



XData

.Net domain object oriented complete feature rich data access layer component

Table of Contents

[XData](#)

[Tutorial](#)

[Introduction](#)

[Glossary](#)

[Basic operations with data](#)

[Versions of XData](#)

[Mapping common rules](#)

[Static mapping](#)

[Dynamic mapping](#)

[Read-only dynamic query building](#)

[Runtime filters](#)

[SQL block](#)

[LINQ support](#)

[Using three tier architecture](#)

[Processing of related objects \(Unit of Work\)](#)

[Security issues](#)

[Concurrency resolving](#)

[Localization](#)

[Logging and debug info](#)

[Install & configuration](#)

[Quick start](#)

[Tips & Tricks](#)

[Appendix I. API Documentation](#)

[XData](#)

[Actions](#)

[ApplyFlag](#)

[AttachedHandler<T>](#)

[AttachedHandlerType](#)

[Calculate<T>](#)

[ClearAttachedHandler](#)

[Combination](#)

[CredentialsType](#)

[CurrentObjectChanged<T>](#)

CurrentObjectChanging<T>
CustomLogic<T>
CustomMapping<T>
DataActionType
DataCombination
DataExpressionType
DataGrouping
DataObjectExtensions
DataObjectState
DataPropertyFlag
DataSourceType
DataStructureFlag
DataSubmitFlag
DataTriggerFlag
DataVersion
DateDiffKind
DefaultFeature
DefaultType
DialectOptions
ExecuteAttachedHandler<T>
ExistsOperation
FilterConstantType
FilterNullable
FilterOperation
FilterType
GetLobAttachedHandler
GetLobSizeAttachedHandler
GetXmlAttachedHandler
IDataEngine
IDataLogic<T>
IDataObject
IDataOptions
IDataQuery
IDataQuery<T>
IDataScope
InitObject<T>
InitRepository<T>

InvalidateObject<T>
InvalidateRepository<T>
IRepository
IRepository<T>
ISqlObject
IWorkScope<T>
IWorkSet
IWorkSet<T>
LockAttachedHandler<T>
Log
Mapping<TFrom, T>
MessageType
ParameterIndexer
ProcedureType
RefreshAttachedHandler<T>
RepositoryAttachedHandler<T>
RepositoryIndexer
RepositoryTrigger<T>
ResultSetIndexer
RuntimeAssemblyLoader
SerializationExtensions
SerializedData
SerializedValue
SetLobAttachedHandler
SetXmlAttachedHandler
SubmitAttachedHandler<T>
SubmitObjectAttachedHandler<T>
SubmitQueryAttachedHandler<T>
Trigger<T>
Variable
VariableIndexer
WithRecursiveType
Work
WorkState
XDataLogic<T>
XDataManager
XDataPureLinqMethodAttribute

XData.Async

AsyncExtensions

XData.Database

DataQueries

FieldInfo

ITransaction

TableInfo

XDataContext

XData.Database.Adapter

BaseDatabaseAdapter<TConnection, TParameter>

CallResult

DatabaseAdapterAttribute

IDatabaseAdapter

IDatabaseAdapterMetadata

Query

QueryParam

QueryResult

QueryResultBase

StoredProcedure

UdtSqlType

XData.Database.Dialect

BaseSqlBuilder

ExpressionContext

MergeFlag

MergeMap

SqlStringType

SqlTypeSize

XConnectionInfo

XData.Database.SqlBlock

IBlockQueryAdapter

IBlockQueryDescription

IBlockQueryDescription<T>

IBlockQueryStructure

IExpressionMergeAdapter

ILoopVariableAdapter<T>

IMergeAttributeAdapter

IMergeBlockAdapter

IPropertyMergeAdapter
ISetBlockAdapter
ISetSqlVarAdapter
ISqlBlock
ISqlBlockAdapter
ISqlBlockAdapter<T>
ISqlParam
ISqlParamAdapter
ISqlTableAdapter
ISqlValue
ISqlValue<T>
ISqlValueAdapter
ISqlVarAdapter
ISqlVariable
SqlBlockDefinition
SqlBlockDefinition<T>
SqlBlockExtensions
SqlType
XData.DataSource.Structure
DataQueryState
DataStructureState
Field
FilterCombinationState
FilterName
FilterState
Grouping
Hierarchy
IConstantFilter
IDataSource
IDataStructure
IDictionaryFilter
IExpressionFilter
IExternalLink
IField
IFilter
IFilterCombination
IInnerLink

- InnerViewLink
- INamedFilter
- IOptionalFilter
- IProcedure
- IRangeFilter
- IRuntimeFilter
- ISqlPart
- ISubqueryFilter
- ISubqueryLink
- ITable
- ITableLike
- ITempTable
- IXmlSource
- Order
- RepositoryState
- RuntimeFilter
- SelectedField
- TaggedValue
- XData.Exceptions
 - AdditionalInfo
 - XDataAdapterException
 - XDataAdapterExceptionType
 - XDataConcurrencyException
 - XDataConfigurationException
 - XDataConfigurationExceptionType
 - XDataException
 - XDataExceptionAttribute
 - XDataQueryStructureException
 - XDataQueryStructureExceptionType
 - XDataRuntimeException
 - XDataRuntimeExceptionType
 - XDataSecurityException
 - XDataSqlException
- XData.Extensions
 - CollectionsExtensions
 - DataObjectQueryableExtensions
 - ExpressionExtensions

- GeneralPropertyComparer<T>
- IndexAttribute
- Indexer<T>
- IProcess<T>
- JoinedEnumerable
- JoinedEnumerable<T>
- Processing
- TypeComparer
- TypeExtensions
- TypeExtensions.DummyType
- XData.Interfaces
 - AdapterConfig
 - ContextConfig
 - DialectConfig
 - IConfiguration
 - ILogWriter
 - IProxy
 - IProxyManager
 - ISecuritySession
 - ISequenceNameRule
 - ISqlBuilder
 - ProxyConfig
- XData.Json
 - HideFlags
 - JsonDataObjectConverter<T>
 - JsonSettings<T>
 - LinkMode
- XData.Mapping
 - ActionAttribute
 - ColumnAttribute
 - ColumnDefaultAttribute
 - ColumnExpressionAttribute
 - ColumnUpdateWithAttribute
 - ConstantFilterAttribute
 - DataObjectAttribute
 - DataSourceAttribute
 - DataTableAttribute

DictionaryFilterAttribute
ExpressionFilterAttribute
ExternalLinkAttribute
FilterAttribute
FilterBaseAttribute
FilterCombinationAttribute
HierarchyAttribute
HintAttribute
HintInfo
HintType
InnerViewAttribute
Link<TVal, TSrc>
LinkAttribute
LinkPropertyAttribute
Lob
OptionalFilterAttribute
ParameterAttribute
ProcedureAttribute
PropertyAttribute
PropertyBaseAttribute
PropertyDefaultAttribute
PropertyExpressionAttribute
PropertyGroupingAttribute
PropertyUpdateWithAttribute
RangeFilterAttribute
ReferenceAttribute
ResultSetAttribute
SqlExpressionAttribute
SubqueryAttribute
SubqueryFilterAttribute
SubqueryLinkAttribute
SubqueryVariableAttribute
WithAttribute
WithRecursiveAttribute
Xml
XData.Mapping.Dynamic
IExpressionPropertyAdapter

IExternalLinkAdapter<T, TChild>
IExternalLinkDefinition
IExternalLinkDefinitionAdapter<T>
IExternalLinkFlag
IFilterAdapter
IFilterDescription
IInheritedStructure<T>
IInnerFilterAdapter
ILinkPropertyMappingAdapter<T>
IParameterDescription
IProcedureParameterAdapter
IProcedureParameterFlag
IPropertyMappingAdapter
IQueryDescription
IQueryDescription<T>
IQueryStructure
IQueryStructureAdapter
IQueryWithAdapter<TRoot>
IQueryWithFlag
IRepositoryDescription<T>
IRepositoryExpressionMappingAdapter
IRepositoryPropertyMappingAdapter<T, TValue>
IRepositoryReadOnlyPropertyMappingAdapter
IRepositoryStructure<T>
IRepositoryStructureAdapter
IRepositoryStructureAdapter<T>
IResultSetDescription
IStructureFlag
ISubqueryLinkAdapter
ISubqueryLinkAdapter<T, TDObj>
ISubqueryLinkAdapter<TDObj>
ITaggedExpression
ParameterDescription
QueryDescriptionExtensions
TaggedExpression<TTag, TResult>
XDataMapping
XData.Security



XData is .Net domain object oriented data access layer component. It is not "yet another ORM", but a complete feature rich data access layer for Your solutions. Basic idea to create this component was a supplying high quality tool to professional developers, having deep competences in data storage modeling and access.

version 1.2.0

Nuget packages

[XData Data Access Layer package](#)

[XData WorkSet \(UnitOfWork\) package](#) - required to use [XData WorkSet \(UnitOfWork\)](#) implementation

[XData Security package](#) - [ISecuritySession](#) interface implementation helper required to use [XData security](#) functionality

[XData Three-tier client proxy package](#) - client proxy to use in three-tier environment (Net 4.0 version only, required full Net 4.0 Framework version)

[XData Cache package](#) - time limited object caching (Net 4.0 version only, required full Net 4.0 Framework version. Net Standard 2.0 version does not require additional packages)

[XData MsSqlSever adapter package](#) - Ms SQL Server database dialect and adapter

[XData Oracle adapter package](#) - Oracle database dialect and adapter based on Oracle ODP provider

[XData PostgreSQL adapter package](#) - PostgreSQL database dialect and adapter based on Npgsql provider

[XData SQLite adapter package](#) - SQLite database dialect and adapter (Net 4.0 version has used System.Data.SQLite provider, Net Standard 2.0 version has used Microsoft.Data.Sqlite provider)

[XData MySql adapter package](#) - MySQL database dialect and adapter based on MySql.Data provider

[XData net 4.0 System.Diagnostics.Trace log writer package](#) - (Net 4.0 only) Log writer over System.Diagnostics.Trace

[XData net 4.0 log4net log writer package](#) - (Net 4.0 only) Log writer over Log4Net

[XData net 4.0 Inversion of Control container package](#) - Support for registering logic modules using the IoC container (Net 4.0 version only, required full Net 4.5 Framework version). Any IoC container with [IDependencyResolver](#) interface implementation is supported. Net Standard 2.0 version does not require additional packages since .Net Standard has built-in support for IoC containers

[XData net standard 2.0 Asynchronous Disposable support package](#) - Support for [IAsyncDisposable](#) interface for disposable objects (Net Standard 2.0 version only, required Net Standard 2.1 specification version support)

[XData net standard 2.0 Asp.Net Core HealthChecks support package](#) - Support for [Asp.Net Core HealthChecks](#) (Net Standard 2.0 version only, required Net Standard 2.1 specification version support)

Plugins

Net 4.0

[XData net 4.0 Three-tier server](#)

Net Standard 2.0

XData Net Standard 2.0 has no plugins. Three-tier architecture is currently not supported in Net Standard 2.0 version.

Online documentation

[Online documentation](#)

PDF documentation

[PDF documentation](#)

Contacts

Denis Dawydenko AKA Mick Fierte – d.dawydenko@gmail.com

Thanx

Special thank to my friends who help me to make it project possible with supporting me, great ideas, working problem solutions and laborious testing:

- Alex Lukinov
- Andrew Ustinov
- Stanislav Kovanovsky
- Pavel Kryukov
- Max Lapotkov
- Olzhas Khalelov

Introduction

Why and when You need XData?

- Independence:
 - Business object hierarchy of Your application not depend XData object hierarchy.
 - Database structure not dictate program object structure and vice versa.
 - Database specific terms (primary key, foreign key, surrogate key value, concurrency token and so on) can be hidden from program object model due object mapping.
 - XData base mapping concept based on one business object mapping with any number of data sources (tables, xml sources, subqueries, recursive trees and procedures) with not restricted recursion deep.
 - Each of mapped business object can operate with hierarchically organized "updatable tree" of tables so business object changes can be stored in multiple linked tables at once. Complicated business objects can be created, updated and deleted with no additional code more then mapping.
 - Same mapping can be used over multiple RDBMS (with some restrictions).
 - SQL dialect modules can be developed independently from adapter modules developed over various ADO.Net providers used to access the same RDBMS.
 - With ability to use XML data sources, temporary tables and procedure support, system can work using proprietary application database schemes where schema changes are not allowed.
 - All technologically required tools (for example DI) can be integrated with XData, but You can choose Your favorite one independently without any dictate or double using it inside XData.
- Functionality:
 - Full IQueryable and LINQ support.
 - XData allowed to use SQL expressions as field data source.
 - Calculated properties can be evaluated on .Net side.
 - [Active record](#) model of data access is supported.
 - Fully declarative using of optimistic locking of data is provided.
 - Pessimistic locking is allowed when it supported by RDBMS functionality.
 - Your application over XData can be deployed in three-tier mode with data aware logic can be took out to the application server layer (only in full .Net Framework 4.0 or above for now).
 - XData introduce data aware business object actions which can be used as in two-tier mode as in three-tier mode.
 - .Net trigger like logic can be programmed over mapped business objects (not over single table!). This feature also can be deployed on application server layer in three-tier mode.
 - Unnamed SQL code blocks can be described as LINQ-like structures that have the same syntax, regardless of the DBMS or SQL dialect used.
 - The ability to inherit the mapping description either using the usual inheritance structure and building a separate inheritance hierarchy for the mapping description.
- Usability
 - Human readable mapping structure with unambiguous correlation with resulting SQL query. Programmer can get query text to check query execution plan for optimization.
 - Feature rich security subsystem over business object level can be simply implemented using separate NuGet package.
 - Declarative realization of [Unit of Work](#) over business object level available when using separate NuGet package.
 - Lazy loading of large objects (XML, BLOB...) is available from the box.
 - Two modes of mapping (using attributes and LINQ style expressions) or their mixing can be used.
 - Dynamic in-place read only queries it's a powerful tool to access data in complicated cases.
 - Virtual data support. You can use database mocking ability for testing.
 - You can debug three-tier XData application in two-tier mode.
 - Attachable localization based on satellite assemblies ability can be used to localize XData internal messages to any languages You interested (only in full .Net Framework 4.0 or above for now).
- Speed:
 - SQL optimization granted by human controlled mapping results.

- Fields containing large objects (LOB or XML) are loaded only when they are needed. Hierarchically organized objects are also loaded only if they are needed and to the extent that they are needed. These and other built-in ways to optimize work with data allow XData to work optimally process complex data objects at the same time without limiting the developer's capabilities.
- Code of XData is extremely optimized to speed with using profiler until no one internal code block is still marked "hot" over business logic and database access.
- Mapping structure caching. Cache can be reset by programmer.
- Data caching. Cache can be reset by programmer.
- SQL hints are supported to improve productivity in most complicated cases.

Is any restrictions of using XData exists?

- XData not supported [Code First](#) approach: Programmer or database administrator can write SQL script. This code is produced by human, real (not artificial) intellect of professional... But still XData has ability to run this script over database using Custom SQL executing feature.
- XData not supported DDL: By the same logic as above, and with the same receipt of using.
- XData not supported [Database First](#) approach: Now I've no idea to provide the same independence level as announced above using this approach (may be in the future...).
- Powerful, but complicated: Try it, simple things still are simple, more complicated is not as scary as they appear at first sight. But the power is never be useless.
- RDBMS SQL specific statements: So many RDBMS specific statements. So many features realized in one only database system, or restricted by version... May be some of them will be implemented, may be not...
- (planned to support in next versions) More dialects and adapters: Now supported MS SQL Server, Oracle, PostgreSQL, MySQL and SQLite. It's a most used RDBMS, but Your favorite database can be out of the list. There are two ways - write me a letter and wait for my code realization, or develop it by Yourself. It's not a "piece of cake", but also not a huge amount megabytes of highly complicated code.

Glossary

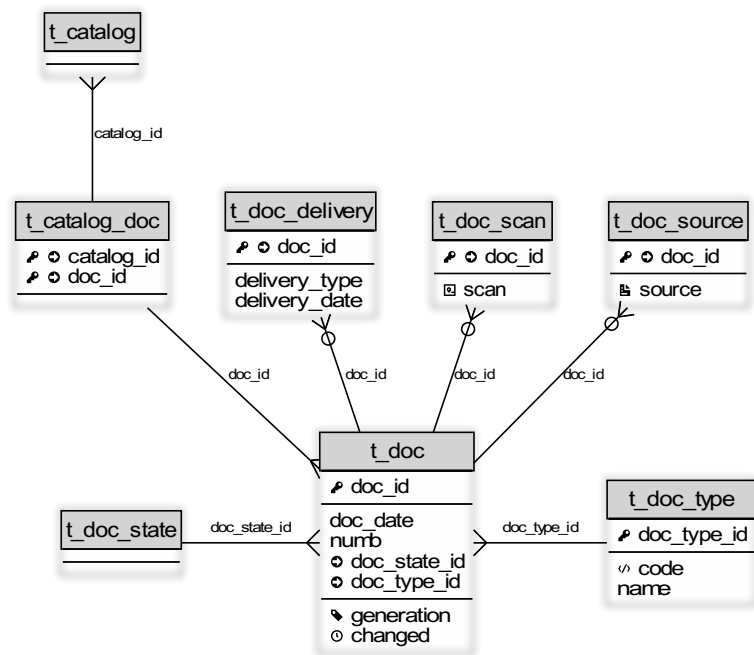
Data object

Application domain objects mapped (statically or dynamically) to database representation. Data object instances can be obtained through generic [repository](#) typed by data object type. Data objects non read-only properties can be changed and that changes may be submitted to database. The data object in XData is considered as exactly the object of the domain (in [DDD](#) terminology). Moreover, its reflection on the database structure is not limited. So a data object can be reflected on the contents of one table:

t_doc_state
doc_state_id
code
name

```
public partial class DocState
{
    public long DocStateId => this.GetProperty(x => x.DocStateId);
    public string Code { get; set; }
    public string Name { get; set; }
}
```

A few tables:



```
public partial class Invoice
{
    public long DocId => this.GetProperty(x => x.DocId);
    public string DocStateCode => this.GetProperty(x => x.DocStateCode);
    public Link<string, DocState> DocState { get; set; }
    public long DocCatalogId { get; set; }
    public string Numb { get; set; }
    public DateTime? DocDate { get; set; }
    public Guid Generation => this.GetProperty(x => x.Generation);
    public DateTime Changed => this.GetProperty(x => x.Changed);
    public Lob Scan { get; set; }
    public Xml Source { get; set; }
    public DeliveryTypeEnum DeliveryType { get; set; }
    public DateTime DeliveryDate { get; set; }
}
```

And also several objects can be reflected in the same tables: for example, different types of documents from the previous example

can be based on the same table T_DOC and having another link to T_DOC_TYPE store their attributes in the same and/or other tables related T_DOC as one-to-one.

Updatable tables hierarchy

The mapping of the data object to the storage structure can include mutable (updatable) tables — which will be updated when the [data object](#) changes and that changes are reflected in the database. And information tables — which contain reference data values, references to which (for example) are contained in the mutable tables but this data itself will not be updated even if we change the link to this directory in the data object. The link will change but the dictionary itself will remain unchanged.

Updatable tables are arranged in a hierarchy according to the relationships between the tables from the root - [base table](#) into which, when inserting, it is necessary to insert data first of all. The base table has no dependencies on mutable tables of this data object, only on informational ones (if any). Further along the hierarchy, updatable tables will be directly dependent on the base table and so on. Information tables are not included in the tree of updated tables and in the description of the mapping of the data object are placed on the first level sibling to the base table.

Real update of one or subtree of a tables in that hierarchy depends on changes of properties that mapped to concrete tables. When [optional references](#) is declared between tables inside updatable tables hierarchy, update of data objects can be automatically translated to insert or delete SQL statement applied to concrete table.

Base table

Root table of [updatable tables hierarchy](#). In case of using read-only repository mapping, base table role can be assigned to any table (or [virtual table](#)) with which it will be logical to begin the construction of the SQL *SELECT* expression.

Optional references

The outer join references between tables within [updatable tables hierarchy](#). Referenced (detail) table must contain one or more properties marked as [mandatory or OuterFlag](#). At least one mandatory property are required and it can not be mapped on primary key (or part of primary key). In most cases that properties is natural key of the table. The values assigned to that properties will be used to make a decision over operation applied to table.

This property value used as indicator of whether or not to insert an entry into the table connected by the outer join (when inserting). If value is null inserting to this table (and their child tables in updatable tables hierarchy) will be skipped. When updating to an empty value, it signals the need to delete the record from the table optionally connected and their child tables in updatable tables hierarchy.

Data scope

Data scope term is required to describe an independent environment of operating the same data inside the application. Each data scope has an GUID identifier - layer. In each data scope application logic can set independent [filters](#) values over the same [repository](#). Same repositories can be linked over different data scopes using different ways and rules.

Repository

Repository grant access to acquire [data objects](#) and apply their changes to database. Repository is represented by XData internal structures and available using interface `IRepository<T> extend IQueryable<T>`, where T is data object type.

Context

String alias associated to one of application database instance and all configuration options to use it inside XData environment. Configuration options can be obtained from configuration file or registered dynamically during runtime (see [Installation & configuration](#)).

Repository alias

Optional [repository](#) alias string. Required to unique identification repository used multiple times within one [data scope](#). Sometimes the same repository (but arranged by different [runtime filters](#) and [variables](#)) can be useful as multiple detail data sources in master-detail relations inside single data scope (master-detail relations is limited to use single data scope).

Filters

Filters statically (by mapping) or dynamically applied to [repository](#).

To define complex filtration rules filters can be grouped in filter groups characterized by name and logical filter combination operation (*AND* or *OR*). Filter groups can combine filters and/or depended filter groups. Root filter group of [repository](#) with filters combined using *AND* logical operation, can be omitted and will be automatically created in runtime. Default filter group name is empty string. By default all filters are linked to default filter group. To specify other filter group to certain filter set filter property *Combination* to name of this filter group.

The master-detail relations is realized on detail repository filters values set up also. In other cases, repository can be filtered by apply [LINQ expressions](#) on it.

Runtime filters

Filters, that allow set the value during runtime. Runtime filter value set can be automated using describing master-detail relation between [repositories](#), or manually, using [SetFilterValue](#) method of repository.

The runtime filter can be defined as part of the mapping of the data object or added dynamically when the filter value is assigned. However, to dynamically add a runtime filter, it is required that the field to which that filter will be applied be described in mapping (as a property, read-only property or a hidden property).

In addition, using the runtime filter, you can change (or set) the filter value described in mapping. In this case, the property is not specified when describing such a filter, and the filter name must match the one described during mapping.

Data source flags

[DataStructureFlags](#) enumeration contains set of flags that defines structure of data source. Data source can be marked with combination of values:

- **None** - default data source structure, plain object structure, can be edited, data grouping not applied.
- **ReadOnly** - data source marked as read-only. In optimization goals is recommended to use this flag for all data sources read-only by application logic.
- **Tree** - data source is marked as hierarchically organized. Allow to use multiple ways to filter detail data sources (when master is marked as **Tree**) - using plain links to active tree node, or to active tree node and subtree.
- **Grouping** - data source is marked as grouped. Required to describe grouped data source.

Property flags

[DataPropertyFlag](#) enumeration contains set flags that defines specific role of property of the [data object](#) during its mapping:

- **None** - default value, property marked with no specific roles
- **Id** - property is part of [data object](#) primary key
- **OuterFlag** - property is marked as mapped to mandatory field of [optional referenced](#) table. The value of this property is used to make a decision over operation applied to optional referenced table.
- **ConcurrencyToken** - concurrency token used within optimistic [locking](#).

SQL expression type

[DataExpressionType](#) enumeration contains types of SQL expression description when using in [data object](#) mapping:

- **PlainSql** - SQL expression is plain SQL code,
- **SubQuery** - SQL expression is subquery with specified alias,
- **LinqExpression** - SQL expression defined as a static private field typed as [Calculate<T>](#), where T is a SQL expression result type, and the field value is LINQ style expression reflects SQL expression logic.

Default value type

When inserting data into a database table, the concept of a default field value is often used. [DefaultType](#) enumeration contains types of values used to set default values of properties:

- **CurrentDate** - current date (database time settings used)
- **CurrentDateTime** - current date and time (database time settings used)
- **CurrentDateTimeUTC** - current date and time UTC (database time settings used)
- **Variable** - [variable](#) value with name stored in *DefaultValue* (see [here](#) for details)
- **Const** - constant value stored in *DefaultValue*
- **AutoIncrement** - value generated by database by auto increment fields or sequence. In case of using sequence is required to match the rules described [here](#).
- **NewGuid** - new GUID value is generated
- **UserName** - current user name. [Security provider subsystem](#) using is required to fill this type of default value.

Default value features

[DefaultFeature](#) enumeration contains extension features used with default values of properties is assigned:

- **UseOnInsert** - use default value on insert (by default, always applied)
- **UseOnUpdate** - use default value on update
- **SkipWhenAssigned** - skip insert into table when PK value assigned explicitly (see [here](#) for details)
- **UpdateWhenAssigned** - change insert operation over table to update when PK value assigned explicitly (see [here](#) for details) Default features can be combined. For example:

```
csharp DefaultFeature.UseOnUpdate | DefaultFeature.SkipWhenAssigned
```

Aggregation type

[DataGrouping](#) enumeration contains data aggregation functions used inside grouped data source:

- **None** - mapped field or SQL expression is one of data source grouped by.
- **Count, Sum, Min Max, Avg** - corresponded function is applied to mapped field or SQL expression.

