<T>

The collection of primary models for which the foreign key field will be loaded.

#### `relmQuickContext` [IRelmQuickContext](#)

The context used to manage data loading operations.

#### `predicate` [Expression](#)<[Func](#)<T, R>>

An expression specifying the foreign key relationship between the primary model and the related model.

#### `customDataLoader` [IRelmDataLoader](#)<S>

A custom data loader used to retrieve the related models.

#### `additionalConstraints` [Expression](#)<[Func](#)<R, [object](#)>>>

An expression specifying additional constraints to apply when resolving the foreign key field.

Returns

#### [ICollection](#)<T>

A collection of the primary models with the foreign key field resolved and populated.

Type Parameters

#### T

The type of the primary model in the collection.

#### R

The type of the related model referenced by the foreign key.

#### S

The type of the model used by the custom data loader.

## LoadForeignKeyField<T, R, S>(ICollection<T>, RelmContextOptionsBuilder, Expression<Func<T, ICollection<R>>>, IRelmDataLoader<S>)

Loads and populates a foreign key collection field for the specified model data.

```
public static ICollection<T> LoadForeignKeyField<T, R, S>(this ICollection<T> modelData,  
RelmContextOptionsBuilder relmContextOptionsBuilder, Expression<Func<T, ICollection<R>>>  
predicate, IRelmDataLoader<S> customDataLoader) where T : IRelmModel, new() where R :  
IRelmModel, new() where S : IRelmModel, new()
```

## Parameters

**modelData** [ICollection](#)<T>

The collection of primary model instances for which the foreign key field will be loaded.

**relmContextOptionsBuilder** [RelmContextOptionsBuilder](#)

The options builder used to configure the data loading context.

**predicate** [Expression](#)<[Func](#)<T, ICollection><R>>>

An expression specifying the foreign key collection property to populate.

**customDataLoader** [IRelmDataLoader](#)<S>

A custom data loader used to retrieve the related data.

## Returns

[ICollection](#)<T>

The original collection of primary model instances with the specified foreign key field populated.

## Type Parameters

**T**

The type of the primary model in the collection.

**R**

The type of the related model in the foreign key collection.

**S**

The type of the model used by the custom data loader.

## Remarks

This method uses a CoreRelm.RelmInternal.Helpers.Utilities.ForeignKeyLoader<T> to load the foreign key collection field for each model instance in the provided `modelData` collection. The `customDataLoader` is used to retrieve the related data, and the `predicate` specifies the foreign key collection property to populate.

**LoadForeignKeyField<T, R, S>(ICollection<T>, RelmContextOptionsBuilder, Expression<Func<T, ICollection<R>>>, IRelmDataLoader<S>, Expression<Func<R, object>>)**

Loads and populates a foreign key collection field for a collection of model data.

```
public static ICollection<T> LoadForeignKeyField<T, R, S>(this ICollection<T> modelData, RelmContextOptionsBuilder relmContextOptionsBuilder, Expression<Func<T, ICollection<R>>> predicate, IRelmDataLoader<S> customDataLoader, Expression<Func<R, object>> additionalConstraints) where T : IRelmModel, new() where R : IRelmModel, new() where S : IRelmModel, new()
```

## Parameters

`modelData` [ICollection<T>](#)

The collection of primary model data for which the foreign key field will be loaded.

`relmContextOptionsBuilder` [RelmContextOptionsBuilder](#)

The options builder used to configure the data loading context.

`predicate` [Expression<Func<T, ICollection<R>>>](#)

An expression specifying the foreign key field to be loaded.

`customDataLoader` [IRelmDataLoader<S>](#)

A custom data loader used to retrieve the related data.

`additionalConstraints` [Expression<Func<R, object>>>](#)

An expression specifying additional constraints to apply when loading the foreign key field.

## Returns

## [ICollection](#)<T>

A collection of the primary model data with the specified foreign key field populated.

### Type Parameters

#### T

The type of the primary model in the collection.

#### R

The type of the related model representing the foreign key field.

#### S

The type of the model used by the custom data loader.

### Remarks

This method uses a foreign key loader to populate the specified foreign key field for the provided collection of model data. The `customDataLoader` allows for custom logic to retrieve the related data, and the `additionalConstraints` parameter can be used to apply additional filtering or constraints.

## LoadForeignKeyField<T, R, S>(ICollection<T>, RelmContextOptionsBuilder, Expression<Func<T, R>>, IRelmDataLoader<S>)

Loads and resolves a foreign key field for a collection of models, using the specified predicate and custom data loader.

```
public static ICollection<T> LoadForeignKeyField<T, R, S>(this ICollection<T> modelData, RelmContextOptionsBuilder relmContextOptionsBuilder, Expression<Func<T, R>> predicate, IRelmDataLoader<S> customDataLoader) where T : IRelmModel, new() where R : IRelmModel, new() where S : IRelmModel, new()
```

### Parameters

#### modelData [ICollection](#)<T>

The collection of primary models for which the foreign key field will be loaded.

## `relmContextOptionsBuilder` [RelmContextOptionsBuilder](#)

The options builder used to configure the data loading context.

## `predicate` [Expression](#)<[Func](#)<T, R>>

An expression specifying the foreign key relationship to resolve.

## `customDataLoader` [IRelmDataLoader](#)<S>

A custom data loader used to fetch the related data.

Returns

## [ICollection](#)<T>

A collection of the primary models with the specified foreign key field resolved.

Type Parameters

### T

The type of the primary model in the collection. Must implement [IRelmModel](#).

### R

The type of the related model being resolved. Must implement [IRelmModel](#).

### S

The type of the model used by the custom data loader. Must implement [IRelmModel](#).

Remarks

This method uses a foreign key loader to resolve the specified relationship for the provided collection of models. The `customDataLoader` allows for custom logic to be applied when loading the related data.

## `LoadForeignKeyField<T, R, S>(ICollection<T>, RelmContextOptionsBuilder, Expression<Func<T, R>>, IRelmDataLoader<S>, Expression<Func<R, object>>)`

Loads and resolves a foreign key field for a collection of models, applying additional constraints if specified.

```
public static ICollection<T> LoadForeignKeyField<T, R, S>(this ICollection<T> modelData,  
RelmContextOptionsBuilder relmContextOptionsBuilder, Expression<Func<T, R>> predicate,  
IRelmDataLoader<S> customDataLoader, Expression<Func<R, object>> additionalConstraints)  
where T : IRelmModel, new() where R : IRelmModel, new() where S : IRelmModel, new()
```

## Parameters

`modelData` [ICollection](#)<T>

The collection of primary models for which the foreign key field will be loaded.

`relmContextOptionsBuilder` [RelmContextOptionsBuilder](#)

The options builder used to configure the data loading context.

`predicate` [Expression](#)<[Func](#)<T, R>>

An expression specifying the foreign key relationship between the primary model and the related model.

`customDataLoader` [IRelmDataLoader](#)<S>

A custom data loader used to fetch the related data.

`additionalConstraints` [Expression](#)<[Func](#)<R, object>>

An optional expression specifying additional constraints to apply when resolving the foreign key field.

## Returns

[ICollection](#)<T>

A collection of the primary models with the foreign key field resolved and loaded.

## Type Parameters

T

The type of the primary model in the collection. Must implement [IRelmModel](#).

R

The type of the related model representing the foreign key. Must implement [IRelmModel](#).

The type of the model used by the custom data loader. Must implement [IRelmModel](#).

## WriteToDatabase<T>(IEnumerable<T>, IRelmContext, string, Type, int, string, bool, bool, bool)

Writes the specified collection of data to a database table in bulk.

```
public static int WriteToDatabase<T>(this IEnumerable<T> modelData, IRelmContext
relmContext, string tableName = null, Type forceType = null, int batchSize = 100, string
databaseName = null, bool allowAutoIncrementColumns = false, bool allowPrimaryKeyColumns =
false, bool allowUniqueColumns = false, bool allowAutoDateColumns = false)
```

### Parameters

**modelData** [IEnumerable](#)<T>

The collection of data objects to write to the database. Cannot be null or empty.

**relmContext** [IRelmContext](#)

The database context used to perform the write operation. Cannot be null.

**tableName** [string](#)

The name of the database table to write to. If null, the table name is inferred from the type T.

**forceType** [Type](#)

An optional type to enforce for the database table schema. If null, the schema is inferred from T.

**batchSize** [int](#)

The number of records to write in each batch. Defaults to 100. Must be greater than zero.

**databaseName** [string](#)

The name of the database to write to. If null, the default database is used.

**allowAutoIncrementColumns** [bool](#)

Indicates whether auto-increment columns are allowed in the table. Defaults to [false](#).

## allowPrimaryKeyColumns [bool](#)

Indicates whether primary key columns are allowed in the table. Defaults to [false](#).

## allowUniqueColumns [bool](#)

Indicates whether unique columns are allowed in the table. Defaults to [false](#).

## allowAutoDateColumns [bool](#)

Indicates whether auto-generated date columns are allowed in the table. Defaults to [false](#).

Returns

## [int](#)

The number of records successfully written to the database.

Type Parameters

## T

The type of the data objects to be written to the database.

Remarks

This method performs a bulk write operation, which is optimized for performance when inserting large amounts of data. The table schema is inferred from the type T unless a specific schema is enforced using the [forceType](#) parameter.

## WriteToDatabase<T>(IEnumerable<T>, IRelmQuickContext, string, Type, int, string, bool, bool, bool)

Writes the specified collection of data to a database table in bulk.

```
public static int WriteToDatabase<T>(this IEnumerable<T> modelData, IRelmQuickContext  
relmContext, string tableName = null, Type forceType = null, int batchSize = 100, string  
databaseName = null, bool allowAutoIncrementColumns = false, bool allowPrimaryKeyColumns =  
false, bool allowUniqueColumns = false, bool allowAutoDateColumns = false)
```

Parameters

`modelData` [IEnumerable](#)<T>

The collection of data objects to write to the database. Cannot be null or empty.

`realmContext` [IRelmQuickContext](#)

The database context used to perform the write operation. Cannot be null.

`tableName` [string](#)

The name of the database table to write to. If null, the table name is inferred from the type T.

`forceType` [Type](#)

An optional type to enforce for the database table schema. If null, the schema is inferred from the type T.

`batchSize` [int](#)

The number of records to write in each batch. Defaults to 100. Must be greater than zero.

`databaseName` [string](#)

The name of the database to write to. If null, the default database associated with realmContext is used.

`allowAutoIncrementColumns` [bool](#)

A value indicating whether columns with auto-increment constraints are allowed to be written.

Defaults to [false](#).

`allowPrimaryKeyColumns` [bool](#)

A value indicating whether primary key columns are allowed to be written. Defaults to [false](#).

`allowUniqueColumns` [bool](#)

A value indicating whether columns with unique constraints are allowed to be written. Defaults to [false](#).

`allowAutoDateColumns` [bool](#)

A value indicating whether columns with automatic date constraints are allowed to be written. Defaults to [false](#).

Returns

[int ↗](#)

The total number of records successfully written to the database.

## Type Parameters

T

The type of the data objects to be written to the database.

## Remarks

This method performs a bulk write operation, which is optimized for performance when inserting large amounts of data. The behavior of the write operation can be customized using the optional parameters.

## WriteToDatabase<T>(IEnumerable<T>, MySqlConnection, MySqlTransaction, string, Type, int, string, bool, bool, bool)

Writes a collection of model data to a database table in bulk.

```
public static int WriteToDatabase<T>(this IEnumerable<T> modelData, MySqlConnection  
existingConnection, MySqlTransaction sqlTransaction = null, string tableName = null, Type  
forceType = null, int batchSize = 100, string databaseName = null, bool  
allowAutoIncrementColumns = false, bool allowPrimaryKeyColumns = false, bool  
allowUniqueColumns = false, bool allowAutoDateColumns = false) where T : IRelmModel
```

## Parameters

**modelData** [IEnumerable ↗<T>](#)

The collection of model data to write to the database.

**existingConnection** MySqlConnection

An open MySql.Data.MySqlClient.MySqlConnection to the database where the data will be written.

**sqlTransaction** MySqlTransaction

An optional MySql.Data.MySqlClient.MySqlTransaction to use for the operation. If null, no transaction is used.

**tableName** [string ↗](#)

The name of the database table to write to. If null, the table name is inferred from the model type.

#### forceType [Type](#)

An optional [Type](#) to enforce a specific model type for the operation. If null, the type of [T](#) is used.

#### batchSize [int](#)

The number of rows to write in each batch. Defaults to 100.

#### databaseName [string](#)

The name of the database to write to. If null, the default database for the connection is used.

#### allowAutoIncrementColumns [bool](#)

Indicates whether auto-increment columns are allowed to be written. Defaults to [false](#).

#### allowPrimaryKeyColumns [bool](#)

Indicates whether primary key columns are allowed to be written. Defaults to [false](#).

#### allowUniqueColumns [bool](#)

Indicates whether unique columns are allowed to be written. Defaults to [false](#).

#### allowAutoDateColumns [bool](#)

Indicates whether auto-generated date columns are allowed to be written. Defaults to [false](#).

Returns

#### [int](#)

The number of rows successfully written to the database.

Type Parameters

#### [T](#)

The type of the model data, which must implement [IRelmModel](#).

Remarks

This method performs a bulk write operation, which is optimized for inserting large amounts of data efficiently. The caller is responsible for ensuring that the [existingConnection](#) is open and valid before

calling this method. If `sqlTransaction` is provided, the operation will be part of the specified transaction.

## WriteToDatabase<T>(IEnumerable<T>, Enum, string, Type, bool, int, string, bool, bool, bool)

Writes a collection of model data to a database table in bulk.

```
public static int WriteToDatabase<T>(this IEnumerable<T> modelData, Enum connectionName,
string tableName = null, Type forceType = null, bool allowUserVariables = false, int
batchSize = 100, string databaseName = null, bool allowAutoIncrementColumns = false, bool
allowPrimaryKeyColumns = false, bool allowUniqueColumns = false, bool allowAutoDateColumns =
false) where T : IRelmModel
```

### Parameters

`modelData` [IEnumerable](#)<T>

The collection of model data to be written to the database.

`connectionName` [Enum](#)

The database connection identifier, represented as an enumeration value.

`tableName` [string](#)

The name of the database table to write to. If [null](#), the default table name for the model type `T` is used.

`forceType` [Type](#)

An optional type to enforce during the write operation. If [null](#), the type of `T` is used.

`allowUserVariables` [bool](#)

A value indicating whether user-defined variables are allowed in the database operation. The default is [false](#).

`batchSize` [int](#)

The number of records to write in each batch. The default is 100.

`databaseName` [string](#)

The name of the database to write to. If [null](#), the default database for the connection is used.

#### **allowAutoIncrementColumns** [bool](#)

A value indicating whether auto-increment columns are allowed to be written. The default is [false](#).

#### **allowPrimaryKeyColumns** [bool](#)

A value indicating whether primary key columns are allowed to be written. The default is [false](#).

#### **allowUniqueColumns** [bool](#)

A value indicating whether unique columns are allowed to be written. The default is [false](#).

#### **allowAutoDateColumns** [bool](#)

A value indicating whether auto-generated date columns are allowed to be written. The default is [false](#).

Returns

#### [int](#)

The number of records successfully written to the database.

Type Parameters

#### **T**

The type of the model data, which must implement [IRelmModel](#).

Remarks

This method performs a bulk write operation, which is optimized for high performance when inserting large amounts of data. Ensure that the database schema matches the structure of the model data to avoid runtime errors.

# Class ModelExtensions

Namespace: [CoreRelm.Extensions](#)

Assembly: CoreRelm.dll

```
public static class ModelExtensions
```

## Inheritance

[object](#) ← ModelExtensions

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

## Methods

**LoadDataLoaderField<T, S>(T, IRelmContext, Expression<Func<T, S>>)**

Loads a related data field for the specified model using the provided predicate.

```
public static T LoadDataLoaderField<T, S>(this T inputModel, IRelmContext relmContext,  
Expression<Func<T, S>> predicate) where T : IRelmModel, new() where S : IRelmModel, new()
```

### Parameters

**inputModel** T

The model instance for which the related data field is to be loaded.

**relmContext** [IRelmContext](#)

The context used to access the data source.

**predicate** [Expression](#)<[Func](#)<T, S>>

An expression specifying the related data field to load.

### Returns

T

The input model with the specified related data field loaded.

## Type Parameters

T

The type of the input model, which must implement [IRelmModel](#).

S

The type of the related data field, which must implement [IRelmModel](#).

## Remarks

This method is an extension method that facilitates loading a specific related data field for a model. The `predicate` parameter is used to identify the field to be loaded.

## LoadDataLoaderField<T, S>(T, IRelmQuickContext, Expression<Func<T, S>>)

Loads a related data loader field for the specified model using the provided context and predicate.

```
public static T LoadDataLoaderField<T, S>(this T inputModel, IRelmQuickContext  
relmQuickContext, Expression<Func<T, S>> predicate) where T : IRelmModel, new() where S :  
IRelmModel, new()
```

## Parameters

`inputModel` T

The model instance for which the data loader field is to be loaded.

`relmQuickContext` [IRelmQuickContext](#)

The context used to resolve the data loader field.

`predicate` [Expression](#)<[Func](#)<T, S>>

An expression specifying the property of the input model that represents the related data loader field.

## Returns

T

The input model with the specified data loader field loaded.

## Type Parameters

T

The type of the input model, which must implement [IRelmModel](#) and have a parameterless constructor.

S

The type of the related model, which must implement [IRelmModel](#) and have a parameterless constructor.

## Remarks

This method is an extension method that facilitates the loading of related data fields in a model. It uses the provided predicate to identify the field to be loaded and the context to resolve the data.

## LoadDataLoaderField<T, S>(T, RelmContextOptionsBuilder, Expression<Func<T, S>>)

Loads a related data loader field for the specified input model using the provided context options and predicate.

```
public static T LoadDataLoaderField<T, S>(this T inputModel, RelmContextOptionsBuilder  
relmContextOptionsBuilder, Expression<Func<T, S>> predicate) where T : IRelmModel, new()  
where S : IRelmModel, new()
```

## Parameters

**inputModel** T

The input model instance for which the related data loader field is to be loaded.

**relmContextOptionsBuilder** [RelmContextOptionsBuilder](#)

The options builder used to configure the [RelmContext](#) for the operation.

## **predicate** [Expression](#)<Func<T, S>>

An expression specifying the related data loader field to load.

## Returns

T

The input model with the specified related data loader field loaded.

## Type Parameters

T

The type of the input model, which must implement [IRelmModel](#) and have a parameterless constructor.

S

The type of the related model to load, which must implement [IRelmModel](#) and have a parameterless constructor.

## Remarks

This method is an extension method that simplifies the process of loading a related data loader field for a given model. It creates a new [RelmContext](#) using the provided options builder and applies the specified predicate to determine the field to load.

## LoadForeignKeyField<T, R>(T, IRelmContext, Expression<Func<T, ICollection<R>>>)

Loads a foreign key collection property for the specified target model using the provided context and predicate.

```
public static T LoadForeignKeyField<T, R>(this T target, IRelmContext relmContext, Expression<Func<T, ICollection<R>>> predicate) where T : IRelmModel, new() where R : IRelmModel, new()
```

## Parameters

target T

The target model instance for which the foreign key collection property will be loaded.

#### **relmContext** [IRelmContext](#)

The context used to load the foreign key collection property.

#### **predicate** [Expression](#)<[Func](#)<T, [ICollection](#)<R>>>

An expression specifying the foreign key collection property to load.

Returns

#### T

The target model instance with the specified foreign key collection property loaded.

Type Parameters

#### T

The type of the target model, which must implement [IRelmModel](#).

#### R

The type of the related model in the collection, which must implement [IRelmModel](#).

Remarks

This method is an extension method that simplifies the process of loading a foreign key collection property for a model. It uses the provided **relmContext** and **predicate** to determine which collection property to load.

## LoadForeignKeyField<T, R>(T, IRelmContext, Expression<Func<T, ICollection<R>>>, Expression<Func<R, object>>)

Loads and initializes a foreign key collection property for the specified target model, applying the given constraints.

```
public static T LoadForeignKeyField<T, R>(this T target, IRelmContext relmContext, Expression<Func<T, ICollection<R>>> predicate, Expression<Func<R, object>> additionalConstraints) where T : IRelmModel, new() where R : IRelmModel, new()
```

## Parameters

**target T**

The target model instance for which the foreign key field will be loaded.

**relmContext [IRelmContext](#)**

The context used to manage the relationship and database operations.

**predicate [Expression<Func<T, ICollection<R>>>](#)**

An expression specifying the foreign key field to be loaded as a collection of related models.

**additionalConstraints [Expression<Func<R, object>>>](#)**

An expression defining additional constraints to filter the related models.

## Returns

**T**

The target model instance with the foreign key field loaded and initialized.

## Type Parameters

**T**

The type of the target model that contains the foreign key field. Must implement [IRelmModel](#).

**R**

The type of the related model referenced by the foreign key field. Must implement [IRelmModel](#).

## LoadForeignKeyField<T, R>(T, IRelmContext, Expression<Func<T, R>>)

Loads a foreign key field for the specified target model using the provided context and predicate.

```
public static T LoadForeignKeyField<T, R>(this T target, IRelmContext relmContext, Expression<Func<T, R>> predicate) where T : IRelmModel, new() where R : IRelmModel, new()
```

## Parameters

### target T

The target model instance for which the foreign key field will be loaded.

### relmContext [IRelmContext](#)

The context used to load the foreign key field.

### predicate [Expression<Func<T, R>>](#)

An expression specifying the foreign key field to load.

## Returns

### T

The target model instance with the specified foreign key field loaded.

## Type Parameters

### T

The type of the target model. Must implement [IRelmModel](#) and have a parameterless constructor.

### R

The type of the related model. Must implement [IRelmModel](#) and have a parameterless constructor.

## Remarks

This method is an extension method that simplifies loading a foreign key field for a model by leveraging the provided context and predicate. The method ensures that the specified foreign key field is populated based on the context's configuration.

## LoadForeignKeyField<T, R>(T, IRelmContext, Expression<Func<T, R>>, Expression<Func<R, object>>)

Loads and resolves a foreign key field for the specified target model, applying the given predicate and additional constraints.

```
public static T LoadForeignKeyField<T, R>(this T target, IRelmContext relmContext,  
Expression<Func<T, R>> predicate, Expression<Func<R, object>> additionalConstraints) where T  
: IRelmModel, new() where R : IRelmModel, new()
```

## Parameters

**target** T

The target model instance for which the foreign key field is being loaded.

**relmContext** [IRelmContext](#)

The context used to resolve the foreign key field, providing configuration and database access.

**predicate** [Expression](#)<[Func](#)<T, R>>

An expression specifying the foreign key relationship between the target model and the related model.

**additionalConstraints** [Expression](#)<[Func](#)<R, object>>

An expression specifying additional constraints to apply when resolving the foreign key field.

## Returns

T

The target model instance with the foreign key field resolved.

## Type Parameters

T

The type of the target model. Must implement [IRelmModel](#).

R

The type of the related model. Must implement [IRelmModel](#).

## Remarks

This method is an extension method that simplifies the process of resolving foreign key fields in models. It uses the provided context and constraints to load the related model data.

# LoadForeignKeyField<T, R>(T, IRelmQuickContext, Expression<Func<T, ICollection<R>>>)

Loads a foreign key collection property for the specified target model.

```
public static T LoadForeignKeyField<T, R>(this T target, IRelmQuickContext relmContext, Expression<Func<T, ICollection<R>>> predicate) where T : IRelmModel, new() where R : IRelmModel, new()
```

## Parameters

**target** T

The target model instance for which the foreign key collection property will be loaded.

**relmContext** [IRelmQuickContext](#)

The context providing access to the database and configuration options.

**predicate** [Expression](#)<[Func](#)<T, [ICollection](#)<R>>>

An expression identifying the foreign key collection property to load.

## Returns

T

The target model instance with the specified foreign key collection property loaded.

## Type Parameters

T

The type of the target model, which must implement [IRelmModel](#).

R

The type of the related model in the collection, which must implement [IRelmModel](#).

## Remarks

This method is an extension method that simplifies loading a foreign key collection property for a model. It uses the provided context and predicate to retrieve and populate the related data.

# LoadForeignKeyField<T, R>(T, IRelmQuickContext, Expression<Func<T, ICollection<R>>>, Expression<Func<R, object>>)

Loads a foreign key collection property for the specified target model, resolving the related collection based on the provided predicate and additional constraints.

```
public static T LoadForeignKeyField<T, R>(this T target, IRelmQuickContext relmContext,  
Expression<Func<T, ICollection<R>>> predicate, Expression<Func<R, object>>  
additionalConstraints) where T : IRelmModel, new() where R : IRelmModel, new()
```

## Parameters

**target** T

The target model instance for which the foreign key field is being loaded.

**relmContext** [IRelmQuickContext](#)

The context providing access to the database and configuration options.

**predicate** [Expression<Func<T, ICollection<R>>>](#)

An expression specifying the foreign key collection property on the target model.

**additionalConstraints** [Expression<Func<R, object>>>](#)

An expression defining additional constraints to filter the related models.

## Returns

T

The target model instance with the foreign key field loaded.

## Type Parameters

T

The type of the target model. Must implement [IRelmModel](#).

R

The type of the related model in the foreign key relationship. Must implement [IRelmModel](#).

## LoadForeignKeyField<T, R>(T, IRelmQuickContext, Expression<Func<T, R>>)

Loads the foreign key field specified by the given predicate for the target model.

```
public static T LoadForeignKeyField<T, R>(this T target, IRelmQuickContext relmContext, Expression<Func<T, R>> predicate) where T : IRelmModel, new() where R : IRelmModel, new()
```

### Parameters

**target** T

The target model instance for which the foreign key field will be loaded.

**relmContext** [IRelmQuickContext](#)

The context used to access the database or data source.

**predicate** [Expression<Func<T, R>>](#)

An expression specifying the foreign key field to load.

### Returns

T

The target model instance with the specified foreign key field loaded.

### Type Parameters

T

The type of the target model, which must implement [IRelmModel](#).

R

The type of the related model, which must implement [IRelmModel](#).

### Remarks

This method is an extension method that simplifies loading a specific foreign key field for a model. The `predicate` parameter should specify the property representing the foreign key relationship.

## LoadForeignKeyField<T, R>(T, IRelmQuickContext, Expression<Func<T, R>>, Expression<Func<R, object>>)

Loads a foreign key field for the specified target model, applying the given predicate and additional constraints.

```
public static T LoadForeignKeyField<T, R>(this T target, IRelmQuickContext relmContext, Expression<Func<T, R>> predicate, Expression<Func<R, object>> additionalConstraints) where T : IRelmModel, new() where R : IRelmModel, new()
```

### Parameters

#### target T

The target model instance for which the foreign key field is to be loaded.

#### relmContext [IRelmQuickContext](#)

The context used to access the database or data source.

#### predicate [Expression<Func<T, R>>](#)

An expression specifying the foreign key relationship between the target model and the related model.

#### additionalConstraints [Expression<Func<R, object>>](#)

An expression specifying additional constraints to apply when loading the related model.

### Returns

#### T

The target model instance with the foreign key field loaded based on the specified predicate and constraints.

### Type Parameters

#### T

The type of the target model. Must implement [IRelmModel](#) and have a parameterless constructor.

R

The type of the related model. Must implement [IRelmModel](#) and have a parameterless constructor.

## Remarks

This method is an extension method that simplifies loading related data for a model by resolving foreign key relationships. Ensure that the `relmContext` is properly configured to access the underlying data source.

## LoadForeignKeyField<T, R>(T, RelmContextOptionsBuilder, Expression<Func<T, ICollection<R>>>)

Loads a foreign key collection property for the specified target model and returns the first related entity.

```
public static T LoadForeignKeyField<T, R>(this T target, RelmContextOptionsBuilder  
relmContextOptionsBuilder, Expression<Func<T, ICollection<R>>> predicate) where T :  
IRelmModel, new() where R : IRelmModel, new()
```

## Parameters

**target** T

The target model instance for which the foreign key field is being loaded.

**relmContextOptionsBuilder** [RelmContextOptionsBuilder](#)

The context options builder used to configure the data context.

**predicate** [Expression](#)<[Func](#)<T, [ICollection](#)<R>>>

An expression specifying the collection property on the target model that represents the foreign key relationship.

## Returns

T

The first related entity of type `R` from the foreign key field, or [null](#) if no related entities are found.

## Type Parameters

T

The type of the target model, which must implement [IRelmModel](#).

R

The type of the related model, which must implement [IRelmModel](#).

## Remarks

This method uses a CoreRelm.RelmInternal.Helpers.Utilities.ForeignKeyLoader<T> to load the foreign key field based on the specified predicate.

**LoadForeignKeyField<T, R>(T, RelmContextOptionsBuilder, Expression<Func<T, ICollection<R>>>, Expression<Func<R, object>>)**

Loads a foreign key collection property for the specified target model, applying the given predicate and additional constraints.

```
public static T LoadForeignKeyField<T, R>(this T target, RelmContextOptionsBuilder  
relmContextOptionsBuilder, Expression<Func<T, ICollection<R>>> predicate, Expression<Func<R,  
object>> additionalConstraints) where T : IRelmModel, new() where R : IRelmModel, new()
```

## Parameters

**target T**

The target model instance for which the foreign key field is being loaded.

**relmContextOptionsBuilder [RelmContextOptionsBuilder](#)**

The options builder used to configure the database context.

**predicate [Expression](#)<[Func](#)<T, [ICollection](#)<R>>>**

An expression specifying the collection navigation property on the target model.

**additionalConstraints [Expression](#)<[Func](#)<R, [object](#)>>>**

An expression specifying additional constraints to apply when loading the related models.

## Returns

T

The first related model that matches the specified predicate and constraints, or [null](#) if no match is found.

## Type Parameters

T

The type of the target model. Must implement [IRelmModel](#).

R

The type of the related model. Must implement [IRelmModel](#).

## LoadForeignKeyField<T, R>(T, RelmContextOptionsBuilder, Expression<Func<T, R>>)

Loads a foreign key field for the specified target model using the provided context options and predicate.

```
public static T LoadForeignKeyField<T, R>(this T target, RelmContextOptionsBuilder  
relmContextOptionsBuilder, Expression<Func<T, R>> predicate) where T : IRelmModel, new()  
where R : IRelmModel, new()
```

## Parameters

**target** T

The target model instance for which the foreign key field is to be loaded.

**relmContextOptionsBuilder** [RelmContextOptionsBuilder](#)

The context options builder used to configure the database context for the operation.

**predicate** [Expression](#)<[Func](#)<T, R>>

An expression specifying the foreign key field to load.

## Returns

T

The first related model instance matching the foreign key field, or [null](#) if no match is found.

## Type Parameters

T

The type of the target model. Must implement [IRelmModel](#) and have a parameterless constructor.

R

The type of the related model. Must implement [IRelmModel](#) and have a parameterless constructor.

## Remarks

This method uses a CoreRelm.RelmInternal.Helpers.Utilities.ForeignKeyLoader<T> to load the foreign key field based on the provided predicate.

## LoadForeignKeyField<T, R>(T, RelmContextOptionsBuilder, Expression<Func<T, R>>, Expression<Func<R, object>>)

Loads a foreign key field for the specified target model, applying the given predicate and additional constraints.

```
public static T LoadForeignKeyField<T, R>(this T target, RelmContextOptionsBuilder  
relmContextOptionsBuilder, Expression<Func<T, R>> predicate, Expression<Func<R, object>>  
additionalConstraints) where T : IRelmModel, new() where R : IRelmModel, new()
```

## Parameters

**target** T

The target model instance for which the foreign key field is being loaded.

**relmContextOptionsBuilder** [RelmContextOptionsBuilder](#)

The options builder used to configure the context for the operation.

**predicate** [Expression](#)<Func<T, R>>

An expression specifying the foreign key relationship to load.

**additionalConstraints** [Expression](#) <Func<R, object>>

An expression specifying additional constraints to apply when loading the foreign key field.

Returns

T

The first related model that matches the specified predicate and constraints, or [null](#) if no match is found.

Type Parameters

T

The type of the target model. Must implement [IRelmModel](#).

R

The type of the related model. Must implement [IRelmModel](#).

Remarks

This method is typically used to load a related entity for a given model instance based on a foreign key relationship. Ensure that the provided expressions are valid and compatible with the underlying data context.

**LoadForeignKeyField<T, R, S>(T, IRelmContext, Expression<Func<T, ICollection<R>>>, IRelmDataLoader<S>)**

Loads and initializes a foreign key collection property for the specified target model using the provided data loader.

```
public static T LoadForeignKeyField<T, R, S>(this T target, IRelmContext relmContext, Expression<Func<T, ICollection<R>>> predicate, IRelmDataLoader<S> customDataLoader) where T : IRelmModel, new() where R : IRelmModel, new() where S : IRelmModel, new()
```

Parameters

**target** T

The target model instance for which the foreign key field will be loaded.

#### `relmContext` [IRelmContext](#)

The context used to manage the data loading operation.

#### `predicate` [Expression](#)<[Func](#)<T, [ICollection](#)<R>>>

An expression specifying the foreign key collection property on the target model.

#### `customDataLoader` [IRelmDataLoader](#)<S>

A custom data loader used to retrieve the related data.

## Returns

### T

The target model instance with the foreign key field loaded.

## Type Parameters

### T

The type of the target model that contains the foreign key field.

### R

The type of the related model in the foreign key collection.

### S

The type of the model used by the custom data loader.

## LoadForeignKeyField<T, R, S>(T, IRelmContext, Expression<Func<T, ICollection<R>>>, IRelmDataLoader<S>, Expression<Func<R, object>>)

Loads a foreign key collection property for the specified target model, resolving related entities based on the provided predicate and constraints.

```
public static T LoadForeignKeyField<T, R, S>(this T target, IRelmContext relmContext, Expression<Func<T, ICollection<R>>> predicate, IRelmDataLoader<S> customDataLoader,
```

```
Expression<Func<R, object>> additionalConstraints) where T : IRelmModel, new() where R : IRelmModel, new() where S : IRelmModel, new()
```

## Parameters

### target T

The target model instance for which the foreign key field will be loaded.

### relmContext [IRelmContext](#)

The context used to manage the data loading operation.

### predicate [Expression](#)<[Func](#)<T, [ICollection](#)<R>>>

An expression specifying the foreign key collection property on the target model.

### customDataLoader [IRelmDataLoader](#)<S>

A custom data loader used to resolve the related entities.

### additionalConstraints [Expression](#)<[Func](#)<R, object>>>

An expression specifying additional constraints to apply when resolving the related entities.

## Returns

### T

The target model instance with the foreign key field loaded.

## Type Parameters

### T

The type of the target model. Must implement [IRelmModel](#).

### R

The type of the related model in the foreign key collection. Must implement [IRelmModel](#).

### S

The type of the model used by the custom data loader. Must implement [IRelmModel](#).

## Remarks

This method facilitates the loading of related entities for a foreign key collection property on the target model. The `customDataLoader` allows for custom logic to be applied during the data loading process, and the `additionalConstraints` can be used to further filter the related entities.

## LoadForeignKeyField<T, R, S>(T, IRelmContext, Expression<Func<T, R>>, IRelmDataLoader<S>)

Loads and assigns a foreign key field for the specified target model using the provided data loader.

```
public static T LoadForeignKeyField<T, R, S>(this T target, IRelmContext relmContext,
Expression<Func<T, R>> predicate, IRelmDataLoader<S> customDataLoader) where T : IRelmModel,
new() where R : IRelmModel, new() where S : IRelmModel, new()
```

## Parameters

### target T

The target model instance for which the foreign key field will be loaded.

### relmContext [IRelmContext](#)

The context used to configure the data loading operation.

### predicate [Expression](#)<[Func](#)<T, R>>

An expression identifying the foreign key field to be loaded.

### customDataLoader [IRelmDataLoader](#)<S>

A custom data loader used to retrieve the foreign key model.

## Returns

### T

The target model instance with the foreign key field loaded and assigned.

## Type Parameters

### T

The type of the target model. Must implement [IRelmModel](#).

R

The type of the foreign key model. Must implement [IRelmModel](#).

S

The type of the model used by the custom data loader. Must implement [IRelmModel](#).

**LoadForeignKeyField<T, R, S>(T, IRelmContext, Expression<Func<T, R>>, IRelmDataLoader<S>, Expression<Func<R, object>>)**

Loads and initializes a foreign key field for the specified model, using the provided data loader and constraints.

```
public static T LoadForeignKeyField<T, R, S>(this T target, IRelmContext relmContext, Expression<Func<T, R>> predicate, IRelmDataLoader<S> customDataLoader, Expression<Func<R, object>> additionalConstraints) where T : IRelmModel, new() where R : IRelmModel, new() where S : IRelmModel, new()
```

## Parameters

**target** T

The instance of the model for which the foreign key field is being loaded.

**relmContext** [IRelmContext](#)

The context used to manage the data loading operation.

**predicate** [Expression](#)<[Func](#)<T, R>>

An expression identifying the foreign key field to be loaded.

**customDataLoader** [IRelmDataLoader](#)<S>

A custom data loader used to retrieve the related data.

**additionalConstraints** [Expression](#)<[Func](#)<R, object>>

An expression specifying additional constraints to apply when loading the related data.

Returns

T

The updated instance of the model with the foreign key field loaded.

Type Parameters

T

The type of the model containing the foreign key field. Must implement [IRelmModel](#).

R

The type of the related model referenced by the foreign key. Must implement [IRelmModel](#).

S

The type of the model used by the custom data loader. Must implement [IRelmModel](#).

Remarks

This method provides a way to load and initialize a foreign key field for a model, allowing for custom data loading and additional constraints. It is particularly useful in scenarios where the default data loading behavior needs to be customized or extended.

**LoadForeignKeyField<T, R, S>(T, IRelmQuickContext, Expression<Func<T, ICollection<R>>>, IRelmDataLoader<S>)**

Loads and initializes a foreign key collection property for the specified target model, using the provided data loader and predicate.

```
public static T LoadForeignKeyField<T, R, S>(this T target, IRelmQuickContext relmContext, Expression<Func<T, ICollection<R>>> predicate, IRelmDataLoader<S> customDataLoader) where T : IRelmModel, new() where R : IRelmModel, new() where S : IRelmModel, new()
```

Parameters

**target T**

The target model instance for which the foreign key field will be loaded.

`relmContext` [IRelmQuickContext](#)

The context providing configuration and options for the operation.

`predicate` [Expression](#)<[Func](#)<T, [ICollection](#)<R>>>

An expression specifying the foreign key collection property on the target model.

`customDataLoader` [IRelmDataLoader](#)<S>

A custom data loader used to retrieve the related data.

Returns

T

The target model instance with the foreign key field loaded.

Type Parameters

T

The type of the target model. Must implement [IRelmModel](#).

R

The type of the related model in the foreign key collection. Must implement [IRelmModel](#).

S

The type of the model used by the custom data loader. Must implement [IRelmModel](#).

Remarks

This method is an extension method that simplifies the process of loading and initializing foreign key fields for models in a relational context. The `customDataLoader` is used to retrieve the related data, and the `predicate` specifies the foreign key collection to be populated.

`LoadForeignKeyField<T, R, S>(T, IRelmQuickContext,  
Expression<Func<T, ICollection<R>>>, IRelmDataLoader<S>,  
Expression<Func<R, object>>)`

Loads and populates a foreign key collection property for the specified target model using the provided data loader and constraints.

```
public static T LoadForeignKeyField<T, R, S>(this T target, IRelmQuickContext relmContext,  
Expression<Func<T, ICollection<R>>> predicate, IRelmDataLoader<S> customDataLoader,  
Expression<Func<R, object>>> additionalConstraints) where T : IRelmModel, new() where R :  
IRelmModel, new() where S : IRelmModel, new()
```

## Parameters

**target** T

The target model instance for which the foreign key field will be loaded.

**relmContext** [IRelmQuickContext](#)

The context providing configuration and services for data loading.

**predicate** [Expression](#)<[Func](#)<T, [ICollection](#)<R>>>

An expression identifying the foreign key collection property on the target model.

**customDataLoader** [IRelmDataLoader](#)<S>

A custom data loader used to retrieve the related data.

**additionalConstraints** [Expression](#)<[Func](#)<R, object>>>

An expression specifying additional constraints to apply when loading the related data.

## Returns

T

The target model instance with the foreign key field populated.

## Type Parameters

T

The type of the target model that contains the foreign key field. Must implement [IRelmModel](#).

R

The type of the related model in the foreign key collection. Must implement [IRelmModel](#).

## S

The type of the model used by the custom data loader. Must implement [IRelmModel](#).

## Remarks

This method is an extension method that simplifies the process of loading related data for a foreign key field. It uses the provided `customDataLoader` and `additionalConstraints` to retrieve and populate the related data.

## LoadForeignKeyField<T, R, S>(T, IRelmQuickContext, Expression<Func<T, R>>, IRelmDataLoader<S>)

Loads and initializes a foreign key field for the specified model using the provided data loader and context.

```
public static T LoadForeignKeyField<T, R, S>(this T target, IRelmQuickContext relmContext, Expression<Func<T, R>> predicate, IRelmDataLoader<S> customDataLoader) where T : IRelmModel, new() where R : IRelmModel, new() where S : IRelmModel, new()
```

## Parameters

### target T

The target model instance for which the foreign key field will be loaded.

### relmContext [IRelmQuickContext](#)

The context providing configuration and options for the operation.

### predicate [Expression<Func<T, R>>](#)

An expression identifying the foreign key field to be loaded.

### customDataLoader [IRelmDataLoader<S>](#)

A custom data loader used to retrieve the foreign key field data.

## Returns

T

The target model instance with the specified foreign key field loaded.

## Type Parameters

T

The type of the target model. Must implement [IRelmModel](#).

R

The type of the foreign key field to be loaded. Must implement [IRelmModel](#).

S

The type of the model used by the custom data loader. Must implement [IRelmModel](#).

## Remarks

This method is an extension method that simplifies the process of loading foreign key fields for models in the Relm framework. It delegates the operation to an overload that uses the context options from the provided `relmContext`.

**LoadForeignKeyField<T, R, S>(T, IRelmQuickContext,  
Expression<Func<T, R>>, IRelmDataLoader<S>,  
Expression<Func<R, object>>)**

Loads and initializes a foreign key field for the specified target model using the provided context, predicate, and data loader.

```
public static T LoadForeignKeyField<T, R, S>(this T target, IRelmQuickContext relmContext,  
Expression<Func<T, R>> predicate, IRelmDataLoader<S> customDataLoader, Expression<Func<R,  
object>> additionalConstraints) where T : IRelmModel, new() where R : IRelmModel, new()  
where S : IRelmModel, new()
```

## Parameters

**target T**

The target model instance for which the foreign key field will be loaded.

`realmContext` [IRelmQuickContext](#)

The context used to configure and manage the data loading operation.

`predicate` [Expression](#)<[Func](#)<T, R>>

An expression identifying the foreign key field to be loaded on the target model.

`customDataLoader` [IRelmDataLoader](#)<S>

A custom data loader used to retrieve the foreign key data.

`additionalConstraints` [Expression](#)<[Func](#)<R, [object](#)>>

An optional expression specifying additional constraints to apply when loading the foreign key data.

Returns

T

The target model instance with the specified foreign key field loaded and initialized.

Type Parameters

T

The type of the target model. Must implement [IRelmModel](#).

R

The type of the foreign key model. Must implement [IRelmModel](#).

S

The type of the model used by the custom data loader. Must implement [IRelmModel](#).

Remarks

This method is an extension method that simplifies the process of loading foreign key fields in models. It delegates the operation to an overload that uses the context options from the provided `realmContext`.

`LoadForeignKeyField<T, R, S>(T, RelmContextOptionsBuilder, Expression<Func<T, ICollection<R>>>, IRelmDataLoader<S>)`

Loads a foreign key collection property for the specified target model using the provided predicate and custom data loader.

```
public static T LoadForeignKeyField<T, R, S>(this T target, RelmContextOptionsBuilder<T> relmContextOptionsBuilder, Expression<Func<T, ICollection<R>>> predicate, IRelmDataLoader<S> customDataLoader) where T : IRelmModel, new() where R : IRelmModel, new() where S : IRelmModel, new()
```

## Parameters

**target** **T**

The target model instance for which the foreign key field is being loaded.

**relmContextOptionsBuilder** [RelmContextOptionsBuilder](#)

The options builder used to configure the data loading context.

**predicate** [Expression](#)<[Func](#)<T, [ICollection](#)<R>>>

An expression specifying the foreign key collection property on the target model.

**customDataLoader** [IRelmDataLoader](#)<S>

A custom data loader used to retrieve the related data.

## Returns

**T**

The first related model from the foreign key collection, or [null](#) if no related models are found.

## Type Parameters

**T**

The type of the target model. Must implement [IRelmModel](#).

**R**

The type of the related model in the foreign key collection. Must implement [IRelmModel](#).

**S**

The type of the model used by the custom data loader. Must implement [IRelmModel](#).

## Remarks

This method uses a CoreRelm.RelmInternal.Helpers.Utilities.ForeignKeyLoader<T> to load the foreign key field based on the specified predicate and custom data loader. The method is designed to return only the first related model from the collection.

## LoadForeignKeyField<T, R, S>(T, RelmContextOptionsBuilder, Expression<Func<T, ICollection<R>>>, IRelmDataLoader<S>, Expression<Func<R, object>>)

Loads a foreign key collection property for the specified target model, applying the given constraints and using a custom data loader.

```
public static T LoadForeignKeyField<T, R, S>(this T target, RelmContextOptionsBuilder  
relmContextOptionsBuilder, Expression<Func<T, ICollection<R>>> predicate, IRelmDataLoader<S>  
customDataLoader, Expression<Func<R, object>> additionalConstraints) where T : IRelmModel,  
new() where R : IRelmModel, new() where S : IRelmModel, new()
```

## Parameters

### target T

The target model instance for which the foreign key field is being loaded.

### relmContextOptionsBuilder [RelmContextOptionsBuilder](#)

The options builder used to configure the context for data loading.

### predicate [Expression](#)<[Func](#)<T, [ICollection](#)<R>>>

An expression specifying the navigation property on the target model that represents the foreign key relationship.

### customDataLoader [IRelmDataLoader](#)<S>

The custom data loader used to retrieve the related data.

### additionalConstraints [Expression](#)<[Func](#)<R, [object](#)>>>

An expression specifying additional constraints to apply when loading the related data.

Returns

T

The first related entity that matches the specified constraints, or [null](#) if no matching entity is found.

Type Parameters

T

The type of the target model that implements [IRelmModel](#).

R

The type of the related model that implements [IRelmModel](#).

S

The type of the model used by the custom data loader that implements [IRelmModel](#).

**LoadForeignKeyField<T, R, S>(T, RelmContextOptionsBuilder, Expression<Func<T, R>>, IRelmDataLoader<S>)**

Loads a foreign key field for the specified target model using the provided predicate and data loader.

```
public static T LoadForeignKeyField<T, R, S>(this T target, RelmContextOptionsBuilder  
relmContextOptionsBuilder, Expression<Func<T, R>> predicate, IRelmDataLoader<S>  
customDataLoader) where T : IRelmModel, new() where R : IRelmModel, new() where S :  
IRelmModel, new()
```

Parameters

**target** T

The target model instance for which the foreign key field is being loaded.

**relmContextOptionsBuilder** [RelmContextOptionsBuilder](#)

The options builder used to configure the context for data loading.

**predicate** [Expression](#)<[Func](#)<T, R>>

An expression that specifies the foreign key field to load.

## customDataLoader [IRelmDataLoader<S>](#)

A custom data loader used to retrieve the related data.

## Returns

T

The first related model of type R loaded for the specified foreign key field, or [null](#) if no related data is found.

## Type Parameters

T

The type of the target model. Must implement [IRelmModel](#).

R

The type of the related model represented by the foreign key. Must implement [IRelmModel](#).

S

The type of the model used by the custom data loader. Must implement [IRelmModel](#).

## Remarks

This method uses a CoreRelm.RelmInternal.Helpers.Utilities.ForeignKeyLoader<T> to load the foreign key field based on the provided predicate and custom data loader. The method returns the first related model found, or [null](#) if no data matches.

## LoadForeignKeyField<T, R, S>(T, RelmContextOptionsBuilder, Expression<Func<T, R>>, IRelmDataLoader<S>, Expression<Func<R, object>>)

Loads a foreign key field for the specified target model using the provided predicate, data loader, and additional constraints.

```
public static T LoadForeignKeyField<T, R, S>(this T target, RelmContextOptionsBuilder  
relmContextOptionsBuilder, Expression<Func<T, R>> predicate, IRelmDataLoader<S>  
customDataLoader, Expression<Func<R, object>> additionalConstraints) where T : IRelmModel,  
new() where R : IRelmModel, new() where S : IRelmModel, new()
```

## Parameters

### target T

The target model instance for which the foreign key field is being loaded.

### relmContextOptionsBuilder [RelmContextOptionsBuilder](#)

The options builder used to configure the context for the operation.

### predicate [Expression](#)<[Func](#)<T, R>>

An expression specifying the foreign key property on the target model.

### customDataLoader [IRelmDataLoader](#)<S>

A custom data loader used to retrieve the related data.

### additionalConstraints [Expression](#)<[Func](#)<R, [object](#)>>

An expression specifying additional constraints to apply when loading the foreign key field.