Property data source

Table field, view field, SQL procedure (or function) parameter or SQL expression mapped to [data object](#) property value.

Virtual table

Various RDBMS has mechanics to select the data without specifying real source ("dual" table in ORACLE, omit "from" section in MsSqlServer and so on). Virtual data table is recommended for use as source of [calculated fields](#), [SQL expressions](#), when [virtual data](#) processing is required. Virtual table is not described in mapping. To set virtual data source as source of property set data source alias as empty string.

Filter group

Named group of filters (or submittal filter groups) linked with the same logical operator (**AND** or **OR**). Root filter group named as empty string and have an **AND** logical operator. Root filter group can be omitted in mapping description. All filters without explicit group specification is assigned to root filter group. Filter group names must to be unique named within data object mapping description.

Filter null processing behavior

[FilterNullable](#) enumeration contains behaviors applied to null value processing while comparing data to filter value:

- **Nullable** - (default) filter null value is omits the filter (no filter limitation applied)
- **NullsAllowed** - null and not null values of filter and data is compared separately (filter within SQL query will looks like *((field is null and :param is null) or field = :param))*)
- **NullsNotAllowed** - filter null value is translated as non valid and query will return no records (filter within SQL query will looks like *(:param is not null and field = :param))*)
- **NullsCompared** - only null values of filter and data is translated as valid (filter within SQL query will looks like *(:param is null and field is null)*)

Variables

Using variables allows you to customize the existing mapping for a specific situation. Using [Constant filters](#) or [Dictionary filters](#) (or see [here](#) and [here](#) for dynamic mapping) with a value of the type *variable value*, you can delay the substitution of a particular value until it is used. To do this, use the "variables" parameter of the [GetRepository](#) method.

Another use of variables is to set stored [procedure parameters](#) (or see [here](#) for dynamic mapping).

In addition, you can use the variables to define the [default field values](#).

In the case of using a subquery (or inner view), you can specify variables for a specific subquery in the [mapping](#) (or see [here](#) for dynamic mapping). This is useful when the subquery structure is used several times with slightly different filter values.

Note that you can use the [SetVar](#) extension method to describe variables:

```
"TestVar".SetVar(1)
```


Basic operations with data

Data engine, data scope and repository

To start using XData You need get [IDataEngine](#) interface reference (see examples below). It will be used to create new [data scopes](#) using [NewDataScope](#) method.

XData use data scope isolation when same typed [repositories](#) used by multiple logic at the same time. Data scope is represented by [IDataScope](#) interface. Data scope identified by [Layer](#) (GUID).

❏ Important

IDataScope is IDisposable. When disposing IDataScope all repositories, opened at this scope are disposed and XData free all memory used by this data scope.

To get [data object](#) instances of type *T* it's required to get typed repository interface [IRepository<T>](#) using method [GetRepository<T>](#) of *IDataScope*.

💡 Tip

IRepository is also IDisposable. When disposing IRepository it's data cache cleared. You can dispose repositories manually, but if not, disposing data scope will free that repository resources. Recommended to dispose large data repositories manually to reduce memory utilized by application.

While repository is not disposed inside data scope it can be requested many times without recreating and database access and [GetRepository<T>](#) method of same *IDataScope* returns the same instance of repository.

Net 4.0:

```
using (var dataEngine = XDataManager.InitXData(x =>
    x.UseConfiguration(ConfigurationManager
        .OpenExeConfiguration(ConfigurationUserLevel.None))))
{
    using (var dataScope = dataEngine.NewDataScope())
    {
        foreach(var data in dataScope.GetRepository<SomeObject>())
            Console.WriteLine(string.Format("{0}, {1}", data.SomeId, data.Name));
    }
}
```

NetStandard 2.0:

```
//Set configuration file
var builder = new ConfigurationBuilder();
builder.AddXmlFile("MyConsoleApp.config");
var configFile = builder.Build();

//Set services
using(var serviceProvider = new ServiceCollection()
    .AddLogging()
    // Localization only for ASP.NET Core
    .AddSingleton(typeof(IStringLocalizerFactory), x => null)
    .AddSingleton(typeof(IConfigurationRoot), x => configFile)
    .AddXData()
    .BuildServiceProvider())
{
    //Configure logging (logger messages minimal allowed levels can be set using config file)
    serviceProvider.GetRequiredService<ILoggerFactory>()
        .AddConsole(LogLevel.Warning).AddDebug(LogLevel.Debug);

    var dataEngine = serviceProvider.GetRequiredService<IDataEngine>();

    using (var dataScope = dataEngine.NewDataScope())
    {
        foreach(var data in dataScope.GetRepository<SomeObject>())
            Console.WriteLine($"{data.SomeId}, {data.Name}");
    }
}
```

Data object properties access

Read/write properties using has no specifics:

```
var invoice = dataScope.GetRepository<Invoice>().First();
invoice.DocNumb = "12345";
```

Except [Link](#), [Lob](#) and [Xml](#) discussed [below](#).

Changing data

It's possible to change data object properties with assignment operator (except [Link](#), [Lob](#) and [Xml](#) discussed [below](#)):

```
var newInvoice = dataScope.GetRepository<Invoice>().New();
newInvoice.DocNumb = String.Format("Inv-{0}", number.ToString(CultureInfo.InvariantCulture).PadLeft(6, '0'));
newInvoice.Submit();
```

To edit data of new object You can also use [New](#) method overload with initialization logic wrapper:

```

return dataScope.GetRepository<Invoice>().New(
    //property DocNumb is a sting assigned as usual
    z => z.DocNumb = String.Format("Inv-{0}",
        number.ToString(CultureInfo.InvariantCulture).PadLeft(6, '0')),
    //property DocCatalog - link to data object typed Catalog
    //assignment using += operator is the same as DocCatalog.Source = ...
    z => z.DocCatalog +=
        dataScope.GetDictionaryValue<Catalog>(x => x.Code == catalogCode),
    //property DeliveryType is enum DeliveryTypeEnum
    z => z.DeliveryType = DeliveryTypeEnum.PickUp,
    //property DeliveryDate is DateTime
    z => z.DeliveryDate = DateTime.Today.AddDays(1),
    //property Source is Xml, assignment using += operator is the same as
    //Source.Document = ...
    z => z.Source += new XDocument(
        new XElement("invoice",
            new XAttribute("number", number),
            new XAttribute("state", z.DocStateCode))),
    //property Source - is a large binary object (Lob) and assignment using += operator
    //is the same as Scan.Value = ...
    z => z.Scan += Encoding.UTF8.GetBytes(z.Source.Document.ToString
    ).Submit(out _);

```

Or [Modify](#) method extension to change object data:

```

var invoice = dataScope.GetRepository<Invoice>()
    .SingleOrDefault(x => x.DocNumb.Like($"%{number}") && x.DocStateCode == oldStateCode);
if (invoice == null) return true;
return invoice.Modify(
    z => z.DocState += dataScope.GetDictionaryValue<DocState>(x => x.Code == newStateCode),
    z => z.Source.Modify(doc => doc.Element("invoice").Attribute("state").Value = newStateCode))
    .Submit(out _);

```

Delete

XData has ability to mark data object as deleted. Before submit changes this flag can be undone. Marking and rejecting the mark can be done with extension method of interface [IDataObject.SetDeleted](#)

```

invoice.SetDeleted(true); // mark data object for delete
...
invoice.SetDeleted(false); // undone delete mark
...
//... and little more complex example:
// set the delete flag and submit changes with submit success flag returning
return invoice.SetDeleted(true).Submit(out _);

```

Tip

To delete a number of objects at once, it's a handy to use [ICollection<T>.SetDeleted](#) extension method.

Submit changes

To submit changes can be used extension method of interface [IDataObject Submit](#)

```

newInvoice.Submit(out var invoice);
var docTypeCode = invoice.DocTypeCode;

```

To submit changes applied to multiple data objects of the same type can be used [IRepository<T>.Submit](#) method accepted ([IEnumerable<T>](#)) as one of parameters. When object is not changed this method will skip it while updating database.

Mapping declaration

To use the repository, required to describe the object-relational mapping rule for the type of repository objects. XData has two

styles of mapping definition static - based on attribute assignment and dynamic - LINQ style mapping description.

SQL representation of samples below:

```
SELECT A.amount AS A_amount /* Amount */,
       A.DOC_SPEC_ID AS A_DOC_SPEC_ID,
       DS.doc_id AS DS_doc_id /* DocId */,
       DS.doc_spec_id AS DS_doc_spec_id /* DocSpecId */,
       DS.DOC_SPEC_TYPE_ID AS DS_DOC_SPEC_TYPE_ID,
       ST.code AS ST_code /* Code */,
       ST.DOC_SPEC_TYPE_ID AS ST_DOC_SPEC_TYPE_ID,
       ST.name AS ST_name /* Name */
FROM T_DOC_SPEC DS
     JOIN T_DOC_SPEC_AMOUNT A ON A.DOC_SPEC_ID = DS.DOC_SPEC_ID
     JOIN T_DOC_SPEC_TYPE ST ON ST.DOC_SPEC_TYPE_ID = DS.DOC_SPEC_TYPE_ID
```

Static mapping description sample:

```
[DataObject("DS"),
 DataTable("T_DOC_SPEC", "DS"),
 DataTable("T_DOC_SPEC_AMOUNT", "A", Parent = "DS"),
 DataTable("T_DOC_SPEC_TYPE", "ST"),
 Link("A", "doc_spec_id", "DS"),
 Link("ST", "doc_spec_type_id", "DS"),
 Column("DocSpecId", typeof(long), "DS", Flags = DataPropertyFlag.Id),
 ColumnDefault("DocSpecId", DefaultType.AutoIncrement),
 Column("DocSpecTypeId", typeof(long), "ST"),
 Column("DocId", typeof(long), "DS")]
public class InvoiceSpec: IDataObject
{
    [Property("ST"),
     LinkProperty,
     LinkProperty(Property = "DocSpecTypeId"),
     LinkProperty(Property = "Code")]
    public Link<string, DocSpecType> Name { get; set; }
    [Property("ST")]
    public string Code { get { return this.GetProperty(x => x.Code); } }
    [Property("A")]
    public decimal Amount { get; set; }
}
```

Dynamic mapping description sample:

```

public partial class InvoiceSpec: IDataObject
{
    private static Expression<CustomMapping<InvoiceSpec>> _mapping = (
        () => XDataMapping.CustomMapping<InvoiceSpec>()
            .DataTable("T_DOC_SPEC", "DS")
            .DataTable("T_DOC_SPEC_AMOUNT", "A", "DS",
                x => x.Link("DS", "doc_spec_id"))
            .DataTable("T_DOC_SPEC_TYPE", "ST",
                x => x.Link("DS", "doc_spec_type_id"))
            .Column("DocSpecId", x => x.Field<long>("DS", string.Empty,
                z => z.Key(), z => z.Default(DefaultType.AutoIncrement)))
            .Column("DocSpecTypeId",
                x => x.Field<long>("ST", string.Empty))
            .Column("DocId", x => x.Field<long>("DS", string.Empty))
            .Map(x => new InvoiceSpec {
                Name = x.Link<string, DocSpecType>("ST", "name",
                    z => z.LinkProperty<DocSpecType>(y => y.Name),
                    z => z.LinkProperty<DocSpecType>((InvoiceSpec y) => y.Code)),
                Amount = x.Field<decimal>("A", string.Empty)
            }).SetBaseTable("DS"));
}

public partial class InvoiceSpec
{
    public Link<string, DocSpecType> Name { get; set; }
    public string Code { get { return this.GetProperty(x => x.Code); } }
    public decimal Amount { get; set; }
}

```

Detailed description of [static](#) and [dynamic](#) methods of mapping described in relevant sections.

Versions of XData

XData delivered as two specific versions: .Net 4.0 package to use with full featured Windows .Net framework version 4.0 or later, and cross platformed .Net Standard 2.0 package.

The versions has similar API but some different implementations of certain features:

1) Different initialization of XData:

Net 4.0:

```
using (var dataEngine = XDataManager.InitXData(x =>
    x.UseConfiguration(ConfigurationManager
        .OpenExeConfiguration(ConfigurationUserLevel.None))))
{
    using (var dataScope = dataEngine.NewDataScope())
    {
        foreach(var data in dataScope.GetRepository<SomeObject>())
            Console.WriteLine(string.Format("{0}, {1}", data.SomeId, data.Name));
    }
}
```

NetStandard 2.0:

```
//Set configuration file
var builder = new ConfigurationBuilder();
builder.AddXmlFile("MyConsoleApp.config");
var configFile = builder.Build();

//Set services
using(var serviceProvider = new ServiceCollection()
    .AddLogging()
    // Localization only for ASP.NET Core
    .AddSingleton(typeof(IStringLocalizerFactory), x => null)
    .AddSingleton(typeof(IConfigurationRoot), x => configFile)
    .AddXData()
    .BuildServiceProvider())
{
    //Configure logging
    serviceProvider.GetRequiredService<ILoggerFactory>()
        .AddConsole(LogLevel.Warning).AddDebug();

    var dataEngine = serviceProvider.GetRequiredService<IDataEngine>();

    using (var dataScope = dataEngine.NewDataScope())
    {
        foreach(var data in dataScope.GetRepository<SomeObject>())
            Console.WriteLine($"{data.SomeId}, {data.Name}");
    }
}
```

2) Logging in .Net 4.0 version is realized with log plugins implements [ILoggerWriter](#) interface. Logging in .Net Standard 2.0 version is realized using Microsoft.Extensions.Logging service.

3) Configuration of .Net 4.0 version is limited by using app.config/web.config files. Configuration in .Net Standard 2.0 version is realized using Microsoft.Extensions.Configuration service. Configuration structure for both versions described [here](#).

4) Localization of .Net 4.0 version use satellite localization assemblies. Localization in .Net Standard 2.0 version is realized using Microsoft.Extensions.Localization service. Localization procedures for both versions described [here](#).

5) Some database adapters not implemented for .Net Standard 2.0 version because no applicable implementation of .Net data providers for .Net Standard 2.0 found now. We are hope they will be implemented in short time future, and XData can be use that

databases too.

6) Three tier architecture is not implemented for .Net Standard 2.0 version for now. Some limitations of current .Net Core realization makes not possible to implement this feature on same technologies now. And no specific requests to implement it quick as possible.

7) Three tier client plugin required full .Net 4.0 framework only (base functionality is available on .Net 4.0 Client version). Life time limited cache (for [IDataScope](#) and [WorkSet](#)) is required full .Net 4.0 framework. Same caching functionality is implemented in .Net Standard 2.0 version.

Mapping common rules

All data objects must be signed with marker interface (empty interface) [IDataObject](#). All objects describes subqueries must to be signed with marker interface [ISqlObject](#).

Tip

Ability of defining mapping to marker interface is key feature of XData allowed to get flexible and natural objects deriving hierarchy with full featured and handy methods to work with it (using extension methods).

Mapping rules can be declared statically with attributes of class and its properties (see [static mapping](#)), and dynamically using LINQ style expression (see [dynamic mapping](#)).

Mapping inheritance

Complex projects can contain many reusable parts. Including query parts (tables, subqueries, stored procedure calls). Sometimes these reused parts correspond to the inheritance hierarchy, sometimes they don't.

XData supports the inheritance of parts of the mapping of [data objects](#) both within the hierarchy of inheritance of objects reflected to the database and the construction of its hierarchy for the reuse of parts of mapping.

Only mapping description is depends on [static](#) or [dynamic](#) method of declaration.

Common types properties

Common types properties (string, bool, int...) including nullable (int?, bool?...), and enum properties is declared as common class properties. Data access to this properties is usual. Only mapping description is depends on [static](#) or [dynamic](#) method of declaration.

Binary properties

To work with large binary object (BLOB) using lazy coupling XData represent property type [Lob](#). This type has followed...

properties:

- `byte[] Value` - binary object value
- (read-only) `bool Assigned` - check object has value (including empty value) or NULL

methods:

- `GetSize()` - returns size of binary object
- (extension) `Modify(Action<byte[]> action)` - helper method to simplify changing of binary object value

operation:

- `+=` - "syntax sugar", `_data.SomeBlob += _someVariable` identical with `_data.SomeBlob.Value = _someVariable`.

Examples:

```
newInvoice.Scan += _image;
...
if(newInvoice.Scan.Assigned)
    _size = newInvoice.Scan.GetSize();
...
invoice.Scan.Modify(x => x = new byte[0]);
```

Xml properties

To work with Xml using lazy coupling XData represent property type [Xml](#). This type has followed...

properties:

- XDocument *Document* - get or set Xml value
- (read-only) bool *Assigned* - check Xml field for NULL value

methods:

- *Extract*(string path, params KeyValuePair<string, string>[] namespaces) - returning result of XPath expression *path* using *namespaces* (pair: namespace - URL)
- (extension) *Modify*(Action action) - helper method to change Xml value

operations:

- += - "syntax sugar", `_data.SomeXml += _someVariable` identical with `_data.SomeXml.Document = _someVariable`.

Example:

```
newInvoice.Source += new XDocument(new XElement("invoice",
    new XAttribute("number", number), new XAttribute("state", newInvoice.DocStateCode)));
...
dataScope.GetRepository<Invoice>(Owner, context: Context)
    .First(x => x.Source.Extract("/invoice/@state)[1]") == "ACTIVE").DocNumb
...
invoice.Source.Modify(x => x = new XDocument());
```

Link properties

To work with external links to data objects outside current object without operating surrogate keys, XData represent special property type - [Link<TVal,TSrc>](#), where *TVal* - type of property for preview, *TSrc* - type of external object. For type *Link* is defined...

properties:

- TVal *Value* - this property to get preview of linked object
- TSrc *Source* - this property can be used to set linked object
- (read-only) bool *Assigned* - check link value for NULL value

operations:

- += - "syntax sugar", `_data.SomeLink += _someDictionaryObject` identical with `_data.SomeLink.Source = _someDictionaryObject`.

```
newInvoice.DocState += dataScope.GetDictionaryValue<DocState>(x => x.Code == "CREATED");
```

To make link properties work correct, at mapping level need to be defined pairs of properties source and current objects, that values will be copied from source object to current when link is applied. Link description is deferent for [static](#) and [dynamic](#) methods of mapping.

Read only properties

Read only properties XData require to perform as in example below:

```
public string DocStateCode { get { return this.GetProperty(x => x.DocStateCode); } }
```

Data access is provided through extension method [GetProperty](#), but property is read only.

Child repositories references

Detail related repositories XData require to perform as:

```
public IRepository<InvoiceSpec> Spec {
    get {
        return this.GetRepository().GetChild<InvoiceSpec>();
    }
}
```

No additional mapping description are needed. Only [external reference](#) is required.

Tip

To use [hierarchical objects](#) and XData Unit of Work realization ([WorkSet](#)), child repositories references are required.

Hidden properties

Some columns in database is needed to full description of mapping, but not required to business logic. This columns is declared as part of mapping named *Column* is selected from database, but not mapped on data object properties.

For example - XData can operate primary (PK) and foreign (FK) keys of business objects and work with master-detail references between them, but mapping to property of [external references](#) is not required to operate with key.

Second case - optimistic concurrency resolving (see [concurrency resolving](#)) required concurrency token field, but this field is not required to business logic.

Third example - when data is grouped, needed to declare some field to link with [external reference](#), but in this case possibly we can't include this field in SELECT expression because of GROUP BY syntax. This case is resolved by [Hidden](#) property of column declaration. When *Hidden* is true - column will be not included into SELECT expression of resulted SQL query.

Hidden properties description is various to [static](#) and [dynamic](#) mapping declaration methods.

Master detail relations

External references is the base element of master-detail references definition in XData. It describes link rules between related detail entities from master entities. XData support multi master and multi detail relation (many-to-many). And it's possible to define multiple relations between every two data objects.

External references is the declarations of *possible* master-detail relations between [data objects](#). Each of external reference is describe master property (or [column](#)) to detail filter relation, used to filter detail collections data when master [current](#) object is set.

Tip

External references applied to [manually attached](#) detail data sources only.

External references definition is differs for [static](#) and [dynamic](#) methods of mapping.

Reference rules is defined at mapping level, and when detail object will be attached using [GetChild](#) method, that rules will be applied.

To fill child repository data in some unmapped array property (when the reference has type "one-to-many") can be used special overload of [GetChild](#) method. And when reference has type "one-to-one" can be used [GetChild](#) overload. That properties will be filled by child data automatically during data acquire when child repositories has been connected using that overloads of method.

```
var rep = dataScope.GetRepository<Invoice>();
var child = rep.GetChild(z => z.InvoiceSpec);
```

Warning

Data hierarchy can be acquired by that way has not limited deep level, but important keep in mind the price of that is performance. Use child repository data loading into unmapped properties when needed only!

To break master-detail reference call [DetachChild](#) method. When [data scope](#) or one of referenced [repositories](#) will be disposed -

reference is beaked automatically.

Detail object reference filter value is set when [SetCurrent](#) extension method is called for master object.

```
if (newInvoice.Submit())
{
    newInvoice.SetCurrent();
    return true;
}
```

Master-detail linked repositories can be used for cascade delete operation over multiple related data objects:

```
newInvoice.SetDeleted(true);
newInvoice.Submit();
```

or cascade delete applied for filtered subset of repository objects and their detail objects:

```
invoices.Clear(x => x.DocStateCode == "REJECTED");
```

Tree

XData can use CTE (common table expression) as part of [repository](#) data source definition. Supported as plain (WITH), as recursive (WITH RECURSIVE) expressions.

CTE is defined in mapping differently for [static](#) and [dynamic](#) mapping paradigm, but it can be used with similar way: defined one or more tables with unique aliases and name equals CTE name.

Recursive CTE can be used as independent data source too. In this case CTE name must to be declared as base table of repository.

SQL functions & procedures

Stored procedures and functions can be used as data source of mapping (if used DBMS and ADO .Net provider is support this).

❗ Important

At this moment we has various limitations of using SQL procedures as on DBMS SQL dialect level, as on ADO .Net provider realization level. Please, choose DBMS and ADO .Net provider with keep in mind limitations they have. XData support many procedure mapping styles and call conventions, but can not garant all of them is supported by DBMS and ADO .Net provider You choose.

XData has support followed SQL procedure mapping styles and call conventions and it combinations:

- Using procedure result set as data source
- Using procedure with multiple result sets
- Using out parameters
- Using return value of stored procedure
- Using simple type array as parameter
- Using cursor as parameter
- Using UTD (user defined type) as parameter
- Using table valued function as data source
- Using scalar function as single property source

❗ Warning

Class mapped to UDT must support native XML serialization!

All this mapping features described as for [static](#) as for [dynamic](#) mapping paradigm.

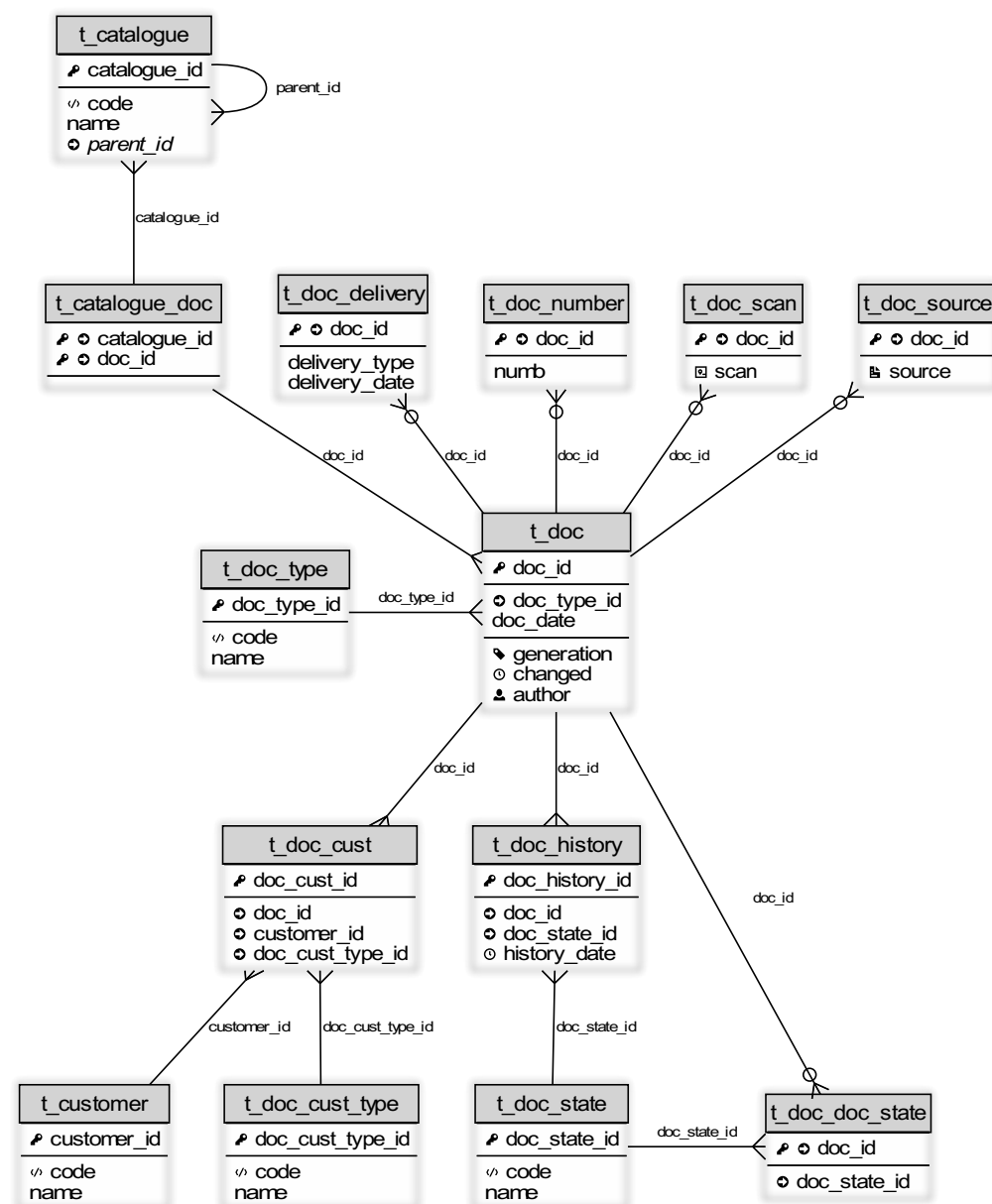
Static mapping

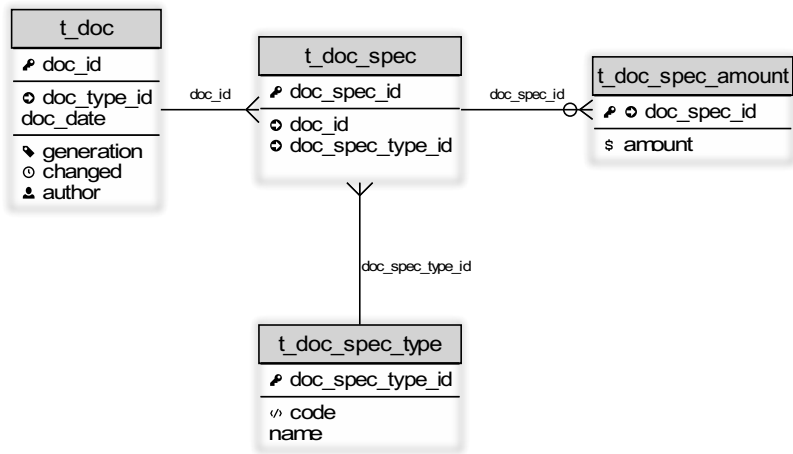
Tip

Before read this topic be sure to already readed [Glossary](#) and [Common mapping rules](#) contains base information to better understanding this.