## Returns

### T

The first related model that matches the specified predicate and constraints, or [null](#) if no match is found.

## Type Parameters

### T

The type of the target model. Must implement [IRelmModel](#) and have a parameterless constructor.

### R

The type of the related model representing the foreign key. Must implement [IRelmModel](#) and have a parameterless constructor.

### S

The type of the model used by the custom data loader. Must implement [IRelmModel](#) and have a parameterless constructor.

# Namespace CoreRelm.Interfaces

## Interfaces

### [IRelmContext](#)

Defines a context for interacting with a Relm database, providing methods for managing connections, transactions, and data operations. This interface supports querying, data manipulation, and bulk operations while ensuring proper resource management.

### [IRelmDataLoader<T>](#)

Defines methods and properties for loading, writing, and managing data, as well as executing commands with associated expressions.

### [IRelmDataSetBase](#)

Base interface for Relm data sets.

### [IRelmDataSet<T>](#)

Represents a strongly-typed, queryable, and updatable collection of data model entities, providing advanced data loading, filtering, and manipulation capabilities.

### [IRelmExecutionCommand](#)

Defines the contract for an execution command within the Relm framework, providing access to the initial command and expression, management of additional commands, and options for foreign key navigation.

### [IRelmFieldLoader](#)

Defines a contract for loading field data within a Relm context.

### [IRelmFieldLoaderBase](#)

Defines the contract for loading field data based on key values in a data source.

### [IRelmMember](#)

Defines the contract for a member entity used in apartmenting within the Relm system, including identification, contact information, and value accessors.

### [IRelmModel](#)

Defines the contract for a Relm-compatible model, providing properties and methods for managing entity state, database interactions, and data transfer object (DTO) generation.

### [IRelmModelApartment](#)

Represents an apartment entity within the Relm model, including associated user and membership information.

# Interface IRelmContext

Namespace: [CoreRelm.Interfaces](#)

Assembly: CoreRelm.dll

Defines a context for interacting with a Relm database, providing methods for managing connections, transactions, and data operations. This interface supports querying, data manipulation, and bulk operations while ensuring proper resource management.

```
public interface IRelmContext
```

## Extension Methods

[DbContextExtensions.SetLockWaitTimeout\(IRelmContext, int\)](#)

## Remarks

Implementations of this interface are designed to facilitate database operations in a structured and transactional manner. The context provides methods for starting and ending connections, managing transactions, and performing CRUD operations on data sets. It also includes support for executing raw queries and bulk operations. The interface extends [IDisposable](#) to ensure that resources are properly released when the context is no longer needed. Thread safety is not guaranteed unless explicitly stated by the implementation. Users should ensure proper synchronization when accessing the context from multiple threads.

## Properties

### ContextOptions

Gets the builder used to configure options for the current Relm context.

```
RelmContextOptionsBuilder ContextOptions { get; }
```

### Property Value

[RelmContextOptionsBuilder](#)

## Methods

# BulkTableWrite<T>(T, string, MySqlTransaction, Type, int, bool, bool, bool)

Performs a bulk write operation to insert or update data in the specified database table.

```
int BulkTableWrite<T>(T sourceData, string tableName = null, MySqlTransaction sqlTransaction = null, Type forceType = null, int batchSize = 100, bool allowAutoIncrementColumns = false, bool allowPrimaryKeyColumns = false, bool allowUniqueColumns = false)
```

## Parameters

**sourceData** T

The data to be written to the table. Must not be null. If a collection is provided, each item will be processed in the bulk operation.

**tableName** [string](#)

The name of the target database table. If null, the table name will be inferred from the type T.

**sqlTransaction** MySqlTransaction

An optional MySql.Data.MySqlClient.MySqlTransaction to associate the bulk write operation with an existing transaction. If null, the operation will execute without a transaction.

**forceType** [Type](#)

An optional [Type](#) to explicitly specify the type of the data being written. If null, the type will be inferred from T.

**batchSize** [int](#)

The maximum number of rows to include in each batch during the bulk write operation. Must be greater than zero. Defaults to 100.

**allowAutoIncrementColumns** [bool](#)

Indicates whether auto-increment columns in the target table are allowed to be explicitly written. Defaults to [false](#).

**allowPrimaryKeyColumns** [bool](#)

Indicates whether primary key columns in the target table are allowed to be explicitly written. Defaults to [false](#).

`allowUniqueColumns` [bool](#)

Indicates whether unique columns in the target table are allowed to be explicitly written. Defaults to [false](#).

Returns

[int](#)

The number of rows successfully written to the database table.

Type Parameters

[T](#)

The type of the source data. This can be a collection or a single object representing the data to be written.

Remarks

This method is optimized for high-performance bulk operations and supports optional batching to handle large datasets efficiently. Ensure that the source data aligns with the schema of the target table to avoid runtime errors.

## CommitTransaction()

Commits the current transaction, making all changes permanent in the database.

[void CommitTransaction\(\)](#)

## DoDatabaseWork(string, Dictionary<string, object>, bool, bool)

Executes a database operation using the specified query and parameters.

```
void DoDatabaseWork\(string query, Dictionary<string, object> parameters = null, bool throwException = true, bool useTransaction = false\)
```

Parameters

### query [string](#)

The SQL query to execute. Cannot be null or empty.

### parameters [Dictionary](#)<[string](#), [object](#)>

An optional dictionary of parameter names and values to be used in the query. If null, no parameters are applied.

### throwException [bool](#)

A value indicating whether an exception should be thrown if the operation fails. If [true](#), an exception is thrown on failure; otherwise, the failure is silently handled.

### useTransaction [bool](#)

A value indicating whether the operation should be executed within a transaction. If [true](#), the operation is wrapped in a transaction; otherwise, it is executed without one.

## Remarks

This method allows for executing parameterized SQL queries with optional transaction support. Use the `throwException` parameter to control error handling behavior.

## DoDatabaseWork(string, Func<MySqlCommand, object>, bool, bool)

Executes a database operation using the specified query and callback function.

```
void DoDatabaseWork(string query, Func<MySqlCommand, object> actionCallback, bool  
throwException = true, bool useTransaction = false)
```

## Parameters

### query [string](#)

The SQL query to execute. Must not be null or empty.

### actionCallback [Func](#)<MySqlCommand, [object](#)>

A callback function that processes the `MySql.Data.MySqlClient.MySqlCommand` object and returns a result. The callback must not be null.

### `throwException` [bool](#)

A value indicating whether to throw an exception if an error occurs during the operation. If [true](#), exceptions will be thrown; otherwise, errors will be suppressed.

### `useTransaction` [bool](#)

A value indicating whether the operation should be executed within a database transaction. If [true](#), the operation will be wrapped in a transaction; otherwise, it will not.

## Remarks

This method provides a flexible way to execute database operations by allowing the caller to define custom logic through the `actionCallback` parameter. Ensure that the `query` is properly sanitized to prevent SQL injection attacks.

## DoDatabaseWork<T>(string, Dictionary<string, object>, bool, bool)

Executes a database query and returns the result as the specified type.

```
T DoDatabaseWork<T>(string query, Dictionary<string, object> parameters = null, bool throwException = true, bool useTransaction = false)
```

## Parameters

### `query` [string](#)

The SQL query to execute. Cannot be null or empty.

### `parameters` [Dictionary](#)<[string](#), [object](#)>

An optional dictionary of parameter names and values to include in the query. Can be null if no parameters are required.

### `throwException` [bool](#)

Specifies whether an exception should be thrown if the query fails. If [true](#), an exception is thrown on failure; otherwise, the method returns the default value of `T`.

### `useTransaction` [bool](#)

Specifies whether the query should be executed within a transaction. If [true](#), the query is executed in a transactional context; otherwise, it is executed without a transaction.

## Returns

T

The result of the query as an instance of type T. Returns the default value of T if throwException is [false](#) and the query fails.

## Type Parameters

T

The type of the result expected from the query.

## Remarks

Use this method to execute queries that return a single result, such as scalar values or objects. Ensure that the type T matches the expected result of the query to avoid runtime errors.

## DoDatabaseWork<T>(string, Func<MySqlCommand, object>, bool, bool)

Executes a database operation using the specified query and callback function.

```
T DoDatabaseWork<T>(string query, Func<MySqlCommand, object> actionCallback, bool throwException = true, bool useTransaction = false)
```

## Parameters

query [string](#)

The SQL query to be executed. Cannot be null or empty.

actionCallback [Func](#)<MySqlCommand, [object](#)>

A callback function that defines the operation to perform with the MySql.Data.MySqlClient.MySqlCommand object. The function should return an object of type T.

throwException [bool](#)

A value indicating whether an exception should be thrown if an error occurs during the operation. The default value is [true](#).

#### **useTransaction** [bool](#)

A value indicating whether the operation should be executed within a database transaction. The default value is [false](#).

#### Returns

T

The result of the operation, as returned by the [actionCallback](#) function.

#### Type Parameters

T

The type of the result returned by the callback function.

#### Remarks

This method provides a flexible way to execute database operations by allowing the caller to define the specific action to perform using the provided  `MySql.Data.MySqlClient.MySqlCommand` object. If [useTransaction](#) is [true](#), the operation will be executed within a transaction, which will be committed or rolled back based on the success or failure of the operation.

## EndConnection(bool)

Ends the current connection and optionally commits any active transaction.

```
void EndConnection(bool commitTransaction = true)
```

#### Parameters

##### **commitTransaction** [bool](#)

A value indicating whether to commit the active transaction before ending the connection. [true](#) to commit the transaction; [false](#) to roll it back. The default is [true](#).

#### Remarks

Use this method to cleanly terminate a connection. If a transaction is active, you can specify whether to commit or roll it back before the connection is closed. Ensure that any necessary operations are completed before calling this method, as the connection will no longer be available afterward.

## FirstOrDefault<T>(Expression<Func<T, bool>>, bool)

Retrieves the first element of a sequence from the data source that satisfies the specified condition, or a default value if no such element is found.

```
T FirstOrDefault<T>(Expression<Func<T, bool>> predicate, bool loadDataLoaders = false) where  
T : IRelmModel, new()
```

### Parameters

**predicate** [Expression<Func<T, bool>>](#)

An expression that defines the condition to test each element for.

**loadDataLoaders** [bool](#)

A boolean value indicating whether to load associated data loaders for the retrieved element. If [true](#), data loaders will be initialized; otherwise, they will not be loaded.

### Returns

**T**

The first element of type **T** that satisfies the specified condition, or the default value of **T** if no such element is found.

### Type Parameters

**T**

The type of the elements in the sequence. Must implement [IRelmModel](#) and have a parameterless constructor.

## GetBulkTableWriter<T>(string, bool, bool, bool, bool, bool)

Creates and returns a [BulkTableWriter<T>](#) instance for performing bulk insert operations on a database table.

```
BulkTableWriter<T> GetBulkTableWriter<T>(string insertQuery = null, bool useTransaction =  
false, bool throwException = true, bool allowAutoIncrementColumns = false, bool  
allowPrimaryKeyColumns = false, bool allowUniqueColumns = false)
```

## Parameters

### `insertQuery` [string](#)

An optional SQL insert query to use for the bulk operation. If [null](#), a default query is generated based on the type `T`.

### `useTransaction` [bool](#)

Specifies whether the bulk operation should be performed within a transaction. If [true](#), the operation is transactional; otherwise, it is not.

### `throwException` [bool](#)

Specifies whether exceptions should be thrown if an error occurs during the bulk operation. If [true](#), exceptions are thrown; otherwise, errors are suppressed.

### `allowAutoIncrementColumns` [bool](#)

Indicates whether auto-increment columns in the database table are allowed to be included in the bulk operation. If [true](#), these columns are included; otherwise, they are excluded.

### `allowPrimaryKeyColumns` [bool](#)

Indicates whether primary key columns in the database table are allowed to be included in the bulk operation. If [true](#), these columns are included; otherwise, they are excluded.

### `allowUniqueColumns` [bool](#)

Indicates whether unique columns in the database table are allowed to be included in the bulk operation. If [true](#), these columns are included; otherwise, they are excluded.

## Returns

### [BulkTableWriter<T>](#)

A [BulkTableWriter<T>](#) instance configured for the specified bulk operation settings.

## Type Parameters

T

The type of the objects to be written to the database table. Each object represents a row in the table.

## Remarks

The [BulkTableWriter<T>](#) provides an efficient way to insert large amounts of data into a database table. Use the optional parameters to customize the behavior of the bulk operation, such as enabling transactions or allowing specific column types.

## GetDataTable<T>(string, Dictionary<string, object>, bool)

Executes the specified query and retrieves a collection of data mapped to the specified type.

```
IEnumerable<T> GetDataTable<T>(string query, Dictionary<string, object> parameters = null,  
bool throwException = true)
```

## Parameters

**query** [string](#)

The SQL query to execute. Must not be null or empty.

**parameters** [Dictionary](#)<[string](#), [object](#)>

An optional dictionary of parameter names and values to be used in the query. Can be null if no parameters are required.

**throwException** [bool](#)

A value indicating whether an exception should be thrown if the query fails. If [true](#), an exception will be thrown on failure; otherwise, the method will return an empty collection.

## Returns

[IEnumerable](#)<T>

An [IEnumerable](#)<T> containing the results of the query mapped to the specified type. Returns an empty collection if no results are found or if **throwException** is [false](#) and the query fails.

## Type Parameters

T

The type to which the query results will be mapped.

## GetDataObject<T>(string, Dictionary<string, object>, bool)

Executes the specified query and retrieves a data object of the specified type.

```
T GetDataObject<T>(string query, Dictionary<string, object> parameters = null, bool throwException = true) where T : IRelmModel, new()
```

## Parameters

**query** [string](#)

The query string used to retrieve the data object. Cannot be [null](#) or empty.

**parameters** [Dictionary](#)<[string](#), [object](#)>

An optional dictionary of parameters to be used in the query. Keys represent parameter names, and values represent their corresponding values.

**throwException** [bool](#)

A value indicating whether an exception should be thrown if the query fails or no data is found. If [true](#), an exception is thrown; otherwise, [default](#) is returned.

## Returns

T

An instance of the specified type T populated with the data retrieved by the query. Returns [default](#) if no data is found and [throwException](#) is [false](#).

## Type Parameters

T

The type of the data object to retrieve. Must implement [IRelmModel](#) and have a parameterless constructor.

## GetDataObjects<T>(string, Dictionary<string, object>?, bool)

Executes a query and retrieves a collection of data objects of the specified type.

```
IEnumerable<T> GetDataObjects<T>(string query, Dictionary<string, object>? parameters = null, bool throwException = true) where T : IRelmModel, new()
```

### Parameters

**query** [string](#)

The query string used to retrieve the data objects. Cannot be null or empty.

**parameters** [Dictionary](#)<[string](#), [object](#)>

An optional dictionary of parameter names and values to be used in the query. If null, no parameters are applied.

**throwException** [bool](#)

A boolean value indicating whether an exception should be thrown if the query fails. If [true](#), an exception is thrown on failure; otherwise, the method returns an empty collection.

### Returns

[IEnumerable](#)<T>

An [IEnumerable](#)<T> containing the data objects retrieved by the query. Returns an empty collection if no data is found or if **throwException** is [false](#) and the query fails.

### Type Parameters

T

The type of data objects to retrieve. Must implement [IRelmModel](#) and have a parameterless constructor.

## GetDataRow(string, Dictionary<string, object>, bool)

Executes the specified query and retrieves the first row of the result set as a [DataRow](#).

```
DataRow GetDataRow(string query, Dictionary<string, object> parameters = null, bool throwException = true)
```

## Parameters

### query [string](#)

The SQL query to execute. Must be a valid SQL statement.

### parameters [Dictionary](#)<[string](#), [object](#)>

An optional dictionary of parameter names and values to be used in the query. If null, no parameters are applied.

### throwException [bool](#)

A boolean value indicating whether an exception should be thrown if the query does not return any rows. If [true](#), an exception is thrown when no rows are found; otherwise, [null](#) is returned.

## Returns

### [DataRow](#)

A [DataRow](#) representing the first row of the result set, or [null](#) if no rows are found and [throwException](#) is set to [false](#).

## GetDataSet(Type)

Retrieves an initialized instance of a dataset based on the specified type.

```
IRelmDataSetBase GetDataSet(Type dataSetType)
```

## Parameters

### dataSetType [Type](#)

The [Type](#) of the dataset to retrieve. This must be a type that implements [IRelmDataSetBase](#).

## Returns

### [IRelmDataSetBase](#)

An instance of the dataset that matches the specified type. Returns [null](#) if no matching dataset is found.

## GetDataSet(Type, bool)

Retrieves an initialized instance of a dataset of the specified type.

`IRelmDataSetBase GetDataSet(Type dataSetType, bool throwException)`

### Parameters

`dataSetType Type`

The [Type](#) of the dataset to retrieve. This must implement [IRelmDataSetBase](#).

`throwException bool`

A value indicating whether an exception should be thrown if the dataset cannot be found. If [true](#), an exception is thrown when the dataset is not found; otherwise, [null](#) is returned.

### Returns

[IRelmDataSetBase](#)

An instance of [IRelmDataSetBase](#) representing the requested dataset, or [null](#) if the dataset is not found and `throwException` is [false](#).

## GetDataSetType(Type)

Retrieves an uninitialized dataset of type specified by `dataSetType`.

`IRelmDataSetBase GetDataSetType(Type dataSetType)`

### Parameters

`dataSetType Type`

The [Type](#) of the dataset to retrieve. This must implement [IRelmDataSetBase](#).

Returns

## [IRelmDataSetBase](#)

An instance of [IRelmDataSetBase](#) that corresponds to the specified `dataSetType`.

## GetDataSetType(Type, bool)

Retrieves an uninitialized dataset of type specified by `dataSetType`.

```
IRelmDataSetBase GetDataSetType(Type dataSetType, bool throwException)
```

Parameters

### `dataSetType` [Type](#)

The [Type](#) of the dataset to retrieve. Must implement [IRelmDataSetBase](#).

### `throwException` [bool](#)

A boolean value indicating whether to throw an exception if the specified `dataSetType` is invalid. [true](#) to throw an exception; otherwise, [false](#).

Returns

## [IRelmDataSetBase](#)

An instance of [IRelmDataSetBase](#) corresponding to the specified `dataSetType`. Returns [null](#) if `throwException` is [false](#) and the dataset type is invalid.

## GetDataSetType<T>()

Retrieves an uninitialized dataset of the specified type.

```
IRelmDataSet<T> GetDataSetType<T>() where T : IRelmModel, new()
```

Returns

## [IRelmDataSet](#)<T>

An instance of [IRelmDataSet<T>](#) representing the dataset of the specified type.

## Type Parameters

T

The type of the dataset to retrieve. Must implement [IRelmModel](#) and have a parameterless constructor.

## Remarks

This method is generic and allows retrieval of datasets for any type that satisfies the constraints.

## GetDataSetType<T>(bool)

Retrieves an uninitialized dataset of the specified type.

```
IRelmDataSet<T> GetDataSetType<T>(bool throwException) where T : IRelmModel, new()
```

## Parameters

**throwException** [bool](#)

A value indicating whether an exception should be thrown if the dataset cannot be retrieved. If [true](#), an exception is thrown; otherwise, [null](#) may be returned.

## Returns

[IRelmDataSet<T>](#)

An instance of [IRelmDataSet<T>](#) representing the dataset of the specified type, or [null](#) if the dataset cannot be retrieved and **throwException** is [false](#).

## Type Parameters

T

The type of the dataset to retrieve. Must implement [IRelmModel](#) and have a parameterless constructor.

## GetDataSet<T>()

Retrieves an initialized instance of a dataset of the specified type.

```
IRelmDataSet<T> GetDataSet<T>() where T : IRelmModel, new()
```

Returns

[IRelmDataSet<T>](#)

An instance of [IRelmDataSet<T>](#) containing the data for the specified type.

Type Parameters

T

The type of the dataset to retrieve. Must implement [IRelmModel](#) and have a parameterless constructor.

Remarks

Use this method to access a dataset for a specific model type. The type parameter T must represent a model that conforms to the [IRelmModel](#) interface.

## GetDataSet<T>(bool)

Retrieves an initialized instance of a dataset of the specified type.

```
IRelmDataSet<T> GetDataSet<T>(bool throwException) where T : IRelmModel, new()
```

Parameters

throwException [bool](#)

A value indicating whether an exception should be thrown if the dataset cannot be retrieved. If [true](#), an exception is thrown on failure; otherwise, [null](#) is returned.

Returns

[IRelmDataSet<T>](#)

An instance of [IRelmDataSet<T>](#) containing the dataset of type `T`. Returns [null](#) if the dataset cannot be retrieved and `throwException` is [false](#).

## Type Parameters

`T`

The type of the dataset to retrieve. Must implement [IRelmModel](#) and have a parameterless constructor.

## GetDataTable(string, Dictionary<string, object>, bool)

Executes the specified SQL query and returns the results as a [DataTable](#).

```
DataTable GetDataTable(string query, Dictionary<string, object> parameters = null, bool throwException = true)
```

## Parameters

`query` [string](#)

The SQL query to execute. This must be a valid SQL statement.

`parameters` [Dictionary](#)<[string](#), [object](#)>

An optional dictionary of parameter names and values to be used in the query. If provided, the parameters will be added to the query to prevent SQL injection. Defaults to [null](#).

`throwException` [bool](#)

A value indicating whether an exception should be thrown if an error occurs during query execution. If [true](#), exceptions will be propagated to the caller. If [false](#), the method will suppress exceptions and return [null](#) in case of an error. Defaults to [true](#).

## Returns

[DataTable](#)

A [DataTable](#) containing the results of the query. Returns [null](#) if `throwException` is [false](#) and an error occurs during execution.

## Remarks

This method is designed to execute read-only queries, such as SELECT statements. It is not intended for executing queries that modify data (e.g., INSERT, UPDATE, DELETE).

## GetIdFromInternalId(string, string)

Converts an internal identifier to its corresponding row identifier for a specified table.

```
string GetIdFromInternalId(string table, string InternalId)
```

### Parameters

**table** [string](#)

The name of the table containing the internal identifier. Cannot be null or empty.

**InternalId** [string](#)

The internal identifier to be converted. Cannot be null or empty.

### Returns

[string](#)

The row identifier corresponding to the specified internal identifier and table.

## GetLastInsertId()

Retrieves the row identifier of the most recently inserted record in the database.

```
string GetLastInsertId()
```

### Returns

[string](#)

The identifier of the last inserted record as a string. The format and value depend on the database implementation.

## Remarks

This method is typically used after an insert operation to obtain the unique identifier generated for the new record. Ensure that the database connection and context are properly configured before calling this method.

## GetScalar<T>(string, Dictionary<string, object>, bool)

Executes the specified SQL query and retrieves a single scalar value of the specified type.

```
T GetScalar<T>(string query, Dictionary<string, object> parameters = null, bool throwException = true)
```

## Parameters

**query** [string](#)

The SQL query to execute. Must not be null or empty.

**parameters** [Dictionary](#)<[string](#), [object](#)>

An optional dictionary of parameter names and values to include in the query. If null, no parameters are added.

**throwException** [bool](#)

A boolean value indicating whether an exception should be thrown if the query fails. If [true](#), an exception is thrown on failure; otherwise, the default value of **T** is returned.

## Returns

**T**

The scalar value of type **T** returned by the query. If the query does not return a result and **throwException** is [false](#), the default value of **T** is returned.

## Type Parameters

**T**

The type of the scalar value to return.

## Remarks

Use this method to retrieve a single value, such as a count or aggregate result, from a database query. Ensure that the query is written to return only one value; otherwise, an exception may occur.

## Get<T>(bool)

Retrieves a collection of entities of type `T` from the data source.

```
ICollection<T> Get<T>(bool loadDataLoaders = false) where T : IRelmModel, new()
```

### Parameters

`loadDataLoaders` [bool](#)

A boolean value indicating whether to load associated data loaders. If [true](#), data loaders will be initialized; otherwise, they will not.

### Returns

[`ICollection`](#)<`T`>

A collection of objects of type `T`. The collection may be empty if no objects are found.

### Type Parameters

`T`

The type of objects to retrieve. Must implement [IRelmModel](#) and have a parameterless constructor.

## Get<T>(Expression<Func<T, bool>>, bool)

Retrieves a collection of entities of type `T` from the data source that satisfy the specified predicate.

```
ICollection<T> Get<T>(Expression<Func<T, bool>> predicate, bool loadDataLoaders = false)
where T : IRelmModel, new()
```

### Parameters

**predicate** [Expression](#)<[Func](#)<T, [bool](#)>>

An expression that defines the conditions the entities must satisfy.

**loadDataLoaders** [bool](#)

A boolean value indicating whether to load associated data loaders for the retrieved entities. [true](#) to load data loaders; otherwise, [false](#).

Returns

[ICollection](#)<T>

A collection of entities of type T that match the specified predicate. Returns an empty collection if no entities match.

Type Parameters

T

The type of the entities to retrieve. Must implement [IRelmModel](#) and have a parameterless constructor.

## HasDataSet(Type, bool)

Determines whether a dataset of the specified type is available.

```
bool HasDataSet(Type dataSetType, bool throwException = true)
```

Parameters

**dataSetType** [Type](#)

The [Type](#) of the dataset to check for availability. This parameter cannot be [null](#).

**throwException** [bool](#)

A value indicating whether an exception should be thrown if the dataset is not available. If [true](#), an exception is thrown when the dataset is not found; otherwise, the method returns [false](#).

Returns

[bool](#)

[true](#) if the dataset of the specified type is available; otherwise, [false](#).

## HasDataSet<T>(bool)

Determines whether a dataset of the specified type exists in the current context.

```
bool HasDataSet<T>(bool throwException = true) where T : IRelmModel, new()
```

### Parameters

[throwException](#) [bool](#)

A value indicating whether to throw an exception if the dataset does not exist. [true](#) to throw an exception; [false](#) to return [false](#) instead.

### Returns

[bool](#)

[true](#) if the dataset of the specified type exists; otherwise, [false](#).

### Type Parameters

[T](#)

The type of the dataset to check for. Must implement [IRelmModel](#) and have a parameterless constructor.

## RollbackTransaction()

Rolls back the current transaction, undoing any changes made since the transaction began.

```
void RollbackTransaction()
```

### Remarks

This method should be called to revert changes if an error occurs or if the transaction cannot be completed successfully. Ensure that a transaction is active before calling this method; otherwise, an exception may be thrown.

## RollbackTransactions()

Rolls back the current transaction, undoing any changes made since the transaction began.

```
void RollbackTransactions()
```

## Remarks

This method should be called to revert changes if an error occurs or if the transaction cannot be completed successfully. Ensure that a transaction is active before calling this method; otherwise, an exception may be thrown.

## Run<T>(string, Dictionary<string, object>)

Executes the specified query and returns a collection of results mapped to the specified type.

```
ICollection<T> Run<T>(string query, Dictionary<string, object> parameters = null) where T : IRelmModel, new()
```

## Parameters

**query** [string](#)

The query string to execute. This string must be a valid query in the context of the underlying data source.

**parameters** [Dictionary](#)<[string](#), [object](#)>

An optional dictionary of parameter names and values to be used in the query. If null, no parameters are applied.

## Returns

[ICollection](#)<T>

A collection of objects of type `T` representing the query results. The collection will be empty if no results are found.

## Type Parameters

`T`

The type of the objects in the result collection. Must implement [IRelmModel](#) and have a parameterless constructor.

## SetDataLoader<T>(IRelmDataLoader<T>)

Configures the data loader for the specified model type.

```
void SetDataLoader<T>(IRelmDataLoader<T> dataLoader) where T : RelmModel, new()
```

## Parameters

`dataLoader` [IRelmDataLoader](#)<T>

The data loader instance to associate with the specified model type. Cannot be [null](#).

## Type Parameters

`T`

The type of the model that the data loader will handle. Must inherit from [RelmModel](#) and have a parameterless constructor.

## StartConnection(bool, int)

Starts a connection to the database.

```
void StartConnection(bool autoOpenTransaction = false, int lockWaitTimeoutSeconds = 0)
```

## Parameters

`autoOpenTransaction` [bool](#)

Specifies whether a transaction should be automatically opened after the connection is established.

Pass [true](#) to open a transaction automatically; otherwise, [false](#).

### lockWaitTimeoutSeconds [int](#)

The lock wait timeout in seconds. A value of 0 indicates the default timeout for the database. Specify a positive integer to set a custom timeout duration.

## Remarks

If [autoOpenTransaction](#) is set to [true](#), ensure that the transaction is committed or rolled back to avoid leaving it open. This method must be called before performing any database operations.

## Where<T>(Expression<Func<T, bool>>)

Retrieves data from the data source, filtered based on the specified predicate.

```
IRelmDataSet<T> Where<T>(Expression<Func<T, bool>> predicate) where T : IRelmModel, new()
```

## Parameters

### [predicate](#) [Expression](#)<[Func](#)<T, [bool](#)>>

An expression that defines the conditions for filtering the dataset.

## Returns

### [IRelmDataSet](#)<T>

A new dataset containing only the elements that satisfy the specified predicate.

## Type Parameters

### T

The type of the model in the dataset. Must implement [IRelmModel](#) and have a parameterless constructor.

## Remarks

The predicate is evaluated for each element in the dataset, and only those elements for which the predicate returns [true](#) are included in the result.

# WriteToDatabase(IRelmModel, int, bool, bool, bool, bool)

Writes the specified [IRelmModel](#) to the database in batches.

```
int WriteToDatabase(IRelmModel relmModel, int batchSize = 100, bool  
allowAutoIncrementColumns = false, bool allowPrimaryKeyColumns = false, bool  
allowUniqueColumns = false, bool allowAutoDateColumns = false)
```

## Parameters

**relmModel** [IRelmModel](#)

The model containing the data to be written to the database. Cannot be [null](#).

**batchSize** [int](#)

The number of records to write in each batch. Must be greater than 0. The default value is 100.

**allowAutoIncrementColumns** [bool](#)

A value indicating whether columns with auto-increment constraints are allowed to be written. If [true](#), auto-increment columns will be included in the write operation; otherwise, they will be excluded.

**allowPrimaryKeyColumns** [bool](#)

A value indicating whether primary key columns are allowed to be written. If [true](#), primary key columns will be included in the write operation; otherwise, they will be excluded.

**allowUniqueColumns** [bool](#)

A value indicating whether columns with unique constraints are allowed to be written. If [true](#), unique columns will be included in the write operation; otherwise, they will be excluded.

**allowAutoDateColumns** [bool](#)

A value indicating whether columns with automatic date generation constraints are allowed to be written. If [true](#), auto-date columns will be included in the write operation; otherwise, they will be excluded.

## Returns

[int](#)

The total number of records successfully written to the database.

## Remarks

This method writes data to the database in batches to optimize performance. The behavior of the write operation can be customized using the boolean parameters to control the inclusion of specific column types.

## WriteToDatabase(IEnumerable<IRelmModel>, int, bool, bool, bool, bool)

Writes a collection of Relm models to the database in batches.

```
int WriteToDatabase(IEnumerable<IRelmModel> relmModels, int batchSize = 100, bool allowAutoIncrementColumns = false, bool allowPrimaryKeyColumns = false, bool allowUniqueColumns = false, bool allowAutoDateColumns = false)
```

## Parameters

`relmModels` [IEnumerable<IRelmModel>](#)

The collection of models to be written to the database. Each model must implement the [IRelmModel](#) interface.

`batchSize` [int](#)

The number of models to include in each batch. Must be a positive integer. The default value is 100.

`allowAutoIncrementColumns` [bool](#)

A value indicating whether columns with auto-increment constraints are allowed to be written. If [true](#), auto-increment columns will be included; otherwise, they will be excluded. The default value is [false](#).

`allowPrimaryKeyColumns` [bool](#)

A value indicating whether primary key columns are allowed to be written. If [true](#), primary key columns will be included; otherwise, they will be excluded. The default value is [false](#).

`allowUniqueColumns` [bool](#)

A value indicating whether columns with unique constraints are allowed to be written. If [true](#), unique columns will be included; otherwise, they will be excluded. The default value is [false](#).

`allowAutoDateColumns` [bool](#)

A value indicating whether columns with automatic date generation constraints are allowed to be written. If [true](#), auto-date columns will be included; otherwise, they will be excluded. The default value is [false](#).

## Returns

[int](#)

The total number of models successfully written to the database.

## Remarks

This method processes the provided models in batches to optimize database writes. The behavior of the write operation can be customized using the optional parameters to control which types of columns are included.

# Interface IRelmDataLoader<T>

Namespace: [CoreRelm.Interfaces](#)

Assembly: CoreRelm.dll

Defines methods and properties for loading, writing, and managing data, as well as executing commands with associated expressions.

```
public interface IRelmDataLoader<T>
```

## Type Parameters

T

The type of data managed by the loader.

## Properties

### LastCommandsExecuted

Gets or sets the collection of the most recently executed commands and their associated execution details.

```
Dictionary<Commands.Command, List<IRelmExecutionCommand>> LastCommandsExecuted { get; set; }
```

## Property Value

[Dictionary](#)<[Commands.Command](#), [List](#)<[IRelmExecutionCommand](#)>>

## Remarks

This property provides a record of the last executed commands along with their execution details. It can be used to analyze or inspect the history of command executions.

## Methods

## AddExpression(Command, Expression)

Adds an expression to the specified command and returns the resulting execution command.

```
IReImExecutionCommand AddExpression(Command command, Expression expression)
```

### Parameters

**command** [Commands.Command](#)

The command to which the expression will be added. Cannot be null.

**expression** [Expression](#)

The expression to add to the command. Cannot be null.

### Returns

[IReImExecutionCommand](#)

An [IReImExecutionCommand](#) representing the updated execution command with the added expression.

## AddSingleExpression(Command, Expression)

Adds a single expression to the specified command and returns the resulting execution command.

```
IReImExecutionCommand AddSingleExpression(Command command, Expression expression)
```

### Parameters

**command** [Commands.Command](#)

The command to which the expression will be added. Cannot be [null](#).

**expression** [Expression](#)

The expression to add to the command. Cannot be [null](#).

### Returns

## [IRelmExecutionCommand](#)

An [IRelmExecutionCommand](#) representing the updated command with the added expression.

## GetLoadData()

Retrieves a collection of data items of type `T`.

`ICollection<T> GetLoadData()`

Returns

[ICollection](#) <T>

A collection of data items of type `T`. The collection will be empty if no data is available.

Remarks

The returned collection contains the data items that are to be loaded. The collection may be empty if no data is available.

## HasUnderscoreProperty(string)

Determines whether the specified property key contains an underscore.

`bool HasUnderscoreProperty(string propertyKey)`

Parameters

`propertyKey` [string](#)

The property key to check. Cannot be null or empty.

Returns

[bool](#)

[true](#) if the property key contains at least one underscore; otherwise, [false](#).

## WriteData()

Writes data to the underlying storage and returns the number of rows written.

```
int WriteData()
```

Returns

[int↗](#)

The number of rows successfully written to the storage.

# Interface IRelmDataSetBase

Namespace: [CoreRelm.Interfaces](#)

Assembly: CoreRelm.dll

Base interface for Relm data sets.

```
public interface IRelmDataSetBase
```

# Interface IRelmDataSet<T>

Namespace: [CoreRelm.Interfaces](#)

Assembly: CoreRelm.dll

Represents a strongly-typed, queryable, and updatable collection of data model entities, providing advanced data loading, filtering, and manipulation capabilities.

```
public interface IRelmDataSet<T> : ICollection<T>, IEnumerable<T>, IEnumerable,
IRelmDataSetBase where T : IRelmModel, new()
```

## Type Parameters

T

The type of data model entities contained in the data set. Must implement IRelmModel and have a parameterless constructor.

## Inherited Members

[ICollection<T>.Clear\(\)](#) , [ICollection<T>.Contains\(T\)](#) , [ICollection<T>.CopyTo\(T\[\], int\)](#) ,  
[ICollection<T>.Remove\(T\)](#) , [ICollection<T>.IsReadOnly](#) , [IEnumerable<T>.GetEnumerator\(\)](#)

## Extension Methods

[ListExtensions.LoadDataLoaderField<T, R>\(ICollection<T>, IRelmContext, Expression<Func<T, R>>\)](#) ,  
[ListExtensions.LoadDataLoaderField<T, R>\(ICollection<T>, IRelmQuickContext, Expression<Func<T, R>>\)](#) ,  
[ListExtensions.LoadDataLoaderField<T, R>\(ICollection<T>, RelmContextOptionsBuilder, Expression<Func<T, R>>\)](#) ,  
[ListExtensions.LoadForeignKeyField<T, R>\(ICollection<T>, IRelmContext, Expression<Func<T, ICollection<R>>>\)](#) ,  
[ListExtensions.LoadForeignKeyField<T, R>\(ICollection<T>, IRelmContext, Expression<Func<T, ICollection<R>>>, Expression<Func<R, object>>\)](#) ,  
[ListExtensions.LoadForeignKeyField<T, R>\(ICollection<T>, IRelmContext, Expression<Func<T, R>>\)](#) ,  
[ListExtensions.LoadForeignKeyField<T, R>\(ICollection<T>, IRelmContext, Expression<Func<T, R>>, Expression<Func<R, object>>\)](#) ,  
[ListExtensions.LoadForeignKeyField<T, R>\(ICollection<T>, IRelmQuickContext, Expression<Func<T, ICollection<R>>>\)](#) ,  
[ListExtensions.LoadForeignKeyField<T, R>\(ICollection<T>, IRelmQuickContext, Expression<Func<T, ICollection<R>>>, Expression<Func<R, object>>\)](#) ,

[ListExtensions.LoadForeignKeyField<T, R>\(ICollection<T>, IRelmQuickContext, Expression<Func<T, R>>\).](#)  
,

[ListExtensions.LoadForeignKeyField<T, R>\(ICollection<T>, IRelmQuickContext, Expression<Func<T, R>>, Expression<Func<R, object>>\),](#)

[ListExtensions.LoadForeignKeyField<T, R>\(ICollection<T>, RelmContextOptionsBuilder, Expression<Func<T, ICollection<R>>>\),](#)

[ListExtensions.LoadForeignKeyField<T, R>\(ICollection<T>, RelmContextOptionsBuilder, Expression<Func<T, ICollection<R>>>, Expression<Func<R, object>>\),](#)

[ListExtensions.LoadForeignKeyField<T, R>\(ICollection<T>, RelmContextOptionsBuilder, Expression<Func<T, R>>\),](#)

[ListExtensions.LoadForeignKeyField<T, R>\(ICollection<T>, RelmContextOptionsBuilder, Expression<Func<T, R>>, Expression<Func<R, object>>\),](#)

[ListExtensions.LoadForeignKeyField<T, R, S>\(ICollection<T>, IRelmContext, Expression<Func<T, ICollection<R>>>, IRelmDataLoader<S>\),](#)

[ListExtensions.LoadForeignKeyField<T, R, S>\(ICollection<T>, IRelmContext, Expression<Func<T, ICollection<R>>>, IRelmDataLoader<S>, Expression<Func<R, object>>\),](#)

[ListExtensions.LoadForeignKeyField<T, R, S>\(ICollection<T>, IRelmContext, Expression<Func<T, R>>, IRelmDataLoader<S>\),](#)

[ListExtensions.LoadForeignKeyField<T, R, S>\(ICollection<T>, IRelmContext, Expression<Func<T, R>>, IRelmDataLoader<S>, Expression<Func<R, object>>\),](#)

[ListExtensions.LoadForeignKeyField<T, R, S>\(ICollection<T>, IRelmQuickContext, Expression<Func<T, ICollection<R>>>, IRelmDataLoader<S>\),](#)

[ListExtensions.LoadForeignKeyField<T, R, S>\(ICollection<T>, IRelmQuickContext, Expression<Func<T, R>>, IRelmDataLoader<S>, Expression<Func<R, object>>\),](#)

[ListExtensions.LoadForeignKeyField<T, R, S>\(ICollection<T>, RelmContextOptionsBuilder, Expression<Func<T, ICollection<R>>>, IRelmDataLoader<S>\),](#)

[ListExtensions.LoadForeignKeyField<T, R, S>\(ICollection<T>, RelmContextOptionsBuilder, Expression<Func<T, R>>, IRelmDataLoader<S>, Expression<Func<R, object>>\),](#)

[ListExtensions.FlattenTreeObject<T>\(IEnumerable<T>, Func<T, ICollection<T>>\),](#)

[ListExtensions.GenerateDTO<T>\(IEnumerable<T>, ICollection<string>, ICollection<string>, string, Func<IRelmModel, Dictionary<string, object>>\),](#)

```
ListExtensions.WriteToDatabase<T>(IEnumerable<T>, IRelmContext, string, Type, int, string, bool, bool, bool, bool),  
ListExtensions.WriteToDatabase<T>(IEnumerable<T>, IRelmQuickContext, string, Type, int, string, bool, bool, bool, bool),  
ListExtensions.WriteToDatabase<T>(IEnumerable<T>, MySqlConnection, MySqlTransaction, string, Type, int, string, bool, bool, bool, bool),  
ListExtensions.WriteToDatabase<T>(IEnumerable<T>, Enum, string, Type, bool, int, string, bool, bool, bool)
```

## Remarks

The [IRelmDataSet<T>](#) interface extends standard collection functionality with methods for querying, loading, and persisting data entities. It supports fluent query composition, including filtering, ordering, grouping, and referencing related entities. Implementations may provide deferred execution and optimized data access patterns. This interface is typically used in data access layers to manage and interact with sets of domain models.

## Methods

### Add(ICollection<T>)

Adds the elements of the specified collection to the current collection.

```
int Add(ICollection<T> items)
```

#### Parameters

**items** [ICollection<T>](#)

The collection of items to add. Cannot be null. All elements in the collection will be added in enumeration order.

#### Returns

[int](#)

The number of items successfully added to the collection.

### Add(ICollection<T>, bool)

Adds the specified collection of items to the underlying store, optionally persisting the changes.

```
int Add(ICollection<T> items, bool persist)
```

## Parameters

**items** [ICollection](#)<T>

The collection of items to add. Cannot be null or contain null elements.

**persist** [bool](#)

A value indicating whether the changes should be persisted immediately. If [true](#), the items are saved to the store; otherwise, the changes remain in memory until persisted.

## Returns

[int](#)

The number of items successfully added to the store.

## Add(T)

Adds the specified item to the data set and returns its identifier.

```
int Add(T item)
```

## Parameters

**item** T

The item to add to the data set.

## Returns

[int](#)

The identifier of the added item.

## Add(T, bool)

Adds the specified item to the collection and optionally persists the change.

```
int Add(T item, bool persist)
```

### Parameters

**item** T

The item to add to the collection. Cannot be null.

**persist** bool

A value indicating whether the addition should be persisted. If [true](#), the change is saved; otherwise, it is not.

### Returns

[int](#)

The zero-based index at which the item was added.

## Count()

Sets a "COUNT(\*)" return column on the data set.

```
IRelmDataSet<T> Count()
```

### Returns

[IRelmDataSet](#)<T>

The current data set with the "count" operation added.

## Count(Expression<Func<T, bool>>)

Sets a "COUNT" return column on the data set with a specified predicate.

```
IRelmDataSet<T> Count(Expression<Func<T, bool>> predicate)
```

## Parameters

**predicate** [Expression<Func<T, bool>>](#)

An expression that defines the criteria for counting items.

## Returns

[IRelmDataSet<T>](#)

The current data set with the "count" operation added.

## DistinctBy(Expression<Func<T, object>>)

Sets a "DISTINCT" return column on the data set with a specified key selector.

```
IRelmDataSet<T> DistinctBy(Expression<Func<T, object>> predicate)
```

## Parameters

**predicate** [Expression<Func<T, object>>](#)

An expression that defines the key by which to filter distinct items.

## Returns

[IRelmDataSet<T>](#)

The current data set with the "distinct" operation added.

## Entry(T)

Creates an entry in the data set for the specified item.

```
IRelmDataSet<T> Entry(T item)
```

## Parameters

**item** T

The item to create an entry for.

## Returns

[IRelmDataSet<T>](#)

An [IRelmDataSet<T>](#) holding the created entry.

## Entry(T, bool)

Creates an entry in the data set for the specified item.

`IRelmDataSet<T> Entry(T item, bool persist = true)`

## Parameters

**item** T

The item to create an entry for.

**persist** [bool](#)

A value indicating whether the entry should be persisted. If [true](#), the entry is saved to the underlying data store; otherwise, it is not persisted.

## Returns

[IRelmDataSet<T>](#)

An [IRelmDataSet<T>](#) holding the created entry.

## Find(int)

Retrieves an item of type T with the specified identifier.

`T Find(int itemId)`

## Parameters

**itemId** [int](#)

The unique identifier of the item to locate.

## Returns

T

The item of type T that matches the specified identifier, or the default value for type T if no matching item is found.

## Find(string)

Retrieves an item of type T that matches the specified internal identifier.

**T** **Find**([string](#) **itemInternalId**)

## Parameters

**itemInternalId** [string](#)

The unique internal identifier of the item to locate. Cannot be null or empty.

## Returns

T

The item of type T that matches the specified internal identifier, or null if no matching item is found.

## FirstOrDefault()

Returns the first element of a sequence, or a default value if the sequence contains no elements.

**T** **FirstOrDefault**()

## Returns

T

The first element in the sequence, or the default value for type T if the sequence is empty.

## FirstOrDefault(bool)

Returns the first element of the sequence, or the default value for type T if the sequence contains no elements.

T **FirstOrDefault**(bool loadItems)

### Parameters

**loadItems** [bool](#)

true to load items before retrieving the first element; otherwise, false to use the current state of the sequence.

### Returns

T

The first element of the sequence if it exists; otherwise, the default value for type T.

## FirstOrDefault(Expression<Func<T, bool>>)

Returns the first element that satisfies the specified condition, or the default value for the type if no such element is found.

T **FirstOrDefault**(Expression<Func<T, bool>> predicate)

### Parameters

**predicate** [Expression](#)<[Func](#)<T, [bool](#)>>

An expression that defines the condition to test each element for. Cannot be null.

### Returns

T

The first element that matches the specified predicate; or the default value for type T if no such element is found.

## FirstOrDefault(Expression<Func<T, bool>>, bool)

Returns the first element that matches the specified predicate, or the default value for type T if no such element is found.

```
T FirstOrDefault(Expression<Func<T, bool>> predicate, bool loadItems)
```

### Parameters

**predicate** [Expression<Func<T, bool>>](#)

An expression that defines the condition to test each element for. Cannot be null.

**loadItems** [bool](#)

true to load items before retrieving the first element; otherwise, false to use the current state of the sequence.

### Returns

T

The first element that matches the predicate, or the default value for type T if no matching element is found.

## GroupBy(Expression<Func<T, object>>)

Sets the "group by" clause for the data set based on the specified key selector.

```
IRelmDataSet<T> GroupBy(Expression<Func<T, object>> predicate)
```

### Parameters

**predicate** [Expression<Func<T, object>>](#)

An expression that defines the key by which to group the elements.

Returns

### [IRelmDataSet<T>](#)

The current data set with the "group by" clause added.

## Limit(int)

Sets the "limit" clause for the data set, restricting the number of items returned.

`IRelmDataSet<T> Limit(int limitCount)`

Parameters

### [limitCount int](#)

The maximum number of items to include in the result set. Must be greater than zero.

Returns

### [IRelmDataSet<T>](#)

The current data set with the "limit" clause added.

## Load()

Loads the items in the data set.

`ICollection<T> Load()`

Returns

### [ICollection<T>](#)

A collection of all items in the data set.

## Load(bool)

Loads a collection of items of type T, optionally including data loaders based on the specified flag.

`ICollection<T> Load(bool loadDataLoaders)`

### Parameters

`loadDataLoaders bool`

true to include data loaders in the returned collection; otherwise, false to exclude them.

### Returns

`ICollection<T>`

A collection of items of type T. The collection may include data loaders if loadDataLoaders is set to true.

## LoadAsDataSet()

Loads the current data as an `IRelmDataSet<T>` instance for querying and manipulation.

`IRelmDataSet<T> LoadAsDataSet()`

### Returns

`IRelmDataSet<T>`

An `IRelmDataSet<T>` containing the loaded data. The returned data set reflects the current state of the underlying source at the time of the call.

## New(bool)

Creates a new instance of type T, optionally persisting it.

`T New(bool persist = true)`

## Parameters

### `persist bool` ↗

true to persist the new instance; otherwise, false. The default is true.

## Returns

T

A new instance of type T. The instance may be persisted depending on the value of the persist parameter.

## New(dynamic, bool)

Creates a new instance of type T using the specified parameters.

```
T New(dynamic newObjectParameters, bool persist = true)
```

## Parameters

### `newObjectParameters dynamic`

An object containing the parameters required to initialize the new instance. The structure and required properties depend on the type T.

### `persist bool` ↗

true to persist the new instance immediately; otherwise, false. The default is true.

## Returns

T

A new instance of type T initialized with the provided parameters.

## Remarks

If persist is set to false, the new instance will not be saved or committed to any underlying data store. The caller is responsible for ensuring that newObjectParameters contains all necessary information for constructing a valid instance of T.

## Offset(int)

Sets the "offset" clause for the data set, specifying the number of items to skip before starting to return items.

`IRelmDataSet<T> Offset(int offsetCount)`

### Parameters

`offsetCount int`

The number of items to skip. Must be greater than or equal to zero.

### Returns

[IRelmDataSet](#)<T>

The current data set with the "offset" clause added.

## OrderBy(Expression<Func<T, object>>)

Sorts the elements of the data set in ascending order according to a specified key.

`IRelmDataSet<T> OrderBy(Expression<Func<T, object>> predicate)`

### Parameters

`predicate Expression<Func<T, object>>`

An expression that specifies the key by which to order the elements. The expression should select a property or value from each element to use as the sort key.

### Returns

[IRelmDataSet](#)<T>

A new data set with the elements ordered in ascending order according to the specified key.

### Remarks

The ordering is stable; elements with equal keys maintain their original relative order. To perform a descending sort, use the corresponding `OrderByDescending` method.

## `OrderByDescending(Expression<Func<T, object>>)`

Sorts the elements of the data set in descending order according to a specified key.

```
IRelmDataSet<T> OrderByDescending(Expression<Func<T, object>> predicate)
```

### Parameters

`predicate Expression<Func<T, object>>`

An expression that selects the key to order the elements by. The key is extracted from each element in the data set.

### Returns

`IRelmDataSet<T>`

A new data set with the elements sorted in descending order based on the specified key.

## `Reference<S>(Expression<Func<T, ICollection<S>>, ICollection<Expression<Func<S, object>>)`

Creates a reference to a related collection of entities based on the specified navigation property and applies additional constraints to the referenced entities.

```
IRelmDataSet<T> Reference<S>(Expression<Func<T, ICollection<S>>> predicate,  
ICollection<Expression<Func<S, object>>> additionalConstraints)
```

### Parameters

`predicate Expression<Func<T, ICollection<S>>>`

An expression that specifies the navigation property representing the collection of related entities to reference.

`additionalConstraints ICollection<Expression<Func<S, object>>>`

A collection of expressions that define additional constraints to apply to the referenced entities. Each expression specifies a property or condition to filter or shape the referenced collection.

## Returns

### [IRelmDataSet<T>](#)

An [IRelmDataSet<T>](#) that represents the current data set with the specified reference and constraints applied.

## Type Parameters

### S

The type of the related entities in the referenced collection.

## Remarks

Use this method to include and constrain related collections when querying or manipulating the data set. This is useful for eager loading or applying filters to related entities in complex queries.

## Reference<S>(Expression<Func<T, ICollection<S>>, Expression<Func<S, object>>)

Creates a reference to a related collection of entities, applying additional constraints to the referenced items.

```
IRelmDataSet<T> Reference<S>(Expression<Func<T, ICollection<S>>> predicate,  
Expression<Func<S, object>>> additionalConstraints)
```

## Parameters

### `predicate` [Expression<Func<T, ICollection<S>>>](#)

An expression that specifies the collection navigation property on the source entity to reference.

### `additionalConstraints` [Expression<Func<S, object>>>](#)

An expression that defines additional constraints or filters to apply to the referenced entities.

## Returns

## [IRelmDataSet<T>](#)

An [IRelmDataSet<T>](#) representing the referenced collection with the specified constraints applied.

### Type Parameters

S

The type of the entities in the referenced collection.

### Remarks

Use this method to navigate and filter related collections in a type-safe manner. The returned data set reflects the specified navigation and constraints, enabling further query composition.

## Reference<S>(Expression<Func<T, S>>)

Creates a reference to a related data set based on the specified property expression.

`IRelmDataSet<T> Reference<S>(Expression<Func<T, S>> predicate)`

### Parameters

`predicate Expression<Func<T, S>>`

An expression that specifies the property of type T to use as the reference. Typically a lambda expression selecting a navigation property.

### Returns

## [IRelmDataSet<T>](#)

An [IRelmDataSet<T>](#) representing the related data set identified by the specified property expression.

### Type Parameters

S

The type of the property referenced by the expression.

### Remarks

Use this method to navigate relationships between data sets, such as foreign key associations. The returned data set allows further querying or operations on the related entities.

## Reference<S>(Expression<Func<T, S>>, ICollection<Expression<Func<S, object>>>)

Creates a reference to a related data set based on the specified navigation property and applies additional constraints to the referenced entities.

```
IRelmDataSet<T> Reference<S>(Expression<Func<T, S>> predicate,  
ICollection<Expression<Func<S, object>>> additionalConstraints)
```

### Parameters

**predicate** [Expression<Func<T, S>>](#)

An expression that specifies the navigation property from the current entity to the related entity.

**additionalConstraints** [ICollection<Expression<Func<S, object>>>](#)

A collection of expressions that define additional constraints to apply to the referenced entities. Each expression specifies a property of the related entity and a constraint to filter the results.

### Returns

[IRelmDataSet<T>](#)

An [IRelmDataSet<T>](#) representing the referenced data set with the applied constraints.

### Type Parameters

**S**

The type of the related entity to reference.

### Remarks

Use this method to navigate relationships between entities and to further filter the referenced data set using additional property constraints. This is useful for scenarios where related data must be accessed with specific filtering criteria.

## Reference<S>(Expression<Func<T, S>>, Expression<Func<S, object>>)

Creates a reference to a related data set based on the specified navigation property and applies additional constraints to the referenced set.

```
IRelmDataSet<T> Reference<S>(Expression<Func<T, S>> predicate, Expression<Func<S, object>> additionalConstraints)
```

### Parameters

**predicate** [Expression<Func<T, S>>](#)

An expression that specifies the navigation property used to reference the related entity.

**additionalConstraints** [Expression<Func<S, object>>](#)

An expression that defines additional constraints to apply to the referenced data set.

### Returns

[IRelmDataSet<T>](#)

An [IRelmDataSet<T>](#) representing the related data set with the specified constraints applied.

### Type Parameters

**S**

The type of the related entity to reference.

### Remarks

Use this method to navigate relationships between entities and further filter the referenced data set. This is useful for querying related data with custom constraints in a type-safe manner.

## Save()

Saves the current changes to the underlying data store.

```
int Save()
```

Returns

[int ↗](#)

The number of objects that were saved to the data store. Returns 0 if no changes were detected.

## Save(T)

Saves the specified item to the underlying data store.

[int Save\(T item\)](#)

Parameters

[item T](#)

The item to be saved. Cannot be null.

Returns

[int ↗](#)

The number of records affected by the save operation. Typically returns 1 if the item was saved successfully; otherwise, returns 0.

## Set(Expression<Func<T, T>>)

Sets the value of the specified property for all elements in the data set.

[IRelmDataSet<T> Set\(Expression<Func<T, T>> predicate\)](#)

Parameters

[predicate Expression<Func<T, T>>](#)

An expression that specifies the property to set and the value to assign.

Returns

## [IRelmDataSet<T>](#)

The current data set with the updated elements.

## **SetDataLoader(IRelmDataLoader<T>)**

Sets the data loader to be used for retrieving data in this dataset.

```
IRelmDataLoader<T> SetDataLoader(IRelmDataLoader<T> dataLoader)
```

### Parameters

**dataLoader** [IRelmDataLoader<T>](#)

The data loader instance that will be used to load data. Cannot be null.

### Returns

[IRelmDataLoader<T>](#)

The data loader instance that was set.

## **SetFieldLoader(string, IRelmFieldLoader)**

Associates a custom field loader with the specified field name.

```
IRelmFieldLoader SetFieldLoader(string fieldName, IRelmFieldLoader dataLoader)
```

### Parameters

**fieldName** [string](#)

The name of the field for which to set the data loader. Cannot be null or empty.

**dataLoader** [IRelmFieldLoader](#)

The field loader to associate with the specified field. Cannot be null.

### Returns

## [IRelmFieldLoader](#)

The updated field loader associated with the specified field name.

## Where(Expression<Func<T, bool>>)

Filters the data set based on a specified predicate expression.

```
IRelmDataSet<T> Where(Expression<Func<T, bool>> predicate)
```

### Parameters

`predicate Expression<Func<T, bool>>`

An expression that defines the conditions each element must satisfy to be included in the returned data set. Cannot be null.

### Returns

[IRelmDataSet<T>](#)

A new data set containing elements that satisfy the specified predicate.

### Remarks

The returned data set is not materialized until it is enumerated. Multiple calls to this method can be chained to apply additional filters.

## Write()

Writes data to the underlying destination.

```
int Write()
```

### Returns

[int](#)

The number of rows written to the destination.

# Interface IRelmExecutionCommand

Namespace: [CoreRelm.Interfaces](#)

Assembly: CoreRelm.dll

Defines the contract for an execution command within the Relm framework, providing access to the initial command and expression, management of additional commands, and options for foreign key navigation.

```
public interface IRelmExecutionCommand
```

## Remarks

Implementations of this interface allow for the composition and extension of execution commands, supporting scenarios where multiple related commands and expressions are processed together. The interface also facilitates navigation of foreign key relationships for collections of items, which can be useful in data access or object-relational mapping contexts.

## Properties

### AdditionalCommandCount

Gets the number of additional commands associated with the current context.

```
int AdditionalCommandCount { get; }
```

### Property Value

[int](#)

### ExecutionCommand

Gets the initial command that was associated with this execution command.

```
Commands.Command ExecutionCommand { get; }
```

Property Value

[Commands.Command](#)

## ExecutionExpression

Gets the initial expression associated with this execution command.

```
Expression ExecutionExpression { get; }
```

Property Value

[Expression](#)

## InitialCommand

Gets the initial command that was associated with this execution command.

```
Commands.Command InitialCommand { get; }
```

Property Value

[Commands.Command](#)

## InitialExpression

Gets the initial expression associated with this execution command.

```
Expression InitialExpression { get; }
```

Property Value

[Expression](#)

## Methods

## AddAdditionalCommand(Command, Expression)

Adds an additional execution command to the current context using the specified command and associated expression.

```
RelmExecutionCommand AddAdditionalCommand(Commands.Command command, Expression expression)
```

### Parameters

**command** [Commands.Command](#)

The command to be added to the execution context. Cannot be null.

**expression** [Expression](#)

The expression associated with the command. Cannot be null.

### Returns

[RelmExecutionCommand](#)

A [RelmExecutionCommand](#) representing the newly added command and its associated expression.

## GetAdditionalCommands()

Retrieves a list of additional execution commands to be processed.

```
List<RelmExecutionCommand> GetAdditionalCommands()
```

### Returns

[List](#) <[RelmExecutionCommand](#)>

A list of [RelmExecutionCommand](#) objects representing additional commands. The list may be empty if there are no additional commands to process.

## GetForeignKeyNavigationOptions<T>(ICollection<T>)

Retrieves navigation options for foreign key relationships based on the provided collection of items.

`ForeignKeyNavigationOptions GetForeignKeyNavigationOptions<T>(ICollection<T> _items)`

## Parameters

`_items` `ICollection<T>`

The collection of items used to determine the available foreign key navigation options. Cannot be null.

## Returns

[ForeignKeyNavigationOptions](#)

A [ForeignKeyNavigationOptions](#) instance representing the available navigation options for the specified items.

## Type Parameters

`T`

The type of elements contained in the collection for which to retrieve navigation options.

# Interface IRelmFieldLoader

Namespace: [CoreRelm.Interfaces](#)

Assembly: CoreRelm.dll

Defines a contract for loading field data within a Relm context.

```
public interface IRelmFieldLoader : IRelmFieldLoaderBase
```

## Inherited Members

[IRelmFieldLoaderBase.FieldName](#) , [IRelmFieldLoaderBase.KeyFields](#) ,  
[IRelmFieldLoaderBase.GetFieldData<S>\(ICollection<S\[\]>\)](#).

## Properties

### RelmContext

Gets the current Relm context associated with this instance.

```
IRelmContext RelmContext { get; }
```

### Property Value

[IRelmContext](#)

# Interface IRelmFieldLoaderBase

Namespace: [CoreRelm.Interfaces](#)

Assembly: CoreRelm.dll

Defines the contract for loading field data based on key values in a data source.

```
public interface IRelmFieldLoaderBase
```

## Remarks

Implementations of this interface provide mechanisms to retrieve field values using one or more key fields. This interface is typically used in scenarios where fields are dynamically loaded or mapped from external data sources based on composite keys.

## Properties

### FieldName

Gets the name of the field represented by this member.

```
string FieldName { get; }
```

### Property Value

[string](#)

### KeyFields

Gets the names of the fields that uniquely identify an entity within the data source.

```
string[] KeyFields { get; }
```

### Property Value

[string](#)[]

# Methods

## GetFieldData<S>(ICollection<S[]>)

Retrieves field data for the specified collection of key arrays.

```
Dictionary<S[], object> GetFieldData<S>(ICollection<S[]> keyData)
```

### Parameters

**keyData** [ICollection](#)<S[]>

A collection of key arrays for which to retrieve associated field data. Cannot be null.

### Returns

[Dictionary](#)<S[], object>

A dictionary mapping each input key array to its corresponding field data. If a key array has no associated data, it may be omitted from the dictionary.

### Type Parameters

**S**

The type of the elements in each key array.

# Interface IRelmMember

Namespace: [CoreRelm.Interfaces](#)

Assembly: CoreRelm.dll

Defines the contract for a member entity used in apartmenting within the Relm system, including identification, contact information, and value accessors.

```
public interface IRelmMember
```

## Remarks

Implementations of this interface represent individual members and provide methods to retrieve and assign values of arbitrary types associated with the member. The generic value accessors enable flexible storage and retrieval of member-specific data.

## Properties

### Email

Gets or sets the email address associated with the user.

```
string Email { get; set; }
```

### Property Value

[string](#)

### Id

Gets or sets the unique identifier for the user.

```
long Id { get; set; }
```

### Property Value

[long](#)

## Login

Gets or sets the login name associated with the user or account.

```
string Login { get; set; }
```

### Property Value

[string](#)

## Name

Gets or sets the name associated with the user.

```
string Name { get; set; }
```

### Property Value

[string](#)

## Methods

### GetValue<T>()

Retrieves the value of the specified type from the underlying source.

```
T GetValue<T>()
```

### Returns

T

The value of type T obtained from the source.

## Type Parameters

T

The type of the value to retrieve.

## SetValue<T>(T)

Sets the value of the current object to the specified value.

```
void SetValue<T>(T value)
```

### Parameters

value T

The value to assign to the current object.

## Type Parameters

T

The type of the value to set.

# Interface IRelmModel

Namespace: [CoreRelm.Interfaces](#)

Assembly: CoreRelm.dll

Defines the contract for a Relm-compatible model, providing properties and methods for managing entity state, database interactions, and data transfer object (DTO) generation.

```
public interface IRelmModel
```

## Remarks

This interface is designed to represent a database entity with common attributes such as identifiers, timestamps, and active state. It also includes methods for resetting attributes, writing data to the database, and generating dynamic DTOs. Implementations of this interface are expected to handle database-specific operations and provide fine-grained control over column constraints during write operations.

## Properties

### Active

Gets or sets a value indicating whether the entity is active.

```
bool Active { get; set; }
```

#### Property Value

[bool](#)

### CreateDate

Gets or sets the date and time when the entity was created.

```
DateTime CreateDate { get; set; }
```

## PropertyValue

[DateTime](#)

### Id

Gets or sets the database row identifier for the entity.

```
long? Id { get; set; }
```

## PropertyValue

[long](#)?

### Remarks

This ID corresponds to the primary key in the database table.

### InternalId

Gets or sets the internal identifier for the entity.

```
string InternalId { get; set; }
```

## PropertyValue

[string](#)

### Remarks

This GUID identifier is globally unique and is intended to uniquely identify this entity across all platforms.

### LastUpdated

Gets or sets the date and time when the object was last updated.

```
DateTime LastUpdated { get; set; }
```

## Property Value

[DateTime](#)

## Methods

**GenerateDTO(IEnumerable<string>, IEnumerable<string>, string, Func<IRelmModel, Dictionary<string, object>>, int)**

Generates a dynamic Data Transfer Object (DTO) based on the specified properties and additional configuration.

```
dynamic GenerateDTO(IEnumerable<string> includeProperties = null, IEnumerable<string>
excludeProperties = null, string sourceObjectName = null, Func<IRelmModel,
Dictionary<string, object>> getAdditionalObjectProperties = null, int iteration = 0)
```

## Parameters

**includeProperties** [IEnumerable](#)<string>

A collection of property names to include in the generated DTO. If [null](#), all properties are included by default.

**excludeProperties** [IEnumerable](#)<string>

A collection of property names to exclude from the generated DTO. If [null](#), no properties are excluded.

**sourceObjectName** [string](#)

The name of the source object from which the DTO is generated. Can be [null](#) if the source object name is not required.

**getAdditionalObjectProperties** [Func](#)<IRelmModel, Dictionary<string, object>>

A function that takes an [IRelmModel](#) instance and returns a dictionary of additional properties to include in the DTO. If [null](#), no additional properties are added.

**iteration** [int](#)

The current iteration count, used to track recursive or iterative calls. Defaults to 0.

## Returns

dynamic

A dynamic object representing the generated DTO, including the specified properties and any additional properties provided.

## Remarks

The method allows fine-grained control over the properties included in the DTO by specifying inclusion and exclusion lists. If both `includeProperties` and `excludeProperties` are provided, the exclusion list takes precedence.

## GetUnderscoreProperties(bool)

Retrieves a list of properties with each name converted to underscore case.

```
List<KeyValuePair<string, Tuple<string, PropertyInfo>>> GetUnderscoreProperties(bool  
getOnlyRelmColumns = true)
```

## Parameters

`getOnlyRelmColumns` [bool](#)

A boolean value indicating whether to include only properties marked as Relm columns. If [true](#), only Realm column properties are included; otherwise, all underscore-prefixed properties are included.

## Returns

[List](#)<[KeyValuePair](#)<[string](#), [Tuple](#)<[string](#), [PropertyInfo](#)>>>

A list of key-value pairs where the key is the property name, and the value is a tuple containing the property's metadata string and its [PropertyInfo](#).

## ResetCoreAttributes(bool, bool)

Resets the core attributes of the model to their default values.

```
IRelmModel ResetCoreAttributes(bool nullInternalId = false, bool resetCreateDate = true)
```

## Parameters

### nullInternalId [bool](#)

A value indicating whether the internal ID should be set to [null](#). Pass [true](#) to nullify the internal ID; otherwise, [false](#).

### resetCreateDate [bool](#)

A value indicating whether the creation date should be reset to the current date and time. Pass [true](#) to reset the creation date; otherwise, [false](#).

## Returns

### [IRelmModel](#)

The updated model instance with the core attributes reset.

## Remarks

Resets fields [Id](#), [Active](#), [InternalId](#), and [CreateDate](#).

## ResetWithData(DataRow, string)

Resets the current model's state using the specified data and optionally sets an alternate table name.

```
IRelmModel ResetWithData(DataRow modelData, string alternateTableName = null)
```

## Parameters

### modelData [DataRow](#)

The [DataRow](#) containing the data to reset the model with. Cannot be [null](#).

### alternateTableName [string](#)

An optional alternate table name to associate with the model. If [null](#), the default table name is used.

## Returns

### [IRelmModel](#)

An updated instance of the model implementing [IRelmModel](#) with the new data applied.

## WriteToDatabase(IRelmContext, int, bool, bool, bool, bool)

Writes data to the database using the specified context and batch size, with options to control column behavior.

```
int WriteToDatabase(IRelmContext relmContext, int batchSize = 10, bool  
allowAutoIncrementColumns = false, bool allowPrimaryKeyColumns = false, bool  
allowUniqueColumns = false, bool allowAutoDateColumns = false)
```

### Parameters

**relmContext** [IRelmContext](#)

The database context used to perform the write operation. This cannot be [null](#).

**batchSize** [int](#)

The number of records to write in each batch. Must be greater than 0. The default value is 10.

**allowAutoIncrementColumns** [bool](#)

If [true](#), allows writing to columns with auto-increment constraints; otherwise, these columns are ignored.

**allowPrimaryKeyColumns** [bool](#)

If [true](#), allows writing to primary key columns; otherwise, these columns are ignored.

**allowUniqueColumns** [bool](#)

If [true](#), allows writing to columns with unique constraints; otherwise, these columns are ignored.

**allowAutoDateColumns** [bool](#)

If [true](#), allows writing to columns with automatic date generation constraints; otherwise, these columns are ignored.

### Returns

[int](#)

The number of records successfully written to the database.

## Remarks

This method provides fine-grained control over how data is written to the database by allowing or disallowing writes to specific types of columns. Use the optional parameters to customize the behavior based on the constraints of your database schema.

## WriteToDatabase(IRelmQuickContext, int, bool, bool, bool, bool)

Writes data to the database using the specified context and batch size, with options to control column constraints.

```
int WriteToDatabase(IRelmQuickContext relmQuickContext, int batchSize = 10, bool  
allowAutoIncrementColumns = false, bool allowPrimaryKeyColumns = false, bool  
allowUniqueColumns = false, bool allowAutoDateColumns = false)
```

## Parameters

**relmQuickContext** [IRelmQuickContext](#)

The database context used to perform the write operation. This context must be properly initialized before calling this method.

**batchSize** [int](#)

The number of records to write in each batch. Must be greater than zero. The default value is 10.

**allowAutoIncrementColumns** [bool](#)

A value indicating whether auto-increment columns are allowed during the write operation. If [true](#), auto-increment columns will be included; otherwise, they will be excluded.

**allowPrimaryKeyColumns** [bool](#)

A value indicating whether primary key columns are allowed during the write operation. If [true](#), primary key columns will be included; otherwise, they will be excluded.

**allowUniqueColumns** [bool](#)

A value indicating whether unique columns are allowed during the write operation. If [true](#), unique columns will be included; otherwise, they will be excluded.

## `allowAutoDateColumns` [bool](#)

A value indicating whether auto-date columns are allowed during the write operation. If [true](#), auto-date columns will be included; otherwise, they will be excluded.

## Returns

### [int](#)

The total number of records successfully written to the database.

## Remarks

This method provides fine-grained control over which types of columns are included during the write operation. Use the boolean parameters to enable or disable specific column constraints as needed.

## `WriteToDatabase(MySqlConnection, MySqlTransaction, int, bool, bool, bool, bool)`

Writes data to the database using the specified connection and optional transaction.

```
int WriteToDatabase(MySqlConnection existingConnection, MySqlTransaction sqlTransaction  
= null, int BatchSize = 10, bool allowAutoIncrementColumns = false, bool  
allowPrimaryKeyColumns = false, bool allowUniqueColumns = false, bool allowAutoDateColumns  
= false)
```

## Parameters

### `existingConnection` [MySqlConnection](#)

An open `MySql.Data.MySqlClient.MySqlConnection` to the database where the data will be written. The connection must remain open for the duration of the operation.

### `sqlTransaction` [MySqlTransaction](#)

An optional `MySql.Data.MySqlClient.MySqlTransaction` to use for the operation. If null, the operation will not be part of a transaction.

### `BatchSize` [int](#)

The number of rows to write in each batch. Must be greater than zero. Defaults to 10.

## `allowAutoIncrementColumns` [bool](#)

A value indicating whether columns with auto-increment constraints are allowed to be written. If [true](#), auto-increment columns will be included; otherwise, they will be excluded.

## `allowPrimaryKeyColumns` [bool](#)

A value indicating whether primary key columns are allowed to be written. If [true](#), primary key columns will be included; otherwise, they will be excluded.

## `allowUniqueColumns` [bool](#)

A value indicating whether unique columns are allowed to be written. If [true](#), unique columns will be included; otherwise, they will be excluded.

## `allowAutoDateColumns` [bool](#)

A value indicating whether columns with automatic date generation constraints are allowed to be written. If [true](#), such columns will be included; otherwise, they will be excluded.

## Returns

### [int](#)

The number of rows successfully written to the database.

## `WriteToDatabase(Enum, int, bool, bool, bool, bool)`

Writes data to the specified database connection in batches.

```
int WriteToDatabase(Enum connectionName, int batchSize = 10, bool allowAutoIncrementColumns = false, bool allowPrimaryKeyColumns = false, bool allowUniqueColumns = false, bool allowAutoDateColumns = false)
```

## Parameters

### `connectionName` [Enum](#)

The name of the database connection to use. Must be a valid connection identifier.

### `batchSize` [int](#)

The number of records to write in each batch. Defaults to 10. Must be greater than 0.

## allowAutoIncrementColumns [bool](#)

Indicates whether auto-increment columns are allowed in the data being written. Defaults to [false](#).

## allowPrimaryKeyColumns [bool](#)

Indicates whether primary key columns are allowed in the data being written. Defaults to [false](#).

## allowUniqueColumns [bool](#)

Indicates whether unique columns are allowed in the data being written. Defaults to [false](#).

## allowAutoDateColumns [bool](#)

Indicates whether auto-generated date columns are allowed in the data being written. Defaults to [false](#).

## Returns

### [int](#)

The number of records successfully written to the database.

## Remarks

This method writes data to the database in batches to optimize performance. Ensure that the specified connection is valid and that the data conforms to the constraints specified by the parameter flags. If any of the flags are set to [true](#), the corresponding column types will be allowed in the data being written.

# Interface IRelmModelApartment

Namespace: [CoreRelm.Interfaces](#)

Assembly: CoreRelm.dll

Represents an apartment entity within the Relm model, including associated user and membership information.

```
public interface IRelmModelApartment
```

## Properties

### ApartmentId

Gets or sets the unique identifier for the apartment.

```
string ApartmentId { get; set; }
```

### Property Value

[string](#)

### Member

Gets or sets the associated member for this instance.

```
IRelmMember Member { get; set; }
```

### Property Value

[IRelmMember](#)

### UserEmail

Gets or sets the email address associated with the user.

```
string UserEmail { get; set; }
```

Property Value

[string](#)

## UserId

Gets or sets the unique identifier for the user.

```
int UserId { get; set; }
```

Property Value

[int](#)

## UserName

Gets or sets the user name associated with the current context.

```
string UserName { get; set; }
```

Property Value

[string](#)

# Namespace CoreRelm.Interfaces.Migrations

## Interfaces

[IMigrationOperation](#)

[IMigrationSqlProvider](#)

[IRelmDatabaseProvisioner](#)

[IRelmMigrationExecutor](#)

[IRelmMigrationPlanner](#)

[IRelmMigrationSqlRenderer](#)

# Interface IMigrationOperation

Namespace: [CoreRelm.Interfaces.Migrations](#)

Assembly: CoreRelm.dll

```
public interface IMigrationOperation
```

## Properties

### Description

```
string Description { get; }
```

### Property Value

[string](#)

# Interface IMigrationSqlProvider

Namespace: [CoreRelm.Interfaces.Migrations](#)

Assembly: CoreRelm.dll

```
public interface IMigrationSqlProvider
```

## Methods

Generate(string, string, string, string,  
List<ValidatedModelType>, bool, CancellationToken, bool, bool,  
MySqlDatabaseProvisioner)

```
Task<MigrationGenerateResult> Generate(string migrationName, string stampUtc,  
string setName, string dbName, List<ValidatedModelType> modelsForDb, bool  
destructive, CancellationToken cancellationToken, bool quiet, bool apply,  
MySqlDatabaseProvisioner provisioner)
```

### Parameters

migrationName [string](#)

stampUtc [string](#)

setName [string](#)

dbName [string](#)

modelsForDb [List](#)<[ValidatedModelType](#)>

destructive [bool](#)

cancellationToken [CancellationToken](#)

quiet [bool](#)

apply [bool](#)

provisioner [MySqlDatabaseProvisioner](#)

Returns

[Task](#) <[MigrationGenerateResult](#)>

# Interface IRelmDatabaseProvisioner

Namespace: [CoreRelm.Interfaces.Migrations](#)

Assembly: CoreRelm.dll

```
public interface IRelmDatabaseProvisioner
```

## Methods

### DatabaseExistsAsync(string, string, CancellationToken)

```
Task<bool> DatabaseExistsAsync(string serverConnectionString, string databaseName,  
CancellationToken ct = default)
```

Parameters

serverConnectionString [string](#)

databaseName [string](#)

ct [CancellationToken](#)

Returns

[Task](#)<bool>

### EnsureDatabaseExistsAsync(string, string, string?, string?, CancellationToken)

```
Task EnsureDatabaseExistsAsync(string serverConnectionString, string databaseName, string?  
charset = null, string? collation = null, CancellationToken ct = default)
```

Parameters

serverConnectionString [string](#)

databaseName [string](#)

charset [string](#)

collation [string](#)

ct [CancellationToken](#)

Returns

[Task](#)

# Interface IRelmMigrationExecutor

Namespace: [CoreRelm.Interfaces.Migrations](#)

Assembly: CoreRelm.dll

```
public interface IRelmMigrationExecutor
```

## Methods

### ExecuteSqlAsync(string, string, CancellationToken)

```
Task ExecuteSqlAsync(string connectionString, string sql, CancellationToken ct = default)
```

#### Parameters

connectionString [string](#)

sql [string](#)

ct [CancellationToken](#)

#### Returns

[Task](#)

# Interface IRelmMigrationPlanner

Namespace: [CoreRelm.Interfaces.Migrations](#)

Assembly: CoreRelm.dll

```
public interface IRelmMigrationPlanner
```

## Methods

### Plan(SchemaSnapshot, SchemaSnapshot, MigrationPlanOptions)

```
MigrationPlan Plan(SchemaSnapshot desired, SchemaSnapshot actual,  
MigrationPlanOptions options)
```

#### Parameters

desired [SchemaSnapshot](#)

actual [SchemaSnapshot](#)

options [MigrationPlanOptions](#)

#### Returns

[MigrationPlan](#)

# Interface IRelmMigrationSqlRenderer

Namespace: [CoreRelm.Interfaces.Migrations](#)

Assembly: CoreRelm.dll

```
public interface IRelmMigrationSqlRenderer
```

## Methods

### Render(MigrationPlan, MySqlRenderOptions?)

```
string Render(MigrationPlan plan, MySqlRenderOptions? options = null)
```

#### Parameters

plan [MigrationPlan](#)

options [MySqlRenderOptions](#)

#### Returns

[string](#) ↗

# Namespace CoreRelm.Interfaces.RelmQuick

## Interfaces

### [IRelmQuickContext](#)

Defines a context for interacting with a Relm database, providing methods for managing connections, transactions, and data operations. This interface supports querying, data manipulation, and bulk operations while ensuring proper resource management.

### [IRelmQuickFieldLoader](#)

Provides access to the field loading quick context for use with RELM field loaders.

# Interface IRelmQuickContext

Namespace: [CoreRelm.Interfaces.RelmQuick](#)

Assembly: CoreRelm.dll

Defines a context for interacting with a Relm database, providing methods for managing connections, transactions, and data operations. This interface supports querying, data manipulation, and bulk operations while ensuring proper resource management.

```
[Obsolete("IRelmQuickContext is deprecated and will be removed in future versions. Please  
use IRelmContext(autoInitializeDataSets: false, autoVerifyTables: false) instead.")]  
public interface IRelmQuickContext : IDisposable
```

## Inherited Members

[IDisposable.Dispose\(\)](#)

## Remarks

Implementations of this interface are designed to facilitate database operations in a structured and transactional manner. The context provides methods for starting and ending connections, managing transactions, and performing CRUD operations on data sets. It also includes support for executing raw queries and bulk operations. The interface extends [IDisposable](#) to ensure that resources are properly released when the context is no longer needed. Thread safety is not guaranteed unless explicitly stated by the implementation. Users should ensure proper synchronization when accessing the context from multiple threads.

## Properties

### ContextOptions

Gets the builder used to configure options for the current Relm context.

```
RelmContextOptionsBuilder ContextOptions { get; }
```

### Property Value

[RelmContextOptionsBuilder](#)

# Methods

## BulkTableWrite<T>(T, string, MySqlTransaction, Type, int, bool, bool, bool)

Performs a bulk write operation to insert or update data in the specified database table.

```
int BulkTableWrite<T>(T sourceData, string tableName = null, MySqlTransaction sqlTransaction = null, Type forceType = null, int batchSize = 100, bool allowAutoIncrementColumns = false, bool allowPrimaryKeyColumns = false, bool allowUniqueColumns = false)
```

### Parameters

#### sourceData T

The data to be written to the table. Must not be null. If a collection is provided, each item will be processed in the bulk operation.

#### tableName string ↗

The name of the target database table. If null, the table name will be inferred from the type T.

#### sqlTransaction MySqlTransaction

An optional MySql.Data.MySqlClient.MySqlTransaction to associate the bulk write operation with an existing transaction. If null, the operation will execute without a transaction.

#### forceType Type ↗

An optional Type ↗ to explicitly specify the type of the data being written. If null, the type will be inferred from T.

#### batchSize int ↗

The maximum number of rows to include in each batch during the bulk write operation. Must be greater than zero. Defaults to 100.

#### allowAutoIncrementColumns bool ↗

Indicates whether auto-increment columns in the target table are allowed to be explicitly written. Defaults to false ↗.

#### allowPrimaryKeyColumns bool ↗

Indicates whether primary key columns in the target table are allowed to be explicitly written. Defaults to [false](#).

### allowUniqueColumns [bool](#)

Indicates whether unique columns in the target table are allowed to be explicitly written. Defaults to [false](#).

Returns

### [int](#)

The number of rows successfully written to the database table.

Type Parameters

### T

The type of the source data. This can be a collection or a single object representing the data to be written.

Remarks

This method is optimized for high-performance bulk operations and supports optional batching to handle large datasets efficiently. Ensure that the source data aligns with the schema of the target table to avoid runtime errors.

## CommitTransaction()

Commits the current transaction, making all changes permanent in the database.

### [void CommitTransaction\(\)](#)

## DoDatabaseWork(string, Dictionary<string, object>, bool, bool)

Executes a database operation using the specified query and parameters.

### [void DoDatabaseWork\(string query, Dictionary<string, object> parameters = null, bool throwException = true, bool useTransaction = false\)](#)

## Parameters

### query [string](#)

The SQL query to execute. Cannot be null or empty.

### parameters [Dictionary](#)<[string](#), [object](#)>

An optional dictionary of parameter names and values to be used in the query. If null, no parameters are applied.

### throwException [bool](#)

A value indicating whether an exception should be thrown if the operation fails. If [true](#), an exception is thrown on failure; otherwise, the failure is silently handled.

### useTransaction [bool](#)

A value indicating whether the operation should be executed within a transaction. If [true](#), the operation is wrapped in a transaction; otherwise, it is executed without one.

## Remarks

This method allows for executing parameterized SQL queries with optional transaction support. Use the `throwException` parameter to control error handling behavior.

## DoDatabaseWork(string, Func<MySqlCommand, object>, bool, bool)

Executes a database operation using the specified query and callback function.

```
void DoDatabaseWork(string query, Func<MySqlCommand, object> actionCallback, bool  
throwException = true, bool useTransaction = false)
```

## Parameters

### query [string](#)

The SQL query to execute. Must not be null or empty.

### actionCallback [Func](#)<MySqlCommand, [object](#)>

A callback function that processes the MySql.Data.MySqlClient.MySqlCommand object and returns a result. The callback must not be null.

#### **throwException** [bool](#)

A value indicating whether to throw an exception if an error occurs during the operation. If [true](#), exceptions will be thrown; otherwise, errors will be suppressed.

#### **useTransaction** [bool](#)

A value indicating whether the operation should be executed within a database transaction. If [true](#), the operation will be wrapped in a transaction; otherwise, it will not.

## Remarks

This method provides a flexible way to execute database operations by allowing the caller to define custom logic through the `actionCallback` parameter. Ensure that the `query` is properly sanitized to prevent SQL injection attacks.

## DoDatabaseWork<T>(string, Dictionary<string, object>, bool, bool)

Executes a database query and returns the result as the specified type.

```
T DoDatabaseWork<T>(string query, Dictionary<string, object> parameters = null, bool throwException = true, bool useTransaction = false)
```

## Parameters

#### **query** [string](#)

The SQL query to execute. Cannot be null or empty.

#### **parameters** [Dictionary](#)<[string](#), [object](#)>

An optional dictionary of parameter names and values to include in the query. Can be null if no parameters are required.

#### **throwException** [bool](#)

Specifies whether an exception should be thrown if the query fails. If [true](#), an exception is thrown on failure; otherwise, the method returns the default value of `T`.

## `useTransaction bool`

Specifies whether the query should be executed within a transaction. If `true`, the query is executed in a transactional context; otherwise, it is executed without a transaction.

## Returns

`T`

The result of the query as an instance of type `T`. Returns the default value of `T` if `throwException` is `false` and the query fails.

## Type Parameters

`T`

The type of the result expected from the query.

## Remarks

Use this method to execute queries that return a single result, such as scalar values or objects. Ensure that the type `T` matches the expected result of the query to avoid runtime errors.

## `DoDatabaseWork<T>(string, Func<MySqlCommand, object>, bool, bool)`

Executes a database operation using the specified query and callback function.

```
T DoDatabaseWork<T>(string query, Func<MySqlCommand, object> actionCallback, bool throwException = true, bool useTransaction = false)
```

## Parameters

### `query string`

The SQL query to be executed. Cannot be null or empty.

### `actionCallback Func<MySqlCommand, object>`

A callback function that defines the operation to perform with the `MySql.Data.MySqlClient.MySqlCommand` object. The function should return an object of type `T`.

### `throwException` [bool](#)

A value indicating whether an exception should be thrown if an error occurs during the operation. The default value is [true](#).

### `useTransaction` [bool](#)

A value indicating whether the operation should be executed within a database transaction. The default value is [false](#).

## Returns

`T`

The result of the operation, as returned by the `actionCallback` function.

## Type Parameters

`T`

The type of the result returned by the callback function.

## Remarks

This method provides a flexible way to execute database operations by allowing the caller to define the specific action to perform using the provided  `MySql.Data.MySqlClient.MySqlCommand` object. If `useTransaction` is [true](#), the operation will be executed within a transaction, which will be committed or rolled back based on the success or failure of the operation.

## EndConnection(bool)

Ends the current connection and optionally commits any active transaction.

```
void EndConnection(bool commitTransaction = true)
```

## Parameters

### `commitTransaction` [bool](#)

A value indicating whether to commit the active transaction before ending the connection. [true](#) to commit the transaction; [false](#) to roll it back. The default is [true](#).

## Remarks

Use this method to cleanly terminate a connection. If a transaction is active, you can specify whether to commit or roll it back before the connection is closed. Ensure that any necessary operations are completed before calling this method, as the connection will no longer be available afterward.

## FirstOrDefault<T>(Expression<Func<T, bool>>, bool)

Retrieves the first element of a sequence from the data source that satisfies the specified condition, or a default value if no such element is found.

```
T FirstOrDefault<T>(Expression<Func<T, bool>> predicate, bool loadDataLoaders = false) where  
T : IRelmModel, new()
```

## Parameters

**predicate** [Expression<Func<T, bool>>](#)

An expression that defines the condition to test each element for.

**loadDataLoaders** [bool](#)

A boolean value indicating whether to load associated data loaders for the retrieved element. If [true](#), data loaders will be initialized; otherwise, they will not be loaded.

## Returns

**T**

The first element of type **T** that satisfies the specified condition, or the default value of **T** if no such element is found.

## Type Parameters

**T**

The type of the elements in the sequence. Must implement [IRelmModel](#) and have a parameterless constructor.

## GetBulkTableWriter<T>(string, bool, bool, bool, bool, bool)

Creates and returns a [BulkTableWriter<T>](#) instance for performing bulk insert operations on a database table.

```
BulkTableWriter<T> GetBulkTableWriter<T>(string insertQuery = null, bool useTransaction =  
false, bool throwException = true, bool allowAutoIncrementColumns = false, bool  
allowPrimaryKeyColumns = false, bool allowUniqueColumns = false)
```

## Parameters

### insertQuery [string](#)

An optional SQL insert query to use for the bulk operation. If [null](#), a default query is generated based on the type [T](#).

### useTransaction [bool](#)

Specifies whether the bulk operation should be performed within a transaction. If [true](#), the operation is transactional; otherwise, it is not.

### throwException [bool](#)

Specifies whether exceptions should be thrown if an error occurs during the bulk operation. If [true](#), exceptions are thrown; otherwise, errors are suppressed.

### allowAutoIncrementColumns [bool](#)

Indicates whether auto-increment columns in the database table are allowed to be included in the bulk operation. If [true](#), these columns are included; otherwise, they are excluded.

### allowPrimaryKeyColumns [bool](#)

Indicates whether primary key columns in the database table are allowed to be included in the bulk operation. If [true](#), these columns are included; otherwise, they are excluded.

### allowUniqueColumns [bool](#)

Indicates whether unique columns in the database table are allowed to be included in the bulk operation. If [true](#), these columns are included; otherwise, they are excluded.

## Returns

### [BulkTableWriter<T>](#)

A [BulkTableWriter<T>](#) instance configured for the specified bulk operation settings.

## Type Parameters

### T

The type of the objects to be written to the database table. Each object represents a row in the table.

## Remarks

The [BulkTableWriter<T>](#) provides an efficient way to insert large amounts of data into a database table. Use the optional parameters to customize the behavior of the bulk operation, such as enabling transactions or allowing specific column types.

## GetDataTable<T>(string, Dictionary<string, object>, bool)

Executes the specified query and retrieves a collection of data mapped to the specified type.

```
IEnumerable<T> GetDataTable<T>(string query, Dictionary<string, object> parameters = null,  
bool throwException = true)
```

## Parameters

### query [string](#)

The SQL query to execute. Must not be null or empty.

### parameters [Dictionary](#)<[string](#), [object](#)>

An optional dictionary of parameter names and values to be used in the query. Can be null if no parameters are required.

### throwException [bool](#)

A value indicating whether an exception should be thrown if the query fails. If [true](#), an exception will be thrown on failure; otherwise, the method will return an empty collection.

## Returns

### [IEnumerable](#)<T>

An [IEnumerable](#)<T> containing the results of the query mapped to the specified type. Returns an empty collection if no results are found or if [throwException](#) is [false](#) and the query fails.

## Type Parameters

T

The type to which the query results will be mapped.

## GetDataObject<T>(string, Dictionary<string, object>, bool)

Executes the specified query and retrieves a data object of the specified type.

```
T GetDataObject<T>(string query, Dictionary<string, object> parameters = null, bool throwException = true) where T : IRelmModel, new()
```

## Parameters

**query** [string](#)

The query string used to retrieve the data object. Cannot be [null](#) or empty.

**parameters** [Dictionary](#)<[string](#), [object](#)>

An optional dictionary of parameters to be used in the query. Keys represent parameter names, and values represent their corresponding values.

**throwException** [bool](#)

A value indicating whether an exception should be thrown if the query fails or no data is found. If [true](#), an exception is thrown; otherwise, [default](#) is returned.

## Returns

T

An instance of the specified type T populated with the data retrieved by the query. Returns [default](#) if no data is found and [throwException](#) is [false](#).

## Type Parameters

T

The type of the data object to retrieve. Must implement [IRelmModel](#) and have a parameterless constructor.

## GetDataObjects<T>(string, Dictionary<string, object>, bool)

Executes a query and retrieves a collection of data objects of the specified type.

```
IEnumerable<T> GetDataObjects<T>(string query, Dictionary<string, object> parameters = null,  
bool throwException = true) where T : IRelmModel, new()
```

### Parameters

**query** [string](#)

The query string used to retrieve the data objects. Cannot be null or empty.

**parameters** [Dictionary](#)<[string](#), [object](#)>

An optional dictionary of parameter names and values to be used in the query. If null, no parameters are applied.

**throwException** [bool](#)

A boolean value indicating whether an exception should be thrown if the query fails. If [true](#), an exception is thrown on failure; otherwise, the method returns an empty collection.

### Returns

[IEnumerable](#)<T>

An [IEnumerable](#)<T> containing the data objects retrieved by the query. Returns an empty collection if no data is found or if **throwException** is [false](#) and the query fails.

### Type Parameters

T

The type of data objects to retrieve. Must implement [IRelmModel](#) and have a parameterless constructor.

## GetDataRow(string, Dictionary<string, object>, bool)

Executes the specified query and retrieves the first row of the result set as a [DataRow](#).

```
DataRow GetDataRow(string query, Dictionary<string, object> parameters = null, bool throwException = true)
```

## Parameters

### query [string](#)

The SQL query to execute. Must be a valid SQL statement.

### parameters [Dictionary](#)<[string](#), [object](#)>

An optional dictionary of parameter names and values to be used in the query. If null, no parameters are applied.

### throwException [bool](#)

A boolean value indicating whether an exception should be thrown if the query does not return any rows. If [true](#), an exception is thrown when no rows are found; otherwise, [null](#) is returned.

## Returns

### [DataRow](#)

A [DataRow](#) representing the first row of the result set, or [null](#) if no rows are found and [throwException](#) is set to [false](#).

## GetDataSet(Type)

Retrieves an initialized instance of a dataset based on the specified type.

```
IRelmDataSetBase GetDataSet(Type dataSetType)
```

## Parameters

### dataSetType [Type](#)

The [Type](#) of the dataset to retrieve. This must be a type that implements [IRelmDataSetBase](#).

## Returns

### [IRelmDataSetBase](#)

An instance of the dataset that matches the specified type. Returns [null](#) if no matching dataset is found.

## GetDataSet(Type, bool)

Retrieves an initialized instance of a dataset of the specified type.

`IRelmDataSetBase GetDataSet(Type dataSetType, bool throwException)`

### Parameters

`dataSetType Type`

The [Type](#) of the dataset to retrieve. This must implement [IRelmDataSetBase](#).

`throwException bool`

A value indicating whether an exception should be thrown if the dataset cannot be found. If [true](#), an exception is thrown when the dataset is not found; otherwise, [null](#) is returned.

### Returns

[IRelmDataSetBase](#)

An instance of [IRelmDataSetBase](#) representing the requested dataset, or [null](#) if the dataset is not found and `throwException` is [false](#).

## GetDataSetType(Type)

Retrieves an uninitialized dataset of type specified by `dataSetType`.

`IRelmDataSetBase GetDataSetType(Type dataSetType)`

### Parameters

`dataSetType Type`

The [Type](#) of the dataset to retrieve. This must implement [IRelmDataSetBase](#).

Returns

## [IRelmDataSetBase](#)

An instance of [IRelmDataSetBase](#) that corresponds to the specified `dataSetType`.

## GetDataSetType(Type, bool)

Retrieves an uninitialized dataset of type specified by `dataSetType`.

```
IRelmDataSetBase GetDataSetType(Type dataSetType, bool throwException)
```

Parameters

### `dataSetType` [Type](#)

The [Type](#) of the dataset to retrieve. Must implement [IRelmDataSetBase](#).

### `throwException` [bool](#)

A boolean value indicating whether to throw an exception if the specified `dataSetType` is invalid. [true](#) to throw an exception; otherwise, [false](#).

Returns

## [IRelmDataSetBase](#)

An instance of [IRelmDataSetBase](#) corresponding to the specified `dataSetType`. Returns [null](#) if `throwException` is [false](#) and the dataset type is invalid.

## GetDataSetType<T>()

Retrieves an uninitialized dataset of the specified type.

```
IRelmDataSet<T> GetDataSetType<T>() where T : IRelmModel, new()
```

Returns

## [IRelmDataSet](#)<T>

An instance of [IRelmDataSet<T>](#) representing the dataset of the specified type.

## Type Parameters

T

The type of the dataset to retrieve. Must implement [IRelmModel](#) and have a parameterless constructor.

## Remarks

This method is generic and allows retrieval of datasets for any type that satisfies the constraints.

## GetDataSetType<T>(bool)

Retrieves an uninitialized dataset of the specified type.

```
IRelmDataSet<T> GetDataSetType<T>(bool throwException) where T : IRelmModel, new()
```

## Parameters

**throwException** [bool](#)

A value indicating whether an exception should be thrown if the dataset cannot be retrieved. If [true](#), an exception is thrown; otherwise, [null](#) may be returned.

## Returns

[IRelmDataSet<T>](#)

An instance of [IRelmDataSet<T>](#) representing the dataset of the specified type, or [null](#) if the dataset cannot be retrieved and **throwException** is [false](#).

## Type Parameters

T

The type of the dataset to retrieve. Must implement [IRelmModel](#) and have a parameterless constructor.

## GetDataSet<T>()

Retrieves an initialized instance of a dataset of the specified type.

```
IRelmDataSet<T> GetDataSet<T>() where T : IRelmModel, new()
```

Returns

[IRelmDataSet<T>](#)

An instance of [IRelmDataSet<T>](#) containing the data for the specified type.

Type Parameters

T

The type of the dataset to retrieve. Must implement [IRelmModel](#) and have a parameterless constructor.

Remarks

Use this method to access a dataset for a specific model type. The type parameter T must represent a model that conforms to the [IRelmModel](#) interface.

## GetDataSet<T>(bool)

Retrieves an initialized instance of a dataset of the specified type.

```
IRelmDataSet<T> GetDataSet<T>(bool throwException) where T : IRelmModel, new()
```

Parameters

throwException [bool](#)

A value indicating whether an exception should be thrown if the dataset cannot be retrieved. If [true](#), an exception is thrown on failure; otherwise, [null](#) is returned.

Returns

[IRelmDataSet<T>](#)

An instance of [IRelmDataSet<T>](#) containing the dataset of type `T`. Returns [null](#) if the dataset cannot be retrieved and `throwException` is [false](#).

## Type Parameters

`T`

The type of the dataset to retrieve. Must implement [IRelmModel](#) and have a parameterless constructor.