Static mapping is defined using attributes of mapped class and it properties.

Static mapping sample





Here is example of data object static mapping. Of course, real mapping is much simpler then this, but as example of XData capabilities it's really representative...

SQL representation of following sample:

```

with recursive T (catalogue_id, code, name, parent_id)
as ((SELECT R.catalogue_id AS R_catalogue_id,
        R.code AS R_code,
        R.name AS R_name,
        R.parent_id AS R_parent_id
FROM T_CATALOGUE R
WHERE ((@FilterByRoot is not null
and R.catalogue_id = @FilterByRoot)
OR (@FilterByRoot is null
and R.parent_id is null))) UNION (SELECT S.catalogue_id AS S_catalogue_id,
        S.code AS S_code,
        S.name AS S_name,
        S.parent_id AS S_parent_id
FROM T_CATALOGUE S, T T
WHERE S.parent_id = (T.catalogue_id)))
SELECT CF.CUSTOMER_ID AS CF_CUSTOMER_ID,
        CF.DOC_CUST_ID AS CF_DOC_CUST_ID,
        CF.DOC_CUST_TYPE_ID AS CF_DOC_CUST_TYPE_ID,
        CF.DOC_ID AS CF_DOC_ID,
        CT.CUSTOMER_ID AS CT_CUSTOMER_ID,
        CT.DOC_CUST_ID AS CT_DOC_CUST_ID,
        CT.DOC_CUST_TYPE_ID AS CT_DOC_CUST_TYPE_ID,
        CT.DOC_ID AS CT_DOC_ID,
        CAST((SELECT Sum(A.amount) AS A_amount
FROM T_DOC_SPEC S
JOIN T_DOC_SPEC_AMOUNT A ON A.DOC_SPEC_ID = S.DOC_SPEC_ID
WHERE S.doc_id = S.doc_id) AS NUMERIC(17, 5)) AS doc_amount /* DocAmount */,
        D.doc_date AS D_doc_date /* DocDate */,
        D.doc_id AS D_doc_id /* DocId */,
        D.DOC_TYPE_ID AS D_DOC_TYPE_ID,
        D.generation AS D_generation /* Generation */,
        D.changed AS D_changed /* Changed */,
        D.author AS D_author /* Author */,
        DD.delivery_date AS DD_delivery_date /* DeliveryDate */,
        DD.delivery_type AS DD_delivery_type /* DeliveryType */,
        DD.DOC_ID AS DD_DOC_ID,
        DS.DOC_DOC_STATE_ID AS DS_DOC_DOC_STATE_ID,
        DS.DOC_ID AS DS_DOC_ID,
        DS.DOC_STATE_ID AS DS_DOC_STATE_ID,
        F.CODE AS F_CODE,
        F.CUSTOMER_ID AS F_CUSTOMER_ID,
        F.name AS F_name /* CustomerFrom */,
        H.H_doc_id AS H_H_doc_id,
        H.H_history_date AS H_H_history_date /* DocLastChange */,

```

```

N.DOC_ID AS N_DOC_ID,
N.numb AS N_numb /* DocNumb */,
S.code AS S_code /* DocStateCode */,
S.DOC_STATE_ID AS S_DOC_STATE_ID,
S.name AS S_name /* DocState */,
SC.DOC_ID AS SC_DOC_ID,
SC.scan AS SC_scan /* Scan */,
SR.DOC_ID AS SR_DOC_ID,
SR.source AS SR_source /* Source */,
ST.S_doc_id AS ST_S_doc_id,
T.CODE AS T_CODE,
T.CUSTOMER_ID AS T_CUSTOMER_ID,
T.name AS T_name /* CustomerTo */
FROM T_DOC D
JOIN T_DOC_DOC_STATE DS ON DS.doc_id = D.doc_id
JOIN T_DOC_STATE S ON S.DOC_STATE_ID = DS.DOC_STATE_ID
JOIN T_DOC_NUMBER N ON N.doc_id = D.doc_id
JOIN T_DOC_CUST CF ON CF.doc_id = D.doc_id
JOIN T_DOC_CATALOGUE DC ON DC.doc_id = D.doc_id
JOIN T_CATALOGUE U ON DC.catalogue_id = U.catalogue_id
JOIN T_CUSTOMER F ON F.CUSTOMER_ID = CF.CUSTOMER_ID
JOIN T_DOC_CUST CT ON CT.doc_id = D.doc_id
JOIN T_CUSTOMER T ON T.CUSTOMER_ID = CF.CUSTOMER_ID
LEFT OUTER JOIN T_DOC_SCAN SC ON SC.doc_id = D.doc_id
LEFT OUTER JOIN T_DOC_SOURCE SR ON SR.doc_id = D.doc_id
LEFT OUTER JOIN T_DOC_DELIVERY DD ON DD.doc_id = D.doc_id
LEFT OUTER JOIN (SELECT H.doc_id AS H_doc_id,
H.history_date AS H_history_date
FROM T_DOC_HISTORY H) H ON D.doc_id = H.H_doc_id
JOIN (SELECT S.doc_id AS S_doc_id
FROM T_DOC_SPEC_TYPE ST
JOIN T_DOC_SPEC S ON S.DOC_SPEC_TYPE_ID = ST.DOC_SPEC_TYPE_ID
WHERE ST.code = @FilterBySpecTypeCode) ST ON D.doc_id = ST.S_doc_id
WHERE D.DOC_TYPE_ID = (select doc_type_id
from T_DOC_TYPE
where code='INVOICE')
AND CF.DOC_CUST_TYPE_ID = (select doc_cust_type_id
from T_DOC_CUST_TYPE
where code='FROM')
AND CF.DOC_CUST_TYPE_ID = (select doc_cust_type_id
from T_DOC_CUST_TYPE
where code='TO')
AND D.doc_id = @FilterByDocId
AND DC.catalogue_id in (SELECT T.catalogue_id AS T_catalogue_id FROM T T)

```

Mapping sample:

```

[DataObject("D", LogicAssemblyName = "XDataObjectTest"),
    DataTable("T_DOC", "D"),
    DictionaryFilter("T_DOC_TYPE", "doc_type_id", "code", "INVOICE", "D"),
    DataTable("T_DOC_DOC_STATE", "DS", Parent = "D"),
    Link("D", "doc_id", "DS"),
    DataTable("T_DOC_STATE", "S"),
    Link("DS", "doc_state_id", "S"),
    DataTable("T_DOC_NUMBER", "N", Parent = "D"),
    Link("D", "doc_id", "N"),
    DataTable("T_DOC_CUST", "CF", Parent = "D"),
    Link("D", "doc_id", "CF"),
    DataTable("T_CUSTOMER", "F"),
    Link("CF", "customer_id", "F"),
    DictionaryFilter("T_DOC_CUST_TYPE", "doc_cust_type_id", "code", "FROM", "CF"),
    DataTable("T_DOC_CUST", "CT", Parent = "D"),
    Link("D", "doc_id", "CT"),
    DataTable("T_CUSTOMER", "T"),
    Link("CT", "customer_id", "T"),

```

```

DictionaryFilter("T_DOC_CUST_TYPE", "doc_cust_type_id", "code", "TO", "CT"),
DataTable("T_DOC_SCAN", "SC", Parent = "D"),
Link("D", "doc_id", "SC", Operation = FilterOperation.OuterJoin),
DataTable("T_DOC_SOURCE", "SR", Parent = "D"),
Link("D", "doc_id", "SR", Operation = FilterOperation.OuterJoin),
DataTable("T_DOC_DELIVERY", "DD", Parent = "D"),
Link("D", "doc_id", "DD", Operation = FilterOperation.OuterJoin),
DataTable("T_DOC_CATALOGUE", "DC", Parent = "D"),
Link("D", "doc_id", "DC"),
DataTable("T_CATALOGUE", "U"),
Link("U", "catalogue_id", "DC"),
Subquery("A", typeof(DocSpecAmounts), "Amount", Grouping = DataGrouping.Sum),
SubqueryLink("A", "DocId"),
InnerView("H", typeof(DocLastChange)),
SubqueryLink("H", "DocId", Operation = FilterOperation.OuterJoin),
ExternalLink(typeof(InvoiceSpec), "DocId"),
InnerView("ST", typeof(DocBySpecType)),
SubqueryLink("ST", "DocId"),
Subquery("UT", typeof(CatalogueTree), "CatalogueId"),
SubqueryFilter("UT", "DC", "catalogue_id", Operation = FilterOperation.In),
Column("DocId", typeof(long?), "D", Flags = DataPropertyFlag.Id),
Column("DocStateId", typeof(long), "S"),
Column("FromCustomerId", typeof(long), "F"),
Column("ToCustomerId", typeof(long), "T"),
Column("CatalogueId", typeof(long), "U"),
ColumnDefault("DocId", DefaultType.AutoIncrement)]
public class Invoice: IDataObject
{
    [Property("S", "code")]
    public string DocStateCode { get { return this.GetProperty(x => x.DocStateCode); } }
    [Property("S", "name"),
        LinkProperty("Name"),
        LinkProperty(Property = "DocStateId"),
        LinkProperty("Code", "DocStateCode")]
    public Link<string, DocState> DocState { get; set; }
    [Property("U", "name"),
        LinkProperty(Property = "CatalogueId"),
        LinkProperty("Name")]
    public Link<string, Catalogue> DocCatalog { get; set; }
    [Property("N", "numb")]
    public string DocNumb { get; set; }
    [Property("D"),
        PropertyDefault(DefaultType.CurrentDate)]
    public DateTime? DocDate { get; set; }
    [Property("D", Flags = DataPropertyFlag.ConcurrencyToken),
        PropertyDefault(DefaultType.AutoIncrement)]
    public long Generation { get { return this.GetProperty(x => x.Generation); } }
    [Property("D"),
        PropertyDefault(DefaultType.CurrentDateTime, DefaultFeature = DefaultFeature.UseOnUpdate)]
    public DateTime Changed { get { return this.GetProperty(x => x.Changed); } }
    [Property("D"),
        PropertyDefault(DefaultType.UserName, DefaultFeature = DefaultFeature.UseOnUpdate)]
    public string Author { get { return this.GetProperty(x => x.Author); } }
    [Property,
        PropertyExpression("A", DataExpressionType.SubQuery, DbType.Decimal,
            ExprSize = 17, ExprScale = 5)]
    public decimal? DocAmount { get { return this.GetProperty(x => x.DocAmount); } }
    [Reference("H", "HistoryDate")]
    public DateTime? DocLastChange {
        get { return this.GetProperty(x => x.DocLastChange); }
    }
    [Property("SC", Flags = DataPropertyFlag.OuterFlag)]
    public Lob Scan { get; set; }
    [Property("SR", Flags = DataPropertyFlag.OuterFlag)]
    public Xml Source { get; set; }
    [Property("F", "name")]

```

```

    [Property( F , name ),
      LinkProperty("CustomerId", "FromCustomerId"),
      LinkProperty("Name")]
    public Link<string, Customer> CustomerFrom { get; set; }
    [Property("T", "name"),
      LinkProperty("CustomerId", "ToCustomerId"),
      LinkProperty("Name")]
    public Link<string, Customer> CustomerTo { get; set; }
    [Property("DD")]
    public DeliveryTypeEnum DeliveryType { get; set; }
    [Property("DD"),
      PropertyDefault(DefaultType.CurrentDate)]
    public DateTime? DeliveryDate { get; set; }
    public IRepository<InvoiceSpec> Spec {
      get { return this.GetRepository().GetChild<InvoiceSpec>(); }
    }
  }
}

```

Mapping parameters

All statically mapped classes must be marked with [DataObjectAttribute](#). Parameters of *DataObjectAttribute*: *BaseTable* - alias of [base table](#) (optional, default is [virtual table](#) with empty alias), *Context* - database [context](#) (optional, context will be specified at runtime), and *Flags* - [data source flags](#) (optional, default is **None**).

Example:

```

[DataObject("D" /* base table alias */,
  Context: "Main", Flags: DataObjectFlags.ReadOnly)]

```

The *DataObjectAttribute* attribute can also be used to inherit the mapping description of the data object. To do this, an alternative [constructor](#) is used that allows you to limit *BaseMappingType* - the base object in the inheritance hierarchy of the mapping description. The *BaseMappingType* class must implement the [ISqlObject](#) interface but does not have to be associated with the current class. The depth of the inheritance hierarchy of the mapping description is not limited.

```

[DataObject(typeof(ParentDescription))]

```

⚠ Warning

When inheriting mapping descriptions, a mixture of the static and dynamic description methods within the a single mapping inheritance hierarchy is not allowed!

Tables

All tables in [repository](#) query must be specified using [DataTableAttribute](#). Parameters of *DataTableAttribute*: *Name* - name of table and *Alias* - table alias in query. Tables in [updatable tables hierarchy](#), but not [base table](#) must to specify *Parent* parameter of this attribute - alias of parent table in updatable tables hierarchy.

```

[DataTable("P" /* table alias */,
  "patient" /* table name */,
  "D" /* parent table alias */)]

```

Same way we can define mapping to view, but all fields of view must to be explicitly defined as [read-only properties](#) or [columns](#).

Subqueries

XData allows use subqueries as part of mapping. This feature can be used for get property values from subquery, for [subquery filters](#), and inner view definitions.

Subquery recursive deep is not limited.

Class used subquery in mapping definition must to marked with attribute [SubqueryAttribute](#) or [InnerViewAttribute](#) (when it used

as inner view definition). Both of them has a parameter *Alias* - subquery alias in resulting query. *InnerViewAttribute* additionally has parameter *InnerViewType* - type mapped to subquery. *SubqueryAttribute* additionally has parameters: *SubqueryType* - type mapped to subquery, *PropertyName* - subquery returned property and *Grouping* - aggregation type of resulting property.

```
[Subquery("A" /* subquery alias */,
    typeof(DocSpecAmounts) /* subquery mapped type */,
    "Amount" /* subquery result property name */,
    Grouping = DataGrouping.Sum /* result property aggregation type */)
...
[InnerView("H" /* inner view alias */,
    typeof(DocLastChange) /* inner view mapped type *)]
```

Filters

Filters can be defined as:

- Constant filters
- SQL expression filters
- Dictionary filters
- Range filters
- Subquery filters
- Inner links

Each of filter type is defined by own special attribute of data object class. Each of attributes has parameters: *Source* - data source alias of filtered field, *FieldName* - filtered field name, *Operation* - filter operation (optional, default is *FilterOperation.Equal*), *Combination* - [filter group](#) name (optional, default is root filter group). Above this every filter attribute has their own specific parameters described below.

Filter operation is defined by enum [FilterOperation](#):

- **Equal** - equal
- **NotEqual** - not equal
- **LessThan** - less then
- **GreaterThan** - greater then
- **LessThanOrEqual** - less or equal
- **GreaterThanOrEqual** - greater or equal
- **In** - field value in range or subquery
- **NotIn** - field value is not in range or subquery
- **Exists** - subquery returns one or more rows
- **NotExists** - subquery returns no rows
- **Like** - field value correlated with find expression
- **NotLike** - field value not correlated with find expression
- **OuterJoin** - left outer join link between query sources (tables or inner views)
- **Contains** - field value correlated with full-text search
- **NotContains** - field value not correlated with full-text search

Filter group

Filter group can be defined by [FilterCombinationAttribute](#) with parameters: *Name* - filter group unique name, *Combination* - logical operation between filters and subgroups inside group (optional, default is *FilterCombination.And*), *Parent* - parent filter group name (optional, can be omitted for filter groups with parent - root filter group).

Logic operation inside filter group is defined by enum [Combination](#):

- **And** - and
- **Or** - or

```
[FilterCombination("OrGroup" /* group filter name */,
    Combination: Combination.OR /* logical operation */)]]
```

Constant filters

Constant filters can restrict query by constant value. This filter is defined by [ConstantFilterAttribute](#), with specific (over described [above](#)) parameters: *Name* - filter name (will be transformed to query parameter name), *ConstantType* - constant type (see below) and *ConstantValue* - constant value depends on *ConstantType*.

Constant types is defined by enum [FilterConstantType](#):

- **Const** - value of *ConstantValue*.
- **Variable** - value of [variable](#) with name equals *ConstantValue*.
- **CurrentDate** - current date.
- **CurrentDateTime** - current date/time.
- **CurrentDateTimeUTC** - current date/time (UTC).

```
[ConstantFilter("FilterByZero" /* filter name */,
    "P" /* table alias */,
    "discount" /* DB field name */,
    0m /* decimal constant value */,
    Operation: FilterOperation.Equal /* filter operation */,
    ConstantType: FilterConstantType.Const /* constant type */)]
```

SQL expression filters

To define filter over SQL expression used [ExpressionFilterAttribute](#) with specific (over described [above](#)) parameter: *ExpressionText* - depends on value can be:

- subquery alias,

```
[ExpressionFilter("P" /* table alias */,
    "discount" /* field name */,
    "D" /* ExpressionText: subquery alias */,
    Operation: FilterOperation.In /* filter operation */)]
```

- private static field name contains dynamic definition of SQL expression,

```
[ExpressionFilter("P" /* table alias */,
    "discount" /* field name */,
    "AllowedDiscount" /* ExpressionText: SQL expression description field name */,
    Operation: FilterOperation.NotEqual /* filter operation */)]]
...
//Private static field of mapped class marked with SqlExpression attribute
[SqlExpression]
private static Calculate<int> AllowedDiscount = x => x.Case<Product, int>(<
    z => z.Field<bool>("is_vip"), z => 0, 1.SetExpression(z => 10));
```

- plain SQL expression text.

```
[ExpressionFilter("P" /* table alias */,
    "discount" /* field name */,
    "case P.is_vip when 1 then 10 else 0 end" /* ExpressionText: plain SQL expression */,
    Operation: FilterOperation.NotEqual /* filter operation */)]
```

Important

If possible do not use plain SQL expressions. It possibly use specific SQL dialect features not supported by other DBMS or providers.

Range filters

To define filter checks field value over array of values used [RangeFilterAttribute](#) with specific (over described [above](#)) parameter: *Range* - array of values to filter on.

```
[RangeFilter("P" /* table alias */,
    "discount" /* field name */,
    new[] {0,5,10} /* range of values */,
    Operation: FilterOperation.In /* filter operation */)]
```

Dictionary filters

Dictionaries surrogate keys may varied over one instance to another, and names can be changed by users. Sometimes business logic need to operate with the concrete value of dictionary. Because of this it's recommended to add into dictionary DB structure a special unique text field - mnemonic code.

XData supports using mnemonic code when define filters on dictionary value - dictionary filters. They defined by [DictionaryFilterAttribute](#) with specific (over described [above](#)) parameters: *DictionaryTable* - dictionary table name, *DictionaryId* - dictionary id field name, *DictionaryCode* - dictionary mnemonic code field name, *DictionaryValue* - mnemonic code value to filter query on, *ValuesVariable* - flag *DictionaryValue* contains not a value but name of [variable](#) with value.

Tip

In case filter field name is equals dictionary id field name, parameter *FieldName* can be omitted.

Important

This filter can be used when dictionary has simple (one field only) identity field and mnemonic code field. But it's a very common case.

```
[DictionaryFilter("t_doc_state" /* dictionary table name */,
    "doc_state_id" /* dictionary id field name */,
    "code" /* dictionary mnemonic code field name */,
    "CREATED" /* mnemonic code value */,
    "D" /* filtered table alias */)]
```

Inner links

To define links between data sources inside query used [LinkAttribute](#) with specific parameters: *LinkedSourceAlias* - linked table alias, *LinkedFieldName* - linked table field name (optional, can be omitted when equals with *FieldName*).

Tip

Inner links can be used to define [optional filters](#). To specify role in optional filter chain need to set parameter *PrimaryFilter* to *true*.

```
[Link("D" /* table alias */,
    "doc_id" /* field name */,
    "DD" /* linked table alias */)]
```

Subquery filters

To define filters checks field value is correspond subquery results used [SubqueryFilterAttribute](#) with specific (over defined [above](#)) parameter: *Subquery* - subquery alias (see [Subqueries](#)). Inner view can not be used with subquery filters.

```
[SubqueryFilter("D" /* table alias */,
    "doc_state_id" /* field name */,
    "A" /* subquery alias */,
    Operation: FilterOperation.In /* filter operation */)]
```

When flag *PrimaryFilter* is set to *true* optional part of query is defined by followed rules: when one of linked data sources is skipped then other one will skipped too. When [base table](#) of subquery is skipped then subquery is skipped completely.

Subquery links

Links with subqueries is used to link not table fields but properties of base class and subquery mapped class including [hidden properties](#). To link with subquery or inner view used [SubqueryLinkAttribute](#) with parameters: *Subquery* - subquery alias, *SubqueryProperty* - linked property from subquery, *PropertyName* - optional, linked property from base class (when omit, base class property name is equal *SubqueryProperty*), also optional properties: *Operation*, *Combination* and *PrimaryFilter* is supported. They was described above in topics [Filters](#) and [Inner links](#).

```
[SubqueryLink("H" /* subquery alias */,
    "DocId" /* subquery property to link */)]
```

When flag *PrimaryFilter* is set to *true* optional part of query is defined by followed rules: when subquery is skipped, then data source linked to subquery by this filter will be skipped too, and skip data sources analysis will continue inside base class (see [Inner links](#)).

Properties

Static mapping is performed as each property with specific attributes mark. Main attribute to statically map property is [PropertyAttribute](#) with parameters: *Source* - data source alias (may be omitted for [virtual table](#)), *FieldName* - DB field name mapped to property (may be omitted when field and property correspond naming rule, for example DB field named as **some_field_name** and mapped to **SomeFieldName** property), *Flags* - optional, mask of [property flags](#), *NativeSqlType* - optional, SQL type name in DB. Parameter *NativeSqlType* is used when type default mapping is conflicted with real SQL type name.

```
[Property("S" /* table alias */, "code" /* field name */)]
```

SQL expression property

[Read only property](#) can be mapped to SQL expression. Static mapping of SQL expression is defined by [PropertyExpressionAttribute](#) with parameters: *ExprText* - depends of *ExprType* value (see below) it can be: subquery alias / SQL expression / private static field name define SQL expression in LINQ style, *ExprType* - optional, [SQL expression type](#), default is *DataExpressionType.PlainSql*, *DbType* - ADO ,Net provider data type (optional, default is *DbType.String*), *ExprSize* - optional, expression field size (if applicable), *ExprScale* - optional, expression field precision (if applicable). Possible to use SQL expressions of three types:

- subquery,

```
[PropertyExpression("A",
    DataExpressionType.SubQuery,
    DbType.Decimal,
    ExprSize = 17,
    ExprScale = 5)]
```

- LINQ expression,

```
[PropertyExpression("AllowedDiscount",
    DataExpressionType.LinqExpression,
    DbType.Decimal,
    ExprSize = 17,
    ExprScale = 5)]
...
//Private static field of mapped class marked with SqlExpression attribute
[SqlExpression]
private static Calculate<int> AllowedDiscount = x => x.Case<Product, int>(
    z => z.Field<bool>("is_vip"), z => 0, 1.SetExpression(z => 10));
```

- plain text SQL expression.

```
[PropertyExpression("case P.is_vip when 1 then 10 else 0 end",  
    DataExpressionType.PlainSql,  
    DbType.Decimal,  
    ExprSize = 17,  
    ExprScale = 5)]
```

Tip

Attribute *PropertyExpressionAttribute* is not replaced *PropertyAttribute* but extend it.