## GetDataTable(string, Dictionary<string, object>, bool)

Executes the specified SQL query and returns the results as a [DataTable](#).

```
DataTable GetDataTable(string query, Dictionary<string, object> parameters = null, bool throwException = true)
```

## Parameters

`query` [string](#)

The SQL query to execute. This must be a valid SQL statement.

`parameters` [Dictionary](#)<[string](#), [object](#)>

An optional dictionary of parameter names and values to be used in the query. If provided, the parameters will be added to the query to prevent SQL injection. Defaults to [null](#).

`throwException` [bool](#)

A value indicating whether an exception should be thrown if an error occurs during query execution. If [true](#), exceptions will be propagated to the caller. If [false](#), the method will suppress exceptions and return [null](#) in case of an error. Defaults to [true](#).

## Returns

[DataTable](#)

A [DataTable](#) containing the results of the query. Returns [null](#) if `throwException` is [false](#) and an error occurs during execution.

## Remarks

This method is designed to execute read-only queries, such as SELECT statements. It is not intended for executing queries that modify data (e.g., INSERT, UPDATE, DELETE).

## GetIdFromInternalId(string, string)

Converts an internal identifier to its corresponding row identifier for a specified table.

```
string GetIdFromInternalId(string table, string InternalId)
```

### Parameters

**table** [string](#)

The name of the table containing the internal identifier. Cannot be null or empty.

**InternalId** [string](#)

The internal identifier to be converted. Cannot be null or empty.

### Returns

[string](#)

The row identifier corresponding to the specified internal identifier and table.

## GetLastInsertId()

Retrieves the row identifier of the most recently inserted record in the database.

```
string GetLastInsertId()
```

### Returns

[string](#)

The identifier of the last inserted record as a string. The format and value depend on the database implementation.

## Remarks

This method is typically used after an insert operation to obtain the unique identifier generated for the new record. Ensure that the database connection and context are properly configured before calling this method.

## GetScalar<T>(string, Dictionary<string, object>, bool)

Executes the specified SQL query and retrieves a single scalar value of the specified type.

```
T GetScalar<T>(string query, Dictionary<string, object> parameters = null, bool throwException = true)
```

## Parameters

**query** [string](#)

The SQL query to execute. Must not be null or empty.

**parameters** [Dictionary](#)<[string](#), [object](#)>

An optional dictionary of parameter names and values to include in the query. If null, no parameters are added.

**throwException** [bool](#)

A boolean value indicating whether an exception should be thrown if the query fails. If [true](#), an exception is thrown on failure; otherwise, the default value of **T** is returned.

## Returns

**T**

The scalar value of type **T** returned by the query. If the query does not return a result and **throwException** is [false](#), the default value of **T** is returned.

## Type Parameters

**T**

The type of the scalar value to return.

## Remarks

Use this method to retrieve a single value, such as a count or aggregate result, from a database query. Ensure that the query is written to return only one value; otherwise, an exception may occur.

## Get<T>(bool)

Retrieves a collection of entities of type `T` from the data source.

```
ICollection<T> Get<T>(bool loadDataLoaders = false) where T : IRelmModel, new()
```

### Parameters

`loadDataLoaders` [bool](#)

A boolean value indicating whether to load associated data loaders. If [true](#), data loaders will be initialized; otherwise, they will not.

### Returns

[ICollection](#)<T>

A collection of objects of type `T`. The collection may be empty if no objects are found.

### Type Parameters

`T`

The type of objects to retrieve. Must implement [IRelmModel](#) and have a parameterless constructor.

## Get<T>(Expression<Func<T, bool>>, bool)

Retrieves a collection of entities of type `T` from the data source that satisfy the specified predicate.

```
ICollection<T> Get<T>(Expression<Func<T, bool>> predicate, bool loadDataLoaders = false)  
where T : IRelmModel, new()
```

### Parameters

**predicate** [Expression](#)<[Func](#)<T, [bool](#)>>

An expression that defines the conditions the entities must satisfy.

**loadDataLoaders** [bool](#)

A boolean value indicating whether to load associated data loaders for the retrieved entities. [true](#) to load data loaders; otherwise, [false](#).

Returns

[ICollection](#)<T>

A collection of entities of type T that match the specified predicate. Returns an empty collection if no entities match.

Type Parameters

T

The type of the entities to retrieve. Must implement [IRelmModel](#) and have a parameterless constructor.

## HasDataSet(Type, bool)

Determines whether a dataset of the specified type is available.

```
bool HasDataSet(Type dataSetType, bool throwException = true)
```

Parameters

**dataSetType** [Type](#)

The [Type](#) of the dataset to check for availability. This parameter cannot be [null](#).

**throwException** [bool](#)

A value indicating whether an exception should be thrown if the dataset is not available. If [true](#), an exception is thrown when the dataset is not found; otherwise, the method returns [false](#).

Returns

[bool](#)

[true](#) if the dataset of the specified type is available; otherwise, [false](#).

## HasDataSet<T>(bool)

Determines whether a dataset of the specified type exists in the current context.

```
bool HasDataSet<T>(bool throwException = true) where T : IRelmModel, new()
```

### Parameters

[throwException](#) [bool](#)

A value indicating whether to throw an exception if the dataset does not exist. [true](#) to throw an exception; [false](#) to return [false](#) instead.

### Returns

[bool](#)

[true](#) if the dataset of the specified type exists; otherwise, [false](#).

### Type Parameters

[T](#)

The type of the dataset to check for. Must implement [IRelmModel](#) and have a parameterless constructor.

## RollbackTransaction()

Rolls back the current transaction, undoing any changes made since the transaction began.

```
void RollbackTransaction()
```

### Remarks

This method should be called to revert changes if an error occurs or if the transaction cannot be completed successfully. Ensure that a transaction is active before calling this method; otherwise, an exception may be thrown.

## RollbackTransactions()

Rolls back the current transaction, undoing any changes made since the transaction began.

```
void RollbackTransactions()
```

## Remarks

This method should be called to revert changes if an error occurs or if the transaction cannot be completed successfully. Ensure that a transaction is active before calling this method; otherwise, an exception may be thrown.

## Run<T>(string, Dictionary<string, object>)

Executes the specified query and returns a collection of results mapped to the specified type.

```
ICollection<T> Run<T>(string query, Dictionary<string, object> parameters = null) where T : IRelmModel, new()
```

## Parameters

**query** [string](#)

The query string to execute. This string must be a valid query in the context of the underlying data source.

**parameters** [Dictionary](#)<[string](#), [object](#)>

An optional dictionary of parameter names and values to be used in the query. If null, no parameters are applied.

## Returns

[ICollection](#)<T>

A collection of objects of type `T` representing the query results. The collection will be empty if no results are found.

## Type Parameters

`T`

The type of the objects in the result collection. Must implement [IRelmModel](#) and have a parameterless constructor.

## SetDataLoader<T>(IRelmDataLoader<T>)

Configures the data loader for the specified model type.

```
void SetDataLoader<T>(IRelmDataLoader<T> dataLoader) where T : RelmModel, new()
```

## Parameters

`dataLoader` [IRelmDataLoader](#)<T>

The data loader instance to associate with the specified model type. Cannot be [null](#).

## Type Parameters

`T`

The type of the model that the data loader will handle. Must inherit from [RelmModel](#) and have a parameterless constructor.

## StartConnection(bool, int)

Starts a connection to the database.

```
void StartConnection(bool autoOpenTransaction = false, int lockWaitTimeoutSeconds = 0)
```

## Parameters

`autoOpenTransaction` [bool](#)

Specifies whether a transaction should be automatically opened after the connection is established.

Pass [true](#) to open a transaction automatically; otherwise, [false](#).

### lockWaitTimeoutSeconds [int](#)

The lock wait timeout in seconds. A value of 0 indicates the default timeout for the database. Specify a positive integer to set a custom timeout duration.

## Remarks

If [autoOpenTransaction](#) is set to [true](#), ensure that the transaction is committed or rolled back to avoid leaving it open. This method must be called before performing any database operations.

## Where<T>(Expression<Func<T, bool>>)

Retrieves data from the data source, filtered based on the specified predicate.

```
IRelmDataSet<T> Where<T>(Expression<Func<T, bool>> predicate) where T : IRelmModel, new()
```

## Parameters

### [predicate](#) [Expression](#)<[Func](#)<T, [bool](#)>>

An expression that defines the conditions for filtering the dataset.

## Returns

### [IRelmDataSet](#)<T>

A new dataset containing only the elements that satisfy the specified predicate.

## Type Parameters

### T

The type of the model in the dataset. Must implement [IRelmModel](#) and have a parameterless constructor.

## Remarks

The predicate is evaluated for each element in the dataset, and only those elements for which the predicate returns [true](#) are included in the result.

# WriteToDatabase(IRelmModel, int, bool, bool, bool, bool)

Writes the specified [IRelmModel](#) to the database in batches.

```
int WriteToDatabase(IRelmModel relmModel, int batchSize = 100, bool  
allowAutoIncrementColumns = false, bool allowPrimaryKeyColumns = false, bool  
allowUniqueColumns = false, bool allowAutoDateColumns = false)
```

## Parameters

**relmModel** [IRelmModel](#)

The model containing the data to be written to the database. Cannot be [null](#).

**batchSize** [int](#)

The number of records to write in each batch. Must be greater than 0. The default value is 100.

**allowAutoIncrementColumns** [bool](#)

A value indicating whether columns with auto-increment constraints are allowed to be written. If [true](#), auto-increment columns will be included in the write operation; otherwise, they will be excluded.

**allowPrimaryKeyColumns** [bool](#)

A value indicating whether primary key columns are allowed to be written. If [true](#), primary key columns will be included in the write operation; otherwise, they will be excluded.

**allowUniqueColumns** [bool](#)

A value indicating whether columns with unique constraints are allowed to be written. If [true](#), unique columns will be included in the write operation; otherwise, they will be excluded.

**allowAutoDateColumns** [bool](#)

A value indicating whether columns with automatic date generation constraints are allowed to be written. If [true](#), auto-date columns will be included in the write operation; otherwise, they will be excluded.

## Returns

[int](#)

The total number of records successfully written to the database.

## Remarks

This method writes data to the database in batches to optimize performance. The behavior of the write operation can be customized using the boolean parameters to control the inclusion of specific column types.

## WriteToDatabase(IEnumerable<IRelmModel>, int, bool, bool, bool, bool)

Writes a collection of Relm models to the database in batches.

```
int WriteToDatabase(IEnumerable<IRelmModel> relmModels, int batchSize = 100, bool allowAutoIncrementColumns = false, bool allowPrimaryKeyColumns = false, bool allowUniqueColumns = false, bool allowAutoDateColumns = false)
```

## Parameters

`relmModels` [IEnumerable<IRelmModel>](#)

The collection of models to be written to the database. Each model must implement the [IRelmModel](#) interface.

`batchSize` [int](#)

The number of models to include in each batch. Must be a positive integer. The default value is 100.

`allowAutoIncrementColumns` [bool](#)

A value indicating whether columns with auto-increment constraints are allowed to be written. If [true](#), auto-increment columns will be included; otherwise, they will be excluded. The default value is [false](#).

`allowPrimaryKeyColumns` [bool](#)

A value indicating whether primary key columns are allowed to be written. If [true](#), primary key columns will be included; otherwise, they will be excluded. The default value is [false](#).

`allowUniqueColumns` [bool](#)

A value indicating whether columns with unique constraints are allowed to be written. If [true](#), unique columns will be included; otherwise, they will be excluded. The default value is [false](#).

`allowAutoDateColumns` [bool](#)

A value indicating whether columns with automatic date generation constraints are allowed to be written. If [true](#), auto-date columns will be included; otherwise, they will be excluded. The default value is [false](#).

## Returns

[int](#)

The total number of models successfully written to the database.

## Remarks

This method processes the provided models in batches to optimize database writes. The behavior of the write operation can be customized using the optional parameters to control which types of columns are included.

# Interface IRelmQuickFieldLoader

Namespace: [CoreRelm.Interfaces.RelmQuick](#)

Assembly: CoreRelm.dll

Provides access to the field loading quick context for use with RELM field loaders.

```
[Obsolete("IRelmQuickFieldLoader is deprecated. Please use the IRelmFieldLoader  
with IRelmContext.")]  
public interface IRelmQuickFieldLoader : IRelmFieldLoaderBase
```

## Inherited Members

[IRelmFieldLoaderBase.FieldName](#) , [IRelmFieldLoaderBase.KeyFields](#) ,  
[IRelmFieldLoaderBase.GetFieldData<S>\(ICollection<S\[\]>\)](#)

## Properties

### RelmContext

Gets the current quick context for Relm operations.

```
IRelmQuickContext RelmContext { get; }
```

### Property Value

[IRelmQuickContext](#)

# Namespace CoreRelm.Interfaces.Resolvers

## Interfaces

### [IRelmResolverBase](#)

Defines methods for obtaining a database connection string builder based on a specified connection type or connection string.

### [IRelmResolver\\_MySQL](#)

Defines methods for obtaining MySQL connection string builders based on connection type, name, or raw connection string.

# Interface IRelmResolverBase

Namespace: [CoreRelm.Interfaces.Resolvers](#)

Assembly: CoreRelm.dll

Defines methods for obtaining a database connection string builder based on a specified connection type or connection string.

```
public interface IRelmResolverBase
```

## Methods

### GetConnectionStringBuilder(Enum)

Creates and returns a new [DbConnectionStringBuilder](#) instance configured for the specified connection type.

```
DbConnectionStringBuilder GetConnectionStringBuilder(Enum ConnectionType)
```

#### Parameters

**ConnectionType** [Enum](#)

An enumeration value that specifies the type of database connection for which to create the connection string builder. The value must correspond to a supported connection type.

#### Returns

[DbConnectionStringBuilder](#)

A [DbConnectionStringBuilder](#) instance preconfigured for the specified connection type.

### GetConnectionStringBuilder(string)

Creates and returns a new [DbConnectionStringBuilder](#) instance initialized with the specified connection string.

```
DbConnectionStringBuilder GetConnectionStringBuilder(string ConnectionString)
```

## Parameters

**ConnectionString** [string](#)

The connection string to use for initializing the DbConnectionStringBuilder. Cannot be null or empty.

## Returns

[DbConnectionStringBuilder](#)

A [DbConnectionStringBuilder](#) initialized with the provided connection string.

# Interface IRelmResolver\_MySQL

Namespace: [CoreRelm.Interfaces.Resolvers](#)

Assembly: CoreRelm.dll

Defines methods for obtaining MySQL connection string builders based on connection type, name, or raw connection string.

```
public interface IRelmResolver_MySQL : IRelmResolverBase
```

## Inherited Members

[IRelmResolverBase.GetConnectionBuilder\(Enum\)](#) , [IRelmResolverBase.GetConnectionBuilder\(string\)](#)

## Remarks

This interface extends IRelmResolverBase to provide MySQL-specific connection string resolution. Implementations are expected to return configured MySQLConnectionStringBuilder instances for use in establishing MySQL database connections.

## Methods

### GetConnectionStringBuilderFromConnectionString(string)

Creates a new MySql.Data.MySqlClient.MySQLConnectionStringBuilder instance based on the specified connection string.

```
MySQLConnectionStringBuilder GetConnectionStringBuilderFromConnectionString(string  
connectionString)
```

#### Parameters

**connectionString** [string](#) ↗

The connection string to parse. Must be a valid MySQL connection string.

#### Returns

MySQLConnectionStringBuilder

A MySql.Data.MySqlClient.MySqlConnectionStringBuilder initialized with the values from the specified connection string.

## GetConnectionStringBuilderFromName(string)

Retrieves a new MySql.Data.MySqlClient.MySqlConnectionStringBuilder instance initialized with the specified connection string.

```
MySqlConnectionStringBuilder GetConnectionStringBuilderFromName(string ConnectionString)
```

### Parameters

**ConnectionString** [string](#)

The name or value of the connection string to use for initializing the connection string builder. Cannot be null or empty.

### Returns

MySqlConnectionStringBuilder

A MySql.Data.MySqlClient.MySqlConnectionStringBuilder configured with the provided connection string.

## GetConnectionStringBuilderFromType(Enum)

Creates a new MySql.Data.MySqlClient.MySqlConnectionStringBuilder instance configured for the specified connection type.

```
MySqlConnectionStringBuilder GetConnectionStringBuilderFromType(Enum ConnectionType)
```

### Parameters

**ConnectionType** [Enum](#)

An enumeration value that specifies the type of connection for which to create the connection string builder. Must be a valid value recognized by the method.

## Returns

MySqlConnectionStringBuilder

A MySql.Data.MySqlClient.MySqlConnectionStringBuilder instance configured according to the specified connection type.

# Namespace CoreRelm.Migrations

## Classes

[DefaultRelmMigrationSqlProvider](#)

# Class DefaultRelmMigrationSqlProvider

Namespace: [CoreRelm.Migrations](#)

Assembly: CoreRelm.dll

```
public sealed class DefaultRelmMigrationSqlProvider : IMigrationSqlProvider
```

## Inheritance

[object](#) ← DefaultRelmMigrationSqlProvider

## Implements

[IMigrationSqlProvider](#)

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

## Methods

Generate(string, string, string, string,  
List<ValidatedModelType>, bool, CancellationToken, bool, bool,  
MySqlDatabaseProvisioner)

```
public Task<MigrationGenerateResult> Generate(string migrationName, string  
stampUtc, string setName, string dbName, List<ValidatedModelType> modelsForDb,  
bool destructive, CancellationToken cancellationToken, bool quiet, bool doApply,  
MySqlDatabaseProvisioner provisioner)
```

## Parameters

migrationName [string](#)

stampUtc [string](#)

setName [string](#)

dbName [string](#)

`modelsForDb` [List](#) <[ValidatedModelType](#)>

`destructive` [bool](#)

`cancellationToken` [CancellationToken](#)

`quiet` [bool](#)

`doApply` [bool](#)

`provisioner`  [MySqlDatabaseProvisioner](#)

Returns

[Task](#) <[MigrationGenerateResult](#)>

# Namespace CoreRelm.Models

## Classes

### [RelmContext](#)

Provides a context for managing database connections, transactions, and data sets within the Relm data access layer. Enables querying, updating, and bulk operations on relational data models using configurable options and connection strategies.

### [RelmDataSet<T>](#)

Represents a dataset that provides functionality for managing, querying, and persisting a collection of entities of type `T`.

### [RelmDefaultDataLoader<T>](#)

Provides a default implementation of the [IRelmDataLoader<T>](#) interface for loading, querying, and writing data for a specified model type. This class is designed to work with models that implement the [IRelmModel](#) interface.

### [RelmExecutionCommand](#)

Represents a command and its associated expression for execution within the Relm framework, supporting additional and child commands for complex execution scenarios.

### [RelmModel](#)

Represents a base model for entities in the Relm framework, providing core attributes and functionality for data manipulation, database operations, and DTO generation.

### [RelmModelApartment](#)

Represents an apartment entity within the Relm data model, including user and member association information.

### [RelmQuickContext](#)

Provides a quick-start context for working with Relm data models and MySQL database connections, supporting configuration, data set management, and transactional operations.

### [RelmTrigger<T>](#)

Represents a database trigger associated with a specific table and operation type.

# Class RelmContext

Namespace: [CoreRelm.Models](#)

Assembly: CoreRelm.dll

Provides a context for managing database connections, transactions, and data sets within the Relm data access layer. Enables querying, updating, and bulk operations on relational data models using configurable options and connection strategies.

```
public class RelmContext : IDisposable, IRelmContext
```

## Inheritance

[object](#) ← RelmContext

## Implements

[IDisposable](#), [IRelmContext](#)

## Inherited Members

[object.Equals\(object\)](#), [object.Equals\(object, object\)](#), [object.GetHashCode\(\)](#), [object.GetType\(\)](#),  
[object.MemberwiseClone\(\)](#), [object.ReferenceEquals\(object, object\)](#), [object.ToString\(\)](#)

## Extension Methods

[DbContextExtensions.SetLockWaitTimeout\(IRelmContext, int\)](#)

## Remarks

RelmContext supports multiple initialization patterns, allowing connections to be established via connection strings, options builders, or existing MySQL connections and transactions. It automatically attaches and manages data sets defined in the context, and provides methods for querying, saving, and performing bulk operations. Transaction management is integrated, with explicit methods for beginning, committing, and rolling back transactions. RelmContext implements IDisposable to ensure proper cleanup of connections and attached resources. Thread safety is not guaranteed; concurrent access should be managed externally if required.

## Constructors

[RelmContext\(IRelmContext, bool, bool, bool, bool, int, bool, bool\)](#)

Initializes a new instance of the RelmContext class using the specified context options and configuration settings.

```
public RelmContext(IRelmContext relmContext, bool autoOpenConnection = true, bool  
autoOpenTransaction = false, bool allowUserVariables = false, bool convertZeroDateTime =  
false, int lockWaitTimeoutSeconds = 0, bool autoInitializeDataSets = true, bool  
autoVerifyTables = true)
```

## Parameters

**relmContext** [IRelmContext](#)

The context options provider used to configure the RelmContext. Cannot be null.

**autoOpenConnection** [bool](#)

true to automatically open the database connection when the context is created; otherwise, false.

**autoOpenTransaction** [bool](#)

true to automatically begin a transaction when the context is created; otherwise, false.

**allowUserVariables** [bool](#)

true to allow user-defined variables in database queries; otherwise, false.

**convertZeroDateTime** [bool](#)

true to convert zero date/time values to null when reading from the database; otherwise, false.

**lockWaitTimeoutSeconds** [int](#)

The maximum time, in seconds, to wait for a database lock before timing out. Specify 0 to use the default timeout.

**autoInitializeDataSets** [bool](#)

true to automatically initialize data sets when the context is created; otherwise, false.

**autoVerifyTables** [bool](#)

true to automatically verify table existence when the context is created; otherwise, false.

## Exceptions

## [ArgumentNullException](#)

Thrown if `relmContext` or `relmContext.ContextOptions` is null.

## RelmContext(RelmContextOptionsBuilder)

Initializes a new instance of the `RelmContext` class with the specified context options and configuration settings.

```
public RelmContext(RelmContextOptionsBuilder optionsBuilder)
```

### Parameters

#### `optionsBuilder` [RelmContextOptionsBuilder](#)

The options builder used to configure the context. Cannot be null.

### Remarks

All configuration settings are validated before the context is initialized. This constructor allows fine-grained control over connection and transaction behavior, as well as SQL compatibility options.

### Exceptions

## [ArgumentNullException](#)

Thrown if `optionsBuilder` is null.

## RelmContext(MySqlConnection, MySqlTransaction, bool, bool, bool, int, bool, bool)

Initializes a new instance of the `RelmContext` class using the specified MySQL connection, transaction, and context options.

```
public RelmContext(MySqlConnection connection, MySqlTransaction transaction, bool autoOpenConnection = true, bool allowUserVariables = false, bool convertZeroDateTime = false, int lockWaitTimeoutSeconds = 0, bool autoInitializeDataSets = true, bool autoVerifyTables = true)
```

## Parameters

**connection** MySqlConnection

The MySqlConnection to use for database operations. Must not be null and should be open or capable of being opened if **autoOpenConnection** is [true](#).

**transaction** MySqlTransaction

The MySqlTransaction to associate with the context. Can be null if no transaction is required.

**autoOpenConnection** [bool](#)

Specifies whether the context should automatically open the connection if it is not already open. If [true](#), the connection will be opened as needed.

**allowUserVariables** [bool](#)

Indicates whether user-defined variables are allowed in SQL statements executed by the context. Set to [true](#) to enable support for user variables.

**convertZeroDateTime** [bool](#)

Specifies whether zero date/time values from the database should be converted to DateTime.MinValue instead of throwing an exception. Set to [true](#) to enable conversion.

**lockWaitTimeoutSeconds** [int](#)

The maximum number of seconds to wait for a database lock before timing out. Specify 0 to use the default server setting.

**autoInitializeDataSets** [bool](#)

true to automatically initialize data sets when the context is created; otherwise, false.

**autoVerifyTables** [bool](#)

true to automatically verify table existence when the context is created; otherwise, false.

**RelmContext(MySqlConnection, bool, bool, bool, bool, int, bool, bool)**

Initializes a new instance of the RelmContext class using the specified MySQL connection and configuration options.

```
public RelmContext(MySqlConnection connection, bool autoOpenConnection = true, bool  
autoOpenTransaction = false, bool allowUserVariables = false, bool convertZeroDateTime =  
false, int lockWaitTimeoutSeconds = 0, bool autoInitializeDataSets = true, bool  
autoVerifyTables = true)
```

## Parameters

**connection** MySqlConnection

The MySqlConnection to use for database operations. Cannot be null.

**autoOpenConnection** [bool](#)

Specifies whether the database connection should be automatically opened when the context is created. If [true](#), the connection is opened; otherwise, it remains closed until explicitly opened.

**autoOpenTransaction** [bool](#)

Specifies whether a database transaction should be automatically started when the context is created. If [true](#), a transaction is started; otherwise, no transaction is started by default.

**allowUserVariables** [bool](#)

Specifies whether user-defined variables are allowed in SQL statements executed by this context. If [true](#), user variables are permitted.

**convertZeroDateTime** [bool](#)

Specifies whether zero date/time values from the database should be converted to DateTime.MinValue. If [true](#), zero date/time values are converted; otherwise, they are not.

**lockWaitTimeoutSeconds** [int](#)

The number of seconds to wait for a database lock before timing out. Specify 0 to use the default server setting.

**autoInitializeDataSets** [bool](#)

true to automatically initialize data sets when the context is created; otherwise, false.

**autoVerifyTables** [bool](#)

true to automatically verify table existence when the context is created; otherwise, false.

# RelmContext(Enum, bool, bool, bool, bool, int, bool, bool)

Initializes a new instance of the RelmContext class with the specified connection options and behavior settings.

```
public RelmContext(Enum connectionStringType, bool autoOpenConnection = true, bool  
autoOpenTransaction = false, bool allowUserVariables = false, bool convertZeroDateTime =  
false, int lockWaitTimeoutSeconds = 0, bool autoInitializeDataSets = true, bool  
autoVerifyTables = true)
```

## Parameters

**connectionStringType** [Enum](#)

The type of connection string to use for configuring the database context. Determines how the context connects to the underlying data source.

**autoOpenConnection** [bool](#)

true to automatically open the database connection when the context is created; otherwise, false. The default is true.

**autoOpenTransaction** [bool](#)

true to automatically begin a transaction when the context is created; otherwise, false. The default is false.

**allowUserVariables** [bool](#)

true to allow user-defined variables in database queries; otherwise, false. The default is false.

**convertZeroDateTime** [bool](#)

true to convert zero date/time values from the database to DateTime.MinValue; otherwise, false. The default is false.

**lockWaitTimeoutSeconds** [int](#)

The maximum number of seconds to wait for a database lock before timing out. Specify 0 to use the default timeout.

**autoInitializeDataSets** [bool](#)

true to automatically initialize data sets when the context is created; otherwise, false.

## autoVerifyTables [bool](#)

true to automatically verify table existence when the context is created; otherwise, false.

## Remarks

Use this constructor to customize context behavior such as connection management, transaction handling, and query options. These settings affect how the context interacts with the database and may impact performance or compatibility depending on the database provider.

## RelmContext(string, bool, bool, bool, bool, int, bool, bool)

Initializes a new instance of the RelmContext class with the specified connection details and configuration options.

```
public RelmContext(string connectionDetails, bool autoOpenConnection = true, bool  
autoOpenTransaction = false, bool allowUserVariables = false, bool convertZeroDateTime =  
false, int lockWaitTimeoutSeconds = 0, bool autoInitializeDataSets = true, bool  
autoVerifyTables = true)
```

## Parameters

### connectionDetails [string](#)

The connection string or details used to establish a database connection.

### autoOpenConnection [bool](#)

true to automatically open the database connection upon context initialization; otherwise, false.

### autoOpenTransaction [bool](#)

true to automatically begin a transaction when the context is initialized; otherwise, false.

### allowUserVariables [bool](#)

true to allow the use of user-defined variables in database queries; otherwise, false.

### convertZeroDateTime [bool](#)

true to convert zero date/time values to null when reading from the database; otherwise, false.

### lockWaitTimeoutSeconds [int](#)

The number of seconds to wait for a database lock before timing out. Specify 0 to use the default timeout.

#### autoInitializeDataSets [bool](#)

true to automatically initialize data sets when the context is created; otherwise, false.

#### autoVerifyTables [bool](#)

true to automatically verify table existence when the context is created; otherwise, false.

## Properties

### ContextOptions

Gets the options builder used to configure the context.

```
public RelmContextOptionsBuilder ContextOptions { get; }
```

### Property Value

#### [RelmContextOptionsBuilder](#)

### Remarks

Use this property to customize context-specific settings before building or initializing the context. Changes to the options should be made prior to finalizing the context configuration.

## Methods

### BeginTransaction()

Begins a database transaction on the current MySQL connection and returns a transaction object for managing the transaction lifecycle.

```
public MySqlTransaction BeginTransaction()
```

### Returns

## MySqlTransaction

A MySql.Data.MySqlClient.MySqlTransaction object representing the started transaction. If a transaction is already active, returns the existing transaction.

## Remarks

If a transaction is already in progress, this method returns the existing transaction rather than starting a new one. The returned transaction must be committed or rolled back to complete the operation. Ensure that the underlying database connection is open before calling this method.

## BulkTableWrite<T>(T, string, MySqlTransaction, Type, int, bool, bool, bool)

Writes a collection of data to a database table in bulk, optionally using batching and transaction support.

```
public int BulkTableWrite<T>(T source, string table = null, MySqlTransaction sqlTransaction = null, Type forceType = null, int batchSize = 100, bool allowAutoIncrementColumns = false, bool allowPrimaryKeyColumns = false, bool allowUniqueColumns = false)
```

## Parameters

**source** T

The source data to write to the table. This can be a collection or a single object of type T.

**table** [string](#)

The name of the target database table. If [null](#), the table name is inferred from the type T.

**sqlTransaction** MySqlTransaction

An optional MySql.Data.MySqlClient.MySqlTransaction to use for the bulk write operation. If [null](#), the operation is executed without an explicit transaction.

**forceType** [Type](#)

An optional type to override the inferred type of T when mapping columns. If [null](#), the type of T is used.

**batchSize** [int](#)

The maximum number of rows to write in each batch. Must be greater than zero.

### `allowAutoIncrementColumns bool`

Indicates whether auto-increment columns are included in the write operation. If [true](#), auto-increment columns are written; otherwise, they are excluded.

### `allowPrimaryKeyColumns bool`

Indicates whether primary key columns are included in the write operation. If [true](#), primary key columns are written; otherwise, they are excluded.

### `allowUniqueColumns bool`

Indicates whether unique columns are included in the write operation. If [true](#), unique columns are written; otherwise, they are excluded.

## Returns

### `int`

The number of rows successfully written to the database table.

## Type Parameters

### `T`

The type of the data objects to be written to the table.

## Remarks

This method is optimized for high-performance bulk inserts and can be used with or without an explicit transaction. Adjusting batch size may affect performance and resource usage. Column inclusion options allow fine-grained control over which table columns are written, which can be useful for tables with auto-increment, primary key, or unique constraints.

## CommitTransaction()

Commits the current database transaction, finalizing all changes made during the transaction.

### `public void CommitTransaction()`

## Remarks

If no active transaction exists, this method has no effect. After committing, the transaction is considered complete and cannot be rolled back. This method should be called only after all intended changes have been made within the transaction scope.

## Dispose()

Releases all resources used by the current instance of the class.

```
public void Dispose()
```

## Remarks

Call this method when you are finished using the object to free unmanaged resources and perform other cleanup operations. After calling `Dispose`, the object should not be used further.

## Dispose(bool)

Releases the unmanaged resources used by the object and optionally releases the managed resources.

```
protected virtual void Dispose(bool disposing)
```

## Parameters

`disposing` [bool](#)

true to release both managed and unmanaged resources; false to release only unmanaged resources.

## Remarks

This method is called by both the public `Dispose()` method and the finalizer. When `disposing` is true, this method disposes of managed resources such as attached properties that implement `IDisposable`. Override this method in a derived class to release additional resources.

## DoDatabaseWork(string, Dictionary<string, object>, bool, bool)

Executes a database operation using the specified SQL query and parameters, with optional exception handling and transaction support.

```
public void DoDatabaseWork(string query, Dictionary<string, object> parameters = null, bool throwException = true, bool useTransaction = false)
```

## Parameters

### query [string](#)

The SQL query to execute against the database. Cannot be null or empty.

### parameters [Dictionary](#)<[string](#), [object](#)>

An optional dictionary containing parameter names and values to be used with the query. If null, the query is executed without parameters.

### throwException [bool](#)

Specifies whether to throw an exception if the database operation fails. Set to [true](#) to throw exceptions; otherwise, errors are suppressed.

### useTransaction [bool](#)

Specifies whether to execute the operation within a database transaction. Set to [true](#) to use a transaction; otherwise, the operation is executed without transactional support.

## Remarks

If [useTransaction](#) is [true](#), the operation is performed within a transaction, which is committed if successful or rolled back on failure. When [throwException](#) is [false](#), errors are suppressed and no exception is thrown, but the operation may not complete as expected.

## DoDatabaseWork(string, Func<MySqlCommand, object>, bool, bool)

Executes a database operation using the specified SQL query and callback, with optional exception handling and transaction support.

```
public void DoDatabaseWork(string query, Func<MySqlCommand, object> actionCallback, bool throwException = true, bool useTransaction = false)
```

## Parameters

### query [string](#)

The SQL query to execute against the database. Cannot be null or empty.

### actionCallback [Func](#)<MySqlCommand, [object](#)>

A callback function that receives the prepared MySql.Data.MySqlClient.MySqlCommand and performs custom logic. The function should return an object representing the result of the operation.

### throwException [bool](#)

Specifies whether to throw an exception if the database operation fails. If [true](#), exceptions are propagated; otherwise, errors are suppressed.

### useTransaction [bool](#)

Specifies whether the database operation should be executed within a transaction. If [true](#), the operation is wrapped in a transaction.

## Remarks

Use this method to perform custom database work, such as executing queries or commands, with control over error handling and transactional behavior. The `actionCallback` allows you to define how the MySql.Data.MySqlClient.MySqlCommand is used, such as reading results or executing non-query commands.

## DoDatabaseWork<T>(string, Dictionary<string, object>, bool, bool)

Executes a database query and returns the result as the specified type.

```
public T DoDatabaseWork<T>(string query, Dictionary<string, object> parameters = null, bool throwException = true, bool useTransaction = false)
```

## Parameters

### query [string](#)

The SQL query to execute against the database. Cannot be null or empty.

### parameters [Dictionary](#)<[string](#), [object](#)>

An optional dictionary of parameter names and values to be used with the query. If null, no parameters are applied.

#### **throwException** [bool](#)

Specifies whether to throw an exception if the query fails. If [true](#), an exception is thrown on error; otherwise, the method returns the default value of [T](#).

#### **useTransaction** [bool](#)

Specifies whether to execute the query within a database transaction. If [true](#), the query is executed in a transaction.

### Returns

#### [T](#)

The result of the query cast to the specified type [T](#). Returns the default value of [T](#) if the query fails and **throwException** is [false](#).

### Type Parameters

#### [T](#)

The type of the result to return. Must be compatible with the query result.

### Remarks

If **useTransaction** is [true](#), the query is executed within a transaction, which may impact performance and rollback behavior. Ensure that **query** and **parameters** are valid for the target database.

## DoDatabaseWork<T>(string, Func<MySqlCommand, object>, bool, bool)

Executes a database operation using the specified query and callback, optionally within a transaction, and returns the result as the specified type.

```
public T DoDatabaseWork<T>(string query, Func<MySqlCommand, object> actionCallback, bool throwException = true, bool useTransaction = false)
```

### Parameters

## query [string](#)

The SQL query to execute against the database. Cannot be null or empty.

## actionCallback [Func](#)<MySqlCommand, [object](#)>

A callback that receives the prepared MySql.Data.MySqlClient.MySqlCommand and performs the desired operation. The result of this callback is returned as type [T](#).

## throwException [bool](#)

Specifies whether to throw an exception if the database operation fails. If [true](#), exceptions are thrown; otherwise, failures are handled silently.

## useTransaction [bool](#)

Specifies whether to execute the database operation within a transaction. If [true](#), the operation is performed in a transaction; otherwise, no transaction is used.

## Returns

### [T](#)

The result of the database operation as type [T](#).

## Type Parameters

### [T](#)

The type of the result returned by the database operation.

## Remarks

If [useTransaction](#) is [true](#), the operation is executed within a transaction, which is committed if successful or rolled back on failure. The behavior when an error occurs depends on the value of [throwException](#).

## EndConnection(bool)

Ends the current database connection and optionally commits the active transaction.

```
public void EndConnection(bool commitTransaction = true)
```

## Parameters

`commitTransaction bool`

Specifies whether to commit the active transaction before closing the connection. Set to `true` to commit; otherwise, the transaction will not be committed.

## Remarks

If a transaction is active and `commitTransaction` is `true`, the transaction is committed before the connection is closed. If no connection is open, this method has no effect.

## `~RelmContext()`

Finalizes the RelmContext instance and releases unmanaged resources before the object is reclaimed by garbage collection.

`protected ~RelmContext()`

## Remarks

This destructor is called automatically by the garbage collector when the object is no longer accessible. It ensures that any unmanaged resources held by the RelmContext are properly released. If you have already called `Dispose`, the finalizer will not release resources again.

## `FirstOrDefault<T>(Expression<Func<T, bool>>, bool)`

Returns the first element of type T that matches the specified predicate, or the default value if no such element is found.

```
public T FirstOrDefault<T>(Expression<Func<T, bool>> predicate, bool loadDataLoaders = false) where T : IRelmModel, new()
```

## Parameters

`predicate Expression<Func<T, bool>>`

An expression that defines the conditions the returned element must satisfy.

`loadDataLoaders bool`

true to load related data loaders for the returned element; otherwise, false. The default is false.

## Returns

T

The first element of type T that matches the predicate, or default(T) if no match is found.

## Type Parameters

T

The type of model to query. Must implement IRelmModel and have a parameterless constructor.

## GetBulkTableWriter<T>(string, bool, bool, bool, bool, bool)

Creates and returns a bulk table writer for efficiently inserting multiple records of type T into the database.

```
public BulkTableWriter<T> GetBulkTableWriter<T>(string insertQuery = null, bool  
useTransaction = false, bool throwException = true, bool allowAutoIncrementColumns = false,  
bool allowPrimaryKeyColumns = false, bool allowUniqueColumns = false)
```

## Parameters

`insertQuery` [string](#)

An optional custom SQL insert query to use for bulk operations. If null, a default query is generated based on the type T.

`useTransaction` [bool](#)

Specifies whether the bulk insert operation should be performed within a database transaction. Set to [true](#) to ensure atomicity; otherwise, [false](#).

`throwException` [bool](#)

Specifies whether exceptions encountered during the bulk operation should be thrown. If [true](#), exceptions are propagated; otherwise, errors are suppressed.

`allowAutoIncrementColumns` [bool](#)

Indicates whether auto-increment columns are included in the insert operation. Set to [true](#) to allow explicit values for such columns.

#### allowPrimaryKeyColumns [bool](#)

Indicates whether primary key columns are included in the insert operation. Set to [true](#) to allow explicit values for primary keys.

#### allowUniqueColumns [bool](#)

Indicates whether unique columns are included in the insert operation. Set to [true](#) to allow explicit values for unique columns.

Returns

#### [BulkTableWriter](#)<T>

A [BulkTableWriter](#)<T> instance configured for bulk insertion of entities of type T into the database.

Type Parameters

#### T

The type of entities to be written to the database table.

Remarks

Use this method to optimize large-scale data insertions. The returned [BulkTableWriter](#)<T> provides methods for writing batches of data efficiently. Ensure that the configuration flags match your table schema and insertion requirements.

## GetDataTable<T>(string, Dictionary<string, object>, bool)

Executes the specified query and returns a collection of results mapped to the specified type.

```
public IEnumerable<T> GetDataTable<T>(string query, Dictionary<string, object> parameters = null, bool throwException = true)
```

Parameters

#### query [string](#)

The SQL query to execute against the data source.

**parameters** [Dictionary<string, object>](#)

An optional dictionary of parameter names and values to be applied to the query. If null, the query is executed without parameters.

**throwException** [bool](#)

true to throw an exception if the query fails; otherwise, false to suppress exceptions and return an empty collection.

Returns

[IEnumerable<T>](#)

An enumerable collection of objects of type T representing the query results. Returns an empty collection if no results are found or if an error occurs and throwException is false.

Type Parameters

T

The type to which each result row will be mapped.

Remarks

The method maps each row in the result set to an instance of type T. If throwException is false and an error occurs during query execution, the method returns an empty collection instead of throwing an exception.

## GetDataObject<T>(string, Dictionary<string, object>, bool)

Retrieves a data object of the specified type by executing the provided query string with optional parameters.

```
public T GetDataObject<T>(string query, Dictionary<string, object> parameters = null, bool throwException = true) where T : IRelmModel, new()
```

Parameters

**query** [string](#)

The query string used to select the data object. Cannot be null or empty.

**parameters** [Dictionary](#)<[string](#), [object](#)>

An optional dictionary of parameter names and values to be used in the query. If null, no parameters are applied.

**throwException** [bool](#)

Specifies whether to throw an exception if the query fails. If [true](#), an exception is thrown on failure; otherwise, the method returns the default value of [T](#).

Returns

[T](#)

An instance of [T](#) representing the retrieved data object. Returns the default value of [T](#) if no data is found and **throwException** is [false](#).

Type Parameters

[T](#)

The type of data object to retrieve. Must implement [IRelmModel](#) and have a parameterless constructor.

## GetDataObjects<T>(string, Dictionary<string, object>, bool)

Executes the specified query and returns a collection of data objects of type [T](#) that match the query criteria.

```
public IEnumerable<T> GetDataObjects<T>(string query, Dictionary<string, object> parameters = null, bool throwException = true) where T : IRelmModel, new()
```

Parameters

**query** [string](#)

The query string used to select data objects. Must be a valid query for the underlying data source.

**parameters** [Dictionary](#)<[string](#), [object](#)>

An optional dictionary of parameter names and values to be used in the query. If [null](#), no parameters are applied.

#### **throwException** [bool](#)

Specifies whether to throw an exception if the query fails. If [true](#), an exception is thrown on error; otherwise, the method returns an empty collection.

#### Returns

#### [IEnumerable](#) <T>

An enumerable collection of data objects of type T that satisfy the query. Returns an empty collection if no matching objects are found or if the query fails and **throwException** is [false](#).

#### Type Parameters

##### T

The type of data object to return. Must implement [IRelmModel](#) and have a parameterless constructor.

## GetDataRow(string, Dictionary<string, object>, bool)

Executes the specified SQL query and returns the first matching data row from the result set.

```
public DataRow GetDataRow(string query, Dictionary<string, object> parameters = null, bool throwException = true)
```

#### Parameters

##### query [string](#)

The SQL query to execute. Must be a valid statement that returns at least one row.

##### parameters [Dictionary](#) <[string](#), [object](#)>

An optional dictionary of parameter names and values to be applied to the query. If null, the query is executed without parameters.

##### **throwException** [bool](#)

true to throw an exception if no matching row is found; otherwise, false to return null.

Returns

## [DataRow](#)

A DataRow containing the first result of the query if found; otherwise, null if no matching row exists and throwException is false.

## GetDataSet(Type)

Retrieves an instance of a data set of the specified type.

```
public IRelmDataSetBase GetDataSet(Type dataSetType)
```

Parameters

### [dataSetType](#) [Type](#)

The type of the data set to retrieve. Must implement [IRelmDataSetBase](#).

Returns

## [IRelmDataSetBase](#)

An instance of [IRelmDataSetBase](#) corresponding to the specified type.

## GetDataSet(Type, bool)

Retrieves the data set instance associated with the specified entity type. If the data set is not initialized, it can be created automatically or an exception can be thrown based on the provided option.

```
public IRelmDataSetBase GetDataSet(Type dataSetType, bool throwException)
```

Parameters

### [dataSetType](#) [Type](#)

The type of the entity for which to retrieve the data set. Must correspond to a table that exists in the current database.

## `throwException` [bool](#)

Specifies whether to throw an exception if the data set for the specified type is not initialized. If [true](#), an exception is thrown; otherwise, a new data set is created if possible.

## Returns

### [IRelmDataSetBase](#)

An instance of [IRelmDataSetBase](#) representing the data set for the specified entity type. Returns a newly created instance if the data set was not previously initialized and `throwException` is [false](#).

## Remarks

If the data set for the specified type is not already initialized and `throwException` is [false](#), a default data loader is used to create and initialize the data set. This method only operates on types that are mapped to existing tables in the current database.

## Exceptions

### [InvalidOperationException](#)

Thrown if the specified type does not correspond to a table in the current database, if no attached property is found for the type, or if the data set cannot be initialized and `throwException` is [true](#).

## GetDataSetType(Type)

Retrieves an instance of a data set corresponding to the specified type.

```
public IRelmDataSetBase GetDataSetType(Type dataSetType)
```

## Parameters

### `dataSetType` [Type](#)

The type of the data set to retrieve. Must be a valid type that implements the required data set interface.

## Returns

### [IRelmDataSetBase](#)

An object representing the data set of the specified type.

## GetDataSetType(Type, bool)

Gets the dataset of the given type.

```
public IRelmDataSetBase GetDataSetType(Type dataSetType, bool throwException)
```

Parameters

**dataSetType** [Type](#)

Type of the dataset.

**throwException** [bool](#)

Whether to throw an exception if the dataset is not found.

Returns

[IRelmDataSetBase](#)

An IDALDataSetBase instance of the given type.

Exceptions

[InvalidOperationException](#)

Thrown when no matching dataset is found.

## GetDataSetType<T>()

Retrieves a data set instance for the specified model type.

```
public IRelmDataSet<T> GetDataSetType<T>() where T : IRelmModel, new()
```

Returns

[IRelmDataSet](#)<T>

An [IRelmDataSet<T>](#) instance associated with the specified model type.

## Type Parameters

T

The type of model for which to retrieve the data set. Must implement [IRelmModel](#) and have a parameterless constructor.

## GetDataSetType<T>(bool)

Gets the dataset of the given type.

```
public IRelmDataSet<T> GetDataSetType<T>(bool throwException) where T : IRelmModel, new()
```

## Parameters

throwException [bool](#)

Whether to throw an exception if the dataset is not found.

## Returns

[IRelmDataSet<T>](#)

An instance of IDALDataSet of the specified type.

## Type Parameters

T

The type of the dataset, which should inherit from CS\_DbModel.

## Exceptions

[InvalidOperationException](#)

Thrown when no matching dataset is found.

## GetDataSet<T>()

Retrieves a data set for the specified model type.

```
public IRelmDataSet<T> GetDataSet<T>() where T : IRelmModel, new()
```

Returns

[IRelmDataSet<T>](#)

An [IRelmDataSet<T>](#) instance for the specified model type [T](#).

Type Parameters

[T](#)

The type of model to retrieve. Must implement [IRelmModel](#) and have a parameterless constructor.

## GetDataSet<T>(bool)

Retrieves a strongly typed data set for the specified model type.

```
public IRelmDataSet<T> GetDataSet<T>(bool throwException) where T : IRelmModel, new()
```

Parameters

[throwException](#) [bool](#)

Specifies whether to throw an exception if the data set cannot be retrieved. If [true](#), an exception is thrown on failure; otherwise, [null](#) is returned.

Returns

[IRelmDataSet<T>](#)

An [IRelmDataSet<T>](#) instance representing the data set for the specified model type, or [null](#) if the data set cannot be retrieved and [throwException](#) is [false](#).

Type Parameters

[T](#)

The type of model for which to retrieve the data set. Must implement [IRelmModel](#) and have a parameterless constructor.

## GetDataTable(string, Dictionary<string, object>, bool)

Executes the specified SQL query and returns the results as a [DataTable](#).

```
public DataTable GetDataTable(string query, Dictionary<string, object> parameters = null,  
    bool throwException = true)
```

### Parameters

**query** [string](#)

The SQL query to execute against the database. Must be a valid query string.

**parameters** [Dictionary](#)<[string](#), [object](#)>

An optional dictionary containing parameter names and values to be used in the query. If [null](#), no parameters are applied.

**throwException** [bool](#)

Specifies whether to throw an exception if the query fails. If [true](#), an exception is thrown on error; otherwise, the method returns [null](#).

### Returns

[DataTable](#)

A [DataTable](#) containing the results of the query, or [null](#) if the query fails and **throwException** is [false](#).

### Remarks

The returned [DataTable](#) will contain all rows and columns produced by the query. If the query does not return any results, the [DataTable](#) will be empty. Ensure that the query and parameters are properly formatted to avoid runtime errors.

## GetIdFromInternalId(string, string)

Retrieves the external identifier associated with the specified internal identifier for a given table.

```
public string GetIdFromInternalId(string table, string InternalId)
```

## Parameters

**table** [string](#)

The name of the table in which to look up the identifier. Cannot be null or empty.

**InternalId** [string](#)

The internal identifier whose corresponding external identifier is to be retrieved. Cannot be null or empty.

## Returns

[string](#)

A string containing the external identifier corresponding to the specified internal identifier. Returns null if no matching identifier is found.