Important

If possible do not use plain SQL expressions. It possibly use specific SQL dialect features not supported by other DBMS or providers.

Tip

It's recommended to use [virtual table](#) as SQL expression source omitting data source alias in *PropertyAttribute*.

Property default value

To set default value of property with static mapping, use [PropertyDefaultAttribute](#) with parameters: *DefaultSource* - [default value type](#), *DefaultValue* optional, depends on *DefaultSource*, *DefaultFeature* - optional, default is **DefaultFeature.UseOnInsert**, default value assignment will be applied with some extended [features](#).

```
[PropertyDefault(DefaultType.UserName, AlwaysUseDefault = true)]
```

Property grouping parameters

Property grouping parameters used when data object mapped to grouped query to specify property role in GROUP BY expression. [PropertyGroupingAttribute](#) has properties: *Grouping* - optional, property [aggregation type](#), default is **None**, *GroupOrder* - optional, property grouping order when *Grouping* = *DataGrouping.None*.

```
[PropertyGrouping(DataGrouping.Sum)]
```

Hidden properties

[Hidden properties](#) statically defined using couple of specific attributes: [ColumnAttribute](#), [ColumnExpressionAttribute](#) and [ColumnDefaultAttribute](#) as analog with similar attributes of [properties](#).

[ColumnAttribute](#) has parameters: *PropertyName* - name of hidden property, *PropertyType* - type of hidden property, *Source* - data source alias, *FieldName* - optional, DB field name (can be omitted when correspond default mapping rule (example: **some_field_name** is mapped to **SomeFieldName**)), *Flags* - optional, bit mask of [property flags](#), *Grouping* - optional, defined [aggregation type](#), default is **None**, *GroupOrder* - optional, group order when *Grouping* = *DataGrouping.None*, *Hidden* - optional, mapped field omitted in SELECT expression of result query (when grouping limitations demand this), default is **false**, *NativeSqlType* - optional, native SQL type name. *NativeSqlType* is used when default data type mapping is conflicted with result set data type.

[ColumnExpressionAttribute](#) has parameters: *PropertyName* - name of hidden property (same as defined by *ColumnAttribute*), *ExprText* - depends on *ExprType* (see below): subquery alias / SQL expression text / field name with LINQ style expression definition, *ExprType* - optional, [SQL expression type](#) default is *DataExpressionType.PlainSql*, *DbType* - data type on ADO .Net provider level (optional, default is *DbType.String*), *ExprSize* - optional, expression field size (if applicable), *ExprScale* - optional, expression field precision (if applicable).

[ColumnDefaultAttribute](#) has parameters: *PropertyName* - hidden property name (same as defined by *ColumnAttribute*), *DefaultSource* - [default value type](#), *DefaultValue* optional, depends on *DefaultSource*, *DefaultFeature* - optional, default is **DefaultFeature.UseOnInsert**, default value assignment will be applied with some extended [features](#).

```
[Column("DocId", typeof(long?), "D", Flags = DataPropertyFlag.Id),  
    ColumnDefault("DocId", DefaultType.AutoIncrement)]
```

Inner view reference

To map property to field from inner view a [ReferenceAttribute](#) is used with parameters: *Source* - inner view alias, *PropertyName* - optional, DB field name (may be omitted when property name is equals with inner view mapped class property), *Flags* - optional, mask of [property flags](#), *NativeSqlType* - optional, SQL type name in DB. Parameter *NativeSqlType* is used when type default mapping is conflicted with real SQL type name.

```
[Reference("H" /* inner view alias */,
    "HistoryDate" /* property name inside inner view */)]
```

Links

To statically define [link property](#) pairs used [LinkPropertyAttribute](#) with parameters: *DictSource* - optional, source object property name, *Property* - optional, property name. When one or both of parameters is omitted, followed rules are used:

- When omit both properties - used property name of link property as target and equal named property from source object
- When *DictSource* is omitted - used *Property* as target property name and equal named property from source object
- When *Property* is omitted - used property name of link property as target and property with name equals *DictSource* from source object

```
[Property("S", "name"),
    LinkProperty("Name") /* DocState.Name -> this.DocState */,
    LinkProperty(Property = "DocStateId") /* column DocState.DocStateId -> column this.DocStateId */,
    LinkProperty("Code", "DocStateCode") /* DocState.Code -> this.DocStateCode */]
public Link<string, DocState> DocState { get; set; }
```

Tree

To statically map tree organized data structure used special data source attributes [With](#) to describe tree root query and [WithRecursive](#) to describe tree childs. Both of them use subquery mappings (see example below) to set subquery structure for tree query construction.

❗ Important

[Data source flags](#) of tree organized data sources must contain `DataStructureFlag.ReadOnly` and `DataStructureFlag.Tree!`

With attribute has properties: *Alias* - alias of tree data source, *SubqueryType* - type of root subquery mapping, *Type* - data source combination within tree query, *Properties* - tree properties order. *WithRecursive* attribute has properties: *Alias* - alias of recursive data source, *SubqueryType* - type of root subquery mapping, *InitialAlias* - alias of tree data source.

```

[DataObject("CAT", Flags = DataStructureFlag.ReadOnly | DataStructureFlag.Tree),
  With("CAT", typeof(CatalogueTreeRoot), WithRecursiveType.RecursiveUnion,
    "CatalogueId", "Name", "Code", "ParentId"),
  WithRecursive("S", typeof(CatalogueTreeFolders), "CAT"),
  SubqueryLink("S", "ParentId", "CatalogueId")]
public class CatalogueTree : IDataObject
{
  public static readonly RuntimeFilter FilterByRoot
    = new RuntimeFilter(null, "FilterByRoot");
  public static readonly RuntimeFilter FilterByCathalogue
    = new RuntimeFilter(null, "FilterByCathalogue");

  [Property("T", Flags = DataPropertyFlag.Id)]
  public long CatalogueId { get { return this.GetProperty(x => x.CatalogueId); } }

  [Property("T")]
  public string Code { get { return this.GetProperty(x => x.Code); } }

  [Property("T")]
  public string Name { get { return this.GetProperty(x => x.Name); } }

  [Property("T")]
  public long? ParentId { get { return this.GetProperty(x => x.ParentId); } }
}

[DataObject("R"),
  DataTable("T_CATALOGUE", "R"),
  Column("CatalogueId", typeof(long), "R"),
  Column("Code", typeof(string), "R"),
  Column("Name", typeof(string), "R"),
  Column("ParentId", typeof(long?), "R"),
  FilterCombination("root", Combination = Combination.Or),
  ConstantFilter("FilterByCathalogue", "R", "catalogue_id", null,
    Combination = "root", Nullable = FilterNullable.NullsNotAllowed),
  ConstantFilter("FilterByRoot", "R", "parent_id", null,
    Combination = "root", Nullable = FilterNullable.NullsCompared)]
public class CatalogueTreeRoot : ISqlObject {}

[DataObject("S"),
  DataTable("T_CATALOGUE", "S"),
  Column("CatalogueId", typeof(long), "S"),
  Column("Code", typeof(string), "S"),
  Column("Name", typeof(string), "S"),
  Column("ParentId", typeof(long?), "S")]
public class CatalogueTreeFolders : ISqlObject {}

```

External link

[External link](#) can be statically defined using [ExternalLinkAttribute](#) with parameters: *ChildType* - detail [repository](#) mapped type, *ChildProperty* - link property in detail repository, *Property* - optional, link property in master repository, default used same name as *ChildProperty*, *FilterName* - optional, detail repository filter name, default used **"FilterBy"** + *ChildProperty*, *MasterRefresh* - optional, refresh master repository current row data on submit detail changes flag (useful when master has calculated properties over detail repository data), default is **false**, *DirectLink* - optional, see [tree master-detail specifics](#) for details, default is **false**, *Operation* - optional, link filter operation (see [Filters](#)), default *FilterOperation.Equal*, *Nullable* - optional, [filter null processing behavior](#), default is *FilterNullable.Nullable*, *Combination* - optional, [filter group](#) name in detail repository. External links can be used in optional filters description (see [Optional filters and optional subqueries](#)). To specify external link is optional filter part You can set primary filter flag *PrimaryFilter*.

```
[ExternalLink(typeof(InvoiceSpec), "DocId")]
```

Variables

The use of variables is described [here](#).

Here we describe a way of setting variable values for reused mapping *Subquery* (and *InnerView*) mapping. To do this, use the [SubqueryVariableAttribute](#) attribute. Parameter *Alias* - subquery alias in resulting query. *Name* - the name of the variable. *Value* - the value of the variable.

```
[SubqueryVariable("A" /* subquery alias */,  
    "TestVar" /* variable name */,  
    1 /* variable value */)]
```

SQL procedures & functions

Common information about mapping to SQL procedures and functions has described in [common mapping rules description](#) and here we just represent static mapping rules for various procedure types.

⚠ Warning

SQLite not supported SQL procedures & functions

⚠ Warning

PostgreSql functions has restrictions:

- returning refcursor or set of refcursor requires transaction!
- parameters use lower case names ONLY!
- result set names (refcursor names) use lower case names ONLY!

SQL procedure as query

To define procedure as data source of mapping used [ProcedureAttribute](#) with parameters: *Alias* - procedure alias, *Name* - procedure or function name, *ProcedureType* - [ProcedureType](#): *ProcedureType*.**Procedure** or *ProcedureType*.**Function**.

SQL procedure parameter

To define SQL procedure parameter used [ParameterAttribute](#) with parameters: *Alias* - procedure alias, *Order* - order number of parameter, *Binding* - name used inside application code (it's handy to use parameter with another name, not the name defined inside DB), *Type* - data type used inside application code, *DbType* - data type on ADO .Net provider level, *Direction* - optional, parameter direction, default is *ParameterDirection.Input*, *Size* - optional, parameter size(if applicable), *Scale* - optional, parameter precision (if applicable), *Name* - optional, parameter name, default is equal with *Binding*, *DefaultType* - [default value type](#), *DefaultValue* optional, depends on *DefaultType*, *NativeSqlType* - optional, SQL type name (*NativeSqlType* is used when type default mapping is conflicted with real SQL type), *UdtTypeName* - optional, user defined type (UDT), *UdtElementTypeName* - optional, user defined type (UDT) of *UdtTypeName* element (in case where *UdtTypeName* - array or table), *IsArray* - optional, parameter is array of *UdtElementTypeName* (when *UdtElementTypeName* is omitted, array of *DbType*), default is **false**.

SQL procedure result set

To describe result set of SQL procedure used [ResultSetAttribute](#) with parameters: *Alias* - procedure alias, *Name* - result set name used inside application code, when result set is not mapped to procedure parameter set it to "" (empty string), *Order* - optional, order number of result set (required to procedures returns multiple result sets), *ResultType* - optional, mapped type of result set, default is attributed class itself, *IsDefault* - optional, result set is default - mapped to attributed class (used when procedure returns multiple result sets), default is **false**.

SQL procedure result set as query sample

- MS SQL Server


```
CREATE PROCEDURE [dbo].[TestProcedure]
(
    @param1 int,
    @param2 varchar(20)
)
AS
    SELECT @param1 as Id, @param2 as Name
```

- Oracle

```
procedure TestProcedure(p_Id int, p_Name varchar2, p_Out out SYS_REFCURSOR)
as
begin
    open p_Out for select p_Id as Id, p_Name as Name from dual;
end TestProcedure;
```

- PostgreSQL

```
CREATE OR REPLACE FUNCTION public.TestProcedure (p_id integer, p_name varchar, refcursor)
RETURNS refcursor AS
$$
begin
    open $3 for select p_id as Id, p_name as Name;
    return $3;
end
$$ LANGUAGE plpgsql;
```

- MySQL

```
create procedure TestProcedure(p_Id int, p_Name varchar(50))
begin
    select p_Id as Id, p_Name as Name;
end
```

- Mapping

```
[DataObject("T"),
    Procedure("T", "dbo.TestProcedure", ProcedureType.Procedure),
    Parameter("T", 1, "param1", typeof(int), DbType.Int32),
    Parameter("T", 2, "param2", typeof(string), DbType.String),
    ResultSet("T")]
public class TestProcedure : IDataObject
{
    [Property("T", "Id", Flags = DataPropertyFlag.Id)]
    public int Id { get { return this.GetProperty(x => x.Id); } }
    [Property("T", "Name")]
    public string Name { get { return this.GetProperty(x => x.Name); } }
}
```

SQL procedure with multiple result sets sample

- MS SQL Server

```
CREATE PROCEDURE [dbo].[TestProcedure2]
(
    @param1 int,
    @param2 varchar(20),
    @param3 int,
    @param4 varchar(20)
)
AS
BEGIN
    SELECT @param1 as Id, @param2 as Name
    SELECT @param3 as Id, @param4 as Name
END
```

- Oracle

```
procedure TestProcedure2(p_Id int, p_Name varchar2, p_Out out SYS_REFCURSOR, p_Out2 out SYS_REFCURSOR)
as
begin
    open p_Out for select p_Id as Id, p_Name as Name from dual;
    open p_Out2 for select p_Id as Id, p_Name as Name from dual;
end TestProcedure2;
```

💡 Tip

Use *refcursor* parameters names as *ResultSet* names!

- PostgreSQL

```
CREATE OR REPLACE FUNCTION public.TestProcedure2 (p_id integer, p_name varchar, refcursor, refcursor)
RETURNS SETOF refcursor AS
$$
begin
    open $3 for select p_id as Id, p_name as Name;
    return NEXT $3;
    open $4 for select p_id as Id, p_name as Name;
    return NEXT $4;
end
$$ LANGUAGE plpgsql;
```

⚠ Warning

Functions witch returns *setof refcursor* is not supported by Npgsql .Net Standard 2.0 Edition :(XData net4.0 version can call that type functions with the same mapping rules as described here...

💡 Tip

If *refcursor* parameters have a names it must be used as *ResultSet* names!

- MySQL

```
create procedure TestProcedure2(p_Id int, p_Name varchar(50))
begin
    select p_Id as Id, p_Name as Name;
    select p_Id as Id, p_Name as Name;
end
```

- Mapping

```
[DataObject("T"),
    Procedure("T", "dbo.TestProcedure2", ProcedureType.Procedure),
    Parameter("T", 1, "param1", typeof(int), DbType.Int32),
    Parameter("T", 2, "param2", typeof(string), DbType.String),
    Parameter("T", 3, "param3", typeof(int), DbType.Int32),
    Parameter("T", 4, "param4", typeof(string), DbType.String),
    ResultSet("T", IsDefault = true),
    ResultSet("T", "SecondResult", ResultType = typeof(TestResult), Order = 2)]
public class TestProcedure2 : IDataObject
{
    [Property("T", "Id", Flags = DataPropertyFlag.Id)]
    public int Id { get { return this.GetProperty(x => x.Id); } }
    [Property("T", "Name")]
    public string Name { get { return this.GetProperty(x => x.Name); } }
    public IEnumerable<TestResult> SecondResult
    { get { return this.GetResultSet(x => x.SecondResult); } }
}
```

SQL procedure with out parameter sample

- MS SQL Server

```
CREATE PROCEDURE [dbo].[TestProcedure3]
(
    @param1 int,
    @param2 varchar(20),
    @param3 varchar(20) OUTPUT
)
AS
SET @param3 = convert(varchar(20), @param1) + ' - ' + @param2
RETURN
```

- Oracle

```
procedure TestProcedure3(p_Id int, p_Name varchar2, p_Out out varchar2)
as
begin
    select p_Id || ' - ' || p_Name into p_Out from dual;
end TestProcedure3;
```

- PostgreSQL

```
CREATE or REPLACE function TestProcedure3(p_id integer, p_name varchar, out p_out varchar) as $$
begin
    p_Out := p_id || ' - ' || p_name;
end
$$ LANGUAGE plpgsql;
```

- MySQL

```
create procedure TestProcedure3(p_Id int, p_Name varchar(50), out p_Out varchar(100))
begin
    set p_Out = CONCAT(p_Id, ' - ', p_Name);
end
```

- Mapping

```
[DataObject("T"),
    Procedure("T", "dbo.TestProcedure3", ProcedureType.Procedure),
    Parameter("T", 1, "param1", typeof(int), DbType.Int32),
    Parameter("T", 2, "param2", typeof(string), DbType.String),
    Parameter("T", 3, "OutParameter", typeof(string),
        DbType.String,
        Size = 20,
        Direction = ParameterDirection.Output,
        Name = "param3")]
public class TestProcedure3 : IDataObject
{
    public string OutParameter
    { get { return this.GetParameter(x => x.OutParameter); } }
}
```

SQL procedure with out parameter and result set sample

- MS SQL Server

```
CREATE PROCEDURE [dbo].[TestProcedure4]
(
    @param1 int,
    @param2 varchar(20)
)
AS
SELECT @param1 as Id, @param2 as Name
RETURN(@param1)
```

- Oracle

```
procedure TestProcedure4(p_Tab in classifier_tab, p_Out out SYS_REFCURSOR)
as
begin
    open p_Out for select Id, Name from table(p_Tab);
end TestProcedure4;
```

Tip

Use *refcursor* parameters names as *ResultSet* names!

- PostgreSQL

```
CREATE OR REPLACE FUNCTION public.TestProcedure4 (p_tab public.classifier_rec [], refcursor)
RETURNS refcursor AS
$$
begin
    open $2 for select Id, Name from unnest(p_tab);
    return $2;
end
$$
LANGUAGE plpgsql;
```

Tip

If *refcursor* parameters have a names it must be used as *ResultSet* names!

- MySQL

Not supported

- Mapping

```
[DataObject("T"),
    Procedure("T", "dbo.TestProcedure4", ProcedureType.Procedure),
    Parameter("T", 1, "param1", typeof(int), DbType.Int32),
    Parameter("T", 2, "param2", typeof(string), DbType.String),
    Parameter("T", 3, "OutParameter", typeof(string),
        DbType.String,
        Size = 20,
        Direction = ParameterDirection.Output,
        Name = "param3"),
    ResultSet("T")]
public class TestProcedure4 : IDataObject
{
    [Property("T", "Id", Flags = DataPropertyFlag.Id)]
    public int Id { get { return this.GetProperty(x => x.Id); } }
    [Property("T", "Name")]
    public string Name { get { return this.GetProperty(x => x.Name); } }
    public string OutParameter
    { get { return this.GetParameter(x => x.OutParameter); } }
}
```

SQL procedure with return value sample

- MS SQL Server

```
CREATE PROCEDURE [dbo].[TestProcedure5]
(
    @param1 int,
    @param2 varchar(20)
)
AS
RETURN(@param1)
```

- Oracle

Not supported

- PostgreSQL

Not supported

- MySQL

Not supported

- Mapping

```
[DataObject("T"),
    Procedure("T", "dbo.TestProcedure5", ProcedureType.Procedure),
    Parameter("T", 1, "param1", typeof(int), DbType.Int32),
    Parameter("T", 2, "param2", typeof(string), DbType.String),
    Parameter("T", 3, "ReturnValue", typeof(string),
        DbType.String,
        Size = 20,
        Direction = ParameterDirection.ReturnValue,
        Name = "result")]
public class TestProcedure5 : IDataObject
{
    public string ReturnValue
    { get { return this.GetParameter(x => x.ReturnValue); } }
}
```

SQL procedure with return value and result set sample

- MS SQL Server

- MS SQL Server

```
CREATE PROCEDURE [dbo].[TestProcedure6]
(
    @param1 int,
    @param2 varchar(20)
)
AS
SELECT @param1 as Id, @param2 as Name
RETURN(@param1)
```

- Oracle

Not supported

- PostgreSQL

Not supported

- MySQL

Not supported

- Mapping

```
[DataObject("T"),
    Procedure("T", "dbo.TestProcedure6", ProcedureType.Procedure),
    Parameter("T", 1, "param1", typeof(int), DbType.Int32),
    Parameter("T", 2, "param2", typeof(string), DbType.String),
    Parameter("T", 3, "ResultParameter", typeof(int),
        DbType.Int32,
        Direction = ParameterDirection.ReturnValue),
    ResultSet("T")]
public class TestProcedure6 : IDataObject
{
    [Property("T", "Id", Flags = DataPropertyFlag.Id)]
    public int Id { get { return this.GetProperty(x => x.Id); } }
    [Property("T", "Name")]
    public string Name { get { return this.GetProperty(x => x.Name); } }
    public int ResultParameter
    { get { return this.GetParameter(x => x.ResultParameter); } }
}
```

SQL procedure with simple array parameter sample

- MS SQL Server

Not supported

- Oracle

Not supported

- PostgreSQL

```
CREATE or REPLACE function TestFunction6(p_arr INTEGER[]) returns integer as $$
begin
    return sum(s) from unnest(p_arr) s;
end
$$ LANGUAGE plpgsql;
```

- MySQL

Not supported

- Mapping

```
[DataObject("T"),
    Procedure("T", "TestFunction", ProcedureType.Function),
    Parameter("T", 1, "param1", typeof(int[]), DbType.Int32, IsArray = true),
    Parameter("T", 2, "Result", typeof(int), DbType.Int32,
        Direction = ParameterDirection.ReturnValue)]
public class TestFunction : IDataObject
{
    public int Result { get { return this.GetParameter(x => x.Result); } }
}
```

SQL procedure with UDT parameter sample

- MS SQL Server

Not supported

Tip

MS SQL Server does not support passing single UDT object as a parameter value, but support array of UDT

- Oracle

Not supported

Warning

Oracle ODP.Net provider does not support UDT

- PostgreSQL

```
CREATE TYPE public.classifier_rec AS ( Id integer, Name varchar(20) );

CREATE or REPLACE function TestFunction5(p_obj classifier_rec) returns varchar as $$
begin
    return p_obj.Id || ' - ' || p_obj.Name;
end
$$ LANGUAGE plpgsql;
```

- MySQL

Not supported

- Mapping

```

/* Type mapped to UDT required to be Xml serializable */
[Serializable]
public class Classifier
{
    [XmlAttribute]
    public int Id { get; set; }
    [XmlAttribute]
    public string Name { get; set; }
}
...
[DataObject("T"),
    Procedure("T", "TestFunction5", ProcedureType.Function),
    Parameter("T", 1, "p_obj",
        typeof(Classifier),
        DbType.Object,
        UdtTypeName = "classifier_rec" /* Full qualified name of UDT */ ),
    Parameter("T", 2, "Result", typeof(string), DbType.String, Size = 20,
        Direction = ParameterDirection.ReturnValue)]
public class TestFunction5 : IDataObject
{
    public string Result => this.GetParameter(x => x.Result);
}

```

SQL procedure with UDT array parameter sample

- MS SQL Server

```

CREATE TYPE [dbo].[Classifier] AS TABLE(
    [Id] [int] NULL,
    [Name] [varchar](20) NULL
)

CREATE PROCEDURE [dbo].[TestProcedure7] (@param1 Classifier READONLY)
AS
SELECT * from @param1

```

- Oracle

Not supported

Warning

Oracle ODP.Net provider does not support UDT

- PostgreSQL

```

CREATE TYPE public.classifier_rec AS ( Id integer, Name varchar(20) );

CREATE or REPLACE function TestProcedure7(p_tab classifier_rec[]) returns setof classifier_rec as $$
    select a.Id, a.Name from unnest(p_tab) a;
$$ LANGUAGE sql;

```

- MySQL

Not supported

- Mapping


```

/* Type mapped to UDT required to be Xml serializable */
[Serializable]
public class Classifier
{
    [XmlAttribute]
    public int Id { get; set; }
    [XmlAttribute]
    public string Name { get; set; }
}
...
[DataObject("T"),
    Procedure("T", "dbo.TestProcedure7", ProcedureType.Procedure),
    Parameter("T", 1, "param1",
        typeof(Classifier[]) /* Parameter type must be an array of mapped type */,
        DbType.Object,
        UdtTypeName = "dbo.Classifier" /* Full qualified name of UDT */ ),
    ResultSet("T")]
public class TestProcedure7 : IDataObject
{
    [Property("T", "Id", Flags = DataPropertyFlag.Id)]
    public int Id => this.GetProperty(x => x.Id);
    [Property("T", "Name")]
    public string Name => this.GetProperty(x => x.Name);
}

```

SQL table valued function as query sample

- MS SQL Server

```

CREATE FUNCTION [dbo].[TestFunction]
(
    @param1 int,
    @param2 varchar(20)
)
RETURNS TABLE
AS
RETURN
(
    SELECT @param1 as Id, @param2 as Name
)

```

- Oracle

```

CREATE OR REPLACE type classifier_rec is object (Id int, Name varchar2(20));
CREATE OR REPLACE type classifier_tab is table of classifier_rec;

function TestFunction(p_Id int, p_Name varchar2) return classifier_tab pipelined
is
begin
for curr in (select p_Id as Id, p_Name as Name from dual) loop
    pipe row (classifier_rec(curr.Id, curr.Name));
end loop;
end TestFunction;

```

- PostgreSQL

```

CREATE or REPLACE function TestFunction(p_id integer, p_name varchar)
returns TABLE (
    Id bigint,
    Name varchar
) as $$
    select p_id as Id, p_name as Name;
$$ LANGUAGE SQL;