## GetLastInsertId()

Retrieves the identifier of the most recently inserted row for the current context.

```
public string GetLastInsertId()
```

## Returns

[string](#)

A string containing the last inserted row identifier. Returns an empty string if no row has been inserted.

## GetScalar<T>(string, Dictionary<string, object>, bool)

Executes the specified SQL query and returns the first column of the first row in the result set, cast to the specified type.

```
public T GetScalar<T>(string query, Dictionary<string, object> parameters = null, bool throwException = true)
```

## Parameters

### query [string](#)

The SQL query to execute. Must be a valid statement that returns a single value.

### parameters [Dictionary](#)<[string](#), [object](#)>

An optional dictionary of parameter names and values to be applied to the query. If null, the query is executed without parameters.

### throwException [bool](#)

true to throw an exception if the query fails or returns no result; otherwise, false to return the default value of type T.

## Returns

### T

The value of the first column of the first row in the result set, cast to type T. Returns the default value of T if no result is found and throwException is false.

## Type Parameters

### T

The type to which the scalar result will be cast.

## Remarks

Use this method to efficiently retrieve a single value from the database, such as a count or aggregate. If throwException is set to true and the query fails or returns no result, an exception will be thrown.

## Get<T>(bool)

Retrieves a collection of entities of type T from the associated data set.

```
public ICollection<T> Get<T>(bool loadDataLoaders = false) where T : IRelmModel, new()
```

## Parameters

### `loadDataLoaders` [bool](#)

Specifies whether to load associated data loaders for each entity. Set to [true](#) to include related data loaders; otherwise, only the entities are loaded.

## Returns

### [ICollection](#)<T>

A collection containing all entities of type `T` from the data set. The collection is empty if no entities are present.

## Type Parameters

### `T`

The type of model to retrieve. Must implement [IRelmModel](#) and have a parameterless constructor.

## Exceptions

### [InvalidOperationException](#)

Thrown if the data set for type `T` is not initialized.

## Get<T>(Expression<Func<T, bool>>, bool)

Retrieves a collection of entities of type `T` that satisfy the specified predicate.

```
public ICollection<T> Get<T>(Expression<Func<T, bool>> predicate, bool loadDataLoaders = false) where T : IRelmModel, new()
```

## Parameters

### `predicate` [Expression](#)<[Func](#)<T, [bool](#)>>

An expression that defines the conditions each entity must satisfy to be included in the result.

## `loadDataLoaders` [bool](#)

true to load related data loaders for each entity; otherwise, false. The default is false.

## Returns

### [ICollection](#)<T>

A collection of entities of type T that match the specified predicate. The collection will be empty if no entities satisfy the predicate.

## Type Parameters

### T

The type of entity to retrieve. Must implement IRelmModel and have a parameterless constructor.

## HasDataSet(Type, bool)

Checks if the DALDataSet of a specific type is attached to the current DALContext instance.

```
public bool HasDataSet(Type dataSetType, bool throwException = true)
```

## Parameters

### `dataSetType` [Type](#)

The System.Type of the dataset to check.

### `throwException` [bool](#)

Whether to throw an exception if the dataset is not found.

## Returns

### [bool](#)

True if the dataset exists, otherwise false.

## HasDataSet<T>(bool)

Checks if the DALDataSet of a specific type is attached to the current DALContext instance.

```
public bool HasDataSet<T>(bool throwException = true) where T : IRelmModel, new()
```

Parameters

**throwException** [bool](#)

Returns

[bool](#)

True if the dataset exists, otherwise false.

Type Parameters

**T**

The type of the dataset, which should inherit from CS\_DbModel.

## HasTransaction()

Determines whether there is an active database transaction associated with the current context.

```
public bool HasTransaction()
```

Returns

[bool](#)

true if a database transaction is currently active; otherwise, false.

## OnConfigure(RelmContextOptionsBuilder)

Configures a triggerable event for after the database options are set, the connection is open, and any transactions have been started.

```
public virtual void OnConfigure(RelmContextOptionsBuilder OptionsBuilder)
```

## Parameters

### [OptionsBuilder](#) [RelmContextOptionsBuilder](#)

A builder used to configure options for the context. Cannot be null.

## Remarks

Override this method to customize how the context is configured, such as specifying database providers or additional options. This method is typically called by the framework during context initialization.

## RollbackTransaction()

Rolls back the current database transaction, reverting all changes made during the transaction.

```
public void RollbackTransaction()
```

## Remarks

If no active transaction exists, this method has no effect. After calling this method, the transaction is considered closed and cannot be committed.

## RollbackTransactions()

Rolls back the current database transaction, reverting all changes made during the transaction.

```
public void RollbackTransactions()
```

## Remarks

If no active transaction exists, this method has no effect. After calling this method, the transaction is considered closed and cannot be committed.

## Run<T>(string, Dictionary<string, object>)

Executes the specified query and returns a collection of data objects of type T that match the query criteria.

```
public ICollection<T> Run<T>(string query, Dictionary<string, object> parameters = null)  
where T : IRelmModel, new()
```

## Parameters

### query [string](#)

The query string to execute. Cannot be null or empty.

### parameters [Dictionary](#)<[string](#), [object](#)>

An optional dictionary of parameters to be used with the query. If null, the query is executed without parameters.

## Returns

### [ICollection](#)<T>

A collection of objects of type T that satisfy the query. The collection will be empty if no matching objects are found.

## Type Parameters

### T

The type of data object to return. Must implement IRelmModel and have a parameterless constructor.

## Exceptions

### [ArgumentNullException](#)

Thrown if query is null or empty.

## SaveAll()

Saves all attached data sets by invoking their respective Save methods.

```
public void SaveAll()
```

## Remarks

This method attempts to persist changes for each data set currently attached to the context. If any data set fails to save, an exception may be thrown from the underlying Save method. The operation is not transactional; if saving one data set fails, others may still be saved successfully.

## SetDataLoader<T>(IRelmDataLoader<T>)

Sets the data loader to be used for the specified data set type.

```
public void SetDataLoader<T>(IRelmDataLoader<T> dataLoader) where T : RelmModel, new()
```

### Parameters

**dataLoader** [IRelmDataLoader<T>](#)

The data loader instance that will be associated with the data set of type **T**.

### Type Parameters

**T**

The type of model for which the data loader is being set. Must inherit from RelmModel and have a parameterless constructor.

### Exceptions

[InvalidOperationException](#)

Thrown if a data set for type **T** does not exist.

## StartConnection(bool, int)

Opens the database connection if it is not already open and optionally begins a new transaction.

```
public void StartConnection(bool autoOpenTransaction = false, int lockWaitTimeoutSeconds = 0)
```

### Parameters

**autoOpenTransaction** [bool](#)

true to automatically begin a new transaction after opening the connection; otherwise, false.

### lockWaitTimeoutSeconds [int](#)

The lock wait timeout, in seconds, to set for the session. Specify a positive value to configure the timeout; otherwise, no change is made.

## Remarks

If the connection is opened and lockWaitTimeoutSeconds is greater than zero, the session's lock wait timeout and transaction isolation level are set. If autoOpenTransaction is true and the connection is open, a new transaction is started automatically.

## Exceptions

### [InvalidOperationException](#)

Thrown if the database connection does not exist.

## Where<T>(Expression<Func<T, bool>>)

Filters the elements of the data set based on a specified predicate.

```
public IRelmDataSet<T> Where<T>(Expression<Func<T, bool>> predicate) where T :  
IRelmModel, new()
```

## Parameters

### [predicate Expression](#)<[Func](#)<T, [bool](#)>>

An expression that defines the conditions each element must satisfy to be included in the result.  
Cannot be null.

## Returns

### [IRelmDataSet](#)<T>

An [IRelmDataSet](#)<T> containing elements that match the specified predicate.

## Type Parameters

T

The type of model contained in the data set. Must implement [IRelmModel](#) and have a parameterless constructor.

## Remarks

The returned data set is filtered according to the provided predicate and may be further queried or enumerated. This method does not modify the original data set.

## Exceptions

### [ArgumentNullException](#)

Thrown if `predicate` is null.

### [InvalidOperationException](#)

Thrown if the data set for type `T` is not initialized.

## WriteToDatabase(IRelmModel, int, bool, bool, bool, bool)

Writes the specified data model to the database using configurable options for batch size and column handling.

```
public int WriteToDatabase(IRelmModel relmModel, int batchSize = 100, bool  
allowAutoIncrementColumns = false, bool allowPrimaryKeyColumns = false, bool  
allowUniqueColumns = false, bool allowAutoDateColumns = false)
```

## Parameters

### `relmModel` [IRelmModel](#)

The data model to be written to the database. Cannot be null.

### `batchSize` [int](#)

The maximum number of records to write in each batch operation. Must be greater than zero. The default is 100.

### `allowAutoIncrementColumns` [bool](#)

Specifies whether columns with auto-increment attributes are included in the write operation. Set to [true](#) to allow writing to auto-increment columns; otherwise, [false](#).

## allowPrimaryKeyColumns [bool](#)

Specifies whether primary key columns are included in the write operation. Set to [true](#) to allow writing to primary key columns; otherwise, [false](#).

## allowUniqueColumns [bool](#)

Specifies whether unique columns are included in the write operation. Set to [true](#) to allow writing to unique columns; otherwise, [false](#).

## allowAutoDateColumns [bool](#)

Specifies whether columns with automatic date attributes are included in the write operation. Set to [true](#) to allow writing to auto-date columns; otherwise, [false](#).

## Returns

### [int](#)

The number of records successfully written to the database.

## Remarks

Adjusting the batch size can impact performance, especially for large data sets. Enabling writing to auto-increment, primary key, unique, or auto-date columns may result in constraint violations depending on the database schema.

## WriteToDatabase(IEnumerable<IRelmModel>, int, bool, bool, bool, bool)

Writes the specified collection of Relm models to the database in batches, with options to control how certain column types are handled during insertion.

```
public int WriteToDatabase(IEnumerable<IRelmModel> relmModels, int batchSize = 100, bool  
allowAutoIncrementColumns = false, bool allowPrimaryKeyColumns = false, bool  
allowUniqueColumns = false, bool allowAutoDateColumns = false)
```

## Parameters

### relmModels [IEnumerable](#)<[IRelmModel](#)>

The collection of Relm models to be written to the database. Cannot be null.

## batchSize [int](#)

The maximum number of models to include in each batch write operation. Must be greater than zero.

## allowAutoIncrementColumns [bool](#)

Specifies whether columns marked as auto-increment are included in the write operation. If [true](#), auto-increment columns are written; otherwise, they are excluded.

## allowPrimaryKeyColumns [bool](#)

Specifies whether primary key columns are included in the write operation. If [true](#), primary key columns are written; otherwise, they are excluded.

## allowUniqueColumns [bool](#)

Specifies whether unique columns are included in the write operation. If [true](#), unique columns are written; otherwise, they are excluded.

## allowAutoDateColumns [bool](#)

Specifies whether columns with automatic date values are included in the write operation. If [true](#), auto-date columns are written; otherwise, they are excluded.

## Returns

### [int](#)

The number of models successfully written to the database.

## Remarks

If the collection contains more models than the specified batch size, the write operation is performed in multiple batches. The behavior of column inclusion is determined by the corresponding boolean parameters. This method does not guarantee transactional integrity across batches.

# Class RelmDataSet<T>

Namespace: [CoreRelm.Models](#)

Assembly: CoreRelm.dll

Represents a dataset that provides functionality for managing, querying, and persisting a collection of entities of type **T**.

```
public class RelmDataSet<T> : IRelmDataSet<T>, ICollection<T>, IEnumerable<T>, IEnumerable, IRelmDataSetBase where T : IRelmModel, new()
```

## Type Parameters

**T**

The type of the entities in the dataset. Must implement [IRelmModel](#) and have a parameterless constructor.

## Inheritance

[object](#) ← RelmDataSet<T>

## Implements

[IRelmDataSet](#)<T>, [ICollection](#)<T>, [IEnumerable](#)<T>, [IEnumerable](#), [IRelmDataSetBase](#)

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

## Extension Methods

[ListExtensions.LoadDataLoaderField<T, R>\(ICollection<T>, IRelmContext, Expression<Func<T, R>>\)](#) ,  
[ListExtensions.LoadDataLoaderField<T, R>\(ICollection<T>, IRelmQuickContext, Expression<Func<T, R>>\)](#) ,  
[ListExtensions.LoadDataLoaderField<T, R>\(ICollection<T>, RelmContextOptionsBuilder, Expression<Func<T, R>>\)](#) ,  
[ListExtensions.LoadForeignKeyField<T, R>\(ICollection<T>, IRelmContext, Expression<Func<T, ICollection<R>>>\)](#) ,  
[ListExtensions.LoadForeignKeyField<T, R>\(ICollection<T>, IRelmContext, Expression<Func<T, ICollection<R>>>, Expression<Func<R, object>>\)](#) ,  
[ListExtensions.LoadForeignKeyField<T, R>\(ICollection<T>, IRelmContext, Expression<Func<T, R>>\)](#) ,



```
ListExtensions.LoadForeignKeyField<T, R, S>(ICollection<T>, RelmContextOptionsBuilder,
Expression<Func<T, R>>, IRelmDataLoader<S>, Expression<Func<R, object>>),
ListExtensions.FlattenTreeObject<T>(IEnumerable<T>, Func<T, ICollection<T>>),
ListExtensions.GenerateDTO<T>(IEnumerable<T>, ICollection<string>, ICollection<string>, string,
Func<IRelmModel, Dictionary<string, object>>),
ListExtensions.WriteToDatabase<T>(IEnumerable<T>, IRelmContext, string, Type, int, string, bool, bool,
bool, bool),
ListExtensions.WriteToDatabase<T>(IEnumerable<T>, IRelmQuickContext, string, Type, int, string, bool,
bool, bool, bool),
ListExtensions.WriteToDatabase<T>(IEnumerable<T>, MySqlConnection, MySqlTransaction, string, Type,
int, string, bool, bool, bool, bool),
ListExtensions.WriteToDatabase<T>(IEnumerable<T>, Enum, string, Type, bool, int, string, bool, bool,
bool, bool)
```

## Remarks

The [RelmDataSet<T>](#) class provides a flexible API for working with collections of entities, including support for filtering, ordering, grouping, and managing relationships between entities. It integrates with a data loader to dynamically load and persist data as needed. The dataset also supports deferred execution for query operations, ensuring efficient data handling. This class is designed to work within the context of a relational data management system, leveraging the provided context and data loader to manage the lifecycle and scope of the dataset.

## Constructors

### RelmDataSet(IRelmContext, IRelmDataLoader<T>)

Initializes a new instance of the [RelmDataSet<T>](#) class with the specified context and data loader.

```
public RelmDataSet(IRelmContext currentContext, IRelmDataLoader<T> dataLoader)
```

#### Parameters

##### currentContext [IRelmContext](#)

The current context used to manage the lifecycle and scope of the dataset. Cannot be [null](#).

##### dataLoader [IRelmDataLoader<T>](#)

The data loader responsible for loading and managing the data for the dataset. Cannot be [null](#).

## Exceptions

### [ArgumentNullException](#)

Thrown if `currentContext` is [null](#).

## RelmDataSet(IRelmQuickContext, IRelmDataLoader<T>)

Initializes a new instance of the [RelmDataSet<T>](#) class with the specified context and data loader.

```
public RelmDataSet(IRelmQuickContext currentContext, IRelmDataLoader<T> dataLoader)
```

## Parameters

### `currentContext` [IRelmQuickContext](#)

The current quick context used to manage the dataset's operational scope. Cannot be [null](#).

### `dataLoader` [IRelmDataLoader](#)<T>

The data loader responsible for loading and managing the dataset's data. Cannot be [null](#).

## Exceptions

### [ArgumentNullException](#)

Thrown if `currentContext` is [null](#).

## Properties

### IsReadOnly

Gets a value indicating whether the collection is read-only.

```
public bool IsReadOnly { get; }
```

## Property Value

[bool](#)

# Modified

Gets or sets a value indicating whether the object has been modified.

```
public bool Modified { get; set; }
```

Property Value

[bool](#)

## Methods

### Add(IICollection<T>)

Adds the specified collection of items to the current instance and persists the change.

```
public int Add(IICollection<T> items)
```

Parameters

[items](#) [IICollection](#)<T>

The collection of items to add. Cannot be null.

Returns

[int](#)

The number of items successfully added.

### Add(IICollection<T>, bool)

Adds a collection of items to the current instance and optionally persists the changes.

```
public int Add(IICollection<T> items, bool Persist)
```

Parameters

**items** [ICollection<T>](#)

The collection of items to add. Cannot be null.

**Persist** [bool](#)

A value indicating whether the changes should be persisted after adding the items. If [true](#), the method persists the changes and returns the result of the save operation. If [false](#), the method only adds the items and returns the count of items added.

Returns

[int](#)

The number of items added if **Persist** is [false](#), or the result of the save operation if **Persist** is [true](#).

## Add(T)

Adds the specified item to the collection and persists the change.

```
public int Add(T item)
```

Parameters

**item** T

The item to add to the collection. Cannot be [null](#).

Returns

[int](#)

The index at which the item was added.

## Add(T, bool)

Adds the specified item to the collection and optionally persists it to the database.

```
public int Add(T item, bool Persist)
```

## Parameters

### **item** T

The item to add to the collection.

### **Persist** bool

A value indicating whether the item should be persisted to the database. If [true](#), the item is written to the database; otherwise, it is only added to the collection.

## Returns

### [int](#)

An integer indicating the result of the operation. Returns 1 if the item is added to the collection without persistence, or the result of the database write operation if **Persist** is [true](#).

## Remarks

If **Persist** is [true](#), the method writes the item to the database using the current database context. The database connection and transaction details are determined by the context options. If **Persist** is [false](#), the collection's state is marked as modified.

## Clear()

Removes all items from the collection.

### `public void Clear()`

## Remarks

After calling this method, the collection will be empty. If the collection is already empty, this method has no effect.

## Contains(T)

Determines whether the collection contains the specified item.

### `public bool Contains(T item)`

## Parameters

### **item** T

The item to locate in the collection.

## Returns

### [bool](#)

[true](#) if the specified item is found in the collection; otherwise, [false](#).

## Remarks

If the collection is null, this method returns [false](#).

## CopyTo(T[], int)

Copies the elements of the collection to the specified array, starting at the specified index.

```
public void CopyTo(T[] array, int arrayIndex)
```

## Parameters

### **array** T[]

The one-dimensional array that is the destination of the elements copied from the collection. The array must have zero-based indexing.

### **arrayIndex** [int](#)

The zero-based index in the **array** at which copying begins.

## Remarks

This method copies the elements in the same order they are enumerated by the collection.

## Count()

Adds a count operation to the current query expression.

```
public IRelmDataSet<T> Count()
```

Returns

[IRelmDataSet<T>](#)

The current [IRelmDataSet<T>](#) instance with the count operation applied.

Remarks

This method modifies the query by appending a count operation, allowing further query composition. The actual count is not executed until the query is evaluated.

## Count(Expression<Func<T, bool>>)

Adds a count operation with the specified filter to the current query.

```
public IRelmDataSet<T> Count(Expression<Func<T, bool>> predicate)
```

Parameters

**predicate** [Expression<Func<T, bool>>](#)

An expression that defines the conditions that the data must satisfy to be included in the count operation.

Returns

[IRelmDataSet<T>](#)

The current dataset instance with the count operation applied. This enables method chaining for building complex queries.

Remarks

This method does not execute the count operation immediately. Instead, it adds the count expression to the query, which will be executed when the query is materialized.

## DistinctBy(Expression<Func<T, object>>)

Filters the dataset to include only distinct elements based on the specified key selector.

```
public IRelmDataSet<T> DistinctBy(Expression<Func<T, object>> predicate)
```

## Parameters

**predicate** [Expression](#)<[Func](#)<T, object>>

An expression that specifies the key used to determine distinct elements. The key is derived from the properties of the dataset elements.

## Returns

[IRelmDataSet](#)<T>

A dataset containing only distinct elements based on the specified key.

## Remarks

This method modifies the current dataset by adding a distinct operation to the query. The distinctness is determined by the value of the key selected by the **predicate** expression.

## Entry(T)

Clears the current dataset, then adds the specified item to the dataset and marks the dataset as modified.

```
public IRelmDataSet<T> Entry(T Item)
```

## Parameters

**Item** T

The item to be added to the dataset. Cannot be null.

## Returns

[IRelmDataSet](#)<T>

The current dataset instance, allowing for method chaining.

## Remarks

The item is added to the cleared dataset without persisting the change. The dataset's state is marked as modified.

## Entry(T, bool)

Clears the current dataset, then adds the specified item to the dataset and optionally persists it.

```
public IRelmDataSet<T> Entry(T Item, bool Persist = true)
```

### Parameters

#### Item T

The item to add to the dataset. Cannot be [null](#).

#### Persist bool

A value indicating whether the item should be persisted. The default value is [true](#).

### Returns

[IRelmDataSet<T>](#)

The current dataset instance with the added item.

## Remarks

This method modifies the dataset by clearing then adding the specified item and marks the dataset as modified. If **Persist** is [true](#), the item will be persisted.

## Find(int)

Finds and returns the first item with the specified identifier.

```
public T Find(int id)
```

### Parameters

**id** [int](#)

The unique identifier of the item to find.

Returns

T

The first item that matches the specified identifier, or [null](#) if no such item is found.

Remarks

This method searches for an item based on its [Id](#) property. If multiple items share the same identifier, only the first match is returned.

## Find(string)

Finds and returns the first element in the collection that matches the specified internal identifier.

```
public T Find(string InternalId)
```

Parameters

**InternalId** [string](#)

The internal identifier to search for. Cannot be null.

Returns

T

The first element that matches the specified internal identifier, or [null](#) if no match is found.

## FirstOrDefault()

Returns the first element of the sequence, or the default value for the type if the sequence is empty.

```
public T FirstOrDefault()
```

Returns

T

The first element of the sequence, or the default value for the type T if the sequence is empty.

Remarks

This method retrieves the first element of the sequence. If the sequence contains no elements, it returns the default value for the type T (e.g., [null](#) for reference types, or the default zero value for value types).

## FirstOrDefault(bool)

Returns the first element of the collection, or the default value for the type if the collection is empty.

```
public T FirstOrDefault(bool loadItems)
```

Parameters

[loadItems](#) [bool](#)

A value indicating whether to load items into the collection before retrieving the first element.

Returns

T

The first element of the collection if it contains any elements; otherwise, the default value for the type T.

Remarks

If [loadItems](#) is [true](#), the method ensures that items are loaded into the collection before attempting to retrieve the first element.

## FirstOrDefault(Expression<Func<T, bool>>)

Loads items from the database using the existing predicate conditions and then returns the first element of a sequence that satisfies the specified condition, or the default value for the type if no such element is found.

```
public T FirstOrDefault(Expression<Func<T, bool>> predicate)
```

## Parameters

**predicate** [Expression<Func<T, bool>>](#)

A function to test each element for a condition. The function must not be [null](#).

## Returns

T

The first element in the sequence that satisfies the condition defined by **predicate**, or the default value of type T if no such element is found.

## FirstOrDefault(Expression<Func<T, bool>>, bool)

Retrieves the first element of a sequence that satisfies the specified condition, or a default value if no such element is found.

```
public T FirstOrDefault(Expression<Func<T, bool>> predicate, bool loadItems)
```

## Parameters

**predicate** [Expression<Func<T, bool>>](#)

An expression that defines the condition to test each element for. If [null](#), no condition is applied.

**loadItems** [bool](#)

A value indicating whether to load the items from the database using the existing predicate conditions before evaluating the condition. If [true](#), the items are loaded from the database; otherwise, the method operates on the existing items.

## Returns

T

The first element that satisfies the condition defined by **predicate**, or the default value of type T if no such element is found.

## Remarks

If `loadItems` is [true](#), the method loads the items and applies the condition specified by `predicate`. If `loadItems` is [false](#), the method evaluates the condition on the existing items.

## GetEnumerator()

Returns an enumerator that iterates through the collection.

```
public IEnumrator<T> GetEnumerator()
```

Returns

[IEnumrator](#)<T>

An enumerator for the collection of items.

## Remarks

If the collection is not already loaded, it will attempt to load the items dynamically. If no items are available, the enumerator will iterate over an empty collection.

## GroupBy(Expression<Func<T, object>>)

Groups the elements of the dataset based on the specified key selector.

```
public IRelmDataSet<T> GroupBy(Expression<Func<T, object>> predicate)
```

Parameters

`predicate` [Expression](#)<[Func](#)<T, [object](#)>>

An expression that specifies the key to group by. The key is derived from the elements of the dataset.

Returns

[IRelmDataSet](#)<T>

A dataset grouped by the specified key.

## Remarks

The grouping is applied to the underlying query, modifying the dataset to reflect the grouping operation. This method does not execute the query; it only adds the grouping condition to the query definition.

## Limit(int)

Limits the number of results returned by the dataset to the specified count.

```
public IRelmDataSet<T> Limit(int limitCount)
```

### Parameters

**limitCount** [int](#)

The maximum number of results to include in the dataset. Must be a non-negative integer.

### Returns

[IRelmDataSet](#)<T>

The current dataset instance with the applied limit, allowing for further query chaining.

## Load()

Loads a collection of items of type [T](#).

```
public ICollection<T> Load()
```

### Returns

[ICollection](#)<T>

A collection of items of type [T](#). The collection may be empty if no items are available.

## Remarks

This method retrieves the items and returns them as a collection. By default, it includes all items unless specified otherwise in an overload.

## Load(bool)

Loads the data for the current context and optionally initializes and executes field loaders.

```
public ICollection<T> Load(bool loadDataLoaders)
```

### Parameters

`loadDataLoaders` [bool](#)

A value indicating whether to initialize and execute field loaders for properties marked with the [Relm DataLoader](#) attribute. If [true](#), field loaders are registered and executed; otherwise, only the data is loaded.

### Returns

[ICollection](#)<T>

A collection of items of type T representing the loaded data. The collection will be empty if no data is available.

### Remarks

If `loadDataLoaders` is [true](#), the method identifies properties in the type T that are marked with the [Relm DataLoader](#) attribute and initializes field loaders for them. These field loaders are then executed to load additional data related to the fields. If the data loader has executed commands related to references, the method also loads all references associated with the data.

## LoadAsDataSet()

Loads the current data and returns it as a dataset.

```
public IRelmDataSet<T> LoadAsDataSet()
```

### Returns

[IRelmDataSet](#)<T>

An [IRelmDataSet](#)<T> representing the loaded dataset.

## Remarks

This method ensures that the data is loaded before returning the dataset. It can be used to retrieve the data in its current state for further processing or querying.

## New(bool)

Creates a new blank instance of the object, with an option to persist it.

```
public T New(bool Persist = true)
```

### Parameters

Persist [bool](#)

A value indicating whether the new instance should be persisted. [true](#) to persist the instance; otherwise, [false](#).

### Returns

T

A new instance of the object.

## New(dynamic, bool)

Creates a new instance of the specified type **T** and initializes its properties based on the provided dynamic parameters.

```
public T New(dynamic NewObjectParameters, bool Persist = true)
```

### Parameters

NewObjectParameters [dynamic](#)

A dynamic object containing property names and values to initialize the new instance. Only properties matching the keys in the underscore properties of **T** will be set.

Persist [bool](#)

A boolean value indicating whether the newly created object should be persisted. The default value is [true](#).

## Returns

T

A new instance of T with its properties initialized based on the provided parameters.

## Remarks

This method uses reflection to set the properties of the new instance. Ensure that the property names in [NewObjectParameters](#) match the expected property names of T.

## Offset(int)

Sets the offset value used for positioning or alignment.

```
public IRelmDataSet<T> Offset(int offsetCount)
```

## Parameters

[offsetCount](#) [int](#)

The offset value to apply. Must be a non-negative integer.

## Returns

[IRelmDataSet](#)<T>

The current dataset instance with the applied offset, allowing for further query chaining.

## OrderBy(Expression<Func<T, object>>)

Orders the elements in the dataset based on the specified key selector.

```
public IRelmDataSet<T> OrderBy(Expression<Func<T, object>> predicate)
```

## Parameters

```
predicate Expression<Func<T, object>>
```

An expression that specifies the key to order the elements by. The key is derived from the dataset elements.

Returns

```
IRelmDataSet<T>
```

A dataset with the elements ordered according to the specified key.

Remarks

This method modifies the query to include an "ORDER BY" clause based on the provided key selector. The ordering is applied in ascending order by default.

## OrderByDescending(Expression<Func<T, object>>)

Orders the elements of the dataset in descending order based on the specified key.

```
public IRelmDataSet<T> OrderByDescending(Expression<Func<T, object>> predicate)
```

Parameters

```
predicate Expression<Func<T, object>>
```

An expression that specifies the key to use for ordering the elements. The key is derived from the property or computed value defined in the expression.

Returns

```
IRelmDataSet<T>
```

A dataset with the elements ordered in descending order based on the specified key.

Remarks

This method modifies the query by appending an "ORDER BY DESC" clause using the specified key. The ordering is applied to the dataset when the query is executed.

## Reference<S>(Expression<Func<T, ICollection<S>>>, ICollection<Expression<Func<S, object>>>)

Configures a reference navigation property for the current entity, allowing additional constraints to be applied.

```
public IRelmDataSet<T> Reference<S>(Expression<Func<T, ICollection<S>>> predicate,  
ICollection<Expression<Func<S, object>>> additionalConstraints)
```

### Parameters

**predicate** [Expression<Func<T, ICollection<S>>>](#)

An expression that specifies the collection navigation property to configure.

**additionalConstraints** [ICollection<Expression<Func<S, object>>>](#)

A collection of expressions representing additional constraints to apply to the related entities.

### Returns

[IRelmDataSet<T>](#)

An [IRelmDataSet<T>](#) representing the configured reference navigation property.

### Type Parameters

**S**

The type of the related entity in the collection.

### Remarks

This method is typically used to define relationships between entities and apply additional filtering or constraints on the related entities. The **predicate** parameter identifies the navigation property, while the **additionalConstraints** parameter allows specifying further conditions on the related entities.

## Reference<S>(Expression<Func<T, ICollection<S>>>, Expression<Func<S, object>>)

Configures a reference navigation property for inclusion in the query, allowing additional constraints to be applied to the related entities.

```
public IRelmDataSet<T> Reference<S>(Expression<Func<T, ICollection<S>>> predicate,  
Expression<Func<S, object>> additionalConstraints)
```

## Parameters

**predicate** [Expression<Func<T, ICollection<S>>>](#)

An expression that specifies the navigation property to include.

**additionalConstraints** [Expression<Func<S, object>>](#)

An expression that defines additional constraints to apply to the related entities.

## Returns

[IRelmDataSet<T>](#)

An [IRelmDataSet<T>](#) that includes the specified navigation property and applies the given constraints.

## Type Parameters

**S**

The type of the related entities in the collection.

## Remarks

This method is typically used to include a collection navigation property in a query and apply additional filtering or constraints to the related entities.

## Reference<S>(Expression<Func<T, S>>)

Creates a reference to the specified property or field of the current entity.

```
public IRelmDataSet<T> Reference<S>(Expression<Func<T, S>> predicate)
```

## Parameters

**predicate** [Expression<Func<T, S>>](#)

An expression that specifies the property or field to reference.

Returns

[IRelmDataSet<T>](#)

An [IRelmDataSet<T>](#) that includes the specified reference.

Type Parameters

S

The type of the property or field being referenced.

Remarks

This method allows you to define a reference to a property or field of the entity, which can be used to include related data in subsequent operations.

**Reference<S>(Expression<Func<T, S>>,  
ICollection<Expression<Func<S, object>>>)**

Creates a reference to a related dataset based on the specified predicate and additional constraints.

```
public IRelmDataSet<T> Reference<S>(Expression<Func<T, S>> predicate,  
ICollection<Expression<Func<S, object>>> additionalConstraints)
```

Parameters

**predicate** [Expression<Func<T, S>>](#)

An expression that specifies the relationship between the current entity and the related entity.

**additionalConstraints** [ICollection<Expression<Func<S, object>>>](#)

A collection of expressions that define additional constraints to apply to the related dataset.

Returns

[IRelmDataSet<T>](#)

An [IRelmDataSet<T>](#) representing the related dataset that matches the specified predicate and constraints.

## Type Parameters

### S

The type of the related entity being referenced.

## Reference<S>(Expression<Func<T, S>>, Expression<Func<S, object>>)

Creates a reference to a related dataset based on the specified predicate and additional constraints.

```
public IRelmDataSet<T> Reference<S>(Expression<Func<T, S>> predicate, Expression<Func<S, object>> additionalConstraints)
```

## Parameters

### predicate [Expression<Func<T, S>>](#)

An expression that specifies the relationship between the current entity and the related entity.

### additionalConstraints [Expression<Func<S, object>>](#)

An expression that defines additional constraints to apply to the related entity.

## Returns

### [IRelmDataSet<T>](#)

An [IRelmDataSet<T>](#) representing the related dataset that matches the specified predicate and constraints.

## Type Parameters

### S

The type of the related entity being referenced.

## Remarks

This method allows you to define a reference to a related dataset by specifying a predicate for the relationship and additional constraints to further filter the related data. Use this method when you need to reference a related dataset with specific filtering criteria.

## Remove(T)

Removes the specified item from the collection.

```
public bool Remove(T item)
```

### Parameters

**item** T

The item to remove from the collection.

### Returns

[bool](#)

[true](#) if the item was successfully removed; otherwise, [false](#). Returns [false](#) if the item was not found in the collection or if the collection is null.

## Save()

Saves the current items to the database and returns the number of rows updated.

```
public int Save()
```

### Returns

[int](#)

The number of rows updated in the database.

### Remarks

The method determines the database connection settings based on the current context options. If the context options specify an open connection, the method uses the provided database connection and

transaction. Otherwise, it uses the connection string type to establish the connection. After the save operation, the [Modified](#) property is set to [false](#).

## Save(T)

Saves the specified item to the collection. If an item with the same internal identifier already exists, it is replaced; otherwise, the item is added to the collection.

```
public int Save(T Item)
```

### Parameters

[Item](#) T

The item to save. The item must have a valid internal identifier.

### Returns

[int](#)

An integer representing the result of the save operation. The exact meaning of the return value depends on the underlying implementation of the save or add operation.

### Remarks

This method ensures that the collection does not contain duplicate items based on their internal identifiers. If an item with the same identifier already exists, it is replaced with the new item. Otherwise, the item is added to the collection, and the operation is persisted.

## Set(Expression<Func<T, T>>)

Updates a single column's value in the dataset based on the specified expression.

```
public IRelmDataSet<T> Set(Expression<Func<T, T>> predicate)
```

### Parameters

[predicate](#) [Expression](#)<[Func](#)<T, T>>

An expression that specifies the column to update and the value to assign. The expression must be in the form of a lambda, such as `x => x.Property = value`.

Returns

### [IRelmDataSet<T>](#)

The current dataset instance, allowing for method chaining.

Remarks

This method modifies the dataset by applying the specified update expression. The changes are not persisted until the dataset is committed.

## SetDataLoader(IRelmDataLoader<T>)

Sets the data loader for the current instance.

```
public IRelmDataLoader<T> SetDataLoader(IRelmDataLoader<T> dataLoader)
```

Parameters

### [dataLoader](#) [IRelmDataLoader<T>](#)

The data loader to be used. Cannot be [null](#).

Returns

### [IRelmDataLoader<T>](#)

The data loader that was set.

## SetFieldLoader(string, IRelmFieldLoader)

Registers a field loader for the specified field name.

```
public IRelmFieldLoader SetFieldLoader(string fieldName, IRelmFieldLoader dataLoader)
```

Parameters

### **fieldName** [string](#)

The name of the field on the model for which the loader is being registered. The field must exist on the model type **T**.

### **dataLoader** [IRelmFieldLoader](#)

The [IRelmFieldLoader](#) instance responsible for loading data for the specified field.

Returns

### [IRelmFieldLoader](#)

The [IRelmFieldLoader](#) instance that was registered for the specified field.

Remarks

This method ensures that the specified field exists on the model type **T** before registering the loader.

Exceptions

### [ArgumentException](#)

Thrown if **fieldName** does not correspond to a property on the model type **T**.

## SetFieldLoader(string, IRelmQuickFieldLoader)

Registers a field loader for the specified field name.

```
public IRelmQuickFieldLoader SetFieldLoader(string fieldName, IRelmQuickFieldLoader dataLoader)
```

Parameters

### **fieldName** [string](#)

The name of the field on the model for which the loader is being registered. The field must exist on the model type **T**.

### **dataLoader** [IRelmQuickFieldLoader](#)

An instance of [IRelmQuickFieldLoader](#) that provides the logic for loading the field's data.

## Returns

### [IRelmQuickFieldLoader](#)

The registered [IRelmQuickFieldLoader](#) instance for the specified field.

## Remarks

This method ensures that the specified field exists on the model type `T` before registering the loader.

## Exceptions

### [ArgumentException](#)

Thrown if the specified `fieldName` does not exist on the model type `T`.

## Where(Expression<Func<T, bool>>)

Filters the dataset based on the specified predicate.

```
public IRelmDataSet<T> Where(Expression<Func<T, bool>> predicate)
```

## Parameters

### `predicate Expression<Func<T, bool>>`

An expression that defines the conditions of the filter. The predicate is a function that takes an element of type `T` and returns [true](#) if the element should be included in the result; otherwise, [false](#).

## Returns

### [IRelmDataSet](#)<T>

A dataset containing only the elements that satisfy the specified predicate.

## Remarks

This method allows you to apply a filtering condition to the dataset. The filtering is deferred, meaning the predicate is not evaluated until the dataset is enumerated or further operations are performed.

## Write()

Writes data using the underlying data loader.

```
public int Write()
```

Returns

[int↗](#)

The number of records successfully written.

# Class RelmDefaultDataLoader<T>

Namespace: [CoreRelm.Models](#)

Assembly: CoreRelm.dll

Provides a default implementation of the [IRelmDataLoader<T>](#) interface for loading, querying, and writing data for a specified model type. This class is designed to work with models that implement the [IRelmModel](#) interface.

```
public class RelmDefaultDataLoader<T> : IRelmDataLoader<T> where T : IRelmModel, new()
```

## Type Parameters

T

The type of the model to be loaded, queried, or written. The type must implement [IRelmModel](#) and have a parameterless constructor.

## Inheritance

[object](#) ← RelmDefaultDataLoader<T>

## Implements

[IRelmDataLoader<T>](#)

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

## Remarks

This class supports operations such as adding query expressions, retrieving data, and writing data to the database. It uses a combination of database column metadata and query expressions to dynamically build SQL queries.

## Constructors

### RelmDefaultDataLoader()

Initializes a new instance of the [RelmDefaultDataLoader\(\)](#) class.

```
public RelmDefaultDataLoader()
```

## Remarks

This constructor performs the initial setup required for the data loader by invoking the CoreRelm.Models.RelmDefaultDataLoader<T>.InitialSetup() method.

## RelmDefaultDataLoader(RelmContextOptionsBuilder)

Initializes a new instance of the [RelmDefaultDataLoader\(\)](#) class with the specified context options builder.

```
public RelmDefaultDataLoader(RelmContextOptionsBuilder contextOptionsBuilder)
```

## Parameters

`contextOptionsBuilder` [RelmContextOptionsBuilder](#)

The builder used to configure options for the [RelmContext](#).

## Properties

### LastCommandsExecuted

Gets or sets the collection of the most recently executed commands and their associated execution details.

```
public Dictionary<Commands.Command, List<IRelmExecutionCommand>> LastCommandsExecuted { get;  
    set; }
```

## Property Value

[Dictionary](#)<[Commands.Command](#), [List](#)<[IRelmExecutionCommand](#)>>

## Remarks

This property provides a record of the last executed commands along with their execution details. The dictionary keys represent the commands, and the associated lists contain the execution information for each command. The property can be used to inspect or analyze the history of command executions.

# Methods

## AddExpression(Command, Expression)

Adds an expression to the specified command and returns a new execution command.

```
public IRelmExecutionCommand AddExpression(Command command, Expression expression)
```

### Parameters

**command** [Commands.Command](#)

The command to which the expression will be added. Must not be [null](#).

**expression** [Expression](#)

The expression to add to the command. Must not be [null](#).

### Returns

[IRelmExecutionCommand](#)

A new [IRelmExecutionCommand](#) instance representing the command with the added expression.

### Remarks

This method creates a new execution command by associating the provided expression with the specified command. The new execution command is added to the prewarmed query for the given command.

## AddSingleExpression(Command, Expression)

Adds a single expression to the specified command and returns the resulting execution command.

```
public IRelmExecutionCommand AddSingleExpression(Command command,  
Expression expression)
```

### Parameters

**command** [Commands.Command](#)

The command to which the expression will be added.

#### **expression** [Expression](#)

The expression to associate with the command.

Returns

#### [IRelmExecutionCommand](#)

An [IRelmExecutionCommand](#) representing the updated execution command with the specified expression.

Remarks

If the command does not already have any associated expressions, a new execution command is created.

## GetLoadData()

Retrieves a collection of data entities based on the current query configuration.

```
public virtual ICollection<T> GetLoadData()
```

Returns

#### [ICollection](#)<T>

A collection of data entities of type **T**. The collection will be empty if no data matches the query.

Remarks

This method constructs a query using the current configuration and retrieves the corresponding data entities. The returned collection may be empty if no matching data is found.

## HasUnderscoreProperty(string)

Determines whether the specified property key exists in the column registry.

```
public bool HasUnderscoreProperty(string PropertyKey)
```

## Parameters

### PropertyKey [string](#)

The key of the property to check for existence.

## Returns

### [bool](#)

[true](#) if the column registry contains the specified property key; otherwise, [false](#).

## Remarks

This method checks the presence of a property key in the internal column registry. If the column registry is null, the method returns [false](#).

## PullData(string, Dictionary<string, object>)

Executes the specified SQL query and retrieves a collection of data objects of type [T](#).

```
public virtual ICollection<T> PullData(string selectQuery, Dictionary<string, object> findOptions)
```

## Parameters

### selectQuery [string](#)

The SQL query to execute. This query should be a valid SELECT statement that matches the structure of the data objects being retrieved.

### findOptions [Dictionary](#)<[string](#), [object](#)>

A dictionary of parameters to be used in the query. The keys represent parameter names, and the values represent their corresponding values.

## Returns

### [ICollection](#)<[T](#)>

A collection of data objects of type [T](#) that match the results of the query. Returns an empty collection if no data is found.

## Remarks

This method supports two modes of database connection: - If the context is configured with an open database connection, the query is executed using the provided connection and transaction. - Otherwise, the query is executed using the configured connection string type.

## WriteData()

Executes a database operation to write data and returns the result of the operation.

```
public int WriteData()
```

## Returns

[int](#)

The result of the database operation as an integer. The value typically represents the number of rows affected by the operation.

## Remarks

This method constructs an update query and performs the database operation based on the configuration of the context options. It supports both open database connections and connection strings, depending on the specified options.

# Class RelmExecutionCommand

Namespace: [CoreRelm.Models](#)

Assembly: CoreRelm.dll

Represents a command and its associated expression for execution within the Relm framework, supporting additional and child commands for complex execution scenarios.

```
public class RelmExecutionCommand : IRelmExecutionCommand
```

## Inheritance

[object](#) ← RelmExecutionCommand

## Implements

[IRelmExecutionCommand](#)

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

## Remarks

A RelmExecutionCommand encapsulates a primary command and expression, and can aggregate additional related commands to support batch or hierarchical execution patterns. This class is typically used to construct and manage command trees or sequences for advanced data operations, such as those involving foreign key navigation or entity relationships. Thread safety is not guaranteed; if used concurrently, external synchronization is required.

## Constructors

### RelmExecutionCommand()

Initializes a new instance of the RelmExecutionCommand class.

```
public RelmExecutionCommand()
```

### RelmExecutionCommand(Command, Expression)

Initializes a new instance of the RelmExecutionCommand class with the specified command and expression.

```
public RelmExecutionCommand(Command command, Expression expression)
```

## Parameters

**command** [Commands.Command](#)

The command to be executed as part of the execution command. Cannot be null.

**expression** [Expression](#)

The expression associated with the execution command. Cannot be null.

# Properties

## AdditionalCommandCount

Gets the number of additional commands associated with this instance.

```
[Obsolete("AdditionalCommandCount is deprecated, please use ChildCommandCount instead.")]  
public int AdditionalCommandCount { get; }
```

## Property Value

[int](#)

## ChildCommandCount

Gets the number of child commands associated with this instance.

```
public int ChildCommandCount { get; }
```

## Property Value

[int](#)

## ExecutionCommand

Gets the command that initiates execution for this operation.

```
public Commands.Command ExecutionCommand { get; }
```

Property Value

[Commands.Command](#)

## ExecutionExpression

Gets the expression that represents the execution logic for this instance.

```
public Expression ExecutionExpression { get; }
```

Property Value

[Expression](#)

## InitialCommand

Gets the command that is executed when the associated component is initialized.

```
[Obsolete("InitialCommand is deprecated, please use ExecutionCommand instead.")]  
public Commands.Command InitialCommand { get; }
```

Property Value

[Commands.Command](#)

## InitialExpression

Gets the initial expression used to define the starting state or value for this instance.

```
[Obsolete("InitialExpression is deprecated, please use ExecutionExpression instead.")]
```

```
public Expression InitialExpression { get; }
```

## Property Value

[Expression](#)

## Methods

### AddAdditionalCommand(Command, Expression)

Adds an additional command and its associated expression to the current execution command sequence.

```
public RelmExecutionCommand AddAdditionalCommand(Command command,  
Expression expression)
```

#### Parameters

**command** [Commands.Command](#)

The command to add to the execution sequence. Cannot be null.

**expression** [Expression](#)

The expression associated with the command. Cannot be null.

#### Returns

[RelmExecutionCommand](#)

The current [RelmExecutionCommand](#) instance with the additional command included.

#### Remarks

This method enables fluent chaining by returning the same instance after adding the command and expression.

### AddAdditionalCommand<T>(Command, Expression<Func<T, object>>)

Adds an additional command to the execution pipeline using a strongly typed expression to specify the target property or member.

```
public RelmExecutionCommand AddAdditionalCommand<T>(Commands.Command command,  
Expression<Func<T, object>> expression)
```

## Parameters

**command** [Commands.Command](#)

The command to add to the execution pipeline.

**expression** [Expression](#)<[Func](#)<T, object>>

An expression that identifies the property or member of type T to which the command applies.

## Returns

[RelmExecutionCommand](#)

The current [RelmExecutionCommand](#) instance, enabling method chaining.

## Type Parameters

T

The type of the object that contains the member referenced by the expression.

## GetAdditionalCommands()

Gets the list of additional execution commands associated with this instance.

```
public List<RelmExecutionCommand> GetAdditionalCommands()
```

## Returns

[List](#)<[RelmExecutionCommand](#)>

A list of [RelmExecutionCommand](#) objects representing additional commands. The list may be empty if no additional commands are present.

# GetForeignKeyNavigationOptions<T>(ICollection<T>)

Retrieves navigation and foreign key mapping options for a collection of entities, enabling resolution of relationships between the provided items and their related entities.

```
public ForeignKeyNavigationOptions GetForeignKeyNavigationOptions<T>(ICollection<T> _items)
```

## Parameters

\_items [ICollection](#)<T>

The collection of entities for which to resolve foreign key navigation options. Cannot be null.

## Returns

[ForeignKeyNavigationOptions](#)

A ForeignKeyNavigationOptions instance containing metadata about the navigation properties, foreign key properties, and primary key values for the specified collection.

## Type Parameters

T

The type of the entities in the collection for which foreign key navigation options are to be determined.

## Remarks

This method analyzes the provided collection and its type metadata to determine the appropriate navigation and foreign key properties. It supports both principal and dependent entity configurations, and will throw exceptions if required attributes or keys are missing. The returned options can be used to facilitate relationship resolution in data access scenarios.

## Exceptions

[InvalidOperationException](#)

Thrown if the initial expression is not a lambda expression in the form 'x => x.PropertyName'.

[Exception](#)

Thrown if no primary keys or reference keys are found for the provided collection.

## [MemberAccessException](#)

Thrown if the foreign key referenced by the RelmForeignKey attribute cannot be found.