```

... or with the same mapping syntax...

```
CREATE TYPE public.classifier_rec AS ( Id integer, Name varchar(20) );

CREATE or REPLACE function TestFunction(p_Id integer, p_Name varchar) returns setof classifier_rec as $$
    select p_Id as Id, p_Name as Name;
$$ LANGUAGE SQL;
```

- MySQL

Not supported

- Mapping

```
[DataObject("T"),
    Procedure("T", "dbo.TestFunction", ProcedureType.Function),
    Parameter("T", 1, "param1", typeof(int), DbType.Int32),
    Parameter("T", 2, "param2", typeof(string), DbType.String),
    ResultSet("T")]
public class TestFunction : IDataObject
{
    [Property("T", "Id", Flags = DataPropertyFlag.Id)]
    public int Id { get { return this.GetProperty(x => x.Id); } }
    [Property("T", "Name")]
    public string Name { get { return this.GetProperty(x => x.Name); } }
}
```

- Mapping table function with joined table

```
[DataObject("D"),
    Procedure("T", "dbo.TestFunction", ProcedureType.Function),
    Parameter("T", 1, "param1", typeof(int), DbType.Int32),
    Parameter("T", 2, "param2", typeof(string), DbType.String),
    ResultSet("T"),
    DataTable("T_DOC_TYPE", "D"),
    Link("D", "doc_type_id", "T", "Id")]
public class TestFunctionJoin : IDataObject
{
    [Property("T", "Id", Flags = DataPropertyFlag.Id)]
    public int Id { get; set; }
    [Property("T", "Name")]
    public string Name { get; set; }
    [Property("D", "Code")]
    public string Code { get; set; }
}
```

SQL scalar function as property data source sample

- MS SQL Server

```
CREATE FUNCTION [dbo].[TestFunction2]
(
    @param1 int,
    @param2 varchar(20)
)
RETURNS varchar(20)
AS
BEGIN
    DECLARE @res varchar(20)
    SET @res = convert(varchar(20), @param1) + ' - ' + @param2
    RETURN @res
END
```

- Oracle

```
function TestFunction2(p_Id int, p_Name varchar2) return varchar2
is
begin
    return p_Id || ' - ' || p_Name;
end TestFunction2;
```

- PostgreSQL

```
CREATE or REPLACE function TestFunction2(p_id integer, p_name varchar) returns varchar as $$
DECLARE
ret varchar;
begin
    select p_id || ' - ' || p_name into ret;
    return ret;
end
$$ LANGUAGE plpgsql;
```

- MySQL

```
create function TestFunction2(p_Id int, p_Name varchar(20)) returns varchar(100)
begin
    declare res varchar(100);
    select CONCAT(p_Id, ' - ', p_Name) into res;
    return(res);
end
```

- Mapping

```
[DataObject("T"),
    DataTable("T_DOC_TYPE", "T"),
    Column("DocTypeId", typeof(long), Flags = DataPropertyFlag.Id)]
public class TestFunction4 : IDataObject
{
    [Property("T")]
    public string Code { get; set; }
    [Property("T")]
    public string Name { get; set; }
    [Property,
        // Property expression attribute "name" is the name of private static field
        // with type Calculate<string> and attributed by SqlExpressionAttribute
        PropertyExpression("test", DataExpressionType.LinqExpression, ExprSize = 20)]
    public string Test { get { return this.GetProperty(x => x.Test); } }

    // Private static field with type Calculate<string> and attributed by
    // SqlExpressionAttribute
    [SqlExpression]
    private static Calculate<string> test z => z.SqlFn<string>("dbo.TestFunction2",
        y => y.Field<long>("T", "doc_type_id"), y => y.Field<string>("T", "name"))
}
```

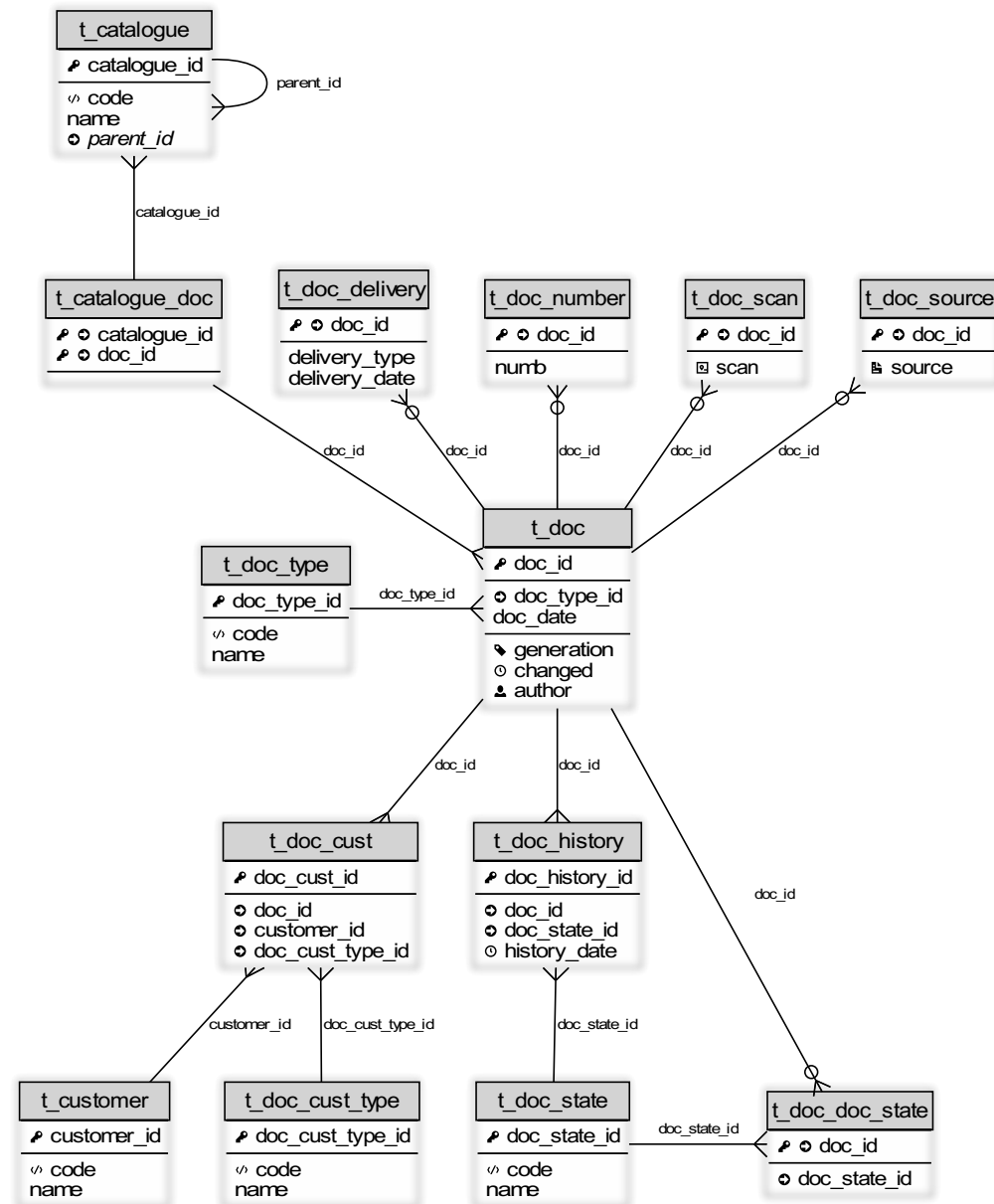
Dynamic mapping

Tip

Before read this topic be sure to already readed [Glossary](#) and [Common mapping rules](#) contains base information to better understanding this.

Dynamic mapping is defined using specific LINQ style expression as private static field of mapped class and it properties.

Dynamic mapping sample




```

N.num AS N_num /* DocNumb */,
S.code AS S_code /* DocStateCode */,
S.DOC_STATE_ID AS S_DOC_STATE_ID,
S.name AS S_name /* DocState */,
SC.DOC_ID AS SC_DOC_ID,
SC.scan AS SC_scan /* Scan */,
SR.DOC_ID AS SR_DOC_ID,
SR.source AS SR_source /* Source */,
ST.S_doc_id AS ST_S_doc_id,
T.CODE AS T_CODE,
T.CUSTOMER_ID AS T_CUSTOMER_ID,
T.name AS T_name /* CustomerTo */
FROM T_DOC D
JOIN T_DOC_DOC_STATE DS ON DS.doc_id = D.doc_id
JOIN T_DOC_STATE S ON S.DOC_STATE_ID = DS.DOC_STATE_ID
JOIN T_DOC_NUMBER N ON N.doc_id = D.doc_id
JOIN T_DOC_CUST CF ON CF.doc_id = D.doc_id
JOIN T_DOC_CATALOGUE DC ON DC.doc_id = D.doc_id
JOIN T_CATALOGUE U ON DC.catalogue_id = U.catalogue_id
JOIN T_CUSTOMER F ON F.CUSTOMER_ID = CF.CUSTOMER_ID
JOIN T_DOC_CUST CT ON CT.doc_id = D.doc_id
JOIN T_CUSTOMER T ON T.CUSTOMER_ID = CF.CUSTOMER_ID
LEFT OUTER JOIN T_DOC_SCAN SC ON SC.doc_id = D.doc_id
LEFT OUTER JOIN T_DOC_SOURCE SR ON SR.doc_id = D.doc_id
LEFT OUTER JOIN T_DOC_DELIVERY DD ON DD.doc_id = D.doc_id
LEFT OUTER JOIN (SELECT H.doc_id AS H_doc_id,
H.history_date AS H_history_date
FROM T_DOC_HISTORY H) H ON D.doc_id = H.H_doc_id
JOIN (SELECT S.doc_id AS S_doc_id
FROM T_DOC_SPEC_TYPE ST
JOIN T_DOC_SPEC S ON S.DOC_SPEC_TYPE_ID = ST.DOC_SPEC_TYPE_ID
WHERE ST.code = @FilterBySpecTypeCode) ST ON D.doc_id = ST.S_doc_id
WHERE D.DOC_TYPE_ID = (select doc_type_id
from T_DOC_TYPE
where code='INVOICE')
AND CF.DOC_CUST_TYPE_ID = (select doc_cust_type_id
from T_DOC_CUST_TYPE
where code='FROM')
AND CF.DOC_CUST_TYPE_ID = (select doc_cust_type_id
from T_DOC_CUST_TYPE
where code='TO')
AND D.doc_id = @FilterByDocId
AND DC.catalogue_id in (SELECT T.catalogue_id AS T_catalogue_id FROM T T)

```

Mapping sample:

```

public partial class Invoice: IDataObject
{
    public static CustomLogic<Invoice> TestCustomLogic;

    private static Expression<CustomMapping<Invoice>> _mapping = (
        () => XDataMapping.CustomMapping<Invoice>()
        .DataTable("T_DOC", "D",
            x => x.DictFilter("T_DOC_TYPE", "doc_type_id", "code", "INVOICE"))
        .DataTable("T_DOC_DOC_STATE", "DS", "D", x => x.Link("D", "doc_id"))
        .DataTable("T_DOC_STATE", "S", x => x.Link("DS", "doc_state_id"))
        .DataTable("T_DOC_CATALOGUE", "DC", "D", x => x.Link("D", "doc_id"),
            x => x.SubqueryFilter("catalogue_id", "UT").SetOperation(FilterOperation.In))
        .DataTable("T_CATALOGUE", "U", x => x.Link("DC", "catalogue_id"))
        .DataTable("T_DOC_NUMBER", "N", "D", x => x.Link("D", "doc_id"))
        .DataTable("T_DOC_CUST", "CF", "D", x => x.Link("D", "doc_id"),
            x => x.DictFilter("T_DOC_CUST_TYPE", "doc_cust_type_id", "code", "FROM"))
        .DataTable("T_CUSTOMER", "F", x => x.Link("CF", "customer_id"))
        .DataTable("T_DOC_CUST", "CT", "D", x => x.Link("D", "doc_id"),
            x => x.DictFilter("T_DOC_CUST_TYPE", "doc_cust_type_id", "code", "TO"))
    );
}

```

```

.DataTable("T_CUSTOMER", "T",
    x => x.Link("CT", "customer_id"))
.DataTable("T_DOC_SCAN", "SC", "D",
    x => x.Link("D", "doc_id").SetOperation(FilterOperation.OuterJoin))
.DataTable("T_DOC_SOURCE", "SR", "D",
    x => x.Link("D", "doc_id").SetOperation(FilterOperation.OuterJoin))
.DataTable("T_DOC_DELIVERY", "DD", "D",
    x => x.Link("D", "doc_id").SetOperation(FilterOperation.OuterJoin))
.Subquery("A", typeof(DocSpecAmounts), "Amount", DataGrouping.Sum,
    x => x.SubqueryLink("DocId"))
.Subquery<CatalogueTree>("UT", x => x.CatalogueId, DataGrouping.None)
.InnerView("H", XDataMapping.GetStructure("H", DataStructureFlag.Grouping)
    .DataTable("T_DOC_HISTORY", "H")
    .Select(x => new {
        DocId = x.Field<long?>("H", string.Empty,
            z => z.Group(DataGrouping.None)),
        HistoryDate = x.Field<DateTime?>("H", string.Empty,
            z => z.Group(DataGrouping.Max))
    }), x => x.SubqueryLink("DocId").SetOperation(FilterOperation.OuterJoin))
.InnerView<DocBySpecType>("ST", x => x.SubqueryLink("DocId"))
.Column("DocId", x => x.Field<long?>("D", string.Empty,
    z => z.Key(), z => z.Default(DefaultType.AutoIncrement)))
.Column("DocStateId", x => x.Field<long>("S", string.Empty))
.Column("FromCustomerId", x => x.Field<long>("F", "customer_id"))
.Column("ToCustomerId", x => x.Field<long>("T", "customer_id"))
.Column("CatalogueId", x => x.Field<long?>("U", string.Empty))
.ReadOnlyProperty(x => x.DocStateCode, x => x.Field<string>("S", "code"))
.ReadOnlyProperty(x => x.Generation,
    x => x.Field<long>("D", string.Empty,
        z => z.ConcurrencyToken(), z => z.Default(DefaultType.AutoIncrement)))
.ReadOnlyProperty(x => x.Changed,
    x => x.Field<DateTime>("D", string.Empty,
        z => z.Default(DefaultType.CurrentDateTime, DefaultFeature.UseOnUpdate)))
.ReadOnlyProperty(x => x.Author,
    x => x.Field<string>("D", string.Empty,
        z => z.Default(DefaultType.UserName, DefaultFeature.UseOnUpdate)))
.ReadOnlyProperty(x => x.DocAmount,
    x => x.Expr<decimal?>(null, DataExpressionType.SubQuery,
        "A", DbType.Decimal, z => z.Size(17, 5)))
.ReadOnlyProperty(x => x.DocLastChange, x => x.Ref<DateTime?>("H", "HistoryDate"))
.Map(x => new Invoice {
    DocState = x.Link<string, DocState>("S", "name",
        z => z.LinkProperty<DocState>(y => y.Name),
        z => z.LinkProperty<DocState>(y => y.Code, y => y.DocStateCode),
        z => z.LinkProperty<DocState>((Invoice y) => y.GetProperty<long>("DocStateId"))),
    DocCatalog = x.Link<string, Catalogue>("U", "name",
        z => z.LinkProperty<Catalogue>(y => y.Name),
        z => z.LinkProperty<Catalogue>((Invoice y) => y.GetProperty<long>("CatalogueId"))),
    DocNumb = x.Field<string>("N", "numb"),
    DocDate = x.Field<DateTime?>("D", string.Empty,
        z => z.Default(DefaultType.CurrentDate)),
    Scan = x.Lob("SC", z => z.OuterFlag()),
    Source = x.Xml("SR", z => z.OuterFlag()),
    CustomerFrom = x.Link<string, Customer>("F", "name",
        z => z.LinkProperty<Customer>(y => y.Name),
        z => z.LinkProperty<Customer>(y => y.GetProperty<long>("CustomerId"),
            y => y.GetProperty<long>("FromCustomerId"))),
    CustomerTo = x.Link<string, Customer>("T", "name",
        z => z.LinkProperty<Customer>(y => y.Name),
        z => z.LinkProperty<Customer>(y => y.GetProperty<long>("CustomerId"),
            y => y.GetProperty<long>("ToCustomerId"))),
    DeliveryType = x.Field<DeliveryTypeEnum>("DD", string.Empty),
    DeliveryDate = x.Field<DateTime?>("DD", string.Empty,
        z => z.Default(DefaultType.CurrentDate))
}, x => x.ExternalLink<InvoiceSpec>("DocId"))
.SetBaseTable("D").SetLogicAssemblv("XDataObjectTest")

```

```

    );
}
...
public partial class Invoice
{
    public string DocStateCode { get
        { return this.GetProperty(x => x.DocStateCode); }
    }
    public Link<string, DocState> DocState { get; set; }
    public Link<string, Catalogue> DocCatalog { get; set; }
    public string DocNumb { get; set; }
    public DateTime? DocDate { get; set; }
    public long Generation { get { return this.GetProperty(x => x.Generation); } }
    public DateTime Changed { get { return this.GetProperty(x => x.Changed); } }
    public string Author { get { return this.GetProperty(x => x.Author); } }
    public decimal? DocAmount { get { return this.GetProperty(x => x.DocAmount); } }
    public DateTime? DocLastChange { get
        { return this.GetProperty(x => x.DocLastChange); }
    }
    public Lob Scan { get; set; }
    public Xml Source { get; set; }
    public Link<string, Customer> CustomerFrom { get; set; }
    public Link<string, Customer> CustomerTo { get; set; }
    public DeliveryTypeEnum DeliveryType { get; set; }
    public DateTime? DeliveryDate { get; set; }

    public IRepository<InvoiceSpec> Spec { get
        { return this.GetRepository().GetChild<InvoiceSpec>(); }
    }
}
}

```

Dynamic mapping provide more representable mechanics to define mapping because of no attribute parameters limitation. Subqueries and inner views can be described in place using [dynamic query](#) mechanics. SQL expressions can be described in LINQ style in place. This abilities enreach representability of dynamic mapping.

Dynamic mapping allows split [data object](#) to partial class definitions with basic class definition and mapping.

Dynamic mapping initialization

Dynamic mapping is described as private static field of type `Expression<CustomMapping<T>>` where `T` - data object type. The value of this property is assigned by call of `XDataMapping.CustomMapping<T>` method returning `IRepositoryStructure<T>`. Interface `IRepositoryStructure<T>` has a number of methods to describe all of mapping entities analog to [static mapping](#) attributes.

The `XDataMapping.UseMapping<T>` method can be used to inherit the mapping description of the data object. The depth of the inheritance hierarchy of the mapping description is not limited. There are several overloads of this method:

`UseMapping<T, TParent>()` - points to type `TParent` as the base mapping description class. The `TParent` class must implement the `ISqlObject` interface but does not have to be associated with the `T` class.

`UseMapping<T>()` - the same as the previous method, only the base class in the usual class inheritance hierarchy acts as the base class in the mapping description hierarchy.

`UseMapping<T, TParent>(params Expression<Func<IExternalLinkDefinitionAdapter<T>, IExternalLinkDefinition>>[] externalLinks)` - allows you to fully inherit the description of mapping and supplement the description with external links. This method of inheriting the mapping description imposes a limitation - the base class must be the ancestor of the `T` class.

`UseMapping<T>(params Expression<Func<IExternalLinkDefinitionAdapter<T>, IExternalLinkDefinition>>[] externalLinks)` - the same as the previous method, only the base class in the usual class inheritance hierarchy acts as the base class in the mapping description hierarchy.

Warning

When inheriting mapping descriptions, a mixture of the static and dynamic description methods within the a single mapping inheritance hierarchy is not allowed!

Mapping parameters

To set parameters of dynamically mapped classes used [IRepositoryDescription<T>](#) interface methods:

- [SetBaseTable](#) - to set [base table](#), (optional, by default used [virtual table](#) with the empty alias)
- [SetContext](#) - database [context](#) (optional, by default context will be assigned in runtime)
- [SetFlags](#) - [data source flags](#) (optional, by default **None**)
- [SetLogicAssemblies](#) - to set data processing logic classes assemblies FullName for this [repository](#) (see [Using three tier architecture](#) and [IDataLogic<T>](#))

To get reference to [IRepositoryDescription<T>](#) interface is required to define mapping of [repository](#) read/write properties using [IRepositoryStructure<T>](#) interface method [Map](#) Example:

```
...
.Map(x => new Invoice {
    DocState = x.Link<string, DocState>("S", "name",
        z => z.LinkProperty<DocState>(y => y.Name),
        z => z.LinkProperty<DocState>(y => y.Code,
            y => y.DocStateCode),
        z => z.LinkProperty<DocState>(
            (Invoice y) => y.GetProperty<long>("DocStateId"))),
    DocCatalog = x.Link<string, Catalogue>("U", "name",
        z => z.LinkProperty(y => y.Name),
        z => z.LinkProperty<Catalogue>(
            (Invoice y) => y.GetProperty<long>("CatalogueId"))),
    DocNumb = x.Field<string>("N", "numb"),
    DocDate = x.Field<DateTime?>("D", string.Empty,
        z => z.Default(DefaultType.CurrentDate)),
    Scan = x.Lob("SC", z => z.OuterFlag()),
    Source = x.Xml("SR", z => z.OuterFlag()),
    CustomerFrom = x.Link<string, Customer>("F", "name",
        z => z.LinkProperty(y => y.Name),
        z => z.LinkProperty<Customer>(y => y.GetProperty<long>("CustomerId"),
            y => y.GetProperty<long>("FromCustomerId"))),
    CustomerTo = x.Link<string, Customer>("T", "name",
        z => z.LinkProperty(y => y.Name),
        z => z.LinkProperty<Customer>(y => y.GetProperty<long>("CustomerId"),
            y => y.GetProperty<long>("ToCustomerId"))),
    DeliveryType = x.Field("DD", string.Empty),
    DeliveryDate = x.Field<DateTime?>("DD", string.Empty,
        z => z.Default(DefaultType.CurrentDate))
}, x => x.ExternalLink("DocId"))
.SetBaseTable("D").SetLogicAssembly("XDataObjectTest")
...
```

More detailed repository properties map description [see below](#).

Tables

All tables of [data object](#) mapping described by [IRepositoryStructure<T>](#).[DataTable](#) method. Tables included into [updatable tables hierarchy](#) excluding [base table](#) must be defined using special [overload](#) allows to set parent table alias.

```
...
.DataTable("T_DOC", "D",
    x => x.DictFilter("T_DOC_TYPE", "doc_type_id", "code", "INVOICE"))
.DataTable("T_DOC_DOC_STATE", "DS", "D", x => x.Link("D", "doc_id"))
...
```

Same way is possible to define mapping to view, but all properties must to mapped as [read only properties](#) or [hidden fields](#).

❏ Important

Filters and links between tables is defined using *filters* parameter of `IRepositoryStructure<T>.DataTable` method or it's [overload](#). Detailed filter definition described [below](#).

Subqueries

Subqueries can be used for a:

- filtration using *[not] exists()*, *[not] in (select)* ([see below](#))
- inner view
- subquery expression as single property source ([see below](#))

Subquery deep level is not limited.

In dynamic mapping of data object subqueries can be defined using one of [Subquery](#) or [InnerView](#) methods overload of [IRepositoryStructure<T>](#) interface to define subqueries or inner views respectively.

While using generic overloads of *Subquery* or *InnerView*, parameter *IQueryDescription<TObj> sub* can be used with in place [dynamic query](#) notation.

Non generic overloads of *Subquery* or *InnerView* must provide `ISqlType` or `IDataObject` marked class type as parameter *Type objType* to reuse of existing mapping. This class can be mapped using static or dynamic style, no matter.

```
...
.Subquery("A", typeof(DocSpecAmounts), "Amount", DataGrouping.Sum,
    x => x.SubqueryLink("DocId"))
.Subquery<CatalogueTree>("UT", x => x.CatalogueId, DataGrouping.None)
.InnerView("H",
    XDataMapping.GetStructure("H", DataStructureFlag.Grouping)
    .DataTable("T_DOC_HISTORY", "H")
    .Select(x => new {
        DocId = x.Field<long?>("H", string.Empty,
            z => z.Group(DataGrouping.None)),
        HistoryDate = x.Field<DateTime?>("H", string.Empty,
            z => z.Group(DataGrouping.Max))
    }),
    x => x.SubqueryLink("DocId").SetOperation(FilterOperation.OuterJoin))
.InnerView<DocBySpecType>("ST", x => x.SubqueryLink("DocId"))
...
```

Filters

Filters can be defined as:

- Constant filters
- SQL expression filters
- Dictionary filters
- Range filters
- Subquery filters
- Inner links

Each of them described by corresponding LINQ expression define their parameters depending on the context of their use and the type of the filter itself (see below). Filter descriptions are provided to *filters* parameter value of [DataTable](#), [Subquery](#), [InnerView](#) and [Procedure](#) methods of [IRepositoryStructure<T>](#) interface, or [WithRecursive](#) method of [IQueryWithAdapter<TRoot>](#). Each of expression has described one filter. Type of filter is declared by one of fabric method of interface inside lambda expression with one of followed filter adapter:

- [IInnerFilterAdapter](#) - filter adapter to define filters and links inside plain query (see [DataTable](#), [Procedure](#)),
- [ISubqueryLinkAdapter](#), [ISubqueryLinkAdapter<TDObj>](#) and [ISubqueryLinkAdapter<T,TDObj>](#) - used in various overloads of subquery definition (see [Subquery](#), [InnerView](#) and [WithRecursive](#)) to declare links with subqueries.