# Class RelmModel

Namespace: [CoreRelm.Models](#)

Assembly: CoreRelm.dll

Represents a base model for entities in the Relm framework, providing core attributes and functionality for data manipulation, database operations, and DTO generation.

```
public class RelmModel : IRelmModel
```

**Inheritance**

[object](#) ← RelmModel

**Implements**

[IRelmModel](#)

**Derived**

[AppliedMigration](#), [RelmModelApartment](#)

**Inherited Members**

[object.Equals\(object\)](#), [object.Equals\(object, object\)](#), [object.GetHashCode\(\)](#), [object.GetType\(\)](#),  
[object.MemberwiseClone\(\)](#), [object.ReferenceEquals\(object, object\)](#), [object.ToString\(\)](#)

**Extension Methods**

[ModelExtensions.LoadDataLoaderField<T, S>\(T, IRelmContext, Expression<Func<T, S>>\)](#),  
[ModelExtensions.LoadDataLoaderField<T, S>\(T, IRelmQuickContext, Expression<Func<T, S>>\)](#),  
[ModelExtensions.LoadDataLoaderField<T, S>\(T, RelmContextOptionsBuilder, Expression<Func<T, S>>\)](#),  
[ModelExtensions.LoadForeignKeyField<T, R>\(T, IRelmContext, Expression<Func<T, ICollection<R>>>\)](#),  
[ModelExtensions.LoadForeignKeyField<T, R>\(T, IRelmContext, Expression<Func<T, ICollection<R>>>,  
Expression<Func<R, object>>\)](#),  
[ModelExtensions.LoadForeignKeyField<T, R>\(T, IRelmContext, Expression<Func<T, R>>\)](#),  
[ModelExtensions.LoadForeignKeyField<T, R>\(T, IRelmContext, Expression<Func<T, R>>,  
Expression<Func<R, object>>\)](#),  
[ModelExtensions.LoadForeignKeyField<T, R>\(T, IRelmQuickContext, Expression<Func<T,  
ICollection<R>>>\)](#),  
[ModelExtensions.LoadForeignKeyField<T, R>\(T, IRelmQuickContext, Expression<Func<T,  
ICollection<R>>>, Expression<Func<R, object>>\)](#),  
[ModelExtensions.LoadForeignKeyField<T, R>\(T, IRelmQuickContext, Expression<Func<T, R>>\)](#),  
[ModelExtensions.LoadForeignKeyField<T, R>\(T, IRelmQuickContext, Expression<Func<T, R>>,  
Expression<Func<R, object>>\)](#),

```
ModelExtensions.LoadForeignKeyField<T, R>(T, RelmContextOptionsBuilder, Expression<Func<T, ICollection<R>>>),
ModelExtensions.LoadForeignKeyField<T, R>(T, RelmContextOptionsBuilder, Expression<Func<T, ICollection<R>>>, Expression<Func<R, object>>),
ModelExtensions.LoadForeignKeyField<T, R>(T, RelmContextOptionsBuilder, Expression<Func<T, R>>),
ModelExtensions.LoadForeignKeyField<T, R>(T, RelmContextOptionsBuilder, Expression<Func<T, R>>, Expression<Func<R, object>>),
ModelExtensions.LoadForeignKeyField<T, R, S>(T, IRelmContext, Expression<Func<T, ICollection<R>>>, IRelmDataLoader<S>),
ModelExtensions.LoadForeignKeyField<T, R, S>(T, IRelmContext, Expression<Func<T, ICollection<R>>>, IRelmDataLoader<S>, Expression<Func<R, object>>),
ModelExtensions.LoadForeignKeyField<T, R, S>(T, IRelmContext, Expression<Func<T, R>>, IRelmDataLoader<S>),
ModelExtensions.LoadForeignKeyField<T, R, S>(T, IRelmContext, Expression<Func<T, R>>, IRelmDataLoader<S>, Expression<Func<R, object>>),
ModelExtensions.LoadForeignKeyField<T, R, S>(T, IRelmQuickContext, Expression<Func<T, ICollection<R>>>, IRelmDataLoader<S>),
ModelExtensions.LoadForeignKeyField<T, R, S>(T, IRelmQuickContext, Expression<Func<T, ICollection<R>>>, IRelmDataLoader<S>, Expression<Func<R, object>>),
ModelExtensions.LoadForeignKeyField<T, R, S>(T, IRelmQuickContext, Expression<Func<T, R>>, IRelmDataLoader<S>),
ModelExtensions.LoadForeignKeyField<T, R, S>(T, IRelmQuickContext, Expression<Func<T, R>>, IRelmDataLoader<S>, Expression<Func<R, object>>),
ModelExtensions.LoadForeignKeyField<T, R, S>(T, RelmContextOptionsBuilder, Expression<Func<T, ICollection<R>>>, IRelmDataLoader<S>),
ModelExtensions.LoadForeignKeyField<T, R, S>(T, RelmContextOptionsBuilder, Expression<Func<T, ICollection<R>>>, IRelmDataLoader<S>, Expression<Func<R, object>>),
ModelExtensions.LoadForeignKeyField<T, R, S>(T, RelmContextOptionsBuilder, Expression<Func<T, R>>, IRelmDataLoader<S>),
ModelExtensions.LoadForeignKeyField<T, R, S>(T, RelmContextOptionsBuilder, Expression<Func<T, R>>, IRelmDataLoader<S>, Expression<Func<R, object>>).
```

## Remarks

The [RelmModel](#) class serves as a foundational class for entities, offering features such as:

- Core attributes like [Id](#), [Active](#), [InternalId](#), [CreateDate](#), and [LastUpdated](#).
- Support for resetting attributes to default values using [ResetCoreAttributes\(bool, bool\)](#).
- Data population from database rows via [ResetWithData\(DataRow, string\)](#).
- Bulk database write operations through various [WriteToDatabase](#) overloads.

- Dynamic Data Transfer Object (DTO) generation with [GenerateDTO\(IEnumerable<string>, IEnumerable<string>, string, Func<IRelmModel, Dictionary<string, object>>, int\)](#).

This class is designed to be extended by specific entity types and provides a flexible foundation for working with relational data in the Relm framework.

## Constructors

### RelmModel()

Initializes a new instance of the [RelmModel](#) class.

```
public RelmModel()
```

#### Remarks

This constructor initializes the core attributes of the model by invoking the [ResetCoreAttributes\(bool, bool\)](#) method.

### RelmModel(IRelmModel)

Initializes a new instance of the [RelmModel](#) class by copying all writable properties from the specified source model.

```
public RelmModel(IRelmModel fromModel)
```

#### Parameters

**fromModel** [IRelmModel](#)

The source model from which to copy property values. Must not be [null](#).

#### Remarks

This constructor performs a shallow copy of all writable properties from the specified source model to the new instance. Only properties with matching names and compatible types between the source and target models are copied. Properties that are read-only or do not exist in the target model are ignored.

#### Exceptions

## [ArgumentNullException](#)

Thrown if `fromModel` is [null](#).

## RelmModel(DataRow, string)

Initializes a new instance of the [RelmModel](#) class and populates its properties using data from the specified database row.

```
public RelmModel(DataRow modelData, string alternateTableName = null)
```

### Parameters

`modelData` [DataRow](#)

A [DataRow](#) containing the data to populate the model. The column names in the `modelData` must match the property names of the model for successful mapping.

`alternateTableName` [string](#)

An optional alternate table name to use when searching for data in the `modelData`. If not specified, the default table name is used.

### Remarks

This constructor automatically maps the data from the provided `modelData` to the properties of the model based on naming conventions. Ensure that the column names in the `modelData` match the property names of the model for accurate mapping.

## Properties

### Active

Gets or sets a value indicating whether the entity is active.

```
[RelmColumn(null, -1, null, false, false, false, false, "1", null, false, false, false)]  
public bool Active { get; set; }
```

### Property Value

[bool](#)

## CreateDate

Gets or sets the creation date and time of the entity.

```
[RelmColumn(null, -1, null, false, false, false, false, "CURRENT_TIMESTAMP", null, false,  
false, false)]  
public DateTime CreateDate { get; set; }
```

## Property Value

[DateTime](#)

## Id

Gets or sets the unique auto-number row identifier for the entity.

```
[RelmColumn(null, -1, null, false, true, true, false, null, null, false, false, false)]  
public long? Id { get; set; }
```

## Property Value

[long](#)?

## InternalId

Gets or sets the unique internal identifier for the entity.

```
[RelmColumn(null, -1, null, false, false, false, true, null, null, false, false, false)]  
public string InternalId { get; set; }
```

## Property Value

[string](#)

# LastUpdated

Gets or sets the timestamp of the last update.

```
[RelmColumn(null, -1, null, false, false, false, false, "CURRENT_TIMESTAMP", null, false,  
false, false)]  
public DateTime LastUpdated { get; set; }
```

Property Value

[DateTime](#)

## Methods

### CopyFromSource<T>(T)

Creates a new instance of the specified type **T** and copies all public properties and fields from the specified **source** object to the new instance.

```
public T CopyFromSource<T>(T source) where T : RelmModel, new()
```

Parameters

**source** T

The source object from which to copy property and field values. Cannot be [null](#).

Returns

T

A new instance of type **T** with all public properties and fields copied from the **source** object.

Type Parameters

T

The type of the object to create and copy values to. Must be a class that derives from [RelmModel](#) and has a parameterless constructor.

## Remarks

This method performs a shallow copy of all public properties and fields. Only properties that are writable and not indexed are copied. Fields are copied regardless of their accessibility.

## Exceptions

### [ArgumentNullException](#)

Thrown if `source` is [null](#).

## Duplicate()

Creates a shallow copy of the current [RelmModel](#) instance.

```
public RelmModel Duplicate()
```

## Returns

### [RelmModel](#)

A new [RelmModel](#) instance that is a shallow copy of the current instance.

## Remarks

The returned object is a shallow copy, meaning that only the top-level fields of the object are duplicated. References to other objects within the instance are not deeply copied.

## GenerateDTO(IEnumerable<string>, IEnumerable<string>, string, Func<IRelmModel, Dictionary<string, object>>, int)

Generates a dynamic Data Transfer Object (DTO) containing properties from the current object based on specific inclusion and exclusion criteria.

```
public dynamic GenerateDTO(IEnumerable<string> includeProperties = null, IEnumerable<string>
excludeProperties = null, string sourceObjectName = null, Func<IRelmModel,
Dictionary<string, object>> getAdditionalObjectProperties = null, int iteration = 0)
```

## Parameters

### `includeProperties` [IEnumerable<string>](#)

A collection of property names to explicitly include in the DTO, even if they are not marked with the [RelmDto](#) attribute.

### `excludeProperties` [IEnumerable<string>](#)

A collection of property names to explicitly exclude from the DTO, even if they are marked with the [RelmDto](#) attribute.

### `sourceObjectName` [string](#)

An optional string representing the source object name, used to resolve nested property paths.

### `getAdditionalObjectProperties` [Func<IRelmModel, Dictionary<string, object>>](#)

A function that provides additional properties to include in the DTO. The function takes the current object as input and returns a dictionary of property names and values.

### `iteration` [int](#)

The current recursion depth, used internally to track nested object processing. Defaults to 0.

## Returns

### `dynamic`

A dynamic object containing the selected properties from the current object. The resulting object includes properties marked with the [RelmDto](#) attribute, explicitly included properties, and additional properties provided by the `getAdditionalObjectProperties` function, while excluding explicitly excluded properties.

## Remarks

This method supports recursive generation of DTOs for nested objects. If a property is a collection of objects, each item in the collection is processed recursively. Properties not marked with the [RelmDto](#) attribute are included only if explicitly specified in `includeProperties`.

## GetUnderscoreProperties(bool)

Retrieves a list of properties for the current object, with their names converted to underscore-separated format.

```
public List<KeyValuePair<string, Tuple<string, PropertyInfo>>> GetUnderscoreProperties(bool  
getOnlyRelmColumns = true)
```

## Parameters

`getOnlyRelmColumns bool`

A value indicating whether to include only properties marked with the `RelmColumn` attribute. If `true`, only such properties are included; otherwise, all properties are included.

## Returns

`List<KeyValuePair<string, Tuple<string, PropertyInfo>>>`

A list of key-value pairs where the key is the underscore-separated name of the property, and the value is a tuple containing the original property name and its `PropertyInfo` metadata.

## ResetCoreAttributes(bool, bool)

Resets the core attributes of the model to their default values.

```
public IRelmModel ResetCoreAttributes(bool nullInternalId = false, bool resetCreateDate  
= true)
```

## Parameters

`nullInternalId bool`

If `true`, sets the internal ID to `null`; otherwise, generates a new unique identifier.

`resetCreateDate bool`

If `true`, resets the creation date to the current date and time.

## Returns

`IRelmModel`

The current instance of the model with updated core attributes.

## ResetWithData(DataRow, string)

Resets the current model instance with data from the specified [DataRow](#).

```
public IRelmModel ResetWithData(DataRow modelData, string alternateTableName = null)
```

### Parameters

**modelData** [DataRow](#)

The [DataRow](#) containing the data to populate the model. The column names in the [DataRow](#) should match the model's property names, using underscore naming conventions.

**alternateTableName** [string](#)

An optional alternate table name used to match column names in the [DataRow](#). If not provided, the model's type name is used as the table name.

### Returns

[IRelmModel](#)

The current model instance, updated with the data from the specified [DataRow](#).

### Remarks

This method maps the columns in the provided [DataRow](#) to the model's properties based on underscore naming conventions. If a property has a `Newtonsoft.Json.JsonConverterAttribute`, the data is deserialized using the specified JSON converter. The method supports both default column names and column names suffixed with the table name.

## WriteToDatabase(IRelmContext, int, bool, bool, bool, bool)

Writes the current object to the database using the table specified in the `RelmTable` attribute.

```
public int WriteToDatabase(IRelmContext relmContext, int batchSize = 100, bool  
allowAutoIncrementColumns = false, bool allowPrimaryKeyColumns = false, bool  
allowUniqueColumns = false, bool allowAutoDateColumns = false)
```

### Parameters

## `relmContext` [IRelmContext](#)

An [IRelmContext](#) instance that provides an open connection and transaction for database operations.

### `batchSize` [int](#)

The number of items to write to the database in each batch. Must be greater than zero. The default value is 100.

### `allowAutoIncrementColumns` [bool](#)

A value indicating whether columns marked as auto-increment are allowed to be written. If [true](#), auto-increment columns will be included in the write operation; otherwise, they will be excluded.

### `allowPrimaryKeyColumns` [bool](#)

A value indicating whether primary key columns are allowed to be written. If [true](#), primary key columns will be included in the write operation; otherwise, they will be excluded.

### `allowUniqueColumns` [bool](#)

A value indicating whether columns with unique constraints are allowed to be written. If [true](#), unique columns will be included in the write operation; otherwise, they will be excluded.

### `allowAutoDateColumns` [bool](#)

A value indicating whether columns with automatic date generation (e.g., timestamps) are allowed to be written. If [true](#), such columns will be included in the write operation; otherwise, they will be excluded.

## Returns

### [int](#)

The number of rows successfully written to the database.

## Remarks

This method performs a bulk write operation to the database. The behavior of the write operation can be customized using the optional parameters to control whether specific types of columns (e.g., auto-increment, primary key, unique, or auto-date columns) are included in the operation. The table to which the object is written is determined by the [RelmTable](#) attribute applied to the object's type.

# WriteToDatabase(IRelmQuickContext, int, bool, bool, bool, bool)

Writes the current object to the database using the table specified in the `RelmTable` attribute.

```
public int WriteToDatabase(IRelmQuickContext relmContext, int batchSize = 100, bool  
allowAutoIncrementColumns = false, bool allowPrimaryKeyColumns = false, bool  
allowUniqueColumns = false, bool allowAutoDateColumns = false)
```

## Parameters

`relmContext` [IRelmQuickContext](#)

The database context used to perform the write operation. This context must be properly configured to connect to the target database.

`batchSize` [int](#)

The number of records to include in each batch during the bulk write operation. The default value is 100.

`allowAutoIncrementColumns` [bool](#)

A value indicating whether columns with auto-increment constraints are allowed to be written. If [true](#), auto-increment columns will be included in the write operation; otherwise, they will be excluded. The default value is [false](#).

`allowPrimaryKeyColumns` [bool](#)

A value indicating whether primary key columns are allowed to be written. If [true](#), primary key columns will be included in the write operation; otherwise, they will be excluded. The default value is [false](#).

`allowUniqueColumns` [bool](#)

A value indicating whether columns with unique constraints are allowed to be written. If [true](#), unique columns will be included in the write operation; otherwise, they will be excluded. The default value is [false](#).

`allowAutoDateColumns` [bool](#)

A value indicating whether columns with automatic date constraints (e.g., timestamps) are allowed to be written. If [true](#), such columns will be included in the write operation; otherwise, they will be excluded. The default value is [false](#).

Returns

[int](#)

The number of records successfully written to the database.

Remarks

This method performs a bulk write operation, which can improve performance when writing large amounts of data. Ensure that the provided `relmContext` is properly configured and that the database schema matches the structure of the data being written. The behavior of the write operation can be customized using the provided options.

## WriteToDatabase(RelmContextOptionsBuilder, int, bool, bool, bool)

Writes the current object to the database using the table specified in the `RelmTable` attribute.

```
public int WriteToDatabase(RelmContextOptionsBuilder relmContextOptions, int batchSize = 100, bool allowAutoIncrementColumns = false, bool allowPrimaryKeyColumns = false, bool allowUniqueColumns = false, bool allowAutoDateColumns = false)
```

Parameters

`relmContextOptions` [RelmContextOptionsBuilder](#)

The options used to configure the database context.

`batchSize` [int](#)

The number of records to process in each batch. Defaults to 100.

`allowAutoIncrementColumns` [bool](#)

A value indicating whether columns with auto-increment constraints are allowed to be written. Defaults to [false](#).

`allowPrimaryKeyColumns` [bool](#)

A value indicating whether primary key columns are allowed to be written. Defaults to [false](#).

`allowUniqueColumns` [bool](#)

A value indicating whether columns with unique constraints are allowed to be written. Defaults to [false](#).

### allowAutoDateColumns [bool](#)

A value indicating whether columns with automatic date generation (e.g., timestamps) are allowed to be written. Defaults to [false](#).

## Returns

### [int](#)

The number of records successfully written to the database.

## Remarks

This method performs a bulk write operation, which can improve performance when writing large datasets. Ensure that the specified options and constraints align with the database schema to avoid runtime errors.

## WriteToDatabase(MySqlConnection, MySqlTransaction, int, bool, bool, bool, bool)

Writes the current object to the database using the table specified in the [RelmTable](#) attribute.

```
public int WriteToDatabase(MySqlConnection existingConnection, MySqlTransaction  
sqlTransaction = null, int batchSize = 100, bool allowAutoIncrementColumns = false, bool  
allowPrimaryKeyColumns = false, bool allowUniqueColumns = false, bool allowAutoDateColumns  
= false)
```

## Parameters

### [existingConnection](#) MySqlConnection

An existing and open MySql.Data.MySqlClient.MySqlConnection to use for writing the data. The connection must remain open for the duration of the operation.

### [sqlTransaction](#) MySqlTransaction

An optional MySql.Data.MySqlClient.MySqlTransaction under which the data will be written. If not provided, the operation will execute outside of a transaction.

## batchSize [int](#)

The number of rows to write to the database in each batch. Must be greater than zero. Defaults to 100.

## allowAutoIncrementColumns [bool](#)

A value indicating whether auto-increment columns should be included in the write operation. If [true](#), auto-increment columns will be written; otherwise, they will be excluded.

## allowPrimaryKeyColumns [bool](#)

A value indicating whether primary key columns should be included in the write operation. If [true](#), primary key columns will be written; otherwise, they will be excluded.

## allowUniqueColumns [bool](#)

A value indicating whether unique columns should be included in the write operation. If [true](#), unique columns will be written; otherwise, they will be excluded.

## allowAutoDateColumns [bool](#)

A value indicating whether auto-generated date columns (e.g., timestamps) should be included in the write operation. If [true](#), auto-date columns will be written; otherwise, they will be excluded.

## Returns

### [int](#)

The number of rows successfully written to the database.

## Remarks

This method performs a bulk write operation to the database. The table to which the data is written is determined by the [RelmTable](#) attribute applied to the object's type. Ensure that the database schema matches the structure of the object being written.

## WriteToDatabase(Enum, int, bool, bool, bool, bool)

Writes the current object to the database using the table specified in the [RelmTable](#) attribute.

```
public int WriteToDatabase(Enum connectionStringType, int batchSize = 100, bool  
allowAutoIncrementColumns = false, bool allowPrimaryKeyColumns = false, bool  
allowUniqueColumns = false, bool allowAutoDateColumns = false)
```

## Parameters

### connectionStringType [Enum](#)

The type of connection to use for the database operation.

### batchSize [int](#)

The number of items to write to the database per batch. Must be a positive integer. Defaults to 100.

### allowAutoIncrementColumns [bool](#)

A value indicating whether columns with auto-increment constraints are allowed to be written. Set to [true](#) to allow writing to such columns; otherwise, [false](#).

### allowPrimaryKeyColumns [bool](#)

A value indicating whether primary key columns are allowed to be written. Set to [true](#) to allow writing to primary key columns; otherwise, [false](#).

### allowUniqueColumns [bool](#)

A value indicating whether unique constraint columns are allowed to be written. Set to [true](#) to allow writing to such columns; otherwise, [false](#).

### allowAutoDateColumns [bool](#)

A value indicating whether columns with automatic date generation (e.g., timestamps) are allowed to be written. Set to [true](#) to allow writing to such columns; otherwise, [false](#).

## Returns

### [int](#)

The number of rows successfully written to the database.

## Remarks

This method performs a bulk write operation to the database. The table to which the data is written is determined by the `RelmTable` attribute applied to the current object's type. Ensure that the database schema matches the structure of the object being written.

## Events

# DtoTypeProcessor

Occurs when a DTO (Data Transfer Object) is processed.

```
public event EventHandler<DtoEventArgs> DtoTypeProcessor
```

## Event Type

[EventHandler](#) <[DtoEventArgs](#)>

## Remarks

This event is triggered after a DTO has been processed, providing the processed DTO as part of the event data. Subscribers can use this event to perform additional actions or handle the processed DTO as needed.

# Class RelmModelApartment

Namespace: [CoreRelm.Models](#)

Assembly: CoreRelm.dll

Represents an apartment entity within the Relm data model, including user and member association information.

```
public class RelmModelApartment : RelmModel, IRelmModel, IRelmModelApartment
```

## Inheritance

[object](#) ← [RelmModel](#) ← RelmModelApartment

## Implements

[IRelmModel](#), [IRelmModelApartment](#)

## Inherited Members

[RelmModel.DtoTypeProcessor](#), [RelmModel.Id](#), [RelmModel.Active](#), [RelmModel.InternalId](#),  
[RelmModel.CreateDate](#), [RelmModel.LastUpdated](#), [RelmModel.ResetCoreAttributes\(bool, bool\)](#),  
[RelmModel.ResetWithData\(DataRow, string\)](#), [RelmModel.GetUnderscoreProperties\(bool\)](#),  
[RelmModel.WriteToDatabase\(Enum, int, bool, bool, bool, bool\)](#),  
[RelmModel.WriteToDatabase\(MySqlConnection, MySqlTransaction, int, bool, bool, bool, bool\)](#),  
[RelmModel.WriteToDatabase\(IRelmContext, int, bool, bool, bool, bool\)](#),  
[RelmModel.WriteToDatabase\(IRelmQuickContext, int, bool, bool, bool, bool\)](#),  
[RelmModel.WriteToDatabase\(RelmContextOptionsBuilder, int, bool, bool, bool, bool\)](#),  
[RelmModel.CopyFromSource<T>\(T\)](#),  
[RelmModel.GenerateDTO\(IEnumerable<string>, IEnumerable<string>, string, Func<IRelmModel, Dictionary<string, object>>, int\)](#),  
[RelmModel.Duplicate\(\)](#), [object.Equals\(object\)](#), [object.Equals\(object, object\)](#),  
[object.GetHashCode\(\)](#), [object.GetType\(\)](#), [object.MemberwiseClone\(\)](#),  
[object.ReferenceEquals\(object, object\)](#), [object.ToString\(\)](#)

## Extension Methods

[ModelExtensions.LoadDataLoaderField<T, S>\(T, IRelmContext, Expression<Func<T, S>>\)](#),  
[ModelExtensions.LoadDataLoaderField<T, S>\(T, IRelmQuickContext, Expression<Func<T, S>>\)](#),  
[ModelExtensions.LoadDataLoaderField<T, S>\(T, RelmContextOptionsBuilder, Expression<Func<T, S>>\)](#),  
[ModelExtensions.LoadForeignKeyField<T, R>\(T, IRelmContext, Expression<Func<T, ICollection<R>>>\)](#),  
[ModelExtensions.LoadForeignKeyField<T, R>\(T, IRelmContext, Expression<Func<T, ICollection<R>>>, Expression<Func<R, object>>\)](#),  
[ModelExtensions.LoadForeignKeyField<T, R>\(T, IRelmContext, Expression<Func<T, R>>\)](#),



## Remarks

This class provides properties for identifying the apartment and its associated user and member. It is typically used in scenarios where apartment membership and user linkage are required within the Relm system. Inherits from RelmModel and implements IRelmModelApartment.

## Constructors

### RelmModelApartment()

Initializes a new instance of the RelmModelApartment class.

```
public RelmModelApartment()
```

### RelmModelApartment(IRelmModel)

Initializes a new instance of the RelmModelApartment class by copying data from the specified model.

```
public RelmModelApartment(IRelmModel fromModel)
```

#### Parameters

`fromModel` [IRelmModel](#)

The source model from which to copy data. Cannot be null.

### RelmModelApartment(DataRow, string)

Initializes a new instance of the RelmModelApartment class using the specified data row and optional alternate table name.

```
public RelmModelApartment(DataRow ModelData, string AlternateTableName = null)
```

#### Parameters

`ModelData` [DataRow](#)

The DataRow containing the data to initialize the apartment model. Cannot be null.

#### **AlternateTableName** [string](#)

An optional table name to use instead of the default. If null, the class name is used.

## Properties

### ApartmentId

Gets or sets the unique identifier for the apartment.

```
[RelmColumn(null, -1, null, true, false, false, false, null, null, false, false, false)]
public string ApartmentId { get; set; }
```

### Property Value

#### [string](#)

### Member

Gets or sets the associated Relm member for this entity.

```
[RelmColumn(null, -1, null, true, false, false, false, null, null, false, false, false)]
public IRelmMember Member { get; set; }
```

### Property Value

#### [IRelmMember](#)

### UserEmail

Gets or sets the email address associated with the user.

```
[RelmColumn(null, -1, null, true, false, false, false, null, null, false, false, false)]
public string UserEmail { get; set; }
```

## Property Value

[string](#) ↗

## UserId

Gets or sets the unique identifier for the user.

```
[RelmColumn(null, -1, null, true, false, false, false, null, null, false, false, false)]  
public int UserId { get; set; }
```

## Property Value

[int](#) ↗

## UserName

Gets or sets the user name associated with the entity.

```
[RelmColumn(null, -1, null, true, false, false, false, null, null, false, false, false)]  
public string UserName { get; set; }
```

## Property Value

[string](#) ↗

## Methods

### SetMemberApartment(int, bool)

Sets the apartment membership information for the specified member and updates tracking fields as specified.

```
public IRelmModelApartment SetMemberApartment(int MemberId, bool UpdateCreate = true)
```

## Parameters

**MemberId** [int](#)

The identifier of the member whose apartment information is to be set.

**UpdateCreate** [bool](#)

true to update the creation tracking fields in addition to the last updated fields; otherwise, false. The default is true.

Returns

[IRelmModelApartment](#)

The current instance with updated membership information.

# Class RelmQuickContext

Namespace: [CoreRelm.Models](#)

Assembly: CoreRelm.dll

Provides a quick-start context for working with Relm data models and MySQL database connections, supporting configuration, data set management, and transactional operations.

```
[Obsolete("RelmQuickContext is deprecated and will be removed in future versions. Please use  
RelmContext(autoInitializeDataSets: false) instead.")]  
public class RelmQuickContext : IRelmQuickContext, IDisposable
```

## Inheritance

[object](#) ← RelmQuickContext

## Implements

[IRelmQuickContext](#), [IDisposable](#)

## Inherited Members

[object.Equals\(object\)](#), [object.Equals\(object, object\)](#), [object.GetHashCode\(\)](#), [object.GetType\(\)](#),  
[object.MemberwiseClone\(\)](#), [object.ReferenceEquals\(object, object\)](#), [object.ToString\(\)](#)

## Remarks

RelmQuickContext simplifies the setup and management of database contexts for Relm-based applications. It supports multiple initialization patterns, including connection strings, MySqlConnection objects, and configuration builders. The context manages the lifecycle of database connections and transactions, and provides methods for accessing, querying, and writing data sets. This class is not thread-safe; each instance should be used by a single thread at a time. Dispose the context when finished to ensure all resources are released.

## Constructors

**RelmQuickContext(IRelmContextOptionsBuilder, bool, bool, bool, bool, int)**

Initializes a new instance of the RelmQuickContext class with the specified context options and configuration settings.

```
public RelmQuickContext(RelmContextOptionsBuilder optionsBuilder, bool autoOpenConnection =  
true, bool autoOpenTransaction = false, bool allowUserVariables = false, bool  
convertZeroDateTime = false, int lockWaitTimeoutSeconds = 0)
```

## Parameters

### [optionsBuilder](#) [RelmContextOptionsBuilder](#)

The options builder used to configure the context. Cannot be null.

### [autoOpenConnection](#) [bool](#)

Specifies whether the database connection should be automatically opened when the context is created. The default is [true](#).

### [autoOpenTransaction](#) [bool](#)

Specifies whether a database transaction should be automatically started when the context is created. The default is [false](#).

### [allowUserVariables](#) [bool](#)

Specifies whether user-defined variables are allowed in SQL statements executed by the context. The default is [false](#).

### [convertZeroDateTime](#) [bool](#)

Specifies whether zero date/time values should be converted to null when reading from the database. The default is [false](#).

### [lockWaitTimeoutSeconds](#) [int](#)

The number of seconds to wait for a database lock before timing out. The default is 0, which uses the server's default lock wait timeout.

## Remarks

All configuration settings are validated before the context is initialized. This constructor allows fine-grained control over connection and transaction behavior, as well as SQL compatibility options.

## Exceptions

### [ArgumentNullException](#)

Thrown if `optionsBuilder` is null.

## RelmQuickContext(MySqlConnection, MySqlTransaction, bool, bool, bool, int)

Initializes a new instance of the RelmQuickContext class using the specified MySQL connection, transaction, and context options.

```
public RelmQuickContext(MySqlConnection connection, MySqlTransaction transaction, bool autoOpenConnection = true, bool allowUserVariables = false, bool convertZeroDateTime = false, int lockWaitTimeoutSeconds = 0)
```

### Parameters

`connection` MySqlConnection

The MySqlConnection to use for database operations. Must not be null and should be open or capable of being opened if `autoOpenConnection` is [true](#).

`transaction` MySqlTransaction

The MySqlTransaction to associate with the context. Can be null if no transaction is required.

`autoOpenConnection` [bool](#)

Specifies whether the context should automatically open the connection if it is not already open. If [true](#), the connection will be opened as needed.

`allowUserVariables` [bool](#)

Indicates whether user-defined variables are allowed in SQL statements executed by the context. Set to [true](#) to enable support for user variables.

`convertZeroDateTime` [bool](#)

Specifies whether zero date/time values from the database should be converted to `DateTime.MinValue` instead of throwing an exception. Set to [true](#) to enable conversion.

`lockWaitTimeoutSeconds` [int](#)

The maximum number of seconds to wait for a database lock before timing out. Specify 0 to use the default server setting.

## RelmQuickContext(MySqlConnection, bool, bool, bool, bool, int)

Initializes a new instance of the RelmQuickContext class using the specified MySQL connection and configuration options.

```
public RelmQuickContext(MySqlConnection connection, bool autoOpenConnection = true, bool  
autoOpenTransaction = false, bool allowUserVariables = false, bool convertZeroDateTime =  
false, int lockWaitTimeoutSeconds = 0)
```

### Parameters

**connection** MySqlConnection

The MySqlConnection to use for database operations. Cannot be null.

**autoOpenConnection** [bool](#)

Specifies whether the database connection should be automatically opened when the context is created. If [true](#), the connection is opened; otherwise, it remains closed until explicitly opened.

**autoOpenTransaction** [bool](#)

Specifies whether a database transaction should be automatically started when the context is created. If [true](#), a transaction is started; otherwise, no transaction is started by default.

**allowUserVariables** [bool](#)

Specifies whether user-defined variables are allowed in SQL statements executed by this context. If [true](#), user variables are permitted.

**convertZeroDateTime** [bool](#)

Specifies whether zero date/time values from the database should be converted to DateTime.MinValue. If [true](#), zero date/time values are converted; otherwise, they are not.

**lockWaitTimeoutSeconds** [int](#)

The number of seconds to wait for a database lock before timing out. Specify 0 to use the default server setting.

## RelmQuickContext(Enum, bool, bool, bool, bool, int)

Initializes a new instance of the RelmQuickContext class with the specified connection options and behavior settings.

```
public RelmQuickContext(Enum connectionStringType, bool autoOpenConnection = true, bool autoOpenTransaction = false, bool allowUserVariables = false, bool convertZeroDateTime = false, int lockWaitTimeoutSeconds = 0)
```

## Parameters

**connectionStringType** [Enum](#)

The type of connection string to use for configuring the database context. Determines how the context connects to the underlying data source.

**autoOpenConnection** [bool](#)

true to automatically open the database connection when the context is created; otherwise, false. The default is true.

**autoOpenTransaction** [bool](#)

true to automatically begin a transaction when the context is created; otherwise, false. The default is false.

**allowUserVariables** [bool](#)

true to allow user-defined variables in database queries; otherwise, false. The default is false.

**convertZeroDateTime** [bool](#)

true to convert zero date/time values from the database to DateTime.MinValue; otherwise, false. The default is false.

**lockWaitTimeoutSeconds** [int](#)

The maximum number of seconds to wait for a database lock before timing out. Specify 0 to use the default timeout.

## Remarks

Use this constructor to customize context behavior such as connection management, transaction handling, and query options. These settings affect how the context interacts with the database and may impact performance or compatibility depending on the database provider.

## RelmQuickContext(string, bool, bool, bool, bool, int)

Initializes a new instance of the RelmQuickContext class with the specified connection details and configuration options.

```
public RelmQuickContext(string connectionDetails, bool autoOpenConnection = true, bool  
autoOpenTransaction = false, bool allowUserVariables = false, bool convertZeroDateTime =  
false, int lockWaitTimeoutSeconds = 0)
```

### Parameters

`connectionDetails` [string](#)

The connection string or details used to establish a database connection.

`autoOpenConnection` [bool](#)

true to automatically open the database connection upon context initialization; otherwise, false.

`autoOpenTransaction` [bool](#)

true to automatically begin a transaction when the context is initialized; otherwise, false.

`allowUserVariables` [bool](#)

true to allow the use of user-defined variables in database queries; otherwise, false.

`convertZeroDateTime` [bool](#)

true to convert zero date/time values to null when reading from the database; otherwise, false.

`lockWaitTimeoutSeconds` [int](#)

The number of seconds to wait for a database lock before timing out. Specify 0 to use the default timeout.

## Properties

### ContextOptions

Gets the options builder used to configure the context.

```
public RelmContextOptionsBuilder ContextOptions { get; }
```

## Property Value

[RelmContextOptionsBuilder](#)

## Remarks

Use this property to customize context-specific settings before building or initializing the context. Changes to the options should be made prior to finalizing the context configuration.

## Methods

### BeginTransaction()

Begins a database transaction on the current MySQL connection and returns a transaction object for managing the transaction lifecycle.

```
public MySqlTransaction BeginTransaction()
```

## Returns

MySqlTransaction

A MySql.Data.MySqlClient.MySqlTransaction object representing the started transaction. If a transaction is already active, returns the existing transaction.

## Remarks

If a transaction is already in progress, this method returns the existing transaction rather than starting a new one. The returned transaction must be committed or rolled back to complete the operation. Ensure that the underlying database connection is open before calling this method.

### BulkTableWrite<T>(T, string, MySqlTransaction, Type, int, bool, bool, bool)

Writes a collection of data to a database table in bulk, optionally using batching and transaction support.

```
public int BulkTableWrite<T>(T source, string table = null, MySqlTransaction sqlTransaction = null, Type forceType = null, int batchSize = 100, bool allowAutoIncrementColumns = false, bool allowPrimaryKeyColumns = false, bool allowUniqueColumns = false)
```

## Parameters

**source** `T`

The source data to write to the table. This can be a collection or a single object of type `T`.

**table** `string`

The name of the target database table. If `null`, the table name is inferred from the type `T`.

**sqlTransaction** `MySqlTransaction`

An optional `MySQL.Data.MySqlClient.MySqlTransaction` to use for the bulk write operation. If `null`, the operation is executed without an explicit transaction.

**forceType** `Type`

An optional type to override the inferred type of `T` when mapping columns. If `null`, the type of `T` is used.

**batchSize** `int`

The maximum number of rows to write in each batch. Must be greater than zero.

**allowAutoIncrementColumns** `bool`

Indicates whether auto-increment columns are included in the write operation. If `true`, auto-increment columns are written; otherwise, they are excluded.

**allowPrimaryKeyColumns** `bool`

Indicates whether primary key columns are included in the write operation. If `true`, primary key columns are written; otherwise, they are excluded.

**allowUniqueColumns** `bool`

Indicates whether unique columns are included in the write operation. If `true`, unique columns are written; otherwise, they are excluded.

## Returns

[int](#)

The number of rows successfully written to the database table.

## Type Parameters

T

The type of the data objects to be written to the table.

## Remarks

This method is optimized for high-performance bulk inserts and can be used with or without an explicit transaction. Adjusting batch size may affect performance and resource usage. Column inclusion options allow fine-grained control over which table columns are written, which can be useful for tables with auto-increment, primary key, or unique constraints.

## CommitTransaction()

Commits the current database transaction, finalizing all changes made during the transaction.

```
public void CommitTransaction()
```

## Remarks

If no active transaction exists, this method has no effect. After committing, the transaction is considered complete and cannot be rolled back. This method should be called only after all intended changes have been made within the transaction scope.

## Dispose()

Releases all resources used by the current instance of the class.

```
public void Dispose()
```

## Remarks

Call this method when you are finished using the object to free unmanaged resources and perform other cleanup operations. After calling Dispose, the object should not be used further.

## Dispose(bool)

Releases the unmanaged resources used by the object and optionally releases the managed resources.

```
protected virtual void Dispose(bool disposing)
```

### Parameters

**disposing** [bool](#)

true to release both managed and unmanaged resources; false to release only unmanaged resources.

### Remarks

This method is called by both the public `Dispose()` method and the finalizer. When `disposing` is true, this method disposes of managed resources such as attached properties that implement `IDisposable`. Override this method in a derived class to release additional resources.

## DoDatabaseWork(string, Dictionary<string, object>, bool, bool)

Executes a database operation using the specified SQL query and parameters, with optional exception handling and transaction support.

```
public void DoDatabaseWork(string query, Dictionary<string, object> parameters = null, bool throwException = true, bool useTransaction = false)
```

### Parameters

**query** [string](#)

The SQL query to execute against the database. Cannot be null or empty.

**parameters** [Dictionary](#)<[string](#), [object](#)>

An optional dictionary containing parameter names and values to be used with the query. If null, the query is executed without parameters.

**throwException** [bool](#)

Specifies whether to throw an exception if the database operation fails. Set to [true](#) to throw exceptions; otherwise, errors are suppressed.

## **useTransaction** [bool](#)

Specifies whether to execute the operation within a database transaction. Set to [true](#) to use a transaction; otherwise, the operation is executed without transactional support.

## Remarks

If **useTransaction** is [true](#), the operation is performed within a transaction, which is committed if successful or rolled back on failure. When **throwException** is [false](#), errors are suppressed and no exception is thrown, but the operation may not complete as expected.

## **DoDatabaseWork(string, Func<MySqlCommand, object>, bool, bool)**

Executes a database operation using the specified SQL query and callback, with optional exception handling and transaction support.

```
public void DoDatabaseWork(string query, Func<MySqlCommand, object> actionCallback, bool throwException = true, bool useTransaction = false)
```

## Parameters

### **query** [string](#)

The SQL query to execute against the database. Cannot be null or empty.

### **actionCallback** [Func](#)<MySqlCommand, [object](#)>

A callback function that receives the prepared MySql.Data.MySqlClient.MySqlCommand and performs custom logic. The function should return an object representing the result of the operation.

### **throwException** [bool](#)

Specifies whether to throw an exception if the database operation fails. If [true](#), exceptions are propagated; otherwise, errors are suppressed.

### **useTransaction** [bool](#)

Specifies whether the database operation should be executed within a transaction. If [true](#), the operation is wrapped in a transaction.

## Remarks

Use this method to perform custom database work, such as executing queries or commands, with control over error handling and transactional behavior. The `actionCallback` allows you to define how the `MySql.Data.MySqlClient.MySqlCommand` is used, such as reading results or executing non-query commands.

## DoDatabaseWork<T>(string, Dictionary<string, object>, bool, bool)

Executes a database query and returns the result as the specified type.

```
public T DoDatabaseWork<T>(string query, Dictionary<string, object> parameters = null, bool throwException = true, bool useTransaction = false)
```

### Parameters

#### query [string](#)

The SQL query to execute against the database. Cannot be null or empty.

#### parameters [Dictionary](#)<[string](#), [object](#)>

An optional dictionary of parameter names and values to be used with the query. If null, no parameters are applied.

#### throwException [bool](#)

Specifies whether to throw an exception if the query fails. If [true](#), an exception is thrown on error; otherwise, the method returns the default value of `T`.

#### useTransaction [bool](#)

Specifies whether to execute the query within a database transaction. If [true](#), the query is executed in a transaction.

### Returns

#### T

The result of the query cast to the specified type `T`. Returns the default value of `T` if the query fails and `throwException` is [false](#).

## Type Parameters

T

The type of the result to return. Must be compatible with the query result.

## Remarks

If `useTransaction` is `true`, the query is executed within a transaction, which may impact performance and rollback behavior. Ensure that `query` and `parameters` are valid for the target database.

## DoDatabaseWork<T>(string, Func<MySqlCommand, object>, bool, bool)

Executes a database operation using the specified query and callback, optionally within a transaction, and returns the result as the specified type.

```
public T DoDatabaseWork<T>(string query, Func<MySqlCommand, object> actionCallback, bool throwException = true, bool useTransaction = false)
```

## Parameters

`query` `string`

The SQL query to execute against the database. Cannot be null or empty.

`actionCallback` `Func<MySqlCommand, object>`

A callback that receives the prepared `MySQL.Data.MySqlClient.MySqlCommand` and performs the desired operation. The result of this callback is returned as type `T`.

`throwException` `bool`

Specifies whether to throw an exception if the database operation fails. If `true`, exceptions are thrown; otherwise, failures are handled silently.

`useTransaction` `bool`

Specifies whether to execute the database operation within a transaction. If `true`, the operation is performed in a transaction; otherwise, no transaction is used.

## Returns

T

The result of the database operation as type T.

## Type Parameters

T

The type of the result returned by the database operation.

## Remarks

If `useTransaction` is [true](#), the operation is executed within a transaction, which is committed if successful or rolled back on failure. The behavior when an error occurs depends on the value of `throwException`.

## EndConnection(bool)

Ends the current database connection and optionally commits the active transaction.

```
public void EndConnection(bool commitTransaction = true)
```

## Parameters

`commitTransaction` [bool](#)

Specifies whether to commit the active transaction before closing the connection. Set to [true](#) to commit; otherwise, the transaction will not be committed.

## Remarks

If a transaction is active and `commitTransaction` is [true](#), the transaction is committed before the connection is closed. If no connection is open, this method has no effect.

## ~RelmQuickContext()

Finalizes the RelmQuickContext instance and releases unmanaged resources before the object is reclaimed by garbage collection.

```
protected ~RelmQuickContext()
```

## Remarks

This destructor is called automatically by the garbage collector when the object is no longer accessible. It ensures that any unmanaged resources held by the RelmQuickContext are properly released. If you have already called Dispose, the finalizer will not release resources again.

## FirstOrDefault<T>(Expression<Func<T, bool>>, bool)

Returns the first element of type T that matches the specified predicate, or the default value if no such element is found.

```
public T FirstOrDefault<T>(Expression<Func<T, bool>> predicate, bool loadDataLoaders =  
false) where T : IRelmModel, new()
```

## Parameters

**predicate** [Expression<Func<T, bool>>](#)

An expression that defines the conditions the returned element must satisfy.

**loadDataLoaders** [bool](#)

true to load related data loaders for the returned element; otherwise, false. The default is false.

## Returns

T

The first element of type T that matches the predicate, or default(T) if no match is found.

## Type Parameters

T

The type of model to query. Must implement IRelmModel and have a parameterless constructor.

## GetBulkTableWriter<T>(string, bool, bool, bool, bool, bool)

Creates and returns a bulk table writer for efficiently inserting multiple records of type T into the database.

```
public BulkTableWriter<T> GetBulkTableWriter<T>(string insertQuery = null, bool  
useTransaction = false, bool throwException = true, bool allowAutoIncrementColumns = false,  
bool allowPrimaryKeyColumns = false, bool allowUniqueColumns = false)
```

## Parameters

### insertQuery [string](#)

An optional custom SQL insert query to use for bulk operations. If null, a default query is generated based on the type T.

### useTransaction [bool](#)

Specifies whether the bulk insert operation should be performed within a database transaction. Set to [true](#) to ensure atomicity; otherwise, [false](#).

### throwException [bool](#)

Specifies whether exceptions encountered during the bulk operation should be thrown. If [true](#), exceptions are propagated; otherwise, errors are suppressed.

### allowAutoIncrementColumns [bool](#)

Indicates whether auto-increment columns are included in the insert operation. Set to [true](#) to allow explicit values for such columns.

### allowPrimaryKeyColumns [bool](#)

Indicates whether primary key columns are included in the insert operation. Set to [true](#) to allow explicit values for primary keys.

### allowUniqueColumns [bool](#)

Indicates whether unique columns are included in the insert operation. Set to [true](#) to allow explicit values for unique columns.

## Returns

### [BulkTableWriter](#)<T>

A [BulkTableWriter](#)<T> instance configured for bulk insertion of entities of type T into the database.

## Type Parameters

T

The type of entities to be written to the database table.

## Remarks

Use this method to optimize large-scale data insertions. The returned [BulkTableWriter<T>](#) provides methods for writing batches of data efficiently. Ensure that the configuration flags match your table schema and insertion requirements.

## GetDataTable<T>(string, Dictionary<string, object>, bool)

Executes the specified query and returns a collection of results mapped to the specified type.

```
public IEnumerable<T> GetDataTable<T>(string query, Dictionary<string, object> parameters = null, bool throwException = true)
```

## Parameters

**query** [string](#)

The SQL query to execute against the data source.

**parameters** [Dictionary](#)<[string](#), [object](#)>

An optional dictionary of parameter names and values to be applied to the query. If null, the query is executed without parameters.

**throwException** [bool](#)

true to throw an exception if the query fails; otherwise, false to suppress exceptions and return an empty collection.

## Returns

[IEnumerable](#)<T>

An enumerable collection of objects of type T representing the query results. Returns an empty collection if no results are found or if an error occurs and throwException is false.

## Type Parameters

T

The type to which each result row will be mapped.

## Remarks

The method maps each row in the result set to an instance of type T. If throwException is false and an error occurs during query execution, the method returns an empty collection instead of throwing an exception.

## GetDataObject<T>(string, Dictionary<string, object>, bool)

Retrieves a data object of the specified type by executing the provided query string with optional parameters.

```
public T GetDataObject<T>(string query, Dictionary<string, object> parameters = null, bool throwException = true) where T : IRelmModel, new()
```

## Parameters

**query** [string](#)

The query string used to select the data object. Cannot be null or empty.

**parameters** [Dictionary](#)<[string](#), [object](#)>

An optional dictionary of parameter names and values to be used in the query. If null, no parameters are applied.

**throwException** [bool](#)

Specifies whether to throw an exception if the query fails. If [true](#), an exception is thrown on failure; otherwise, the method returns the default value of T.

## Returns

T

An instance of T representing the retrieved data object. Returns the default value of T if no data is found and **throwException** is [false](#).

## Type Parameters

T

The type of data object to retrieve. Must implement [IRelmModel](#) and have a parameterless constructor.

## GetDataObjects<T>(string, Dictionary<string, object>, bool)

Executes the specified query and returns a collection of data objects of type T that match the query criteria.

```
public IEnumerable<T> GetDataObjects<T>(string query, Dictionary<string, object> parameters  
= null, bool throwException = true) where T : IRelmModel, new()
```

## Parameters

query [string](#)

The query string used to select data objects. Must be a valid query for the underlying data source.

parameters [Dictionary](#)<[string](#), [object](#)>

An optional dictionary of parameter names and values to be used in the query. If [null](#), no parameters are applied.

throwException [bool](#)

Specifies whether to throw an exception if the query fails. If [true](#), an exception is thrown on error; otherwise, the method returns an empty collection.

## Returns

[IEnumerable](#)<T>

An enumerable collection of data objects of type T that satisfy the query. Returns an empty collection if no matching objects are found or if the query fails and [throwException](#) is [false](#).

## Type Parameters

T

The type of data object to return. Must implement [IRelmModel](#) and have a parameterless constructor.

## GetDataRow(string, Dictionary<string, object>, bool)

Executes the specified SQL query and returns the first matching data row from the result set.

```
public DataRow GetDataRow(string query, Dictionary<string, object> parameters = null, bool throwException = true)
```

### Parameters

**query** [string](#)

The SQL query to execute. Must be a valid statement that returns at least one row.

**parameters** [Dictionary](#)<[string](#), [object](#)>

An optional dictionary of parameter names and values to be applied to the query. If null, the query is executed without parameters.

**throwException** [bool](#)

true to throw an exception if no matching row is found; otherwise, false to return null.

### Returns

[DataRow](#)

A DataRow containing the first result of the query if found; otherwise, null if no matching row exists and throwException is false.

## GetDataSet(Type)

Retrieves an instance of a data set of the specified type.

```
public IRelmDataSetBase GetDataSet(Type dataSetType)
```

### Parameters

**dataSetType** [Type](#)

The type of the data set to retrieve. Must implement [IRelmDataSetBase](#).

## Returns

### [IRelmDataSetBase](#)

An instance of [IRelmDataSetBase](#) corresponding to the specified type.

## GetDataSet(Type, bool)

Retrieves the data set instance associated with the specified entity type. If the data set is not initialized, it can be created automatically or an exception can be thrown based on the provided option.

```
public IRelmDataSetBase GetDataSet(Type dataSetType, bool throwException)
```

## Parameters

### dataSetType [Type](#)

The type of the entity for which to retrieve the data set. Must correspond to a table that exists in the current database.

### throwException [bool](#)

Specifies whether to throw an exception if the data set for the specified type is not initialized. If [true](#), an exception is thrown; otherwise, a new data set is created if possible.

## Returns

### [IRelmDataSetBase](#)

An instance of [IRelmDataSetBase](#) representing the data set for the specified entity type. Returns a newly created instance if the data set was not previously initialized and [throwException](#) is [false](#).

## Remarks

If the data set for the specified type is not already initialized and [throwException](#) is [false](#), a default data loader is used to create and initialize the data set. This method only operates on types that are mapped to existing tables in the current database.

## Exceptions

## [InvalidOperationException](#)

Thrown if the specified type does not correspond to a table in the current database, if no attached property is found for the type, or if the data set cannot be initialized and `throwException` is [true](#).

## GetDataSetType(Type)

Retrieves an instance of a data set corresponding to the specified type.

```
public IRelmDataSetBase GetDataSetType(Type dataSetType)
```

### Parameters

`dataSetType` [Type](#)

The type of the data set to retrieve. Must be a valid type that implements the required data set interface.

### Returns

[IRelmDataSetBase](#)

An object representing the data set of the specified type.

## GetDataSetType(Type, bool)

Gets the dataset of the given type.

```
public IRelmDataSetBase GetDataSetType(Type dataSetType, bool throwException)
```

### Parameters

`dataSetType` [Type](#)

Type of the dataset.

`throwException` [bool](#)

Whether to throw an exception if the dataset is not found.

Returns

### [IRelmDataSetBase](#)

An IDALDataSetBase instance of the given type.

Exceptions

### [InvalidOperationException](#)

Thrown when no matching dataset is found.

## GetDataSetType<T>()

Retrieves a data set instance for the specified model type.

```
public IRelmDataSet<T> GetDataSetType<T>() where T : IRelmModel, new()
```

Returns

### [IRelmDataSet<T>](#)

An [IRelmDataSet<T>](#) instance associated with the specified model type.

Type Parameters

T

The type of model for which to retrieve the data set. Must implement [IRelmModel](#) and have a parameterless constructor.

## GetDataSetType<T>(bool)

Gets the dataset of the given type.

```
public IRelmDataSet<T> GetDataSetType<T>(bool throwException) where T : IRelmModel, new()
```

Parameters

`throwException bool`

Whether to throw an exception if the dataset is not found.

Returns

[IRelmDataSet<T>](#)

An instance of IDALDataSet of the specified type.

Type Parameters

`T`

The type of the dataset, which should inherit from CS\_DbModel.

Exceptions

[InvalidOperationException](#)

Thrown when no matching dataset is found.

## GetDataSet<T>()

Retrieves a data set for the specified model type.

```
public IRelmDataSet<T> GetDataSet<T>() where T : IRelmModel, new()
```

Returns

[IRelmDataSet<T>](#)

An [IRelmDataSet<T>](#) instance for the specified model type `T`.

Type Parameters

`T`

The type of model to retrieve. Must implement [IRelmModel](#) and have a parameterless constructor.

## GetDataSet<T>(bool)

Retrieves a strongly typed data set for the specified model type.

```
public IRelmDataSet<T> GetDataSet<T>(bool throwException) where T : IRelmModel, new()
```

### Parameters

**throwException** [bool](#)

Specifies whether to throw an exception if the data set cannot be retrieved. If [true](#), an exception is thrown on failure; otherwise, [null](#) is returned.

### Returns

[IRelmDataSet](#)<T>

An [IRelmDataSet](#)<T> instance representing the data set for the specified model type, or [null](#) if the data set cannot be retrieved and **throwException** is [false](#).

### Type Parameters

T

The type of model for which to retrieve the data set. Must implement [IRelmModel](#) and have a parameterless constructor.

## GetDataTable(string, Dictionary<string, object>, bool)

Executes the specified SQL query and returns the results as a [DataTable](#).

```
public DataTable GetDataTable(string query, Dictionary<string, object> parameters = null,  
    bool throwException = true)
```

### Parameters

**query** [string](#)

The SQL query to execute against the database. Must be a valid query string.

**parameters** [Dictionary](#)<[string](#), [object](#)>

An optional dictionary containing parameter names and values to be used in the query. If [null](#), no parameters are applied.

**throwException** [bool](#)

Specifies whether to throw an exception if the query fails. If [true](#), an exception is thrown on error; otherwise, the method returns [null](#).

Returns

[DataTable](#)

A [DataTable](#) containing the results of the query, or [null](#) if the query fails and **throwException** is [false](#).

Remarks

The returned [DataTable](#) will contain all rows and columns produced by the query. If the query does not return any results, the [DataTable](#) will be empty. Ensure that the query and parameters are properly formatted to avoid runtime errors.

## GetIdFromInternalId(string, string)

Retrieves the external identifier associated with the specified internal identifier for a given table.

```
public string GetIdFromInternalId(string table, string InternalId)
```

Parameters

**table** [string](#)

The name of the table in which to look up the identifier. Cannot be null or empty.

**InternalId** [string](#)

The internal identifier whose corresponding external identifier is to be retrieved. Cannot be null or empty.

Returns

## [string](#)

A string containing the external identifier corresponding to the specified internal identifier. Returns null if no matching identifier is found.

## GetLastInsertId()

Retrieves the identifier of the most recently inserted row for the current context.

```
public string GetLastInsertId()
```

Returns

## [string](#)

A string containing the last inserted row identifier. Returns an empty string if no row has been inserted.

## GetScalar<T>(string, Dictionary<string, object>, bool)

Executes the specified SQL query and returns the first column of the first row in the result set, cast to the specified type.

```
public T GetScalar<T>(string query, Dictionary<string, object> parameters = null, bool throwException = true)
```

Parameters

### [query](#) [string](#)

The SQL query to execute. Must be a valid statement that returns a single value.

### [parameters](#) [Dictionary](#)<[string](#), [object](#)>

An optional dictionary of parameter names and values to be applied to the query. If null, the query is executed without parameters.

### [throwException](#) [bool](#)

true to throw an exception if the query fails or returns no result; otherwise, false to return the default value of type T.

## Returns

T

The value of the first column of the first row in the result set, cast to type T. Returns the default value of T if no result is found and throwException is false.

## Type Parameters

T

The type to which the scalar result will be cast.

## Remarks

Use this method to efficiently retrieve a single value from the database, such as a count or aggregate. If throwException is set to true and the query fails or returns no result, an exception will be thrown.

## Get<T>(bool)

Retrieves a collection of entities of type T from the associated data set.

```
public ICollection<T> Get<T>(bool loadDataLoaders = false) where T : IRelmModel, new()
```

## Parameters

loadDataLoaders [bool](#)

Specifies whether to load associated data loaders for each entity. Set to [true](#) to include related data loaders; otherwise, only the entities are loaded.

## Returns

[ICollection](#)<T>

A collection containing all entities of type T from the data set. The collection is empty if no entities are present.

## Type Parameters

T

The type of model to retrieve. Must implement [IRelmModel](#) and have a parameterless constructor.

## Exceptions

[InvalidOperationException](#)

Thrown if the data set for type T is not initialized.

## Get<T>(Expression<Func<T, bool>>, bool)

Retrieves a collection of entities of type T that satisfy the specified predicate.

```
public ICollection<T> Get<T>(Expression<Func<T, bool>> predicate, bool loadDataLoaders = false) where T : IRelmModel, new()
```

## Parameters

**predicate** [Expression](#)<[Func](#)<T, [bool](#)>>

An expression that defines the conditions each entity must satisfy to be included in the result.

**loadDataLoaders** [bool](#)

true to load related data loaders for each entity; otherwise, false. The default is false.

## Returns

[ICollection](#)<T>

A collection of entities of type T that match the specified predicate. The collection will be empty if no entities satisfy the predicate.

## Type Parameters

T

The type of entity to retrieve. Must implement IRelmModel and have a parameterless constructor.

## HasDataSet(Type, bool)

Checks if the DALDataSet of a specific type is attached to the current DALContext instance.

```
public bool HasDataSet(Type dataSetType, bool throwException = true)
```

### Parameters

**dataSetType** [Type](#)

The System.Type of the dataset to check.

**throwException** [bool](#)

Whether to throw an exception if the dataset is not found.

### Returns

[bool](#)

True if the dataset exists, otherwise false.

## HasDataSet<T>(bool)

Checks if the DALDataSet of a specific type is attached to the current DALContext instance.

```
public bool HasDataSet<T>(bool throwException = true) where T : IRelmModel, new()
```

### Parameters

**throwException** [bool](#)

### Returns

[bool](#)

True if the dataset exists, otherwise false.

### Type Parameters

The type of the dataset, which should inherit from CS\_DbModel.

## HasTransaction()

Determines whether there is an active database transaction associated with the current context.

```
public bool HasTransaction()
```

Returns

[bool](#)

true if a database transaction is currently active; otherwise, false.

## OnConfigure(RelmContextOptionsBuilder)

Configures a triggerable event for after the database options are set, the connection is open, and any transactions have been started.

```
public virtual void OnConfigure(RelmContextOptionsBuilder OptionsBuilder)
```

Parameters

[OptionsBuilder](#) [RelmContextOptionsBuilder](#)

A builder used to configure options for the context. Cannot be null.

Remarks

Override this method to customize how the context is configured, such as specifying database providers or additional options. This method is typically called by the framework during context initialization.

## RollbackTransaction()

Rolls back the current database transaction, reverting all changes made during the transaction.

```
public void RollbackTransaction()
```

## Remarks

If no active transaction exists, this method has no effect. After calling this method, the transaction is considered closed and cannot be committed.

## RollbackTransactions()

Rolls back the current database transaction, reverting all changes made during the transaction.

```
public void RollbackTransactions()
```

## Remarks

If no active transaction exists, this method has no effect. After calling this method, the transaction is considered closed and cannot be committed.

## Run<T>(string, Dictionary<string, object>)

Executes the specified query and returns a collection of data objects of type T that match the query criteria.

```
public ICollection<T> Run<T>(string query, Dictionary<string, object> parameters = null)  
where T : IRelmModel, new()
```

## Parameters

**query** [string](#)

The query string to execute. Cannot be null or empty.

**parameters** [Dictionary](#)<[string](#), [object](#)>

An optional dictionary of parameters to be used with the query. If null, the query is executed without parameters.

## Returns

## [ICollection](#)<T>

A collection of objects of type T that satisfy the query. The collection will be empty if no matching objects are found.

### Type Parameters

T

The type of data object to return. Must implement IRelmModel and have a parameterless constructor.

### Exceptions

#### [ArgumentNullException](#)

Thrown if query is null or empty.

## SetDataLoader<T>(IRelmDataLoader<T>)

Sets the data loader to be used for the specified data set type.

```
public void SetDataLoader<T>(IRelmDataLoader<T> dataLoader) where T : RelmModel, new()
```

### Parameters

#### dataLoader [IRelmDataLoader](#)<T>

The data loader instance that will be associated with the data set of type T.

### Type Parameters

T

The type of model for which the data loader is being set. Must inherit from RelmModel and have a parameterless constructor.

### Exceptions

#### [InvalidOperationException](#)

Thrown if a data set for type T does not exist.

## StartConnection(bool, int)

Opens the database connection if it is not already open and optionally begins a new transaction.

```
public void StartConnection(bool autoOpenTransaction = false, int lockWaitTimeoutSeconds = 0)
```

### Parameters

**autoOpenTransaction** [bool](#)

true to automatically begin a new transaction after opening the connection; otherwise, false.

**lockWaitTimeoutSeconds** [int](#)

The lock wait timeout, in seconds, to set for the session. Specify a positive value to configure the timeout; otherwise, no change is made.

### Remarks

If the connection is opened and lockWaitTimeoutSeconds is greater than zero, the session's lock wait timeout and transaction isolation level are set. If autoOpenTransaction is true and the connection is open, a new transaction is started automatically.

### Exceptions

[InvalidOperationException](#)

Thrown if the database connection does not exist.

## Where<T>(Expression<Func<T, bool>>)

Filters the elements of the data set based on a specified predicate.

```
public IRelmDataSet<T> Where<T>(Expression<Func<T, bool>> predicate) where T : IRelmModel, new()
```

### Parameters

**predicate** [Expression](#)<[Func](#)<T, [bool](#)>>

An expression that defines the conditions each element must satisfy to be included in the result.  
Cannot be null.

Returns

[IRelmDataSet<T>](#)

An [IRelmDataSet<T>](#) containing elements that match the specified predicate.

Type Parameters

T

The type of model contained in the data set. Must implement [IRelmModel](#) and have a parameterless constructor.

Remarks

The returned data set is filtered according to the provided predicate and may be further queried or enumerated. This method does not modify the original data set.

Exceptions

[ArgumentNullException](#)

Thrown if `predicate` is null.

[InvalidOperationException](#)

Thrown if the data set for type T is not initialized.

**WriteToDatabase(IRelmModel, int, bool, bool, bool, bool)**

Writes the specified data model to the database using configurable options for batch size and column handling.

```
public int WriteToDatabase(IRelmModel relmModel, int batchSize = 100, bool  
allowAutoIncrementColumns = false, bool allowPrimaryKeyColumns = false, bool  
allowUniqueColumns = false, bool allowAutoDateColumns = false)
```

Parameters

## `relmModel` [IRelmModel](#)

The data model to be written to the database. Cannot be null.

## `batchSize` [int](#)

The maximum number of records to write in each batch operation. Must be greater than zero. The default is 100.

## `allowAutoIncrementColumns` [bool](#)

Specifies whether columns with auto-increment attributes are included in the write operation. Set to [true](#) to allow writing to auto-increment columns; otherwise, [false](#).

## `allowPrimaryKeyColumns` [bool](#)

Specifies whether primary key columns are included in the write operation. Set to [true](#) to allow writing to primary key columns; otherwise, [false](#).

## `allowUniqueColumns` [bool](#)

Specifies whether unique columns are included in the write operation. Set to [true](#) to allow writing to unique columns; otherwise, [false](#).

## `allowAutoDateColumns` [bool](#)

Specifies whether columns with automatic date attributes are included in the write operation. Set to [true](#) to allow writing to auto-date columns; otherwise, [false](#).

## Returns

### [int](#)

The number of records successfully written to the database.

## Remarks

Adjusting the batch size can impact performance, especially for large data sets. Enabling writing to auto-increment, primary key, unique, or auto-date columns may result in constraint violations depending on the database schema.

## `WriteToDatabase(IEnumerable<IRelmModel>, int, bool, bool, bool, bool)`

Writes the specified collection of Relm models to the database in batches, with options to control how certain column types are handled during insertion.

```
public int WriteToDatabase(IEnumerable<IRelmModel> relmModels, int batchSize = 100, bool  
allowAutoIncrementColumns = false, bool allowPrimaryKeyColumns = false, bool  
allowUniqueColumns = false, bool allowAutoDateColumns = false)
```

## Parameters

`relmModels` [IEnumerable](#)<[IRelmModel](#)>

The collection of Relm models to be written to the database. Cannot be null.

`batchSize` [int](#)

The maximum number of models to include in each batch write operation. Must be greater than zero.

`allowAutoIncrementColumns` [bool](#)

Specifies whether columns marked as auto-increment are included in the write operation. If [true](#), auto-increment columns are written; otherwise, they are excluded.

`allowPrimaryKeyColumns` [bool](#)

Specifies whether primary key columns are included in the write operation. If [true](#), primary key columns are written; otherwise, they are excluded.

`allowUniqueColumns` [bool](#)

Specifies whether unique columns are included in the write operation. If [true](#), unique columns are written; otherwise, they are excluded.

`allowAutoDateColumns` [bool](#)

Specifies whether columns with automatic date values are included in the write operation. If [true](#), auto-date columns are written; otherwise, they are excluded.

## Returns

[int](#)

The number of models successfully written to the database.

## Remarks

If the collection contains more models than the specified batch size, the write operation is performed in multiple batches. The behavior of column inclusion is determined by the corresponding boolean parameters. This method does not guarantee transactional integrity across batches.

# Class RelmTrigger<T>

Namespace: [CoreRelm.Models](#)

Assembly: CoreRelm.dll

Represents a database trigger associated with a specific table and operation type.

```
public class RelmTrigger<T>
```

## Type Parameters

T

The type representing the database table. The type must have a [RelmTable](#) attribute to define the table name.

## Inheritance

[object](#) ← RelmTrigger<T>

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#)

## Remarks

The [RelmTrigger<T>](#) class is used to define and manage database triggers for a table represented by the type T. It provides properties to configure the trigger's behavior, such as the trigger type, body, and associated database details. The table name is derived from the [RelmTable](#) attribute applied to the type.

## Properties

### DatabaseName

Gets or sets the name of the database.

```
public string DatabaseName { get; set; }
```

### Property Value

[string](#)

## DefinerUser

Gets or sets the name of the user who defined the current entity or operation.

```
public string DefinerUser { get; set; }
```

### Property Value

[string](#)

## StatementDelimiter

Gets or sets the delimiter used to separate individual statements in a batch.

```
public string StatementDelimiter { get; set; }
```

### Property Value

[string](#)

### Remarks

This property is typically used in scenarios where multiple statements are processed together, such as in database scripts or command batches. Ensure the delimiter matches the expected syntax of the target system.

## TableName

Gets the name of the database table associated with this instance.

```
public string TableName { get; }
```

### Property Value

[string](#)

## TriggerBody

Gets or sets the body of the trigger event.

```
public string TriggerBody { get; set; }
```

Property Value

[string](#)

## TriggerType

Gets the type of trigger associated with the current operation.

```
public Triggers.TriggerTypes TriggerType { get; }
```

Property Value

[Triggers.TriggerTypes](#)

## Methods

### ToString()

Generates a string representation of the SQL statement required to create a database trigger.

```
public override string ToString()
```

Returns

[string](#)

A string containing the complete SQL script to create the database trigger.

Remarks

The generated SQL includes the necessary statements to drop an existing trigger, set the delimiter, specify the definer, and create the trigger with the provided body and configuration. The output string is formatted to include the database name (if specified), the trigger name, and the trigger body. This method is useful for dynamically generating SQL scripts for database triggers.

# Namespace CoreRelm.Models.EventArgs

## Classes

### [DtoEventArgs](#)

Provides event data that includes additional object properties for data transfer operations.

# Class DtoEventArgs

Namespace: [CoreRelm.Models.EventArgs](#)

Assembly: CoreRelm.dll

Provides event data that includes additional object properties for data transfer operations.

```
public class DtoEventArgs : EventArgs
```

## Inheritance

[object](#) ← [EventArgs](#) ← DtoEventArgs

## Inherited Members

[EventArgs.Empty](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,  
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) ,  
[object.ToString\(\)](#)

## Remarks

Use this class to pass extra key-value pairs with event notifications when working with data transfer objects (DTOs). This enables event handlers to access supplementary information that may not be part of the standard event arguments.

## Properties

### AdditionalObjectProperties

Gets or sets a collection of additional properties associated with the object.

```
public Dictionary<string, object> AdditionalObjectProperties { get; set; }
```

## Property Value

[Dictionary](#) <[string](#), [object](#)>

## Remarks

This dictionary allows for the storage of custom key-value pairs that are not explicitly defined by other properties. It can be used to extend the object with extra data as needed.



# Namespace CoreReIm.Models.Migrations

## Classes

[MigrationGenerateResult](#)

[ModelSetDefinition](#)

[ModelSetValidateReport](#)

[ModelSetValidatedModel](#)

[ModelSetsFile](#)

[ResolvedModelSet](#)

[ResolvedModelSetDiagnostics](#)

[ValidatedModelType](#)

# Class MigrationGenerateResult

Namespace: [CoreRelm.Models.Migrations](#)

Assembly: CoreRelm.dll

```
public sealed record MigrationGenerateResult : IEquatable<MigrationGenerateResult>
```

## Inheritance

[object](#) ← MigrationGenerateResult

## Implements

[IEquatable](#)<[MigrationGenerateResult](#)>

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

## Constructors

MigrationGenerateResult(string, bool, string?, string)

```
public MigrationGenerateResult(string DatabaseName, bool HasChanges, string? Sql,  
string Message)
```

## Parameters

DatabaseName [string](#)

HasChanges [bool](#)

Sql [string](#)

Message [string](#)

## Properties

DatabaseName

```
public string DatabaseName { get; init; }
```

Property Value

[string](#)

## HasChanges

```
public bool HasChanges { get; init; }
```

Property Value

[bool](#)

## Message

```
public string Message { get; init; }
```

Property Value

[string](#)

## Sql

```
public string? Sql { get; init; }
```

Property Value

[string](#)

## Methods

### Changes(string, string, string)

```
public static MigrationGenerateResult Changes(string dbName, string sql, string message)
```

Parameters

dbName [string](#)

sql [string](#)

message [string](#)

Returns

[MigrationGenerateResult](#)

## NoChanges(string, string)

```
public static MigrationGenerateResult NoChanges(string dbName, string message)
```

Parameters

dbName [string](#)

message [string](#)

Returns

[MigrationGenerateResult](#)

# Class ModelSetDefinition

Namespace: [CoreRelm.Models.Migrations](#)

Assembly: CoreRelm.dll

```
public sealed class ModelSetDefinition
```

## Inheritance

[object](#) ← ModelSetDefinition

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

# Properties

## NamespacePrefixes

```
[JsonPropertyName("namespacePrefixes")]
public List<string> NamespacePrefixes { get; set; }
```

## Property Value

[List](#)<[string](#)>

## Types

```
[JsonPropertyName("types")]
public List<string> Types { get; set; }
```

## Property Value

[List](#)<[string](#)>

# Class ModelSetValidateReport

Namespace: [CoreRelm.Models.Migrations](#)

Assembly: CoreRelm.dll

```
public sealed record ModelSetValidateReport : IEquatable<ModelSetValidateReport>
```

## Inheritance

[object](#) ← ModelSetValidateReport

## Implements

[IEquatable](#)<[ModelSetValidateReport](#)>

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

## Constructors

ModelSetValidateReport(string, string, string, string, string?, ResolvedModelSetDiagnostics, IReadOnlyDictionary<string, List<ModelSetValidatedModel>>)

```
public ModelSetValidateReport(string Tool, string Command, string GeneratedUtc, string  
SetName, string? DatabaseFilter, ResolvedModelSetDiagnostics Diagnostics,  
IReadOnlyDictionary<string, List<ModelSetValidatedModel>> ModelsByDatabase)
```

## Parameters

Tool [string](#)

Command [string](#)

GeneratedUtc [string](#)

SetName [string](#)

DatabaseFilter [string](#)

Diagnostics [ResolvedModelSetDiagnostics](#)

ModelsByDatabase [IReadOnlyDictionary](#)<[string](#), [List](#)<[ModelSetValidatedModel](#)>>

## Properties

### Command

```
public string Command { get; init; }
```

#### Property Value

[string](#)

### DatabaseFilter

```
public string? DatabaseFilter { get; init; }
```

#### Property Value

[string](#)

### Diagnostics

```
public ResolvedModelSetDiagnostics Diagnostics { get; init; }
```

#### Property Value

[ResolvedModelSetDiagnostics](#)

### GeneratedUtc

```
public string GeneratedUtc { get; init; }
```

Property Value

[string](#)

## ModelsByDatabase

```
public IReadOnlyDictionary<string, List<ModelSetValidatedModel>> ModelsByDatabase { get;  
    init; }
```

Property Value

[IReadOnlyDictionary](#)<[string](#), [List](#)<[ModelSetValidatedModel](#)>>

## SetName

```
public string SetName { get; init; }
```

Property Value

[string](#)

## Tool

```
public string Tool { get; init; }
```

Property Value

[string](#)

# Class ModelSetValidatedModel

Namespace: [CoreRelm.Models.Migrations](#)

Assembly: CoreRelm.dll

```
public sealed record ModelSetValidatedModel : IEquatable<ModelSetValidatedModel>
```

## Inheritance

[object](#) ← ModelSetValidatedModel

## Implements

[IEquatable](#)<[ModelSetValidatedModel](#)>

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

## Constructors

### ModelSetValidatedModel(string, string, string)

```
public ModelSetValidatedModel(string TypeName, string DatabaseName, string TableName)
```

## Parameters

TypeName [string](#)

DatabaseName [string](#)

TableName [string](#)

## Properties

### DatabaseName

```
public string DatabaseName { get; init; }
```

Property Value

[string](#) ↗

TableName

```
public string TableName { get; init; }
```

Property Value

[string](#) ↗

TypeName

```
public string TypeName { get; init; }
```

Property Value

[string](#) ↗

# Class ModelSetsFile

Namespace: [CoreRelm.Models.Migrations](#)

Assembly: CoreRelm.dll

```
public sealed class ModelSetsFile
```

## Inheritance

[object](#) ← ModelSetsFile

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

# Properties

## Sets

```
[JsonPropertyName("sets")]
public Dictionary<string, ModelSetDefinition> Sets { get; set; }
```

## Property Value

[Dictionary](#)<[string](#), [ModelSetDefinition](#)>

## Version

```
[JsonPropertyName("version")]
public int Version { get; set; }
```

## Property Value

[int](#)

# Class ResolvedModelSet

Namespace: [CoreRelm.Models.Migrations](#)

Assembly: CoreRelm.dll

```
public sealed record ResolvedModelSet : IEquatable<ResolvedModelSet>
```

## Inheritance

[object](#) ← ResolvedModelSet

## Implements

[IEquatable](#)<[ResolvedModelSet](#)>

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

## Constructors

ResolvedModelSet(string, IReadOnlyList<ValidatedModelType>,  
IReadOnlyDictionary<string, List<ValidatedModelType>>)

```
public ResolvedModelSet(string SetName, IReadOnlyList<ValidatedModelType> AllModels,  
IReadOnlyDictionary<string, List<ValidatedModelType>> ModelsByDatabase)
```

## Parameters

SetName [string](#)

AllModels [IReadOnlyList](#)<[ValidatedModelType](#)>

ModelsByDatabase [IReadOnlyDictionary](#)<[string](#), [List](#)<[ValidatedModelType](#)>>

## Properties

### AllModels

```
public IReadOnlyList<ValidatedModelType> AllModels { get; init; }
```

Property Value

[IReadOnlyList](#) <[ValidatedModelType](#)>

## ModelsByDatabase

```
public IReadOnlyDictionary<string, List<ValidatedModelType>> ModelsByDatabase { get; init; }
```

Property Value

[IReadOnlyDictionary](#) <[string](#), [List](#) <[ValidatedModelType](#)>>

## SetName

```
public string SetName { get; init; }
```

Property Value

[string](#)

# Class ResolvedModelSetDiagnostics

Namespace: [CoreRelm.Models.Migrations](#)

Assembly: CoreRelm.dll

```
public sealed record ResolvedModelSetDiagnostics : IEquatable<ResolvedModelSetDiagnostics>
```

## Inheritance

[object](#) ← ResolvedModelSetDiagnostics

## Implements

[IEquatable](#)<[ResolvedModelSetDiagnostics](#)>

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

## Constructors

ResolvedModelSetDiagnostics(string, int, IReadOnlyList<string>)

```
public ResolvedModelSetDiagnostics(string SetName, int AssemblyTypeCount, int  
ExplicitTypeNameCount, int ExplicitTypesResolvedCount, int NamespacePrefixCount, int  
NamespaceMatchedCount, int CandidateCountBeforeFilter, int AbstractExcludedCount, int  
NotRelmModelExcludedCount, int IncludedCount, int MissingRelmDatabaseCount, int  
MissingRelmTableCount, int AttributeValueErrorCode, IReadOnlyList<string> Errors)
```

## Parameters

SetName [string](#)

AssemblyTypeCount [int](#)

ExplicitTypeNameCount [int](#)

ExplicitTypesResolvedCount [int](#)

NamespacePrefixCount [int](#)

NamespaceMatchedCount [int](#)

CandidateCountBeforeFilter [int](#)

AbstractExcludedCount [int](#)

NotRelmModelExcludedCount [int](#)

IncludedCount [int](#)

MissingRelmDatabaseCount [int](#)

MissingRelmTableCount [int](#)

AttributeValueErrorCount [int](#)

Errors  [IReadOnlyList<string>](#)

## Properties

### AbstractExcludedCount

```
public int AbstractExcludedCount { get; init; }
```

Property Value

[int](#)

### AssemblyTypeCount

```
public int AssemblyTypeCount { get; init; }
```

Property Value

[int](#)

### AttributeValueErrorCount

```
public int AttributeValueErrorCount { get; init; }
```

Property Value

[int ↗](#)

## CandidateCountBeforeFilter

```
public int CandidateCountBeforeFilter { get; init; }
```

Property Value

[int ↗](#)

## Errors

```
public IReadOnlyList<string> Errors { get; init; }
```

Property Value

[IReadOnlyList ↗ <string ↗ >](#)

## ExplicitTypeNameCount

```
public int ExplicitTypeNameCount { get; init; }
```

Property Value

[int ↗](#)

## ExplicitTypesResolvedCount

```
public int ExplicitTypesResolvedCount { get; init; }
```

Property Value

[int ↗](#)

## IncludedCount

```
public int IncludedCount { get; init; }
```

Property Value

[int ↗](#)

## MissingRelmDatabaseCount

```
public int MissingRelmDatabaseCount { get; init; }
```

Property Value

[int ↗](#)

## MissingRelmTableCount

```
public int MissingRelmTableCount { get; init; }
```

Property Value

[int ↗](#)

## NamespaceMatchedCount

```
public int NamespaceMatchedCount { get; init; }
```

Property Value

[int ↗](#)

## NamespacePrefixCount

```
public int NamespacePrefixCount { get; init; }
```

Property Value

[int ↗](#)

## NotRelmModelExcludedCount

```
public int NotRelmModelExcludedCount { get; init; }
```

Property Value

[int ↗](#)

## SetName

```
public string SetName { get; init; }
```

Property Value

[string ↗](#)

# Class ValidatedModelType

Namespace: [CoreRelm.Models.Migrations](#)

Assembly: CoreRelm.dll

```
public sealed record ValidatedModelType : IEquatable<ValidatedModelType>
```

## Inheritance

[object](#) ← ValidatedModelType

## Implements

[IEquatable](#)<[ValidatedModelType](#)>

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

## Constructors

### ValidatedModelType(Type, string, string)

```
public ValidatedModelType(Type ClrType, string DatabaseName, string TableName)
```

## Parameters

ClrType [Type](#)

DatabaseName [string](#)

TableName [string](#)

## Properties

### ClrType

```
public Type ClrType { get; init; }
```

Property Value

[Type ↗](#)

## DatabaseName

```
public string DatabaseName { get; init; }
```

Property Value

[string ↗](#)

## TableName

```
public string TableName { get; init; }
```

Property Value

[string ↗](#)

# Namespace CoreRelm.Models.Migrations.Execution

## Classes

[AppliedMigration](#)

# Class AppliedMigration

Namespace: [CoreRelm.Models.Migrations.Execution](#)

Assembly: CoreRelm.dll

```
public class AppliedMigration : RelmModel, IRelmModel
```

## Inheritance

[object](#) ← [RelmModel](#) ← AppliedMigration

## Implements

[IRelmModel](#)

## Inherited Members

[RelmModel.DtoTypeProcessor](#), [RelmModel.Id](#), [RelmModel.Active](#), [RelmModel.InternalId](#),  
[RelmModel.CreateDate](#), [RelmModel.LastUpdated](#), [RelmModel.ResetCoreAttributes\(bool, bool\)](#),  
[RelmModel.ResetWithData\(DataRow, string\)](#), [RelmModel.GetUnderscoreProperties\(bool\)](#),  
[RelmModel.WriteToDatabase\(Enum, int, bool, bool, bool, bool\)](#),  
[RelmModel.WriteToDatabase\(MySqlConnection, MySqlTransaction, int, bool, bool, bool, bool\)](#),  
[RelmModel.WriteToDatabase\(IRelmContext, int, bool, bool, bool, bool\)](#),  
[RelmModel.WriteToDatabase\(IRelmQuickContext, int, bool, bool, bool, bool\)](#),  
[RelmModel.WriteToDatabase\(RelmContextOptionsBuilder, int, bool, bool, bool, bool\)](#),  
[RelmModel.CopyFromSource<T>\(T\)](#),  
[RelmModel.GenerateDTO\(IEnumerable<string>, IEnumerable<string>, string, Func<IRelmModel, Dictionary<string, object>>, int\)](#),  
[RelmModel.Duplicate\(\)](#), [object.Equals\(object\)](#), [object.Equals\(object, object\)](#),  
[object.GetHashCode\(\)](#), [object.GetType\(\)](#), [object.MemberwiseClone\(\)](#),  
[object.ReferenceEquals\(object, object\)](#), [object.ToString\(\)](#)

## Extension Methods

[ModelExtensions.LoadDataLoaderField<T, S>\(T, IRelmContext, Expression<Func<T, S>>\)](#),  
[ModelExtensions.LoadDataLoaderField<T, S>\(T, IRelmQuickContext, Expression<Func<T, S>>\)](#),  
[ModelExtensions.LoadDataLoaderField<T, S>\(T, RelmContextOptionsBuilder, Expression<Func<T, S>>\)](#),  
[ModelExtensions.LoadForeignKeyField<T, R>\(T, IRelmContext, Expression<Func<T, ICollection<R>>>\)](#),  
[ModelExtensions.LoadForeignKeyField<T, R>\(T, IRelmContext, Expression<Func<T, ICollection<R>>>, Expression<Func<R, object>>\)](#),  
[ModelExtensions.LoadForeignKeyField<T, R>\(T, IRelmContext, Expression<Func<T, R>>\)](#),  
[ModelExtensions.LoadForeignKeyField<T, R>\(T, IRelmContext, Expression<Func<T, R>>, Expression<Func<R, object>>\)](#),



# Constructors

## AppliedMigration()

```
public AppliedMigration()
```

## AppliedMigration(string, string, DateTime)

```
public AppliedMigration(string fileName, string checksumSha256, DateTime appliedUtc)
```

### Parameters

fileName [string](#)

checksumSha256 [string](#)

appliedUtc [DateTime](#)

# Properties

## AppliedUtc

```
public DateTime AppliedUtc { get; init; }
```

### Property Value

[DateTime](#)

## ChecksumSha256

```
public string ChecksumSha256 { get; init; }
```

### Property Value

[string](#)

## FileName

```
public string FileName { get; init; }
```

Property Value

[string](#) ↗

# Namespace CoreRelm.Models.Migrations.Introspection

## Classes

[ColumnSchema](#)

[ForeignKeySchema](#)

[FunctionSchema](#)

[IndexColumnSchema](#)

[IndexSchema](#)

[SchemaSnapshot](#)

[TableSchema](#)

[TriggerSchema](#)

# Class ColumnSchema

Namespace: [CoreRelm.Models.Migrations.Introspection](#)

Assembly: CoreRelm.dll

```
public sealed record ColumnSchema : IEquatable<ColumnSchema>
```

## Inheritance

[object](#) ← ColumnSchema

## Implements

[IEquatable](#)<[ColumnSchema](#)>

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

## Constructors

ColumnSchema(string, string, bool, bool, bool, bool, bool,  
string?, string?, bool, string?, int)

```
public ColumnSchema(string ColumnName, string ColumnType, bool IsNullable, bool  
IsPrimaryKey, bool IsForeignKey, bool IsReadOnly, bool IsUnique, string? DefaultValue,  
string? DefaultValueSql, bool IsAutoIncrement, string? Extra, int OrdinalPosition)
```

## Parameters

ColumnName [string](#)

ColumnType [string](#)

IsNullable [bool](#)

IsPrimaryKey [bool](#)

IsForeignKey [bool](#)

IsReadOnly [bool](#)

IsUnique [bool](#)

DefaultValue [string](#)

DefaultValueSql [string](#)

IsAutoIncrement [bool](#)

Extra [string](#)

OrdinalPosition [int](#)

## Properties

### ColumnName

```
public string ColumnName { get; init; }
```

### Property Value

[string](#)

### ColumnType

```
public string ColumnType { get; init; }
```

### PropertyValue

[string](#)

### DefaultValue

```
public string? DefaultValue { get; init; }
```

### PropertyValue

[string](#)

## DefaultValueSql

```
public string? DefaultValueSql { get; init; }
```

Property Value

[string](#) ↗

## Extra

```
public string? Extra { get; init; }
```

Property Value

[string](#) ↗

## IsAutoIncrement

```
public bool IsAutoIncrement { get; init; }
```

Property Value

[bool](#) ↗

## IsForeignKey

```
public bool IsForeignKey { get; init; }
```

Property Value

[bool](#) ↗

## IsNullable

```
public bool IsNullable { get; init; }
```

Property Value

[bool](#) ↗

## IsPrimaryKey

```
public bool IsPrimaryKey { get; init; }
```

Property Value

[bool](#) ↗

## IsReadOnly

```
public bool IsReadOnly { get; init; }
```

Property Value

[bool](#) ↗

## IsUnicode

```
public bool IsUnicode { get; init; }
```

Property Value

[bool](#) ↗

## OrdinalPosition

```
public int OrdinalPosition { get; init; }
```

Property Value

[int ↗](#)

# Class ForeignKeySchema

Namespace: [CoreRelm.Models.Migrations.Introspection](#)

Assembly: CoreRelm.dll

```
public sealed record ForeignKeySchema : IEquatable<ForeignKeySchema>
```

## Inheritance

[object](#) ← ForeignKeySchema

## Implements

[IEquatable](#)<[ForeignKeySchema](#)>

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

## Constructors

ForeignKeySchema(string, string, IReadOnlyList<string>, string,  
IReadOnlyList<string>, string, string)

```
public ForeignKeySchema(string ConstraintName, string TableName, IReadOnlyList<string>  
ColumnNames, string ReferencedTableName, IReadOnlyList<string> ReferencedColumnNames, string  
UpdateRule, string DeleteRule)
```

## Parameters

ConstraintName [string](#)

TableName [string](#)

ColumnNames [IReadOnlyList](#)<[string](#)>

ReferencedTableName [string](#)

ReferencedColumnNames [IReadOnlyList](#)<[string](#)>

UpdateRule [string](#)

DeleteRule [string](#)

## Properties

### ColumnNames

```
public IReadOnlyList<string> ColumnNames { get; init; }
```

#### Property Value

[IReadOnlyList](#)<[string](#)>

### ConstraintName

```
public string ConstraintName { get; init; }
```

#### Property Value

[string](#)

### DeleteRule

```
public string DeleteRule { get; init; }
```

#### Property Value

[string](#)

### ReferencedColumnNames

```
public IReadOnlyList<string> ReferencedColumnNames { get; init; }
```

#### Property Value

[IReadOnlyList](#) <[string](#)>

## ReferencedTableName

```
public string ReferencedTableName { get; init; }
```

Property Value

[string](#)

## TableName

```
public string TableName { get; init; }
```

Property Value

[string](#)

## UpdateRule

```
public string UpdateRule { get; init; }
```

Property Value

[string](#)

# Class FunctionSchema

Namespace: [CoreRelm.Models.Migrations.Introspection](#)

Assembly: CoreRelm.dll

```
public sealed record FunctionSchema : IEquatable<FunctionSchema>
```

Inheritance

[object](#) ← FunctionSchema

Implements

[IEquatable](#)<[FunctionSchema](#)>

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

## Constructors

### FunctionSchema(string, string, string)

```
public FunctionSchema(string FunctionName, string ReturnType, string RoutineDefinition)
```

Parameters

FunctionName [string](#)

ReturnType [string](#)

RoutineDefinition [string](#)

## Properties

### FunctionName

```
public string FunctionName { get; init; }
```

Property Value

[string ↗](#)

ReturnType

```
public string ReturnType { get; init; }
```

Property Value

[string ↗](#)

RoutineDefinition

```
public string RoutineDefinition { get; init; }
```

Property Value

[string ↗](#)

# Class IndexColumnSchema

Namespace: [CoreRelm.Models.Migrations.Introspection](#)

Assembly: CoreRelm.dll

```
public sealed record IndexColumnSchema : IEquatable<IndexColumnSchema>
```

## Inheritance

[object](#) ← IndexColumnSchema

## Implements

[IEquatable](#)<[IndexColumnSchema](#)>

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

## Constructors

### IndexColumnSchema(string, int, string?)

```
public IndexColumnSchema(string ColumnName, int SeqInIndex, string? Collation)
```

## Parameters

ColumnName [string](#)

SeqInIndex [int](#)

Collation [string](#)

## Properties

### Collation

```
public string? Collation { get; init; }
```

Property Value

[string ↗](#)

ColumnName

```
public string ColumnName { get; init; }
```

Property Value

[string ↗](#)

SeqInIndex

```
public int SeqInIndex { get; init; }
```

Property Value

[int ↗](#)

# Class IndexSchema

Namespace: [CoreRelm.Models.Migrations.Introspection](#)

Assembly: CoreRelm.dll

```
public sealed record IndexSchema : IEquatable<IndexSchema>
```

## Inheritance

[object](#) ← IndexSchema

## Implements

[IEquatable](#)<[IndexSchema](#)>

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

## Constructors

IndexSchema(string, bool,  
 IReadOnlyList<IndexColumnSchema>)

```
public IndexSchema(string IndexName, bool IsUnique, IReadOnlyList<IndexColumnSchema>  
Columns)
```

## Parameters

IndexName [string](#)

IsUnique [bool](#)

Columns [IReadOnlyList](#)<[IndexColumnSchema](#)>

## Properties

### Columns

```
public IReadOnlyList<IndexColumnSchema> Columns { get; init; }
```

Property Value

[IReadOnlyList](#) <[IndexColumnSchema](#)>

IndexName

```
public string IndexName { get; init; }
```

Property Value

[string](#)

IsUnique

```
public bool IsUnique { get; init; }
```

Property Value

[bool](#)

# Class SchemaSnapshot

Namespace: [CoreRelm.Models.Migrations.Introspection](#)

Assembly: CoreRelm.dll

```
public sealed record SchemaSnapshot : IEquatable<SchemaSnapshot>
```

## Inheritance

[object](#) ← SchemaSnapshot

## Implements

[IEquatable](#)<[SchemaSnapshot](#)>

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

## Constructors

SchemaSnapshot(string, IReadOnlyDictionary<string, TableSchema>, IReadOnlyDictionary<string, FunctionSchema>)

```
public SchemaSnapshot(string DatabaseName, IReadOnlyDictionary<string, TableSchema> Tables,  
IReadOnlyDictionary<string, FunctionSchema> Functions)
```

## Parameters

DatabaseName [string](#)

Tables [IReadOnlyDictionary](#)<[string](#), [TableSchema](#)>

Functions [IReadOnlyDictionary](#)<[string](#), [FunctionSchema](#)>

## Properties

DatabaseName

```
public string DatabaseName { get; init; }
```

Property Value

[string](#)

## Functions

```
public IReadOnlyDictionary<string, FunctionSchema> Functions { get; init; }
```

Property Value

[IReadOnlyDictionary](#)<[string](#), [FunctionSchema](#)>

## Tables

```
public IReadOnlyDictionary<string, TableSchema> Tables { get; init; }
```

Property Value

[IReadOnlyDictionary](#)<[string](#), [TableSchema](#)>

# Class TableSchema

Namespace: [CoreRelm.Models.Migrations.Introspection](#)

Assembly: CoreRelm.dll

```
public sealed record TableSchema : IEquatable<TableSchema>
```

Inheritance

[object](#) ← TableSchema

Implements

[IEquatable](#)<[TableSchema](#)>

Inherited Members

[object.Equals\(object\)](#), [object.Equals\(object, object\)](#), [object.GetHashCode\(\)](#), [object.GetType\(\)](#),  
[object.ReferenceEquals\(object, object\)](#), [object.ToString\(\)](#)

## Constructors

TableSchema(string, IReadOnlyDictionary<string, ColumnSchema>, IReadOnlyDictionary<string, IndexSchema>, IReadOnlyDictionary<string, ForeignKeySchema>, IReadOnlyDictionary<string, TriggerSchema>)

```
public TableSchema(string TableName, IReadOnlyDictionary<string, ColumnSchema> Columns,  
IReadOnlyDictionary<string, IndexSchema> Indexes, IReadOnlyDictionary<string,  
ForeignKeySchema> ForeignKeys, IReadOnlyDictionary<string, TriggerSchema> Triggers)
```

Parameters

TableName [string](#)

Columns [IReadOnlyDictionary](#)<[string](#), [ColumnSchema](#)>

Indexes [IReadOnlyDictionary](#)<[string](#), [IndexSchema](#)>

ForeignKeys [IReadOnlyDictionary](#)<[string](#), [ForeignKeySchema](#)>

Triggers [IReadOnlyDictionary](#)<[string](#), [TriggerSchema](#)>

## Properties

### Columns

```
public IReadOnlyDictionary<string, ColumnSchema> Columns { get; init; }
```

### Property Value

[IReadOnlyDictionary](#)<[string](#), [ColumnSchema](#)>

### ForeignKeys

```
public IReadOnlyDictionary<string, ForeignKeySchema> ForeignKeys { get; init; }
```

### Property Value

[IReadOnlyDictionary](#)<[string](#), [ForeignKeySchema](#)>

### Indexes

```
public IReadOnlyDictionary<string, IndexSchema> Indexes { get; init; }
```

### Property Value

[IReadOnlyDictionary](#)<[string](#), [IndexSchema](#)>

### TableName

```
public string TableName { get; init; }
```

### Property Value

[string](#)

## Triggers

```
public IReadOnlyDictionary<string, TriggerSchema> Triggers { get; init; }
```

Property Value

[IReadOnlyDictionary](#)<[string](#), [TriggerSchema](#)>

# Class TriggerSchema

Namespace: [CoreRelm.Models.Migrations.Introspection](#)

Assembly: CoreRelm.dll

```
public sealed record TriggerSchema : IEquatable<TriggerSchema>
```

Inheritance

[object](#) ← TriggerSchema

Implements

[IEquatable](#)<[TriggerSchema](#)>

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

## Constructors

TriggerSchema(string, string, string, string)

```
public TriggerSchema(string TriggerName, string EventManipulation, string ActionTiming,  
string ActionStatement)
```

Parameters

TriggerName [string](#)

EventManipulation [string](#)

ActionTiming [string](#)

ActionStatement [string](#)

## Properties

ActionStatement

```
public string ActionStatement { get; init; }
```

Property Value

[string](#) ↗

## ActionTiming

```
public string ActionTiming { get; init; }
```

Property Value

[string](#) ↗

## EventManipulation

```
public string EventManipulation { get; init; }
```

Property Value

[string](#) ↗

## TriggerName

```
public string TriggerName { get; init; }
```

Property Value

[string](#) ↗

# Namespace CoreRelm.Models.Migrations. MigrationPlans

## Classes

[MigrationPlan](#)

[MigrationPlanOptions](#)

# Class MigrationPlan

Namespace: [CoreRelm.Models.Migrations.MigrationPlans](#)

Assembly: CoreRelm.dll

```
public sealed record MigrationPlan : IEquatable<MigrationPlan>
```

## Inheritance

[object](#) ← MigrationPlan

## Implements

[IEquatable](#)<[MigrationPlan](#)>

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

## Constructors

MigrationPlan(string, IReadOnlyList<IMigrationOperation>,  
IReadOnlyList<string>, IReadOnlyList<string>)

```
public MigrationPlan(string DatabaseName, IReadOnlyList<IMigrationOperation> Operations,  
IReadOnlyList<string> Warnings, IReadOnlyList<string> Blockers)
```

## Parameters

DatabaseName [string](#)

Operations [IReadOnlyList](#)<[IMigrationOperation](#)>

Warnings [IReadOnlyList](#)<[string](#)>

Blockers [IReadOnlyList](#)<[string](#)>

## Properties

## Blockers

```
public IReadOnlyList<string> Blockers { get; init; }
```

### Property Value

[IReadOnlyList](#)<[string](#)>

## DatabaseName

```
public string DatabaseName { get; init; }
```

### Property Value

[string](#)

## HasChanges

```
public bool HasChanges { get; }
```

### Property Value

[bool](#)

## Operations

```
public IReadOnlyList<IMigrationOperation> Operations { get; init; }
```

### Property Value

[IReadOnlyList](#)<[IMigrationOperation](#)>

## Warnings

```
public IReadOnlyList<string> Warnings { get; init; }
```

## Property Value

[IReadOnlyList](#) <[string](#)>

# Class MigrationPlanOptions

Namespace: [CoreRelm.Models.Migrations.MigrationPlans](#)

Assembly: CoreRelm.dll

```
public sealed record MigrationPlanOptions : IEquatable<MigrationPlanOptions>
```

## Inheritance

[object](#) ← MigrationPlanOptions

## Implements

[IEquatable](#)<[MigrationPlanOptions](#)>

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

## Constructors

MigrationPlanOptions(bool, ISet<string>)

```
public MigrationPlanOptions(bool Destructive, ISet<string> ScopeTables)
```