Specific parameters for each type of filter is described in filter fabric methods described below in specific topics. And some optional modifiers can be defined using continue LINQ calls with extension methods:

- [SetOperation](#) - to set [filter operation](#)
- [AsPrimary](#) - to set [optional filter](#) flag
- [SetCombination](#) - to set [filter combination](#)

Each of filter type is defined by own special method of corresponding filter adapter.

Filter operation is defined by enum [FilterOperation](#):

- **Equal** - equal
- **NotEqual** - not equal
- **LessThan** - less then
- **GreaterThan** - greater then
- **LessThanOrEqual** - less or equal
- **GreaterThanOrEqual** - greater or equal
- **In** - field value in range or subquery
- **NotIn** - field value is not in range or subquery
- **Exists** - subquery returns one or more rows
- **NotExists** - subquery returns no rows
- **Like** - field value correlated with find expression
- **NotLike** - field value not correlated with find expression
- **OuterJoin** - left outer join link between query sources (tables or inner views)
- **Contains** - field value correlated with full-text search
- **NotContains** - field value not correlated with full-text search

Filter group

Filter group of filter can be assigned to it declaration by [SetCombination](#) extension of [IFilterDescription](#) interface.

```
...
, x => x.ConstFilter("FilterByCathalogue", "catalogue_id",
    FilterConstantType.Const, null, FilterNullable.NullsNotAllowed)
    .SetCombination("root")
...
```

Constant filters

Constant filters allows to restrict selection of data by some constant value. This filter defined by one of [ConstFilter](#) method overloads of [IInnerFilterAdapter](#) interface.

Constant type list restricted by enum [FilterConstantType](#):

- **Const** - constant with value set by *ConstantValue*.
- **Variable** - value of [variable](#) with name set by *ConstantValue*.
- **CurrentDate** - current date.
- **CurrentDateTime** - current date and time.
- **CurrentDateTimeUTC** - current date and time (UTC).

```
...
, x => x.ConstFilter("FilterByCathalogue", "catalogue_id",
    FilterConstantType.Const, null, FilterNullable.NullsNotAllowed)
    .SetCombination("root")
...
```

SQL expression filters

To define filter over SQL expression used one of *ExprFilter* method overloads of [IInnerFilterAdapter](#) interface. There are two overloads of *ExprFilter* method:

- [IFilterDescription ExprFilter\(string field, string exprText\)](#) - to use with plain SQL expressions or subquery alias
- [IFilterDescription ExprFilter<T>\(string field, Expression<Func<IQueryStructureAdapter, T>> expression\)](#) - to use with SQL expressions as LINQ style expressions

Depending style of SQL expression description that overloads can be used with:

- subquery alias,

```
, ExprFilter("discount" /* field name */,
    "D" /* ExpressionText: subquery alias */)
    .SetOperation(FilterOperation.In)
```

- dynamic definition of SQL expression,

```
, ExprFilter("discount" /* field name */,
    /* expression: SQL expression LINQ style definition */
    x => x.Case<Product, int>(
        z => z.Field<bool>("is_vip"), z => 0, 1.SetExpression(z => 10)),
    Operation: FilterOperation.NotEqual /* filter operation */)]
```

- plain SQL expression text.

```
, ExprFilter("discount" /* field name */,
    /* ExpressionText: plain SQL expression */
    "case P.is_vip when 1 then 10 else 0 end")
    .SetOperation(FilterOperation.NotEqual)
```

❗ Important

If possible do not use plain SQL expressions. It possibly use specific SQL dialect features not supported by other DBMS or providers.

Range filters

To define filter checks field value over array of values used [RangeFilter](#) methods of [IInnerFilterAdapter](#) interface.

```
...
, RangeFilter("discount", 1, 5, 10)
    .SetOperation(FilterOperation.In)
...
```

Dictionary filters

Dictionaries surrogate keys may varied over one instance to another, and names can be changed by users. Sometimes business logic need to operate with the concrete value of dictionary. Because of this it's recommended to add into dictionary DB structure a special unique text field - mnemonic code.

XData supports using mnemonic code when define filters on dictionary value - dictionary filters. They defined by one of [DictFilter](#) overloads of [IInnerFilterAdapter](#) interface.

❗ Important

This filter can be used when dictionary has simple (one field only) identity field and mnemonic code field. But it's a very common case.

```
...
, x => x.DictFilter("T_DOC_CUST_TYPE", "doc_cust_type_id", "code", "T0")
...
```

Inner links

To define links between data sources inside query used [Link](#) overloads of [IInnerFilterAdapter](#) interface.

```
...
, x => x.Link("DS", "doc_state_id")
...
```

Tip

Inner links can be used to define [optional filters](#). To specify role in optional filter chain need to call extension method [AsPrimary](#) as continuation of *Link* method call.

Subquery filters

To define filters checks field value is correspond subquery results used [SubqueryFilter](#) method of [IInnerFilterAdapter](#).

```
...
.DataTable("T_DOC_CATALOGUE", "DC", "D", x => x.Link("D", "doc_id"),
    x => x.SubqueryFilter("catalogue_id", "UT")
    .SetOperation(FilterOperation.In))
.Subquery<CatalogueTree>("UT", x => x.CatalogueId, DataGrouping.None)
...
```

Tip

Subquery filters can be used to define [optional filters](#). To specify role in optional filter chain need to call extension method [AsPrimary](#) as continuation of *Link* method call.

Subquery links

Links with subqueries is used to link not table fields but properties of base class and subquery mapped class including [hidden properties](#). To link with subquery or inner view used one of *SubqueryLink* method overload of [ISubqueryLinkAdapter](#), [ISubqueryLinkAdapter<TDObj>](#) or [ISubqueryLinkAdapter<T,TDObj>](#) depending by outer method signature and subquery declaration specific.

```
...
.InnerView<DocBySpecType>("ST", x => x.SubqueryLink("DocId"))
...
```

Tip

Subquery links can be used to define [optional filters](#). To specify role in optional filter chain need to call extension method [AsPrimary](#) as continuation of *Link* method call.

Properties

Properties dynamic mapping is performed as method call of [IRepositoryStructure<T>](#) interface:

- [Column](#) method call for each of [hidden properties](#) mapping
- [ReadOnlyProperty](#) method call for each of [read only properties](#) mapping
- [Map](#) method call for declare all of read/write properties mapping

```
...
.Column("DocId", x => x.Field<long?>("D", string.Empty,
    z => z.Key(), z => z.Default(DefaultType.AutoIncrement)))
...
.ReadOnlyProperty(x => x.Changed,
    x => x.Field<DateTime>("D", string.Empty,
        z => z.Default(DefaultType.CurrentDateTime, DefaultFeature.UseOnUpdate)))
...
```

Parameter *mapper* of *Map* method allows to define mapping rule for multiple read/write properties using lambda expression with [IQueryStructureAdapter](#) interface as parameter.

```
...
.Map(x => new Invoice {
    DocState = x.Link<string, DocState>("S", "name",
        z => z.LinkProperty<DocState>(y => y.Name),
        z => z.LinkProperty<DocState>(y => y.Code, y => y.DocStateCode),
        z => z.LinkProperty<DocState>(
            (Invoice y) => y.GetProperty<long>("DocStateId"))),
    DocCatalog = x.Link<string, Catalogue>("U", "name",
        z => z.LinkProperty<Catalogue>(y => y.Name),
        z => z.LinkProperty<Catalogue>(
            (Invoice y) => y.GetProperty<long>("CatalogueId"))),
    DocNumb = x.Field<string>("N", "numb"),
    DocDate = x.Field<DateTime?>("D", string.Empty,
        z => z.Default(DefaultType.CurrentDate)),
    Scan = x.Lob("SC", z => z.OuterFlag()),
    Source = x.Xml("SR", z => z.OuterFlag()),
    CustomerFrom = x.Link<string, Customer>("F", "name",
        z => z.LinkProperty<Customer>(y => y.Name),
        z => z.LinkProperty<Customer>(y => y.GetProperty<long>("CustomerId"),
            y => y.GetProperty<long>("FromCustomerId"))),
    CustomerTo = x.Link<string, Customer>("T", "name",
        z => z.LinkProperty<Customer>(y => y.Name),
        z => z.LinkProperty<Customer>(y => y.GetProperty<long>("CustomerId"),
            y => y.GetProperty<long>("ToCustomerId"))),
    DeliveryType = x.Field<DeliveryTypeEnum>("DD", string.Empty),
    DeliveryDate = x.Field<DateTime?>("DD", string.Empty,
        z => z.Default(DefaultType.CurrentDate))
}, x => x.ExternalLink<InvoiceSpec>("DocId"))
...
```

Interfaces [IRepositoryStructureAdapter<T>](#) and [IQueryStructureAdapter](#) as successor has some overloads of methods to define mapping:

- [Field<TVal>](#) - for mapping property to data source field
- [Expr<TVal>](#) - for mapping property to SQL expression (see detailed description [below](#))
- [Ref<TVal>](#) - for mapping property to inner view property (see detailed description [below](#))

[IRepositoryStructureAdapter<T>](#) also has a special methods to map [Lob](#), [Xml](#) and [Link<TVal,TSource>](#)

- [Lob](#) - for mapping Lob property
- [Xml](#) - for mapping Xml property
- [Link<TVal,TSource>](#) - for mapping [link](#) property (see detailed description [below](#))

These methods of single property mapping has parameter *attributes* to define additional mapping properties using interface [IPropertyMappingAdapter](#):

- [Key](#) - primary unique key property flag, possible to mark multiple properties or [hidden properties](#) as key properties
- [Hidden](#) - flag, the field should not be included in the *SELECT* expression, but can be used in filters and/or links between data sources (tables, subqueries...)

- [Group](#) - property grouping parameters (see detailed description [below](#))

And using interface [IRepositoryPropertyMappingAdapter<T, TValue>](#):

- [ConcurrencyToken](#) - [concurrency token](#) flag for property
- [Default](#) - [default value](#) assign rules for property
- [OuterFlag](#) - outer join mandatory flag, values of properties marked this flag is analyzed during making decision of operations over tables in [optional references](#)

□ Important

Required to set at least one property of mapped class as primary unique key. Subqueries and inner views can has no key properties.

Interface [IQueryStructureAdapter<T>](#) is used in [dynamic queries](#) only. Interfaces [IRepositoryStructureAdapter](#) and [IRepositoryStructureAdapter<T>](#) is used in dynamic repository mapping only. Interface [IPropertyMappingAdapter](#) is used as in [dynamic queries](#) as in dynamic repository mapping. Interface [IRepositoryPropertyMappingAdapter](#) is inherited from [\[IPropertyMappingAdapter\]](#) and used in dynamic repository mapping only.

SQL expression property

[Read only property](#) can be mapped to SQL expression. Dynamic mapping of SQL expression is defined by one of [IRepositoryStructureAdapter](#) interface [Expr](#) method overload:

- [Expr<TValue>\(string fieldName, Expression<Func<IQueryStructureAdapter, TValue>> expr, DbType dbType, params Expression<Func<IRepositoryExpressionMappingAdapter, IStructureFlag>>\[\] attributes\)](#)
- [Expr<TValue>\(string fieldName, DataExpressionType type, string exprText, DbType dbType, params Expression<Func<IRepositoryExpressionMappingAdapter, IStructureFlag>>\[\] attributes\)](#) - used to define SQL expression [DataExpressionType.SubQuery](#) or [DataExpressionType.PlainSql](#)

When first overload used, SQL expression is defined by [expr](#) parameter value as lambda function with [IQueryStructureAdapter](#) interface extension methods inside:

- [Case<T, TResult>\(this IQueryStructureAdapter adapter, Expression<Func<IQueryStructureAdapter, T>> expression, Expression<Func<IQueryStructureAdapter, TResult>> elseValue, params TaggedExpression<T, TResult>\[\] switchValues\)](#) - *case when ... then ... else ... end* SQL construction
- [DateDiff<T>\(this IQueryStructureAdapter adapter, Expression<Func<IQueryStructureAdapter, DateTime>> finalDate, Expression<Func<IQueryStructureAdapter, DateTime>> startDate, DateDiffKind kind\)](#) - date difference
- [Exists\(this IQueryStructureAdapter adapter, string subQueryAlias\)](#) - *exists(select ...)* SQL construction
- [HasFlag<T>\(this IQueryStructureAdapter adapter, Expression<Func<IQueryStructureAdapter, T>> expression, Expression<Func<IQueryStructureAdapter, T>> flagValue, Expression<Func<IQueryStructureAdapter, T>> trueValue, Expression<Func<IQueryStructureAdapter, T>> falseValue\)](#) - bitwise mask checking for value
- [SqlExpr<TResult>\(this IQueryStructureAdapter adapter, string format, params Expression<Func<IQueryStructureAdapter, object>>\[\] expressions\)](#) - mixing plain SQL expressions with LINQ style defined expression subparts, parameter *format* is the string.Format mask
- [SqlFn<TResult>\(this IQueryStructureAdapter adapter, string name, params Expression<Func<IQueryStructureAdapter, object>>\[\] parameters\)](#) - SQL function call expression
- [SqlParameter<TResult>\(this IQueryStructureAdapter adapter, string binding, DbType type, params Expression<Func<IProcedureParameterAdapter, IProcedureParameterFlag>>\[\] attributes\)](#) - SQL function parameter value expression

Mapping parameters of SQL expressions is defined by [IRepositoryExpressionMappingAdapter](#) interface methods:

- [NativeSqlType\(string nativeSqlType\)](#) - set native SQL type name of expression result, used when expression result type has specific SQL type
- [Size\(int size\)](#) - set number or varchar length of SQL expression result type
- [Size\(int size, int scale\)](#) - set length and scale for floating point SQL data types

- [Size\(int size, int scale\)](#) - set length and scale for floating point SQL data types

- subquery,

```
...
.ReadOnlyProperty(x => x.DocAmount,
    x => x.Expr<decimal?>(null, DataExpressionType.SubQuery,
        "A", DbType.Decimal, z => z.Size(17, 5)))
...
```

- LINQ expression,

```
...
.ReadOnlyProperty(x => x.AllowedDiscount,
    x => x.Expr<int>(null, y => y.Case<Product, int>(
        z => z.Field<bool>("is_vip"), z => 0, 1.SetExpression(z => 10)),
        DbType.Decimal, z => z.Size(17, 5)))
...
```

- plain text SQL expression.

```
...
.ReadOnlyProperty(x => x.AllowedDiscount,
    x => x.Expr<int>(null, DataExpressionType.PlainSql,
        "case P.is_vip when 1 then 10 else 0 end",
        DbType.Decimal, z => z.Size(17, 5)))
...
```

❏ Important

If possible do not use plain SQL expressions. It possibly use specific SQL dialect features not supported by other DBMS or providers.

Property default value

To set default value of property with dynamic mapping, use one of [IRepositoryPropertyMappingAdapter<T, TValue>](#) interface *Default* method overload:

- [Default\(DefaultType source\)](#) - use with *DefaultType.AutoIncrement*, *DefaultType.CurrentDate*, *DefaultType.CurrentDateTime*, *DefaultType.CurrentDateTimeUtc*, *DefaultType.NewGuid*, *DefaultType.UserName* where default value assigned data row on insert only
- [Default\(DefaultType source, DefaultFeature features\)](#) - use with *DefaultType.AutoIncrement*, *DefaultType.CurrentDate*, *DefaultType.CurrentDateTime*, *DefaultType.CurrentDateTimeUtc*, *DefaultType.NewGuid*, *DefaultType.UserName* where default value assigned data row on insert or update
- [Default\(DefaultType source, TValue value\)](#) - use with *DefaultType.Const* or *DefaultType.Variable* where default value assigned data row on insert only
- [Default\(DefaultType source, TValue value, DefaultFeature features\)](#) - use with *DefaultType.Const* or *DefaultType.Variable* where default value assigned data row on insert or update

```
...
.ReadOnlyProperty(x => x.Changed,
    x => x.Field<DateTime>("D", string.Empty,
        z => z.Default(DefaultType.CurrentDateTime, DefaultFeature.UseOnUpdate)))
...
```

Property grouping parameters

Property grouping parameters used when data object mapped to grouped query to specify property role in GROUP BY expression. Use one of [IPropertyMappingAdapter](#) interface *Group* method overload:

- [Group\(int groupOrder\)](#) - when property mapped to field query grouped by with N-th order

- [Group\(DataGrouping grouping\)](#) - when property mapped to aggregation function over field

```
...
HistoryDate = x.Field<DateTime?>("H", string.Empty,
    z => z.Group(DataGrouping.Max))
...
```

Hidden properties

[Hidden properties](#) dynamically defined using [Column](#) method call of [IRepositoryStructure<T>](#) interface.

Hidden property can be mapped to:

- data source field - [Field<TVal>](#)
- SQL expression (see detailed description [above](#)) - [Expr<TVal>](#)
- inner view property (see detailed description [below](#)) - [Ref<TVal>](#)

```
...
.Column("DocId", x => x.Field<long?>("D", string.Empty,
    z => z.Key(), z => z.Default(DefaultType.AutoIncrement)))
...
```

Inner view reference

To map property to field from inner view used one of [IRepositoryStructureAdapter](#) interface *Ref* method overload:

- [Ref<TValue>\(string sourceAlias, string property, params Expression<Func<IRepositoryPropertyMappingAdapter, IStructureFlag>>\[\] attributes\)](#) - used when referenced property is a [hidden property](#)
- [Ref<TValue, TObj>\(string sourceAlias, Expression<Func<TObj, object>> property, params Expression<Func<IRepositoryPropertyMappingAdapter, IStructureFlag>>\[\] attributes\)](#)

```
.ReadOnlyProperty(x => x.DocLastChange, x => x.Ref<DateTime?>("H", "HistoryDate"))
```

Links

To dynamically define [link property](#) used one of [IRepositoryStructureAdapter<T>](#) interface *Link* method overload:

- [Link<TValue, TSource> Link<TValue, TSource>\(string sourceAlias, params Expression<Func<ILinkPropertyMappingAdapter<T>, IStructureFlag>>\[\] attributes\)](#) - used when source property is correlated with property name with default name mapping rule (*hungarian_notation* in DB and *CamelCaseNotation* property name in code)
- [Link<TValue, TSource> Link<TValue, TSource>\(string sourceAlias, string fieldName, params Expression<Func<ILinkPropertyMappingAdapter<T>, IStructureFlag>>\[\] attributes\)](#)

Parameter *attributes* of method *Link* value used to define link properties pairs with one of [ILinkPropertyMappingAdapter<T>](#) interface *LinkProperties* method overload call:

- [LinkProperty\(\)](#) - used property of source object to associate with link property defined by *Link* method
- [LinkProperty<TSource>\(Expression<Func<TSource, object>> source\)](#) - used *source* as source property and equal named property from target object
- [LinkProperty<TSource>\(Expression<Func<TSource, object>> source, Expression<Func<T, object>> property\)](#) - specifying both properties

```

...
DocState = x.Link<string, DocState>("S", "name",
    z => z.LinkProperty<DocState>(y => y.Name),
    z => z.LinkProperty<DocState>(y => y.Code, y => y.DocStateCode)
    z => z.LinkProperty<DocState>(
        (Invoice y) => y.GetProperty<long>("DocStateId"))),
...

```

Tree

To dynamically map tree organized data structure used special method [With](#) of [RepositoryStructure<T>](#) interface to describe tree root query and tree childs. Both of them use subquery mappings (see example below) to set subquery structure for tree query construction.

❏ Important

[Data source flags](#) of tree organized data sources must contain [DataStructureFlag.ReadOnly](#) and [DataStructureFlag.Tree](#)!

```

private static Expression<CustomMapping<CatalogueTree>> _unused = (
    () => XDataMapping.CustomMapping<CatalogueTree>()
        .With("CAT", XDataMapping.GetStructure("R")
            .DataTable("T_CATALOGUE", "R",
                x => x.Combination("root", Combination.Or),
                x => x.ConstFilter("FilterByCathalogue", "catalogue_id",
                    FilterConstantType.Const, null,
                    FilterNullable.NullsNotAllowed).SetCombination("root"),
                x => x.ConstFilter("FilterByRoot", "parent_id",
                    FilterConstantType.Const, null,
                    FilterNullable.NullsCompared)
                    .SetCombination("root"))
            .Select(x => new {
                CatalogueId = x.Field<long>("R", string.Empty, z => z.Key()),
                Code = x.Field<string>("R", string.Empty),
                Name = x.Field<string>("R", string.Empty),
                ParentId = x.Field<long?>("R", string.Empty),
            })), x => x.Properties(z => z.CatalogueId,
                z => z.Code, z => z.Name, z => z.ParentId),
    x => x.WithRecursive("S", XDataMapping
        .GetStructure("S")
        .DataTable("T_CATALOGUE", "S")
        .Select(y => new {
            CatalogueId = y.Field<long>("S", string.Empty, z => z.Key()),
            Code = y.Field<string>("S", string.Empty),
            Name = y.Field<string>("S", string.Empty),
            ParentId = y.Field<long?>("S", string.Empty)
        })),
        z => z.SubqueryLink(y => y.ParentId, y => y.CatalogueId)))
    .ReadOnlyProperty(x => x.CatalogueId,
        x => x.Field<long>("CAT", string.Empty, z => z.Key()))
    .ReadOnlyProperty(x => x.Code,
        x => x.Field<string>("CAT", string.Empty))
    .ReadOnlyProperty(x => x.Name,
        x => x.Field<string>("CAT", string.Empty))
    .ReadOnlyProperty(x => x.ParentId,
        x => x.Field<long?>("CAT", string.Empty))
    .Map()
    .SetFlags(DataStructureFlag.ReadOnly | DataStructureFlag.Tree)
);

```

External link

[External link](#) can be dynamically defined using expression of type [Expression](#) as part of call [Map](#) method of [IRepositoryStructure](#) interface.

Expression<IExternalLinkDefinition> can be obtained using multiple overloads of *ExternalLink* method of *ExternalLinkDefinitionAdapter<T>*:

- *IExternalLinkDefinition ExternalLink<TChild>(params Expression<Func<IExternalLinkAdapter<T, TChild>, IExternalLinkFlag>>[] linkFlags)*
- *IExternalLinkDefinition ExternalLink<TChild>(Expression<Func<T, object>> property, params Expression<Func<IExternalLinkAdapter<T, TChild>, IExternalLinkFlag>>[] linkFlags)*
- *IExternalLinkDefinition ExternalLink<TChild>(string property, params Expression<Func<IExternalLinkAdapter<T, TChild>, IExternalLinkFlag>>[] linkFlags)*

IExternalLinkAdapter<T, TChild> used to specify properties of external link:

- *Filter* - used to specify external link runtime filter,
- *FilterCombination* - used to set filter group of filter,
- *DirectLink* - used to set direct link flag, see [tree master-detail specifics](#) for details,
- *MasterRefresh* - used to set refresh master repository current row data on submit details changes flag (useful when master has calculated properties over detail repository data),
- *Nullable* - filter null processing behavior,
- *Operation* - link filter operation (see [Filters](#)),
- *PrimaryFilter* - used to specify external link is optional filter,
- *Property* - link property in master repository (used when property is [hidden property](#)),
- *Property<TChildLink>* - link property in master repository (used when property is actual property),
- *TreeFilter* - used to set external link filter's tree filter flag

```
// when external link used hidden property
..., x => x.ExternalLink<InvoiceSpec>("DocId"))
// when external link used read-only or regular property
..., x => x.ExternalLink<InvoiceSpec>(z => z.DocId))
```

Variables

The use of variables is described [here](#).

To describe the variables, special overloads of [Subquery](#) and [InnerView](#) methods are used. Variables are described in the *variables* parameter.

```
.Subquery<DocPropertyValue>("PH", x => x.GetProperty("ParamValue"), DataGrouping.None,
    new[] {"ParamCode".SetVar("AUTHOR_PHONE")}, /* variables */
    x => x.SubqueryLink("DocId"))
```

SQL procedures & functions

Common information about mapping to SQL procedures and functions described in [common mapping rules description](#) and here we just represent dynamic mapping rules for various procedure types.

Warning

SQLite not supported SQL procedures & functions

Warning

PostgreSql functions has restrictions:

- returning refcursor or set of refcursor requires transaction!
- parameters use lower case names ONLY!
- result set names (refcursor names) use lower case names ONLY!

SQL procedure as query

To define procedure as data source of mapping used [Procedure](#) method call of *IRepositoryStructure<T>* interface.

SQL procedure parameter

To define SQL procedure parameter used [Parameter](#) method call of [IRepositoryStructure<T>](#) interface.

SQL procedure result set

To describe result set of SQL procedure used [Return](#) method call of [IRepositoryStructure<T>](#) interface.