## Parameters

Destructive [bool](#)

ScopeTables [ISet](#)<[string](#)>

## Properties

Destructive

```
public bool Destructive { get; init; }
```

## Property Value

[bool](#)

## ScopeTables

`public ISet<string> ScopeTables { get; init; }`

Property Value

[ISet](#)<[string](#)>

# Namespace CoreRelm.Models.Migrations.Rendering

## Classes

[AddColumnOperation](#)

[AddForeignKeyOperation](#)

[AlterColumnOperation](#)

[CreateFunctionOperation](#)

[CreateIndexOperation](#)

[CreateTableOperation](#)

[CreateTriggerOperation](#)

[DropForeignKeyOperation](#)

[DropIndexOperation](#)

[DropTriggerOperation](#)

[MySqlRenderOptions](#)

# Class AddColumnOperation

Namespace: [CoreRelm.Models.Migrations.Rendering](#)

Assembly: CoreRelm.dll

```
public sealed record AddColumnOperation : IMigrationOperation,  
IEquatable<AddColumnOperation>
```

## Inheritance

[object](#) ← AddColumnOperation

## Implements

[IMigrationOperation](#), [IEquatable](#)<[AddColumnOperation](#)>

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

# Constructors

## AddColumnOperation(string, ColumnSchema)

```
public AddColumnOperation(string TableName, ColumnSchema Column)
```

## Parameters

TableName [string](#)

Column [ColumnSchema](#)

# Properties

## Column

```
public ColumnSchema Column { get; init; }
```

Property Value

[ColumnSchema](#)

Description

```
public string Description { get; }
```

Property Value

[string](#)

TableName

```
public string TableName { get; init; }
```

Property Value

[string](#)

# Class AddForeignKeyOperation

Namespace: [CoreRelm.Models.Migrations.Rendering](#)

Assembly: CoreRelm.dll

```
public sealed record AddForeignKeyOperation : IMigrationOperation,  
IEquatable<AddForeignKeyOperation>
```

## Inheritance

[object](#) ← AddForeignKeyOperation

## Implements

[IMigrationOperation](#), [IEquatable](#)<[AddForeignKeyOperation](#)>

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

## Constructors

AddForeignKeyOperation(string, ForeignKeySchema)

```
public AddForeignKeyOperation(string TableName, ForeignKeySchema ForeignKey)
```

## Parameters

TableName [string](#)

ForeignKey [ForeignKeySchema](#)

## Properties

### Description

```
public string Description { get; }
```

Property Value

[string](#) ↗

## ForeignKey

```
public ForeignKeySchema ForeignKey { get; init; }
```

Property Value

[ForeignKeySchema](#)

## TableName

```
public string TableName { get; init; }
```

Property Value

[string](#) ↗

# Class AlterColumnOperation

Namespace: [CoreRelm.Models.Migrations.Rendering](#)

Assembly: CoreRelm.dll

```
public sealed record AlterColumnOperation : IMigrationOperation,  
IEquatable<AlterColumnOperation>
```

## Inheritance

[object](#) ← AlterColumnOperation

## Implements

[IMigrationOperation](#), [IEquatable](#)<[AlterColumnOperation](#)>

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

## Constructors

AlterColumnOperation(string, ColumnSchema, string)

```
public AlterColumnOperation(string TableName, ColumnSchema Desired, string Reason)
```

## Parameters

TableName [string](#)

Desired [ColumnSchema](#)

Reason [string](#)

## Properties

### Description

```
public string Description { get; }
```

Property Value

[string](#) ↗

Desired

```
public ColumnSchema Desired { get; init; }
```

Property Value

[ColumnSchema](#)

Reason

```
public string Reason { get; init; }
```

Property Value

[string](#) ↗

TableName

```
public string TableName { get; init; }
```

Property Value

[string](#) ↗

# Class CreateFunctionOperation

Namespace: [CoreRelm.Models.Migrations.Rendering](#)

Assembly: CoreRelm.dll

```
public sealed record CreateFunctionOperation : IMigrationOperation,  
IEquatable<CreateFunctionOperation>
```

## Inheritance

[object](#) ← CreateFunctionOperation

## Implements

[IMigrationOperation](#), [IEquatable](#)<[CreateFunctionOperation](#)>

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

# Constructors

## CreateFunctionOperation(string, string)

```
public CreateFunctionOperation(string FunctionName, string CreateSql)
```

## Parameters

FunctionName [string](#)

CreateSql [string](#)

# Properties

## CreateSql

```
public string CreateSql { get; init; }
```

Property Value

[string](#) ↗

Description

```
public string Description { get; }
```

Property Value

[string](#) ↗

FunctionName

```
public string FunctionName { get; init; }
```

Property Value

[string](#) ↗

# Class CreateIndexOperation

Namespace: [CoreRelm.Models.Migrations.Rendering](#)

Assembly: CoreRelm.dll

```
public sealed record CreateIndexOperation : IMigrationOperation,  
IEquatable<CreateIndexOperation>
```

## Inheritance

[object](#) ← CreateIndexOperation

## Implements

[IMigrationOperation](#), [IEquatable](#)<[CreateIndexOperation](#)>

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

## Constructors

CreateIndexOperation(string, IndexSchema)

```
public CreateIndexOperation(string TableName, IndexSchema Index)
```

## Parameters

TableName [string](#)

Index [IndexSchema](#)

## Properties

### Description

```
public string Description { get; }
```

Property Value

[string](#) ↗

Index

```
public IndexSchema Index { get; init; }
```

Property Value

[IndexSchema](#)

TableName

```
public string TableName { get; init; }
```

Property Value

[string](#) ↗

# Class CreateTableOperation

Namespace: [CoreRelm.Models.Migrations.Rendering](#)

Assembly: CoreRelm.dll

```
public sealed record CreateTableOperation : IMigrationOperation,  
IEquatable<CreateTableOperation>
```

## Inheritance

[object](#) ← CreateTableOperation

## Implements

[IMigrationOperation](#), [IEquatable](#)<[CreateTableOperation](#)>

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

# Constructors

## CreateTableOperation(TableSchema)

```
public CreateTableOperation(TableSchema Table)
```

## Parameters

Table [TableSchema](#)

# Properties

## Description

```
public string Description { get; }
```

## Property Value

[string](#) ↗

## Table

```
public TableSchema Table { get; init; }
```

Property Value

[TableSchema](#)

# Class CreateTriggerOperation

Namespace: [CoreRelm.Models.Migrations.Rendering](#)

Assembly: CoreRelm.dll

```
public sealed record CreateTriggerOperation : IMigrationOperation,  
IEquatable<CreateTriggerOperation>
```

## Inheritance

[object](#) ← CreateTriggerOperation

## Implements

[IMigrationOperation](#), [IEquatable](#)<[CreateTriggerOperation](#)>

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

## Constructors

CreateTriggerOperation(string, TriggerSchema)

```
public CreateTriggerOperation(string TableName, TriggerSchema Trigger)
```

## Parameters

TableName [string](#)

Trigger [TriggerSchema](#)

## Properties

### Description

```
public string Description { get; }
```

Property Value

[string](#) ↗

TableName

```
public string TableName { get; init; }
```

Property Value

[string](#) ↗

Trigger

```
public TriggerSchema Trigger { get; init; }
```

Property Value

[TriggerSchema](#)

# Class DropForeignKeyOperation

Namespace: [CoreRelm.Models.Migrations.Rendering](#)

Assembly: CoreRelm.dll

```
public sealed record DropForeignKeyOperation : IMigrationOperation,  
IEquatable<DropForeignKeyOperation>
```

## Inheritance

[object](#) ← DropForeignKeyOperation

## Implements

[IMigrationOperation](#), [IEquatable](#)<[DropForeignKeyOperation](#)>

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

# Constructors

## DropForeignKeyOperation(string, string)

```
public DropForeignKeyOperation(string TableName, string ConstraintName)
```

## Parameters

TableName [string](#)

ConstraintName [string](#)

# Properties

## ConstraintName

```
public string ConstraintName { get; init; }
```

Property Value

[string ↗](#)

Description

```
public string Description { get; }
```

Property Value

[string ↗](#)

TableName

```
public string TableName { get; init; }
```

Property Value

[string ↗](#)

# Class DropIndexOperation

Namespace: [CoreRelm.Models.Migrations.Rendering](#)

Assembly: CoreRelm.dll

```
public sealed record DropIndexOperation : IMigrationOperation,  
IEquatable<DropIndexOperation>
```

## Inheritance

[object](#) ← DropIndexOperation

## Implements

[IMigrationOperation](#), [IEquatable](#)<[DropIndexOperation](#)>

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

# Constructors

## DropIndexOperation(string, string)

```
public DropIndexOperation(string TableName, string IndexName)
```

## Parameters

TableName [string](#)

IndexName [string](#)

# Properties

## Description

```
public string Description { get; }
```

Property Value

[string](#) ↗

IndexName

```
public string IndexName { get; init; }
```

Property Value

[string](#) ↗

TableName

```
public string TableName { get; init; }
```

Property Value

[string](#) ↗

# Class DropTriggerOperation

Namespace: [CoreRelm.Models.Migrations.Rendering](#)

Assembly: CoreRelm.dll

```
public sealed record DropTriggerOperation : IMigrationOperation,  
IEquatable<DropTriggerOperation>
```

## Inheritance

[object](#) ← DropTriggerOperation

## Implements

[IMigrationOperation](#), [IEquatable](#)<[DropTriggerOperation](#)>

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

# Constructors

## DropTriggerOperation(string, string)

```
public DropTriggerOperation(string TableName, string TriggerName)
```

## Parameters

TableName [string](#)

TriggerName [string](#)

# Properties

## Description

```
public string Description { get; }
```

Property Value

[string](#) ↗

TableName

```
public string TableName { get; init; }
```

Property Value

[string](#) ↗

TriggerName

```
public string TriggerName { get; init; }
```

Property Value

[string](#) ↗

# Class MySqlRenderOptions

Namespace: [CoreRelm.Models.Migrations.Rendering](#)

Assembly: CoreRelm.dll

```
public sealed record MySqlRenderOptions : IEquatable<MySqlRenderOptions>
```

## Inheritance

[object](#) ← MySqlRenderOptions

## Implements

[IEquatable](#)<[MySqlRenderOptions](#)>

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

## Constructors

### MySqlRenderOptions(bool, bool, string)

```
public MySqlRenderOptions(bool IncludeUseDatabase = true, bool WrapTriggersWithDelimiter =  
true, string TriggerDelimiter = "##")
```

## Parameters

IncludeUseDatabase [bool](#)

WrapTriggersWithDelimiter [bool](#)

TriggerDelimiter [string](#)

## Properties

### IncludeUseDatabase

```
public bool IncludeUseDatabase { get; init; }
```

Property Value

[bool](#)

## TriggerDelimiter

```
public string TriggerDelimiter { get; init; }
```

Property Value

[string](#)

## WrapTriggersWithDelimiter

```
public bool WrapTriggersWithDelimiter { get; init; }
```

Property Value

[bool](#)

# Namespace CoreRelm.Options

## Classes

### [RelmContextOptionsBuilder](#)

Provides a builder for configuring options required to establish a connection to a relational database context. Supports multiple initialization patterns, including connection strings, named connections, and open MySQL connections.

## Enums

### [RelmContextOptionsBuilder.OptionsBuilderTypes](#)

Specifies the types of options that can be used to configure a database context using an options builder.

# Class RelmContextOptionsBuilder

Namespace: [CoreRelm.Options](#)

Assembly: CoreRelm.dll

Provides a builder for configuring options required to establish a connection to a relational database context. Supports multiple initialization patterns, including connection strings, named connections, and open MySQL connections.

```
public class RelmContextOptionsBuilder
```

## Inheritance

[object](#) ← RelmContextOptionsBuilder

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

## Remarks

Use this class to specify database connection details such as server, database name, user credentials, or to provide an existing MySqlConnection or named connection. The builder validates configuration based on the selected connection method. Once configured, the options can be used to initialize a database context. This class is not thread-safe.

## Constructors

### RelmContextOptionsBuilder()

Initializes a new instance of the RelmContextOptionsBuilder class.

```
public RelmContextOptionsBuilder()
```

### RelmContextOptionsBuilder(MySqlConnection)

Initializes a new instance of the RelmContextOptionsBuilder class using the specified MySQL database connection.

```
public RelmContextOptionsBuilder(MySqlConnection connection)
```

## Parameters

**connection** MySqlConnection

The MySqlConnection to use for configuring the context options. Cannot be null.

## RelmContextOptionsBuilder(MySqlConnection, MySqlTransaction)

Initializes a new instance of the RelmContextOptionsBuilder class using the specified MySqlConnection and MySqlTransaction.

```
public RelmContextOptionsBuilder(MySqlConnection connection, MySqlTransaction transaction)
```

## Parameters

**connection** MySqlConnection

The MySqlConnection to be used for database operations. Cannot be null.

**transaction** MySqlTransaction

The MySqlTransaction to associate with the context. Cannot be null.

## Remarks

Use this constructor to configure the context to operate within an existing MySQL connection and transaction. This is useful when managing connection and transaction lifetimes externally.

## RelmContextOptionsBuilder(Enum)

Initializes a new instance of the RelmContextOptionsBuilder class using the specified connection string type.

```
public RelmContextOptionsBuilder(Enum connectionStringType)
```

## Parameters

### `connectionStringType` [Enum ↗](#)

An enumeration value that specifies the type of connection string to use. Must be a valid enum representing a supported connection string type.

## RelmContextOptionsBuilder(string)

Initializes a new instance of the RelmContextOptionsBuilder class using the specified connection string.

```
public RelmContextOptionsBuilder(string connectionString)
```

## Parameters

### `connectionString` [string ↗](#)

The connection string used to configure the context options. Cannot be null or empty.

## Exceptions

### [ArgumentNullException ↗](#)

Thrown if the `connectionString` parameter is null or empty.

## RelmContextOptionsBuilder(string, string, string, string)

Initializes a new instance of the RelmContextOptionsBuilder class with the specified database connection settings.

```
public RelmContextOptionsBuilder(string databaseServer, string databaseName, string
databaseUser, string databasePassword)
```

## Parameters

### `databaseServer` [string ↗](#)

The name or network address of the database server to connect to. Cannot be null or empty.

### `databaseName` [string ↗](#)

The name of the database to use. Cannot be null or empty.

#### databaseUser [string](#)

The username to use when connecting to the database. Cannot be null or empty.

#### databasePassword [string](#)

The password associated with the specified database user. Cannot be null or empty.

## Properties

### AllowUserVariables

Gets a value indicating whether user-defined variables are permitted in SQL statements.

```
public bool AllowUserVariables { get; }
```

#### Property Value

[bool](#)

### AutoInitializeDataSets

Gets a value indicating whether data sets are automatically initialized when the component is created.

```
public bool AutoInitializeDataSets { get; }
```

#### Property Value

[bool](#)

### AutoOpenConnection

Gets a value indicating whether the connection should be automatically opened when required.

```
public bool AutoOpenConnection { get; }
```

Property Value

[bool](#) ↗

## AutoOpenTransaction

Gets a value indicating whether a transaction is automatically opened when a database connection is established.

```
public bool AutoOpenTransaction { get; }
```

Property Value

[bool](#) ↗

## AutoVerifyTables

Gets a value indicating whether table verification is performed automatically before database operations.

```
public bool AutoVerifyTables { get; }
```

Property Value

[bool](#) ↗

## ConnectionStringType

Gets the type of the connection string used by the data source.

```
public Enum ConnectionStringType { get; }
```

Property Value

[Enum](#) ↗

## ConvertZeroDateTime

Gets a value indicating whether date and time values of zero are converted to `DateTime.MinValue` when retrieved from the database.

```
public bool ConvertZeroDateTime { get; }
```

Property Value

[bool](#)

Remarks

When enabled, date and time fields with a value of '0000-00-00' or '0000-00-00 00:00:00' are returned as `DateTime.MinValue` instead of causing an exception or being treated as invalid. This option is useful when working with databases that allow zero date values.

## DatabaseConnection

Gets the active MySQL database connection used by the application.

```
public MySqlConnection DatabaseConnection { get; }
```

Property Value

[MySqlConnection](#)

## DatabaseConnectionString

Gets the connection string used to connect to the database.

```
public string DatabaseConnectionString { get; }
```

Property Value

[string](#)

## DatabaseName

Gets the name of the database associated with this instance.

```
public string DatabaseName { get; }
```

### Property Value

[string](#)

## DatabasePassword

Gets the password used to connect to the database.

```
public string DatabasePassword { get; }
```

### Property Value

[string](#)

## DatabaseServer

Gets the name or network address of the database server to which the application is connected.

```
public string DatabaseServer { get; }
```

### Property Value

[string](#)

## DatabaseTransaction

Gets the current database transaction associated with the connection.

```
public MySqlTransaction DatabaseTransaction { get; }
```

## Property Value

MySqlTransaction

## Remarks

Use this property to access the active MySQL transaction for executing commands within a transactional context. The property is null if no transaction is in progress.

## DatabaseUser

Gets the user name used to connect to the database.

```
public string DatabaseUser { get; }
```

## Property Value

[string](#)

## LockWaitTimeoutSeconds

Gets the maximum number of seconds to wait for a lock to be acquired before timing out.

```
public int LockWaitTimeoutSeconds { get; }
```

## Property Value

[int](#)

## NamedConnection

Gets or sets the name of the connection to use for database operations.

```
public string NamedConnection { get; set; }
```

## Property Value

[string](#)

## OptionsBuilderType

Gets the type of options builder used to configure options for this instance.

```
public RelmContextOptionsBuilder.OptionsBuilderTypes OptionsBuilderType { get; }
```

### Property Value

[RelmContextOptionsBuilder.OptionsBuilderTypes](#)

## Methods

### ParseConnectionDetails(string)

Parses the specified connection details string and configures the database connection settings accordingly.

```
public void ParseConnectionDetails(string connectionDetails)
```

### Parameters

`connectionDetails` [string](#)

A string containing the connection details. Must be in the format 'name=connectionString' to use a named connection, or 'server=serverName;database=databaseName;user=userName;password=password' to specify individual connection parameters.

### Remarks

The method supports two formats for specifying connection details: a named connection (e.g., 'name=MyConnection') or explicit connection parameters (e.g., 'server=localhost;database=MyDb;user=admin;password=secret'). Only one format may be used at a time. Additional or missing parameters will result in an exception.

### Exceptions

## [ArgumentNullException](#)

Thrown if the connectionDetails parameter is null, empty, or consists only of white-space characters.

## [ArgumentException](#)

Thrown if the connectionDetails parameter does not match the required format or is missing required connection parameters.

# SetAllowUserVariables(bool)

Enables or disables the use of user-defined variables in SQL statements.

```
public RelmContextOptionsBuilder SetAllowUserVariables(bool allowUserVariables)
```

Parameters

### [allowUserVariables bool](#)

true to allow user-defined variables in SQL statements; otherwise, false.

Returns

### [RelmContextOptionsBuilder](#)

The current instance of the [RelmContextOptionsBuilder](#) class.

# SetAutoInitializeDataSets(bool)

Enables or disables automatic initialization of data sets.

```
public RelmContextOptionsBuilder SetAutoInitializeDataSets(bool autoInitializeDataSets)
```

Parameters

### [autoInitializeDataSets bool](#)

true to enable automatic initialization of data sets; otherwise, false.

Returns

## [RelmContextOptionsBuilder](#)

The current instance of the [RelmContextOptionsBuilder](#) class.

## SetAutoOpenConnection(bool)

Enables or disables automatic opening of the connection when required by operations.

```
public RelmContextOptionsBuilder SetAutoOpenConnection(bool autoOpenConnection)
```

Parameters

### [autoOpenConnection bool](#)

true to enable automatic opening of the connection; false to require manual connection management.

Returns

## [RelmContextOptionsBuilder](#)

The current instance of the [RelmContextOptionsBuilder](#) class.

Remarks

When automatic connection opening is enabled, the connection will be opened as needed by operations that require it. If disabled, the caller is responsible for ensuring the connection is open before performing such operations.

## SetAutoOpenTransaction(bool)

Enables or disables automatic opening of a transaction when executing database operations.

```
public RelmContextOptionsBuilder SetAutoOpenTransaction(bool autoOpenTransaction)
```

Parameters

### [autoOpenTransaction bool](#)

true to automatically open a transaction for each operation; otherwise, false.

Returns

### [RelmContextOptionsBuilder](#)

The current instance of the [RelmContextOptionsBuilder](#) class.

## SetAutoVerifyTables(bool)

Enables or disables automatic verification of tables before performing operations.

```
public RelmContextOptionsBuilder SetAutoVerifyTables(bool autoVerifyTables)
```

Parameters

### [autoVerifyTables](#) [bool](#)

true to enable automatic table verification; false to disable it.

Returns

### [RelmContextOptionsBuilder](#)

The current instance of the [RelmContextOptionsBuilder](#) class.

## SetConnectionStringType(Enum)

Sets the type of the connection string using the specified enumeration value.

```
public RelmContextOptionsBuilder SetConnectionStringType(Enum connectionStringType)
```

Parameters

### [connectionStringType](#) [Enum](#)

An enumeration value that specifies the type of connection string to use. Must not be null.

Returns

## [RelmContextOptionsBuilder](#)

The current instance of the [RelmContextOptionsBuilder](#) class.

## SetConnectionStringType(Type, Enum)

Sets the type of the connection string to use for database operations.

```
public RelmContextOptionsBuilder SetConnectionStringType(Type enumType,  
Enum connectionStringType)
```

### Parameters

**enumType** [Type](#)

The enumeration type that defines the valid connection string types. Must be an enum type.

**connectionStringType** [Enum](#)

The specific connection string type to set. Must be a defined value of **enumType**.

### Returns

[RelmContextOptionsBuilder](#)

The current instance of the [RelmContextOptionsBuilder](#) class.

### Exceptions

[ArgumentNullException](#)

Thrown if **connectionStringType** is not a defined value of **enumType**.

## SetConvertZeroDateTime(bool)

Specifies whether zero date values should be converted to DateTime.MinValue when retrieving data.

```
public RelmContextOptionsBuilder SetConvertZeroDateTime(bool convertZeroDateTime)
```

### Parameters

`convertZeroDateTime` `bool`

true to convert zero date values to DateTime.MinValue; otherwise, false.

Returns

[RelmContextOptionsBuilder](#)

The current instance of the [RelmContextOptionsBuilder](#) class.

Remarks

Zero date values are commonly used in some databases to represent an undefined or missing date. Enabling this option allows such values to be mapped to DateTime.MinValue in .NET, which may simplify handling of missing dates in application code.

## SetDatabaseConnection(MySqlConnection)

Sets the database connection to be used by the current instance.

```
public RelmContextOptionsBuilder SetDatabaseConnection(MySqlConnection connection)
```

Parameters

`connection` MySqlConnection

The open MySql.Data.MySqlClient.MySqlConnection to associate with this instance. The connection must not be null.

Returns

[RelmContextOptionsBuilder](#)

The current instance of the [RelmContextOptionsBuilder](#) class.

Remarks

After calling this method, the instance will use the provided connection for all subsequent database operations. The caller is responsible for managing the lifetime of the connection.

Exceptions

## [ArgumentNullException](#)

Thrown if `connection` is null.

## SetDatabaseConnectionString(string)

Sets the connection string used to connect to the database.

```
public RelmContextOptionsBuilder SetDatabaseConnectionString(string  
DatabaseConnectionString)
```

### Parameters

#### `DatabaseConnectionString` [string](#)

The connection string to use for database connections. Cannot be null or empty.

### Returns

#### [RelmContextOptionsBuilder](#)

The current instance of the [RelmContextOptionsBuilder](#) class.

### Exceptions

#### [ArgumentNullException](#)

Thrown if `DatabaseConnectionString` is null or empty.

## SetDatabaseName(string)

Sets the name of the database to be used for the connection.

```
public RelmContextOptionsBuilder SetDatabaseName(string databaseName)
```

### Parameters

#### `databaseName` [string](#)

The name of the database. Must be a non-empty string containing only alphanumeric characters, underscores (\_), dollar signs (\$), or Unicode characters in the range U+0080 to U+FFFF.

Returns

#### [RelmContextOptionsBuilder](#)

The current instance of the [RelmContextOptionsBuilder](#) class.

Exceptions

#### [ArgumentNullException](#)

Thrown if the databaseName parameter is null or an empty string.

#### [ArgumentException](#)

Thrown if databaseName contains invalid characters. The name must be alphanumeric and may include underscores (\_), dollar signs (\$), or Unicode characters in the range U+0080 to U+FFFF.

## SetDatabasePassword(string)

Sets the password used to connect to the database.

```
public RelmContextOptionsBuilder SetDatabasePassword(string databasePassword)
```

Parameters

#### [databasePassword](#) [string](#)

The password to use for authenticating the database connection. Cannot be null or empty.

Returns

#### [RelmContextOptionsBuilder](#)

The current instance of the [RelmContextOptionsBuilder](#) class.

Exceptions

#### [ArgumentNullException](#)

Thrown if `databasePassword` is null or empty.

## SetDatabaseServer(string)

Sets the database server to use for establishing connections.

```
public RelmContextOptionsBuilder SetDatabaseServer(string databaseServer)
```

Parameters

`databaseServer` [string](#)

The name or address of the database server. Cannot be null or empty.

Returns

[RelmContextOptionsBuilder](#)

The current instance of the [RelmContextOptionsBuilder](#) class.

Exceptions

[ArgumentNullException](#)

Thrown if `databaseServer` is null or empty.

## SetDatabaseTransaction(MySqlTransaction)

Sets the database transaction to be used for subsequent database operations.

```
public RelmContextOptionsBuilder SetDatabaseTransaction(MySqlTransaction transaction)
```

Parameters

`transaction` [MySqlTransaction](#)

The MySqlTransaction instance to associate with database operations. Can be null to clear the current transaction.

Returns

### [RelmContextOptionsBuilder](#)

The current instance of the [RelmContextOptionsBuilder](#) class.

Remarks

Use this method to specify an existing transaction for database commands. If a transaction is set, all subsequent operations will be executed within the context of that transaction until it is cleared or replaced.

## SetDatabaseUser(string)

Sets the database user name to be used for the connection.

```
public RelmContextOptionsBuilder SetDatabaseUser(string databaseUser)
```

Parameters

### databaseUser [string](#)

The user name to associate with the database connection. Cannot be null or empty.

Returns

### [RelmContextOptionsBuilder](#)

The current instance of the [RelmContextOptionsBuilder](#) class.

Exceptions

### [ArgumentNullException](#)

Thrown if `databaseUser` is null or empty.

## SetLockWaitTimeoutSeconds(int)

Sets the lock wait timeout period, in seconds, for acquiring a lock.

```
public RelmContextOptionsBuilder SetLockWaitTimeoutSeconds(int lockWaitTimeoutSeconds)
```

## Parameters

**lockWaitTimeoutSeconds** [int](#)

The maximum number of seconds to wait for a lock to be acquired. Must be a non-negative value.

## Returns

[RelmContextOptionsBuilder](#)

The current instance of the [RelmContextOptionsBuilder](#) class.

## Remarks

If the timeout is set to zero, the method will not wait and will attempt to acquire the lock immediately. Setting an appropriate timeout can help prevent deadlocks in concurrent scenarios.

## SetNamedConnection(string)

Sets the current database connection using the specified named connection string.

```
public RelmContextOptionsBuilder SetNamedConnection(string namedConnection)
```

## Parameters

**namedConnection** [string](#)

The name of the connection string to use. Cannot be null or empty.

## Returns

[RelmContextOptionsBuilder](#)

The current instance of the [RelmContextOptionsBuilder](#) class.

## Exceptions

[ArgumentNullException](#)

Thrown if the namedConnection parameter is null or empty.

## ValidateAllSettings(bool)

Validates all required database connection settings based on the configured options builder type.

```
public bool ValidateAllSettings(bool throwExceptions = true)
```

### Parameters

**throwExceptions** [bool](#)

true to throw an exception if a required setting is missing; false to return false instead of throwing an exception. The default is true.

### Returns

[bool](#)

true if all required settings are valid; otherwise, false if a required setting is missing and throwExceptions is false.

### Remarks

The required settings depend on the options builder type. For example, a named connection string requires DatabaseConnectionString, an open connection requires DatabaseConnection, and a standard connection string requires DatabaseServer, DatabaseName, DatabaseUser, and DatabasePassword.

### Exceptions

[ArgumentNullException](#)

Thrown if a required setting is missing and throwExceptions is true.

[ArgumentException](#)

Thrown if the configured connection string type is invalid.

# Enum RelmContextOptionsBuilder.OptionsBuilderTypes

Namespace: [CoreRelm.Options](#)

Assembly: CoreRelm.dll

Specifies the types of options that can be used to configure a database context using an options builder.

```
public enum RelmContextOptionsBuilder.OptionsBuilderTypes
```

## Fields

**ConnectionString = 0**

Sets the option builder connection type to use a raw connection string.

**NamedConnectionString = 1**

Sets the option builder connection type to use a named connection string.

**OpenConnection = 2**

Sets the option builder connection type to use an open connection.

## Remarks

Use this enumeration to indicate how the options builder should obtain or interpret connection information when configuring a database context. The selected value determines whether a raw connection string, a named connection string, or an existing open connection is used.

# Namespace CoreRelm.Persistence

## Classes

### [BulkTableWriter<T>](#)

Provides functionality for performing high-performance bulk insert operations into a database table with configurable options for batching, transactions, error handling, and column mapping.

# Class BulkTableWriter<T>

Namespace: [CoreRelm.Persistence](#)

Assembly: CoreRelm.dll

Provides functionality for performing high-performance bulk insert operations into a database table with configurable options for batching, transactions, error handling, and column mapping.

```
public class BulkTableWriter<T>
```

## Type Parameters

T

The type of the data items to be written to the database table. Each instance of T represents a row in the target table.

## Inheritance

[object](#) ← BulkTableWriter<T>

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

## Remarks

The [BulkTableWriter<T>](#) class is designed for efficient bulk data insertion scenarios, such as importing large datasets or synchronizing data between sources. It supports customization of batch size, transaction usage, and column inclusion rules, and allows mapping between object properties and database columns. This class is intended for internal use and is typically instantiated via factory methods or helper classes within the data access layer. Thread safety is not guaranteed; use separate instances for concurrent operations.

## Methods

### AddColumn(string, MySqlDbType, int, string, string)

Adds a column definition to the bulk table writer.

```
public BulkTableWriter<T> AddColumn(string columnName, MySqlDbType dbType, int size, string alternatePropertyName = null, string columnDefault = null)
```

## Parameters

**columnName** [string](#)

The name of the column to add. This value cannot be null or empty.

**dbType** [MySqlDbType](#)

The MySQL data type of the column.

**size** [int](#)

The maximum size of the column, in bytes. Must be a non-negative value.

**alternatePropertyName** [string](#)

An optional alternate property name to map to this column. If null, the column will use the default property mapping.

**columnDefault** [string](#)

An optional default value for the column. If null, no default value will be applied.

## Returns

[BulkTableWriter](#)<T>

The current [BulkTableWriter](#)<T> instance, allowing for method chaining.

## AddColumns(IEnumerable<Tuple<string, MySqlDbType, int, string, string>>)

Adds the specified columns to the current table schema for bulk writing operations.

```
public BulkTableWriter<T> AddColumns(IEnumerable<Tuple<string, MySqlDbType, int, string, string>> columns)
```

## Parameters

`columns IEnumerable<Tuple<string, MySqlDbType, int, string, string>>`

A collection of tuples representing the columns to add. Each tuple contains the following elements:

- The column name (`string`).
- The MySQL data type of the column (`MySql.Data.MySqlClient.MySqlDbType`).
- The column size (`int`).
- The column collation (`string`).
- The column default value (`string`).

Returns

`BulkTableWriter<T>`

The current instance of `BulkTableWriter<T>`, allowing for method chaining.

Remarks

Columns with a null data type (`MySql.Data.MySqlClient.MySqlDbType`) are ignored. If the `columns` parameter is null, no columns are added.

## AddColumns(IEnumerable<Tuple<string, MySqlDbType, int>>) (1/2)

Adds a collection of columns to the bulk table writer.

```
public BulkTableWriter<T> AddColumns(IEnumerable<Tuple<string, MySqlDbType, int>> columns)
```

Parameters

`columns IEnumerable<Tuple<string, MySqlDbType, int>>`

A collection of tuples, where each tuple specifies the column name, the MySQL data type, and the column size.

Returns

`BulkTableWriter<T>`

The current instance of `BulkTableWriter<T>` with the specified columns added.

Remarks

This method allows you to define columns for the bulk table writer by providing their names, data types, and sizes. The additional metadata for each column, such as default values or constraints, is not specified in this overload.

## AllowAutoDateColumns(bool)

Enables or disables the automatic inclusion of date columns in the bulk table writer.

```
public BulkTableWriter<T> AllowAutoDateColumns(bool allowAutoDateColumns)
```

### Parameters

`allowAutoDateColumns` [bool](#)

A value indicating whether automatic date columns should be included. [true](#) to enable automatic date columns; otherwise, [false](#).

### Returns

[BulkTableWriter](#)<T>

The current instance of [BulkTableWriter](#)<T>, allowing for method chaining.

## AllowAutoIncrementColumns(bool)

Configures whether auto-increment columns are allowed during bulk table writing.

```
public BulkTableWriter<T> AllowAutoIncrementColumns(bool allowAutoIncrementColumns)
```

### Parameters

`allowAutoIncrementColumns` [bool](#)

A value indicating whether auto-increment columns should be included. Specify [true](#) to allow auto-increment columns; otherwise, [false](#).

### Returns

[BulkTableWriter](#)<T>

The current [BulkTableWriter<T>](#) instance, enabling method chaining.

## AllowPrimaryKeyColumns(bool)

Configures whether primary key columns are allowed to be included in bulk write operations.

```
public BulkTableWriter<T> AllowPrimaryKeyColumns(bool allowPrimaryKeyColumns)
```

### Parameters

`allowPrimaryKeyColumns bool`

A value indicating whether primary key columns should be included. [true](#) to allow primary key columns; otherwise, [false](#).

### Returns

[BulkTableWriter<T>](#)

The current [BulkTableWriter<T>](#) instance, enabling method chaining.

### Remarks

By default, primary key columns are typically excluded from bulk write operations to prevent unintended modifications to primary keys. Use this method to explicitly allow their inclusion if required.

## AllowUniqueColumns(bool)

Configures whether unique columns are allowed during bulk table writing.

```
public BulkTableWriter<T> AllowUniqueColumns(bool allowUniqueColumns)
```

### Parameters

`allowUniqueColumns bool`

A value indicating whether unique columns should be allowed. [true](#) to allow unique columns; otherwise, [false](#).

Returns

## [BulkTableWriter<T>](#)

The current [BulkTableWriter<T>](#) instance, enabling method chaining.

## AllowUserVariables(bool)

Enables or disables the use of user-defined variables in the bulk table writer.

```
public BulkTableWriter<T> AllowUserVariables(bool allowUserVariables)
```

Parameters

### allowUserVariables [bool](#)

A value indicating whether user-defined variables are allowed. [true](#) to allow user-defined variables; otherwise, [false](#).

Returns

## [BulkTableWriter<T>](#)

The current instance of [BulkTableWriter<T>](#), allowing for method chaining.

## RemoveColumn(string)

Removes the specified column from the table writer.

```
public BulkTableWriter<T> RemoveColumn(string columnName)
```

Parameters

### columnName [string](#)

The name of the column to remove. This parameter is case-sensitive.

Returns

## [BulkTableWriter](#)<T>

The current [BulkTableWriter](#)<T> instance, allowing for method chaining.

### Remarks

If the specified column does not exist in the table, the method performs no action.

## ResetBatchSize()

Resets the batch size to its default value.

```
public BulkTableWriter<T> ResetBatchSize()
```

### Returns

## [BulkTableWriter](#)<T>

The current [BulkTableWriter](#)<T> instance with the batch size reset.

### Remarks

This method sets the batch size to the default value defined by the implementation. It can be used to revert any custom batch size previously set.

## SetBatchSize(int)

Sets the batch size for bulk write operations.

```
public BulkTableWriter<T> SetBatchSize(int batchSize)
```

### Parameters

batchSize [int](#)

The number of items to include in each batch. Must be a positive integer.

### Returns

## [BulkTableWriter](#)<T>

The current [BulkTableWriter<T>](#) instance, allowing for method chaining.

## Remarks

The batch size determines how many items are processed together during a bulk write operation. Setting an appropriate batch size can help optimize performance based on the workload and system resources.

## SetDatabaseName(string)

Sets the name of the database to be used for bulk table operations.

```
public BulkTableWriter<T> SetDatabaseName(string databaseName)
```

### Parameters

**databaseName** [string](#)

The name of the database. Cannot be null or empty.

### Returns

[BulkTableWriter](#)<T>

The current instance of [BulkTableWriter<T>](#) to allow method chaining.

## SetInsertQuery(string)

Sets the SQL insert query to be used for bulk data operations.

```
public BulkTableWriter<T> SetInsertQuery(string insertQuery)
```

### Parameters

**insertQuery** [string](#)

The SQL insert query string to execute during bulk operations. This query should be properly formatted and compatible with the target database.

### Returns

## [BulkTableWriter<T>](#)

The current instance of [BulkTableWriter<T>](#), allowing for method chaining.

### [SetSourceData\(IEnumerable<T>\)](#)

Sets the source data to be written in bulk operations.

```
public BulkTableWriter<T> SetSourceData(IEnumerable<T> sourceData)
```

#### Parameters

**sourceData** [IEnumerable<T>](#)

The collection of data items to be used as the source for bulk writing. Cannot be null.

#### Returns

[BulkTableWriter<T>](#)

The current instance of [BulkTableWriter<T>](#) to allow method chaining.

### [SetSourceData\(T\)](#)

Sets the source data for the bulk table writer.

```
public BulkTableWriter<T> SetSourceData(T sourceData)
```

#### Parameters

**sourceData** T

The source data to be written, represented as an instance of type **T**.

#### Returns

[BulkTableWriter<T>](#)

The current instance of [BulkTableWriter<T>](#), allowing for method chaining.

## SetTableName(string)

Sets the name of the database table to which data will be written.

```
public BulkTableWriter<T> SetTableName(string tableName)
```

### Parameters

**tableName** [string](#)

The name of the table. Cannot be null or empty.

### Returns

[BulkTableWriter](#)<T>

The current [BulkTableWriter](#)<T> instance, allowing for method chaining.

## SetTransaction(MySqlTransaction)

Sets the transaction to be used for the bulk table write operation.

```
public BulkTableWriter<T> SetTransaction(MySqlTransaction sqlTransaction)
```

### Parameters

**sqlTransaction** MySqlTransaction

The MySql.Data.MySqlClient.MySqlTransaction instance to associate with the bulk table writer. This transaction will be used to execute the database operations.

### Returns

[BulkTableWriter](#)<T>

The current [BulkTableWriter](#)<T> instance, allowing for method chaining.

## ThrowException(bool)

Configures whether the writer should throw an exception when an error occurs during bulk writing.

```
public BulkTableWriter<T> ThrowException(bool throwException)
```

## Parameters

`throwException` [bool](#)

A value indicating whether exceptions should be thrown. [true](#) to throw exceptions on errors; otherwise, [false](#).

## Returns

[BulkTableWriter](#)<T>

The current instance of [BulkTableWriter](#)<T>, allowing for method chaining.

## UseTransaction(bool)

Configures whether the bulk table writer should use a transaction during its operations.

```
public BulkTableWriter<T> UseTransaction(bool useTransaction)
```

## Parameters

`useTransaction` [bool](#)

A value indicating whether to enable transaction usage. [true](#) to use a transaction; otherwise, [false](#).

## Returns

[BulkTableWriter](#)<T>

The current instance of [BulkTableWriter](#)<T>, allowing for method chaining.

## Write()

Writes data to the underlying output stream.

```
public int Write()
```

Returns

[int](#)

The result of the write operation, as returned by the [Write\(Func<string, T, object>\)](#) method.

Remarks

This overload calls the [Write\(Func<string, T, object>\)](#) method with a [null](#) argument.

## Write(Func<string, T, object>)

Executes a batch operation to process and insert data into a database using the specified function to populate a data table.

```
public int Write(Func<string, T, object> DataTableFunction)
```

Parameters

**DataTableFunction** [Func<string, T, object>](#)

A function that takes a column name and a data item of type **T** and returns the value to be inserted into the data table.

Returns

[int](#)

The total number of records successfully inserted into the database. Returns 0 if there is no source data to process.

Remarks

This method processes the source data in batches, determined by the configured batch size, and inserts the data into the database. The method supports multiple database contexts and connections, and it uses the appropriate one based on the current configuration.

# Namespace CoreRelm.RelmInternal.Helpers.Migrations.Execution

## Classes

[MigrationApplier](#)

[MySqlScriptRunner](#)

[SchemaMigrationsStore](#)

# Class MigrationApplier

Namespace: [CoreRelm.RelmInternal.Helpers.Migrations.Execution](#)

Assembly: CoreRelm.dll

```
public sealed class MigrationApplier
```

## Inheritance

[object](#) ← MigrationApplier

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

## Methods

ApplyAsync(string, string, string, string, string, bool,  
CancellationToken)

```
public Task<bool> ApplyAsync(string serverConn, string dbTemplate, string dbName, string  
migrationFileName, string sql, bool quiet, CancellationToken ct)
```

## Parameters

serverConn [string](#)

dbTemplate [string](#)

dbName [string](#)

migrationFileName [string](#)

sql [string](#)

quiet [bool](#)

ct [CancellationToken](#)

Returns

[Task](#) <[bool](#)>

# Class MySqlScriptRunner

Namespace: [CoreRelm.RelmInternal.Helpers.Migrations.Execution](#)

Assembly: CoreRelm.dll

```
public sealed class MySqlScriptRunner
```

## Inheritance

[object](#) ← MySqlScriptRunner

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

## Methods

### ExecuteScriptAsync(IRelmContext, string, CancellationToken)

```
public Task ExecuteScriptAsync(IRelmContext context, string sql, CancellationToken ct  
= default)
```

#### Parameters

context [IRelmContext](#)

sql [string](#)

ct [CancellationToken](#)

#### Returns

[Task](#)

# Class SchemaMigrationsStore

Namespace: [CoreRelm.RelmInternal.Helpers.Migrations.Execution](#)

Assembly: CoreRelm.dll

```
public sealed class SchemaMigrationsStore
```

## Inheritance

[object](#) ← SchemaMigrationsStore

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

## Methods

### EnsureTableAsync(IRelmContext, CancellationToken)

```
public Task<int> EnsureTableAsync(IRelmContext context, CancellationToken ct = default)
```

#### Parameters

context [IRelmContext](#)

ct [CancellationToken](#)

#### Returns

[Task](#)<[int](#)>

### GetAppliedAsync(IRelmContext, CancellationToken)

```
public Task<Dictionary<string, AppliedMigration>> GetAppliedAsync(IRelmContext context,  
CancellationToken ct = default)
```

#### Parameters

context [IRelmContext](#)

ct [CancellationToken](#)

Returns

[Task](#)<[Dictionary](#)<[string](#), [AppliedMigration](#)>>

RecordAppliedAsync(IRelmContext, string, string, CancellationToken)

```
public Task<int> RecordAppliedAsync(IRelmContext context, string migrationFile, string checksumSha256, CancellationToken ct = default)
```

Parameters

context [IRelmContext](#)

migrationFile [string](#)

checksumSha256 [string](#)

ct [CancellationToken](#)

Returns

[Task](#)<[int](#)>

# Namespace CoreRelm.RelmInternal.Helpers. Migrations.Introspection

## Classes

[DbAvailabilityHelper](#)

[SchemaSnapshotFactory](#)

# Class DbAvailabilityHelper

Namespace: [CoreRelm.RelmInternal.Helpers.Migrations.Introspection](#)

Assembly: CoreRelm.dll

```
public static class DbAvailabilityHelper
```

## Inheritance

[object](#) ← DbAvailabilityHelper

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

## Methods

EnsureForApplyOrMigrateAsync(IRelmDatabaseProvisioner,  
string, string, Action<string>, Action<string>,  
CancellationToken)

```
public static Task<bool> EnsureForApplyOrMigrateAsync(IRelmDatabaseProvisioner provisioner,  
string serverConnectionString, string databaseName, Action<string> logInfo, Action<string>  
logWarn, CancellationToken ct)
```

## Parameters

provisioner [IRelmDatabaseProvisioner](#)

serverConnectionString [string](#)

databaseName [string](#)

logInfo [Action<string>](#)

logWarn [Action<string>](#)

ct [CancellationToken](#)

Returns

[Task](#) <bool>

**WarnIfMissingAsync(IRelmDatabaseProvisioner, string, string, Action<string>, CancellationToken)**

```
public static Task<bool> WarnIfMissingAsync(IRelmDatabaseProvisioner provisioner, string  
serverConnectionString, string databaseName, Action<string> logWarn, CancellationToken ct)
```

Parameters

provisioner [IRelmDatabaseProvisioner](#)

serverConnectionString [string](#)

databaseName [string](#)

logWarn [Action](#) <string>

ct [CancellationToken](#)

Returns

[Task](#) <bool>

# Class SchemaSnapshotFactory

Namespace: [CoreRelm.RelmInternal.Helpers.Migrations.Introspection](#)

Assembly: CoreRelm.dll

```
public static class SchemaSnapshotFactory
```

## Inheritance

[object](#) ← SchemaSnapshotFactory

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

# Methods

## Empty(string)

```
public static SchemaSnapshot Empty(string dbName)
```

### Parameters

dbName [string](#)

### Returns

[SchemaSnapshot](#)

# Namespace CoreRelm.RelmInternal.Helpers.Migrations.MigrationPlans

## Classes

[RelmMigrationPlanner](#)

# Class RelmMigrationPlanner

Namespace: [CoreRelm.RelmInternal.Helpers.Migrations.MigrationPlans](#)

Assembly: CoreRelm.dll

```
public sealed class RelmMigrationPlanner : IRelmMigrationPlanner
```

## Inheritance

[object](#) ← RelmMigrationPlanner

## Implements

[IRelmMigrationPlanner](#)

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

## Methods

### Plan(SchemaSnapshot, SchemaSnapshot, MigrationPlanOptions)

```
public MigrationPlan Plan(SchemaSnapshot desired, SchemaSnapshot actual,  
                           MigrationPlanOptions options)
```

#### Parameters

desired [SchemaSnapshot](#)

actual [SchemaSnapshot](#)

options [MigrationPlanOptions](#)

#### Returns

[MigrationPlan](#)

# Namespace CoreRelm.RelmInternal.Helpers. Migrations.Provisioning

## Classes

[MySqlDatabaseProvisioner](#)

# Class MySqlDatabaseProvisioner

Namespace: [CoreRelm.RelmInternal.Helpers.Migrations.Provisioning](#)

Assembly: CoreRelm.dll

```
public sealed class MySqlDatabaseProvisioner : IRelmDatabaseProvisioner
```

## Inheritance

[object](#) ← MySqlDatabaseProvisioner

## Implements

[IRelmDatabaseProvisioner](#)

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

## Methods

### DatabaseExistsAsync(string, string, CancellationToken)

```
public Task<bool> DatabaseExistsAsync(string serverConnectionString, string databaseName,  
CancellationToken ct = default)
```

## Parameters

serverConnectionString [string](#)

databaseName [string](#)

ct [CancellationToken](#)

## Returns

[Task](#)<[bool](#)>

## EnsureDatabaseExistsAsync(string, string, string?, string?, CancellationToken)

```
public Task EnsureDatabaseExistsAsync(string serverConnectionString, string databaseName,  
    string? charset = null, string? collation = null, CancellationToken ct = default)
```

### Parameters

serverConnectionString [string](#)

databaseName [string](#)

charset [string](#)

collation [string](#)

ct [CancellationToken](#)

### Returns

[Task](#)

# Namespace CoreRelm.RelmInternal.Helpers.Migrations.Rendering

## Classes

[MySqlMigrationSqlRenderer](#)

# Class MySqlMigrationSqlRenderer

Namespace: [CoreRelm.RelmInternal.Helpers.Migrations.Rendering](#)

Assembly: CoreRelm.dll

```
public sealed class MySqlMigrationSqlRenderer : IRelmMigrationSqlRenderer
```

## Inheritance

[object](#) ← MySqlMigrationSqlRenderer

## Implements

[IRelmMigrationSqlRenderer](#)

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

## Methods

### Render(MigrationPlan, MySqlRenderOptions?)

```
public string Render(MigrationPlan plan, MySqlRenderOptions? options = null)
```

#### Parameters

plan [MigrationPlan](#)

options [MySqlRenderOptions](#)

#### Returns

[string](#)

# Namespace CoreRelm.RelmInternal.Models

## Classes

### [ForeignKeyNavigationOptions](#)

Class to hold foreign key navigation options, used internally by CoreRelm to manage relationships between models.

# Class ForeignKeyNavigationOptions

Namespace: [CoreRelm.RelmInternal.Models](#)

Assembly: CoreRelm.dll

Class to hold foreign key navigation options, used internally by CoreRelm to manage relationships between models.

```
public class ForeignKeyNavigationOptions
```

## Inheritance

[object](#) ← ForeignKeyNavigationOptions

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)