* - when result set is not mapped to procedure parameter set result set name to *string.Empty*

SQL procedure result set as query sample

- MS SQL Server

```
CREATE PROCEDURE [dbo].[TestProcedure]
(
    @param1 int,
    @param2 varchar(20)
)
AS
    SELECT @param1 as Id, @param2 as Name
```

- Oracle

```
procedure TestProcedure(p_Id int, p_Name varchar2, p_Out out SYS_REFCURSOR)
as
begin
    open p_Out for select p_Id as Id, p_Name as Name from dual;
end TestProcedure;
```

- PostgreSQL

```
CREATE OR REPLACE FUNCTION public.TestProcedure (p_id integer, p_name varchar, refcursor)
RETURNS refcursor AS
$$
begin
    open $3 for select p_id as Id, p_name as Name;
    return $3;
end
$$ LANGUAGE plpgsql;
```

- MySQL

```
create procedure TestProcedure(p_Id int, p_Name varchar(50))
begin
    select p_Id as Id, p_Name as Name;
end
```

- Dynamic query

```
Expression<Func<IQueryStructureAdapter, PostgreSQLClassifier>> convertor =
    x => new PostgreSQLClassifier { Id = x.Field<int>(z => z.Key()), Name = x.Field<string>() };
var res = XDataMapping.GetStructure("T")
    .Procedure("T", "dbo.TestProcedure", ProcedureType.Procedure)
    .Parameter<int>("T", 1, "param1", DbType.Int32)
    .Parameter<string>("T", 2, "param2", DbType.String)
    .Return("T", string.Empty, 1, convertor)
    .Call<Classifier>(DataScope, string.Empty,
        "param1".SetVar(1), "param2".SetVar("Test"));
foreach (var classifier in res)
    Console.WriteLine("{0} - \"{1}\"", classifier.Id, classifier.Name);
```

- Mapping

```

...
.Procedure("T", "dbo.TestProcedure", ProcedureType.Procedure)
.Parameter<int>("T", 1, "param1", DbType.Int32)
.Parameter<string>("T", 2, "param2", DbType.String)
.Return("T")
.Map(x => new TestProcedure
{
    Id = x.Field<int>("T", string.Empty, z => z.Key()),
    Name = x.Field<string>("T", string.Empty)
})
.SetBaseTable("T")
...

```

SQL procedure with multiple result sets sample

- MS SQL Server

```

CREATE PROCEDURE [dbo].[TestProcedure2]
(
    @param1 int,
    @param2 varchar(20),
    @param3 int,
    @param4 varchar(20)
)
AS
BEGIN
    SELECT @param1 as Id, @param2 as Name
    SELECT @param3 as Id, @param4 as Name
END

```

- Oracle

```

procedure TestProcedure2(p_Id int, p_Name varchar2, p_Out out SYS_REFCURSOR, p_Out2 out SYS_REFCURSOR)
as
begin
    open p_Out for select p_Id as Id, p_Name as Name from dual;
    open p_Out2 for select p_Id as Id, p_Name as Name from dual;
end TestProcedure2;

```

Tip

Use *refcursor* parameters names as *ResultSet* names!

- PostgreSQL

```

CREATE OR REPLACE FUNCTION public.TestProcedure2 (p_id integer, p_name varchar, refcursor, refcursor)
RETURNS SETOF refcursor AS
$$
begin
    open $3 for select p_id as Id, p_name as Name;
    return NEXT $3;
    open $4 for select p_id as Id, p_name as Name;
    return NEXT $4;
end
$$ LANGUAGE plpgsql;

```

Warning

Functions witch returns *setof refcursor* is not supported by Npgsql .Net Standard 2.0 Edition :(XData net4.0 version can call that type functions with the same mapping rules as described here...

Tip

If *refcursor* parameters have a names it must be used as *ResultSet* names!

- MySQL

```
create procedure TestProcedure2(p_Id int, p_Name varchar(50))
begin
    select p_Id as Id, p_Name as Name;
    select p_Id as Id, p_Name as Name;
end
```

- Dynamic query

```
Expression<Func<IQueryStructureAdapter, PostgreSQLClassifier>> convertor =
    x => new PostgreSQLClassifier { Id = x.Field<int>(z => z.Key()), Name = x.Field<string>() };
var res2 = XDataMapping.GetStructure("T")
    .Procedure("T", "dbo.TestProcedure2", ProcedureType.Procedure)
    .Parameter<int>("T", 1, "param1", DbType.Int32)
    .Parameter<string>("T", 2, "param2", DbType.String)
    .Parameter<int>("T", 3, "param3", DbType.Int32)
    .Parameter<string>("T", 4, "param4", DbType.String)
    .Return("T", "Result1", 1, convertor)
    .Return("T", "Result2", 2, convertor)
    .Call<Classifier>(DataScope, "Result1", "param1".SetVar(1),
        "param2".SetVar("Test"), "param3".SetVar(2), "param4".SetVar("Some"));
foreach (var classifier in res2)
    Console.WriteLine("{0} - \"{1}\"", classifier.Id, classifier.Name);
foreach (var classifier in res2.GetResultSet<Classifier>("Result2"))
    Console.WriteLine("{0} - \"{1}\"", classifier.Id, classifier.Name);
```

- Mapping

```
...
.Procedure("T", "dbo.TestProcedure2", ProcedureType.Procedure)
.Parameter<int>("T", 1, "param1", DbType.Int32)
.Parameter<string>("T", 2, "param2", DbType.String)
.Parameter<int>("T", 3, "param3", DbType.Int32)
.Parameter<string>("T", 4, "param4", DbType.String)
.Return("T")
.Return("T", "Result2", 2,
    x => new PostgreSQLClassifier {
        Id = x.Field<int>(z => z.Key()),
        Name = x.Field<string>() })
.Map(x => new TestProcedure2
{
    Id = x.Field<int>("T", string.Empty, z => z.Key()),
    Name = x.Field<string>("T", string.Empty)
})
.SetBaseTable("T")
...
```

SQL procedure with out parameter sample

- MS SQL Server

```
CREATE PROCEDURE [dbo].[TestProcedure3]
(
    @param1 int,
    @param2 varchar(20),
    @param3 varchar(20) OUTPUT
)
AS
SET @param3 = convert(varchar(20), @param1) + ' - ' + @param2
RETURN
```

- Oracle

```

procedure TestProcedure3(p_Id int, p_Name varchar2, p_Out out varchar2)
as
begin
    select p_Id || ' - ' || p_Name into p_Out from dual;
end TestProcedure3;

```

- PostgreSQL

```

CREATE or REPLACE function TestProcedure3(p_id integer, p_name varchar, out p_out varchar) as $$
begin
    p_Out := p_id || ' - ' || p_name;
end
$$ LANGUAGE plpgsql;

```

- MySQL

```

create procedure TestProcedure3(p_Id int, p_Name varchar(50), out p_Out varchar(100))
begin
    set p_Out = CONCAT(p_Id, ' - ', p_Name);
end

```

- Dynamic query

```

var res3 = XDataMapping.GetStructure("T")
    .Procedure("T", "dbo.TestProcedure3", ProcedureType.Procedure)
    .Parameter<int>("T", 1, "param1", DbType.Int32)
    .Parameter<string>("T", 2, "param2", DbType.String)
    .Parameter<string>("T", 3, "param3", DbType.String,
        z => z.Size(20), z => z.Direction(ParameterDirection.Output))
    .Call(DataScope.Layer,
        "param1".SetVar(1), "param2".SetVar("Test"));
Console.WriteLine(res3.GetParameter<string>("param3"));

```

- Mapping

```

...
.Procedure("T", "dbo.TestProcedure3", ProcedureType.Procedure)
.Parameter<int>("T", 1, "param1", DbType.Int32)
.Parameter<string>("T", 2, "param2", DbType.String)
.Parameter<string>("T", 3, "param3", DbType.String,
    z => z.Size(20), z => z.Direction(ParameterDirection.Output))
.Map()
.SetBaseTable("T")
...

```

SQL procedure with out parameter and result set sample

- MS SQL Server

```

CREATE PROCEDURE [dbo].[TestProcedure4]
(
    @param1 int,
    @param2 varchar(20)
)
AS
SELECT @param1 as Id, @param2 as Name
RETURN(@param1)

```

- Oracle

```

procedure TestProcedure4(p_Tab in classifier_tab, p_Out out SYS_REFCURSOR)
as
begin
    open p_Out for select Id, Name from table(p_Tab);
end TestProcedure4;

```

❗ Tip

Use *refcursor* parameters names as *ResultSet* names!

- PostgreSQL

```

CREATE OR REPLACE FUNCTION public.TestProcedure4 (p_tab public.classifier_rec [], refcursor)
RETURNS refcursor AS
$$
begin
    open $2 for select Id, Name from unnest(p_tab);
    return $2;
end
$$
LANGUAGE plpgsql;

```

❗ Tip

If *refcursor* parameters have a names it must be used as *ResultSet* names!

- MySQL

Not supported

- Dynamic query

```

var res4 = XDataMapping.GetStructure("T")
    .Procedure("T", "dbo.TestProcedure4", ProcedureType.Procedure)
    .Parameter<int>("T", 1, "param1", DbType.Int32)
    .Parameter<string>("T", 2, "param2", DbType.String)
    .Parameter<int>("T", 3, "result", DbType.Int32,
        z => z.Direction(ParameterDirection.ReturnValue))
    .Return("T", string.Empty, 1, x => new Classifier {
        Id = x.Field<int>(z => z.Key()),
        Name = x.Field<string>() })
    .Call<Classifier>(DataScope, string.Empty,
        "param1".SetVar(1), "param2".SetVar("Test"));
Console.WriteLine(res4.GetParameter<int>("result"));
foreach (var classifier in res4)
    Console.WriteLine("{0} - \"{1}\"", classifier.Id, classifier.Name);

```

- Mapping

```

...
.Procedure("T", "dbo.TestProcedure4", ProcedureType.Procedure)
.Parameter<int>("T", 1, "param1", DbType.Int32)
.Parameter<string>("T", 2, "param2", DbType.String)
.Parameter<int>("T", 3, "result", DbType.Int32,
    z => z.Direction(ParameterDirection.ReturnValue))
.Return("T")
.Map(x => new TestProcedure4
{
    Id = x.Field<int>("T", string.Empty, z => z.Key()),
    Name = x.Field<string>("T", string.Empty)
})
.SetBaseTable("T")

```

SQL procedure with return value sample

- MS SQL Server

```
CREATE PROCEDURE [dbo].[TestProcedure5]
(
    @param1 int,
    @param2 varchar(20)
)
AS
RETURN(@param1)
```

- Oracle

Not supported

- PostgreSQL

Not supported

- MySQL

Not supported

- Dynamic query

```
var res5 = XDataMapping.GetStructure("T")
    .Procedure("T", "dbo.TestProcedure5", ProcedureType.Procedure)
    .Parameter<int>("T", 1, "param1", DbType.Int32)
    .Parameter<string>("T", 2, "param2", DbType.String)
    .Parameter<int>("T", 3, "result", DbType.Int32,
        z => z.Direction(ParameterDirection.ReturnValue))
    .Call<Classifier>(DataScope.Layer, string.Empty,
        "param1".SetVar(1), "param2".SetVar("Test"));
Console.WriteLine(res5.GetParameter<int>("result"));
```

- Mapping

```
...
.Procedure("T", "dbo.TestProcedure5", ProcedureType.Procedure)
.Parameter<int>("T", 1, "param1", DbType.Int32)
.Parameter<string>("T", 2, "param2", DbType.String)
.Parameter<int>("T", 3, "result", DbType.Int32,
    z => z.Direction(ParameterDirection.ReturnValue))
.Map()
.SetBaseTable("T")
...
public int ResultParameter
    { get { return this.GetParameter<int>("result"); } }
```

SQL procedure with return value and result set sample

- MS SQL Server

- MS SQL Server

```
CREATE PROCEDURE [dbo].[TestProcedure6]
(
    @param1 int,
    @param2 varchar(20)
)
AS
SELECT @param1 as Id, @param2 as Name
RETURN(@param1)
```

- Oracle

Not supported

- PostgreSQL

Not supported

- MySQL

Not supported

- Dynamic query

```
var res6 = XDataMapping.GetStructure("T")
    .Procedure("T", "dbo.TestProcedure6", ProcedureType.Procedure)
    .Parameter<int>("T", 1, "param1", DbType.Int32)
    .Parameter<string>("T", 2, "param2", DbType.String)
    .Parameter<int>("T", 3, "result", DbType.Int32,
        z => z.Direction(ParameterDirection.ReturnValue))
    .Return("T", string.Empty, 1, x => new Classifier {
        Id = x.Field<int>(z => z.Key()), Name = x.Field<string>() })
    .Call<Classifier>(DataScope.Layer, string.Empty, "param1".SetVar(param));
foreach (var classifier in res6)
    Console.WriteLine("{0} - \"{1}\"", classifier.Id, classifier.Name);
```

- Mapping

```
...
.Procedure("T", "dbo.TestProcedure6", ProcedureType.Procedure)
.Parameter<int>("T", 1, "param1", DbType.Int32)
.Parameter<string>("T", 2, "param2", DbType.String)
.Parameter<int>("T", 3, "result", DbType.Int32,
    z => z.Direction(ParameterDirection.ReturnValue))
.Return("T")
.Map(x => new TestProcedure6
{
    Id = x.Field<int>("T", string.Empty, z => z.Key()),
    Name = x.Field<string>("T", string.Empty)
})
.SetBaseTable("T")
...
public int ResultParameter
{ get { return this.GetParameter(x => x.ResultParameter); } }
```

SQL procedure with simple array parameter sample

- MS SQL Server

Not supported

- Oracle

Not supported

- PostgreSQL

```
CREATE or REPLACE function TestFunction6(p_arr INTEGER[]) returns integer as $$
begin
    return sum(s) from unnest(p_arr) s;
end
$$ LANGUAGE plpgsql;
```

- MySQL

Not supported

- Dynamic query

```
var arr = new[] { 1, 2, 3 };
var f6 = XDataMapping.GetStructure("T")
    .Procedure("T", "TestFunction6", ProcedureType.Function)
    .Parameter<int[]>("T", 1, "p_arr", DbType.Int32, z => z.Array())
    .Parameter<int>("T", 2, "result", DbType.Int32,
        z => z.Direction(ParameterDirection.ReturnValue))
    .Call(DataScope, "p_arr".SetVar(arr));
Console.WriteLine(f6.GetParameter<int>("result"));
```

- Mapping

```
...
.Procedure("T", "TestFunction6", ProcedureType.Function)
.Parameter<int[]>("T", 1, "p_arr", DbType.Int32, z => z.Array())
.Parameter<int>("T", 2, "result", DbType.Int32,
    z => z.Direction(ParameterDirection.ReturnValue))
...
```

SQL procedure with UDT parameter sample

- MS SQL Server

Not supported

Tip

MS SQL Server does not support passing single UDT object as a parameter value, but support array of UDT

- Oracle

Not supported

Warning

Oracle ODP.Net provider does not support UDT

- PostgreSQL

```
CREATE TYPE public.classifier_rec AS ( Id integer, Name varchar(20) );

CREATE or REPLACE function TestFunction5(p_obj classifier_rec) returns varchar as $$
begin
    return p_obj.Id || ' - ' || p_obj.Name;
end
$$ LANGUAGE plpgsql;
```

- MySQL

Not supported

- Dynamic query

```

/* Type mapped to UDT required to be Xml serializable */
[Serializable]
public class Classifier
{
    [XmlAttribute]
    public int Id { get; set; }
    [XmlAttribute]
    public string Name { get; set; }
}
...
var f5 = XDataMapping.GetStructure("T")
    .Procedure("T", "TestFunction5", ProcedureType.Function)
    .Parameter<Classifier>("T", 1, "p_obj", DbType.Object,
        z => z.UdtDataType("classifier_rec"))
    .Parameter<string>("T", 3, "result", DbType.String,
        z => z.Size(20),
        z => z.Direction(ParameterDirection.ReturnValue))
    .Call(DataScope, "p_obj".SetVar(
        new Classifier { Id = 1, Name = "Test" }));
Console.WriteLine(f5.GetParameter<string>("result"));

```

- Mapping

```

/* Type mapped to UDT required to be Xml serializable */
[Serializable]
public class Classifier
{
    [XmlAttribute]
    public int Id { get; set; }
    [XmlAttribute]
    public string Name { get; set; }
}
...
.Procedure("T", "TestFunction5", ProcedureType.Function)
.Parameter<Classifier>("T", 1, "p_obj", DbType.Object,
    z => z.UdtDataType("classifier_rec"))
.Parameter<string>("T", 3, "result", DbType.String,
    z => z.Size(20),
    z => z.Direction(ParameterDirection.ReturnValue))
.Map()
.SetBaseTable("T")
...
public class TestFunction5 : IDataObject
{
    public string Result => this.GetParameter(x => x.Result);
}

```

SQL procedure with UDT array parameter sample

- MS SQL Server

```

CREATE TYPE [dbo].[Classifier] AS TABLE(
    [Id] [int] NULL,
    [Name] [varchar](20) NULL
)

CREATE PROCEDURE [dbo].[TestProcedure7] (@param1 Classifier READONLY)
AS
SELECT * from @param1

```

- Oracle

Not supported

Warning

Oracle ODP.Net provider does not support UDT

- PostgreSQL

```
CREATE TYPE public.classifier_rec AS ( Id integer, Name varchar(20) );

CREATE or REPLACE function TestProcedure7(p_tab classifier_rec[]) returns setof classifier_rec as $$
    select a.Id, a.Name from unnest(p_tab) a;
$$ LANGUAGE sql;
```

- MySQL

Not supported

- Dynamic query

```
/* Type mapped to UDT required to be Xml serializable */
[Serializable]
public class Classifier
{
    [XmlAttribute]
    public int Id { get; set; }
    [XmlAttribute]
    public string Name { get; set; }
}
...
var param = new[] { new Classifier { Id = 1, Name = "Test" },
    new Classifier { Id = 2, Name = "Some" } };
var f4 = XDataMapping.GetStructure("T")
    .Procedure("T", "dbo.TestProcedure7", ProcedureType.Procedure)
    .Parameter<Classifier[]>("T", 1, "param1", DbType.Object,
        z => z.UdtDataType(null, "dbo.Classifier"))
    .Return("T")
    .Call(DataScope, string.Empty, "param1".SetVar(param));
foreach (var classifier in res)
    Console.WriteLine("{0} - \"{1}\"", classifier.Id, classifier.Name);
```

- Mapping

```
/* Type mapped to UDT required to be Xml serializable */
[Serializable]
public class Classifier
{
    [XmlAttribute]
    public int Id { get; set; }
    [XmlAttribute]
    public string Name { get; set; }
}
...
.Procedure("T", "dbo.TestProcedure7", ProcedureType.Procedure)
.Parameter<Classifier[]>("T", 1, "param1", DbType.Object,
    z => z.UdtDataType(null, "dbo.Classifier"))
.Return("T")
.Map(x => new TestProcedure
{
    Id = x.Field<int>("T", string.Empty, z => z.Key()),
    Name = x.Field<string>("T", string.Empty)
})
.SetBaseTable("T")
```

SQL table valued function as query sample

- MS SQL Server

```
CREATE FUNCTION [dbo].[TestFunction]
(
    @param1 int,
    @param2 varchar(20)
)
RETURNS TABLE
AS
RETURN
(
    SELECT @param1 as Id, @param2 as Name
)
```

- Oracle

```
CREATE OR REPLACE type classifier_rec is object (Id int, Name varchar2(20));
CREATE OR REPLACE type classifier_tab is table of classifier_rec;

function TestFunction(p_Id int, p_Name varchar2) return classifier_tab pipelined
is
begin
for curr in (select p_Id as Id, p_Name as Name from dual) loop
    pipe row (classifier_rec(curr.Id, curr.Name));
end loop;
end TestFunction;
```

- PostgreSQL

```
CREATE or REPLACE function TestFunction(p_id integer, p_name varchar)
returns TABLE (
    Id bigint,
    Name varchar
) as $$
    select p_id as Id, p_name as Name;
$$ LANGUAGE SQL;
```

... or with the same mapping syntax...

```
CREATE TYPE public.classifier_rec AS ( Id integer, Name varchar(20) );

CREATE or REPLACE function TestFunction(p_Id integer, p_Name varchar) returns setof classifier_rec as $$
    select p_Id as Id, p_Name as Name;
$$ LANGUAGE SQL;
```

- MySQL

Not supported

- Dynamic query

```
var f3 = XDataMapping.GetStructure()
    .Procedure("T", "dbo.TestFunction", ProcedureType.Function)
    .Parameter<int>("T", 1, "param1", DbType.Int32)
    .Parameter<string>("T", 2, "param2", DbType.String)
    .Return("T")
    .Select(x => new Classifier {
        Id = x.Field<int>("T", "Id", z => z.Key()),
        Name = x.Field<string>("T", "Name") })
    .AsQuery(DataScope, "param1".SetVar(param));
foreach (var classifier in f3)
    Console.WriteLine("{0} - \"{1}\"", classifier.Id, classifier.Name);
```

- Dynamic query table function with joined table

```
var f1Join = XDataMapping.GetStructure()
    .DataTable("T_DOC_TYPE", "D")
    .Procedure("T", "dbo.TestFunction", ProcedureType.Function,
        z => z.Link("D", "doc_type_id", "Id"))
    .Parameter<int>("T", 1, "param1", DbType.Int32)
    .Parameter<string>("T", 2, "param2", DbType.String)
    .Return("T")
    .Select(x => new {
        Id = x.Field<int>("T", "Id", z => z.Key()),
        Name = x.Field<string>("T", "Name"),
        Code = x.Field<string>("D", string.Empty)
    })
    .AsQuery(DataScope, "param1".SetVar(1), "param2".SetVar("Test"));
foreach (var res in f1Join)
    Console.WriteLine("{0} - \"{1}\" - \"{2}\"", res.Id, res.Name, res.Code);
```

- Mapping

```
...
.Procedure("T", "dbo.TestFunction", ProcedureType.Function)
.Parameter<int>("T", 1, "param1", DbType.Int32)
.Parameter<string>("T", 2, "param2", DbType.String)
.Return("T")
.Map(x => new TestFunction
{
    Id = x.Field<int>("T", "Id", z => z.Key()),
    Name = x.Field<string>("T", "Name")
})
.SetBaseTable("T")
```

- Mapping table function with joined table

```
.DataTable("T_DOC_TYPE", "D")
.Procedure("T", "dbo.TestFunction", ProcedureType.Function,
    z => z.Link("D", "doc_type_id", "Id"))
.Parameter<int>("T", 1, "param1", DbType.Int32)
.Parameter<string>("T", 2, "param2", DbType.String)
.Return("T")
.Map(x => new TestFunctionJoin
{
    Id = x.Field<int>("T", "Id", z => z.Key()),
    Name = x.Field<string>("T", "Name"),
    Code = x.Field<string>("D", string.Empty)
})
.SetBaseTable("D")
```

SQL scalar function as property data source sample

- MS SQL Server

```
CREATE FUNCTION [dbo].[TestFunction2]
(
    @param1 int,
    @param2 varchar(20)
)
RETURNS varchar(20)
AS
BEGIN
    DECLARE @res varchar(20)
    SET @res = convert(varchar(20), @param1) + ' - ' + @param2
    RETURN @res
END
```

- Oracle

```
function TestFunction2(p_Id int, p_Name varchar2) return varchar2
is
begin
    return p_Id || ' - ' || p_Name;
end TestFunction2;
```

- PostgreSQL

```
CREATE or REPLACE function TestFunction2(p_id integer, p_name varchar) returns varchar as $$
DECLARE
ret varchar;
begin
    select p_id || ' - ' || p_name into ret;
    return ret;
end
$$ LANGUAGE plpgsql;
```

- MySQL

```
create function TestFunction2(p_Id int, p_Name varchar(20)) returns varchar(100)
begin
    declare res varchar(100);
    select CONCAT(p_Id, ' - ', p_Name) into res;
    return(res);
end
```

- Mapping

```
var t1 = XDataMapping.GetStructure("T")
.DataTable("T_DOC_TYPE", "T").Select(x => new
{
    DocTypeId = x.Field<long>("T", "doc_type_id", z => z.Key()),
    Code = x.Field<string>("T", "code"),
    Name = x.Field<string>("T", "name"),
    Test = x.Expr("test", z => z.SqlFn<string>("dbo.TestFunction2",
        y => y.Field<long>("T", "doc_type_id"), y => y.Field<string>("T", "name")), DbType.String, z =>
z.Size(20))
}).AsQuery(DataScope);
foreach (var z in t1)
    Console.WriteLine("{0} - \"{1}\" - \"{2}\" - \"{3}\"", z.DocTypeId, z.Code, z.Name, z.Test);
```


Read only dynamic query building

XData have a feature allow the programmer to build read-only data query in dynamic LINQ like style:

```
var query = XDataMapping.GetStructure("D")
    .DataTable("T_DOC", "D",
        x => x.DictFilter("T_DOC_TYPE", "doc_type_id", "code", "INVOICE"))
    .DataTable("T_DOC_NUMBER", "N", x => x.Link("D", "doc_id"))
    .DataTable("T_DOC_SCAN", "SC", x => x.Link("D", "doc_id"))
    .DataTable("T_DOC_DELIVERY", "DD", x => x.Link("D", "doc_id"))
    .DataTable("T_DOC_SOURCE", "SR", x => x.Link("D", "doc_id"))
    .Subquery("A", typeof(DocSpecAmounts), "Amount",
        DataGrouping.Sum, x => x.SubqueryLink("DocId"))
    .InnerView("H", XDataMapping.GetStructure("H")
        .DataTable("T_DOC_HISTORY", "H")
        .Select(x => new {
            DocId = x.Field<int?>("H", string.Empty,
                z => z.Key(), z => z.Group(DataGrouping.None)),
            HistoryDate = x.Field<DateTime?>("H", string.Empty,
                z => z.Group(DataGrouping.Max))
        }), x => x.SubqueryLink(z => z.DocId)
        .SetOperation(FilterOperation.OuterJoin))
    .Column("DocId", x => x.Field<int?>("D", "doc_id", z => z.Key()))
    .Select(x => new {
        Number = x.Field<string>("N", "numb"),
        Date = x.Field<DateTime>("D", "doc_date"),
        Amount = x.Expr<decimal?>("doc_amount", DataExpressionType.SubQuery,
            "A", DbType.Decimal, z => z.Size(17, 5)), //SUBQUERY
        LastChange = x.Ref<object>("H", "HistoryDate"), //INNER VIEW
        Scan = x.Field<byte[]>("SC", string.Empty), //BLOB
        DeliveryType = x.Field<DeliveryTypeEnum>("DD", string.Empty), //ENUM
        Source = x.Field<string>("SR", string.Empty) //XML
    }).AsQuery(dataScope);
foreach (var x in query)
    Console.WriteLine(
        "Number:{0}, Date:{1}, Amount:{2}, Scan:{3}, LastChange:{4}, DeliveryType:{5}, Source:{6}",
        x.Number, x.Date, x.Amount, Encoding.UTF8.GetString(x.Scan), x.LastChange, x.DeliveryType,
        x.Source);
```

Static class XDataMapping has method [GetStructure](#) to obtain [IQueryStructure<T>](#) interface having various methods to describe very complicated query structure dynamically. Rules of building queries dynamically is similar as [dynamic mapping](#) but query more lightweight, safe and can be extended in runtime.

Method GetStructure has overloaded version to get structure of existing repository to extend it with adding data sources as [data tables](#), [subqueries](#) and [inner views](#), applying [filters](#) including filters by subqueries. Query mapping logic can be declared by [Select](#) method returning [IQueryDescription<T>](#) interface. No matter [statically](#) or [dynamically](#) is mapped repository we use as source structure, it can be extended.

To access query data get [IQueryable](#) implementation [IDataQuery<T>](#) from [IQueryDescription<T>](#) use [AsQuery](#) method. With dynamic query You can query and map data to any object including real-time typed object as showed in example.

Dynamic mapping has no method to add filters in runtime. To add new filter to *QueryStructure* used method [AddFilters](#).

Runtime filters

Runtime filters used for filtering data by field (or SQL expression) where filter value will be accessible during runtime only. Similar results can be reached using LINQ, but may be field to filter on is not selected. Or the result we need is not a read-only dynamic dataset (see [LINQ support](#) and [Dynamic queries](#)), but feature rich [repository](#), only filtered.

Runtime filters in XData are defined as static fields of [data object](#) typed as [RuntimeFilter](#). Name of runtime filter is mast to be unique over data object mapping structure (with subqueries). When value of runtime filter is set and filter with same name is not found in mapping, it will be dynamically added to mapping structure.

To set value of runtime filter, use [SetFilterValue](#) method of [repository](#). To clear one runtime filter value, set *null* value. To reset all runtime filters, use [Reset](#) method of repository.

Tip

When runtime filter value is set to *null* check [filter null processing behavior](#) assigned to this filter!

```
var rep = invoice.GetRepository();
var hist = dataScope.GetRepository<DocHistory>(rep.Layer, context: rep.Context)
    .Reset()
    .SetFilterValue(DocHistory.FilterByDocId, invoice.GetProperty<long>("DocId"))
    .SetFilterValue(DocHistory.FilterByDocStateId, invoice.DocState.Key);
var newHist = hist.New();
newHist.HistoryDate = DateTime.UtcNow;
return hist.Submit(ref newHist);
```

Runtime filters can be described on place:

```
//Using column name
.SetFilterValue("DocId", doc_id)
//Using property expression
.SetFilterValue(x => DocNumber, docNumber)
//Using FilterName helper
.SetFilterValue(new FilterName("FilterByDocId"), doc_id)
```

SQL block

XData has the ability to describe and execute an anonymous SQL block in LINQ style. To call an SQL block that does not return a result set, you can call it using the [ExecuteSqlBlock](#) extension method, and [GetSqlBlockData<T>](#) can be used to get a collection of objects of type T.

An expression of type [Expression<SqlBlockDefinition>](#) is used to describe the SQL block and an expression of type [Expression<SqlBlockDefinition<T>>](#) is used as the value of the block parameter of the methods described above to return a result of type T. These types are delegates of the functions parameterized by [ISqlBlockAdapter](#) and [ISqlBlockAdapter<T>](#), respectively.

An array called parameters is used to describe the parameter values of an unnamed SQL block.

Lets explain in more detail with specific examples:

- **to DELETE SQL block...**

```
delete T from T_TEST T where T.test2_id = 1
```

```
dataScope.ExecuteSqlBlock(x => x.Delete(t => t.GetStructure("T",  
    y => y.DataTable("T_TEST", "T",  
        a => a.ConstFilter("FilterByTest2Id", "test2_id", 1)))));
```

- **to INSERT VALUES SQL block...**

```
insert into T_TEST(test2_id, date) values(@id, @date)
```

```
dataScope.ExecuteSqlBlock(x => x.Insert("T_TEST", z => new {  
    TestId = z.Value(y => y.Default<long>(DefaultType.AutoIncrement)),  
    Test2Id = z.Value("test2_id", y => y.Data<long>("id")),  
    DateField = z.Value(y => y.Data<DateTime>("date_param")) }},  
    x => x.Param<long>("id", DbType.Int32).Set(1),  
    x => x.Param<DateTime>("date_param", DbType.DateTime).Set(DateTime.Now));
```

Tip

Note that the key auto-increment field is described separately here. For some DBMSs (such as Oracle for example), this is important. But for the rest, this does not interfere with the SQL block.

- **to INSERT MULTIPLE VALUES SQL block...**

```
insert into T_TEST(test2_id, date)  
values(1, '2017-01-01'),  
(1, '2017-01-02'),  
(2, '2017-01-03')
```

```

using (var table = new DataTable())
{
    table.Columns.AddRange(new[] { new DataColumn("id", typeof(long)),
                                    new DataColumn("date_param", typeof(DateTime)) });
    table.Rows.Add(1, DateTime.Parse("2017-01-01"));
    table.Rows.Add(1, DateTime.Parse("2017-01-02"));
    table.Rows.Add(2, DateTime.Parse("2017-01-03"));
    table.AcceptChanges();

    dataScope.ExecuteSqlBlock(
        x => x.Insert("T_TEST", table, z => new {
            TestId = z.Value(y => y.Default<long>(DefaultType.AutoIncrement)),
            Test2Id = z.Value("test2_id", y => y.Data<long>("id")),
            DateField = z.Value(y => y.Data<DateTime>("date_param")) }));
}

```

- **to INSERT INTO SELECT SQL block...**

```

insert into T_TEST(test2_id, date_field)
select 2 as id, dateadd(day, -4, X.date_field) as date_val
from T_TEST X
where test2_id = 1

```

```

dataScope.ExecuteSqlBlock(
    x => x.Insert("T_TEST",
        t => t.GetStructure("X", y => y.DataTable("T_TEST", "X",
            z => z.ConstFilter("FilterByTest2Id", "test2_id", 1)))
        .Select(z => new {
            //!!! Auto increment primary key description is required
            //!!! here only for Oracle and leads to an error for
            //!!! others!
            TestId = z.Default<long>(DefaultType.AutoIncrement),
            Test2Id = z.Expr("id", s => 2, DbType.Int64),
            DateField = z.Expr("date_val",
                s => s.Field<DateTime>("X",
                    "date_field").AddDays(-4), DbType.DateTime) }));

```

⚠ Warning

Oracle has required explicit description of auto increment primary key in that sort SQL block and other RDBMS threat that as error. Due to some features of the implementation at the moment, it was not possible to reduce this to a single syntax. Perhaps this will change further.

- **to UPDATE SQL block...**

```

update T_TEST T
set date_field = @date
where T.test2_id in (select T2.test2_id from T_TEST2 T2 where T2.some_field = 2)

```

```

dataScope.ExecuteSqlBlock(
    x => x.Update(t => t.GetStructure("T", y => y.DataTable("T_TEST", "T",
        z => z.SubqueryFilter("test2_id", "X")
        .SetOperation(FilterOperation.In))
        .Subquery("X", XDataMapping.GetStructure("T2", null,
            DataStructureFlag.ReadOnly)
            .DataTable("T_TEST2", "T2",
                z => z.ConstFilter("SomeFilter", "some_field",
                    FilterConstantType.Const, 2))
        .Select(z => new { Test2Id = z.Field<long>("T2", "test2_id") },
            "Test2Id", DataGrouping.None)),
        z => new { DateField = z.Value(y => y.Data<DateTime>("date_param")) },
    x => x.Param<DateTime>("date_param", DbType.DateTime).Set(DateTime.Now.AddDays(-8)));

```

- **to UPDATE FROM SUBQUERY SQL block...**

```
update T_TEST T
set date_field = (select min(Z.date_field) from T_TEST Z where Z.test2_id = 2)
where T.test2_id in (select T2.test2_id from T_TEST2 T2 where T2.some_field = 2)
```

```
dataScope.ExecuteSqlBlock(
    x => x.Update(t => t.GetStructure("T", y => y.DataTable("T_TEST", "T",
        z => z.SubqueryFilter("test2_id", "X")
        .SetOperation(FilterOperation.In))
        .Subquery("X", XDataMapping.GetStructure("T2", null,
            DataStructureFlag.ReadOnly)
            .DataTable("T_TEST2", "T2",
                z => z.ConstFilter("SomeFilter", "some_field",
                    FilterConstantType.Const, 2))
            .Select(z => new {
                Test2Id = z.Field<long>("T2", "test2_id") }),
                "Test2Id", DataGrouping.None)),
        z => new {
            DateField = z.Value(y => y.Subquery(xa =>
                xa.GetStructure("Z", ya => ya.DataTable("T_TEST", "Z",
                    za => za.ConstFilter("FkFilter", "test2_id",
                        FilterConstantType.Const, minT2 + 1))
                    .Column("TestId", f => f.Field<long>("Z", "test_id",
                        k => k.Key(), k => k.Hidden()))),
                    za => za.Field<DateTime>("Z", "date_field",
                        ya => ya.Group(DataGrouping.Min)),
                    DataStructureFlag.Grouping)) })),
```

Note

It is necessary to consider the fact that in MySQL there is a restriction that you cannot access the same table or subqueries based on it during update.

- **to UPDATE JOINED TABLE SQL block...**

```
update T set date = @date from T_TEST T
join T_TEST2 T2 on T.test2_id = T2.test2_id
where T2.some_field = 1
```

```
dataScope.ExecuteSqlBlock(
    x => x.Update(t => t.GetStructure("T", y => y.DataTable("T_TEST", "T")
        .DataTable("T_TEST2", "T2",
            z => z.Link("T", "test2_id"),
            z => z.ConstFilter("SomeFilter", "some_field",
                FilterConstantType.Const, 1))),
    z => new { DateField = z.Value(y => y.Data<DateTime>("date_param")) }),
    x => x.Param<DateTime>("date_param", DbType.DateTime).Set(DateTime.Now.AddDays(-8)));
```

Note

It is necessary to consider the fact that in MySQL there is a restriction that you cannot access the same table or subqueries based on it during update.

Warning

Not supported in SQLite

- **to UPDATE SET FIELD VALUE FROM JOINED TABLE SQL block...**

```
update T set some_field = T2.some_field from T_TEST2 T
join T_TEST2 T2 on T.test2_id = T2.test2_id
where T2.some_field = 1
```

```
dataScope.ExecuteSqlBlock(
    x => x.Update(t => t.GetStructure("T", y => y.DataTable("T_TEST2", "T")
        .DataTable("T_TEST2", "T2",
            z => z.Link("T", "test2_id"),
            z => z.ConstFilter("SomeFilter", "some_field",
                FilterConstantType.Const, 1))),
    z => new { SomeField = z.Value(y => y.Field<int>("T2", "some_field")) }));
```

Note

It is necessary to consider the fact that in MySQL there is a restriction that you cannot access the same table or subqueries based on it during update.

Warning

Not supported in SQLite

- **to MERGE TABLE WITH VALUES...**

```
merge T_TEST2 T
using (select @test2_id, @some) S (test2_id, some_field)
on (T.test2_id = S.test2_id)
when matched then update set some_field = S.some_field
when not matched then insert(some_field) values(@some)
```

```
dataScope.ExecuteSqlBlock(
    x => x.Merge("T_TEST2", "T", "S", z => new {
        Test2Id = z.Field("test2_id", y => y.Data<long>("Test2Id"),
            y => y.Insert(DefaultType.AutoIncrement), y => y.Link()),
        SomeField = z.Field(y => y.Data<int>("Some"),
            y => y.Update(), y => y.Insert())    }),
    x => x.Param<int>("Test2Id", DbType.Int64).Set(3),
    x => x.Param<int>("Some", DbType.Int32).Set(3));
```

Warning

Not supported in SQLite

- **to MERGE TABLE WITH ANOTHER TABLE...**

```
merge T_TEST2 T
using (select doc_state_id DS_doc_state_id, 1 some_field
    from T_DOC_STATE DS where DS.doc_state_id=1) S
(doc_state_id, some_field)
on (T.test2_id = S.doc_state_id)
when matched then update set some_field = @some
when not matched then insert(some_field) values(@some)
```

```

dataScope.ExecuteSqlBlock(
    x => x.Merge("T_TEST2", "T",
        z => z.GetStructure("DS", y => y.DataTable("T_DOC_STATE", "DS",
            f => f.ConstFilter("FilterById", "doc_state_id",
                FilterConstantType.Const, 1)))
        .Select(s => new {
            DocStateId = s.Field<long>("DS", "doc_state_id"),
            //!!! PostgreSql and MySql don't support merge statement
            //!!! only insert with conflict handling so second field
            //!!! leads to error but required for MS SQL Server and
            //!!! Oracle!
            SomeField = s.Expr<int>("some_field",
                DataExpressionType.PlainSql, "1", DbType.Int32) }),
    "S",
    z => new {
        Test2Id = z.Field("test2_id",
            y => y.Field<long>("S", "doc_state_id"),
            y => y.Insert(DefaultType.AutoIncrement),
            y => y.Link()),
        SomeField = z.Field(y => y.Data<int>("Some"),
            y => y.Update(), y => y.Insert()) },
    x => x.Param<int>("Some", DbType.Int32).Set(4));

```

⚠ Warning

PostgreSql and MySql don't support merge statement only insert with conflict handling so second field leads to error but required for MS SQL Server and Oracle!

⚠ Warning

Not supported in SQLite

• to IF BLOCK...

```

if @id >= 1
begin
    insert into T_TEST(test2_id, date_field) values(@id, @date_param)
end

```

```

dataScope.ExecuteSqlBlock(
    x => x.If(z => z.Data<long>("id"),
        z => z.Const(1, DbType.Int64), FilterOperation.GreaterThanOrEqual,
        s => s.Insert("T_TEST", z => new {
            TestId = z.Value(y => y.Default<long>(DefaultType.AutoIncrement)),
            Test2Id = z.Value("test2_id", y => y.Data<long>("id")),
            DateField = z.Value(y => y.Data<DateTime>("date_param")) })),
    x => x.Param<long>("id", DbType.Int64).Set(1),
    x => x.Param<DateTime>("date_param", DbType.DateTime).Set(DateTime.Now));

```

⚠ Warning

Unnamed procedure SQL blocks are not supported in SQLite and MySQL

• to IF EXISTS...

```

if not exists(select 1 from T_TEST where test2_id = @id)
begin
    insert into T_TEST(test2_id, date_field) values(@id, @date_param)
end

```

```
dataScope.ExecuteSqlBlock(
    x => x.If(t => t.GetStructure("T", y => y.DataTable("T_TEST", "T",
        z => z.ConstFilter("FilterByTest2", "test2_id",
            FilterConstantType.Variable, "id"))),
        FilterOperation.NotExists,
        s => s.Insert("T_TEST", z => new {
            TestId = z.Value(y => y.Default<long>(DefaultType.AutoIncrement)),
            Test2Id = z.Value("test2_id", y => y.Data<long>("id")),
            DateField = z.Value(y => y.Data<DateTime>("date_param")) })),
    x => x.Param<long>("id", DbType.Int64).Set(1),
    x => x.Param<DateTime>("date_param", DbType.DateTime).Set(DateTime.Now));
```

Warning

Unnamed procedure SQL blocks are not supported in SQLite and MySQL

- **some other sample**

```
declare @test table(test_id int, test2_id int, date_field datetime),
    @pid int, @pfk int, @pdate datetime;
begin
    select test_id, test2_id, date_field into @test from T_TEST;
    declare test_cursor cursor for select test_id, test2_id, date_field
        from @test where test2_id = 2;
    open test_cursor;
    fetch next from test_cursor into @pid, @pfk, @pdate;
    while @@fetch_status = 0
    begin
        fetch next from test_cursor into @pid, @pfk, @pdate;
    end
    close test_cursor;
    deallocate test_cursor;
    select test_id, test2_id, date_field from @test;
end
```



```

var sqlBlockData = dataScope.GetSqlBlockData<TestObj>(
    b => b.Block<TestObj>(<
        x => x.Declare(
            z => z.Var<DataTable>("test",
                y => y.Field("test_id", DbType.Int64),
                y => y.Field("test2_id", DbType.Int64),
                y => y.Field("date_field", DbType.DateTime)),
            z => z.Var<long>("pid", DbType.Int64),
            z => z.Var<long>("pfk", DbType.Int64),
            z => z.Var<DateTime>("pdate", DbType.DateTime)),
        x => x.SetVar(t => t.GetStructure("T",
            y => y.DataTable("T_TEST", "T"))
            .Select(z => new {
                TestId = z.Field<long>("T", string.Empty),
                Test2Id = z.Field<long>("T", "test2_id"),
                DateField = z.Field<DateTime>("T", string.Empty)
            }), z => z.Var<DataTable>("test")),
        x => x.ForEach(t => t.GetStructure("T",
            y => y.DataTable(t.Var("test"), "T",
                z => z.ConstFilter("FilterByTest2", "test2_id",
                    FilterConstantType.Const, 2)))
            .Select(z => new {
                TestId = z.Field<long>("T", string.Empty),
                Test2Id = z.Field<long>("T", "test2_id"),
                DateField = z.Field<DateTime>("T", string.Empty) })),
        "test_cursor",
        z => null,
        z => z.Map("pid", y => y.TestId),
        z => z.Map("pfk", y => y.Test2Id),
        z => z.Map("pdate", y => y.DateField)),
        x => x.Return(t => t.GetStructure("T",
            y => y.DataTable(t.Var("test"), "T"))
            .Select(z => new TestObj {
                TestId = z.Field<long>("T", string.Empty),
                Test2Id = z.Field<long>("T", "test2_id"),
                DateField = z.Field<DateTime>("T", string.Empty)
            })))
    ));
foreach (TestObj testObj in sqlBlockData)
{
    Console.WriteLine(testObj.DateField);
}

```

Warning

Oracle support predefined table variables types only so table variable type declaration in unnamed block is not supported in Oracle and block need to be redesigned

Warning

Oracle support return table from predefined pipelined functions only so unnamed SQL blocks that returns collection of objects conflicts with Oracle concept and not supported at RDMS level. This block in Oracle can be realized using pipelined functions and XData procedure call ability

Warning

Unnamed procedure SQL blocks are not supported in SQLite and MySQL

LINQ support

XData [repositories](#) and [dynamic queries](#) implements interface [IOrderedQueryable<T>](#) and granted full support for IQueryable methods.

```
// full data object has many fields but we want to select only two of them
dataScope.GetRepository<SomeObject>().Select(x => new { x.One, x.Two });
```

❗ Important

Methods *SkipWhile*, *TakeWhile*, *Join*, *SelectMany* is supported, but results with full fetch of source data and executed over IEnumerable sources on Linq2Object level. This limitations is related to unexpected complexity of translation handlers to SQL for first two methods, and dynamic changes of result type in others. To effective access to joined or restricted by fields data sources with single database query use [dynamic queries](#)

Instead of multiple data object joining or sub-selecting You can define various mappings and/or dynamic queries over same data as many as You need. This specific mappings or dynamic queries can be fine tuned for high speed and effective data access according any specialized requirements. Some of them can be reused, some not. But all of them will be as effective and low cost as You can write.

Using three tier architecture

Important

This is feature of [.Net 4.0 version](#) only.

One of unique features of XData is using same application logic modules as in two-tier-architecture (client-db), as in three-tier-architecture (client-application server-db). And You can switch three-tier application in two-tier mode to simplify debugging changing configuration file only!!!

Tip

In three-tier mode no database client software needed on client workstation. And no database credential stored in configuration file.

To extract part of data aware application logic on application server level You will need:

- extract mapped classes in separated assembly (or assemblies) called "data modules"
- extract server data logic in separated assembly (or assemblies) called "server modules"
- in server modules create logic classes for some [data objects](#) extend abstract class [XDataLogic<T>](#), where T - data object class type that uses server logic including [trigger logic](#) and call server logic from client code.

```

public abstract class InvoiceLogic : XDataLogic<Invoice>
{
    [Action(DataActionType.AfterInsert), Action(DataActionType.AfterUpdate)]
    public readonly static Trigger<Invoice> UpdateHistory =
    ((ref Invoice invoice, ref DataTriggerFlag flag) =>
    {
        if (!invoice.CheckState(DataObjectState.New)
            && !invoice.IsChanged(x => x.DocState)) return true;
        var rep = invoice.GetRepository();
        var hist = GetRepository<DocHistory>(rep.Layer, context: rep.Context)
            .Reset()
            .SetFilterValue(DocHistory.FilterByDocId,
                invoice.GetProperty<long>("DocId"))
            .SetFilterValue(DocHistory.FilterByDocStateId,
                invoice.DocState.Key);
        var newHist = hist.New();
        newHist.HistoryDate = DateTime.UtcNow;
        return hist.Submit(ref newHist);
    });

    [Action(DataActionType.BeforeDelete)]
    public readonly static Trigger<Invoice> ClearHistory =
    ((ref Invoice invoice, ref DataTriggerFlag flag) =>
    {
        var i = invoice;
        return GetRepository<DocHistory>(i.GetLayer(), context: i.GetContext())
            .Reset()
            .Clear(x => x.GetProperty<long>("DocId") == i.GetProperty<long>("DocId"));
    });

    [Action(DataActionType.BeforeClear)]
    public readonly static RepositoryTrigger<Invoice> ClearHistoryBatch =
    ((IRepository<Invoice> invoiceRepository, ref DataTriggerFlag flag) =>
        GetRepository<DocHistory>(invoiceRepository.Layer, context: invoiceRepository.Context)
            .Reset()
            .Clear(x => invoiceRepository
                .Any(z => x.GetProperty<long>("DocId") == z.GetProperty<long>("DocId"))));

    public static readonly CustomLogic<Invoice> TestCustomLogic = (objects =>
    {
        Log.Write(MessageType.Information,
            () => String.Format("TestCustomLogic called with {0} objects", objects.Length));
        foreach (var invoice in objects)
        {
            var i = invoice;
            invoice.PostData("testPost", () => Encoding.UTF8.GetBytes(i.DocNumb));
            var p = Encoding.UTF8.GetBytes(i.DocNumb);
            var r = i.Callback("testCall", ref p);
            Log.Write(MessageType.Information,
                () => String.Format("Call for \"{0}\" returned \"{1}\" with data \"{2}\"",
                    i.DocNumb, r, p == null ? null : Encoding.UTF8.GetString(p)));
        }
        return true;
    });
}

```

- set server module assembly name in data object mapping in data modules

```

[DataObject("D", LogicAssemblyName = "InvoiceServerLogic")] //for static mapping
//or
...
.SetLogicAssembly("InvoiceServerLogic") //for dynamic mapping

```

- register custom logic handlers inside mapped types in data modules

```
public static CustomLogic<Invoice> TestCustomLogic;
```

- write [custom logic](#) calls from client modules

```
var random = new Random();
return dataScope.GetRepository<Invoice>().ToArray().Execute(
() => Invoice.TestCustomLogic,
"testPost".SetValue((Action<byte[]>)(data =>
    Console.WriteLine("Post message received (data=\"{0}\")", data == null
        ? null
        : Encoding.UTF8.GetString(data))))).AsEnum().ToDictionary(),
"testCall".SetValue((Func<byte[], byte[]>)(data =>
    {
        Console.WriteLine("Call received \"{0}\\"",
            data == null ? null : Encoding.UTF8.GetString(data));
        return random.NextDouble() >= 0.5
            ? null
            : Encoding.UTF8.GetBytes(
                string.Format("reply for \"{0}\\"", data == null
                    ? null
                    : Encoding.UTF8.GetString(data))));
    }
)).AsEnum().ToDictionary());
```

Tip

Custom logic handlers can use synchronous and asynchronous callbacks to client code and perform analysis of synchronous callbacks result. This functionality allows to realize very complex and interactive distributed business processes.

Processing of related objects (Unit of Work)

XData has supported processing data object mapped to number of tricky related tables. Transactional support is required to apply changes over various but related data objects.

For example we have a business object having one or more related objects collections. Every one of them can have related collections of other objects and so on. And some business logic works with this related conglomerate as single complicated entity required submitted into DB using one transaction.

XData is support using [transaction](#), but some non terminal data objects can be inserted inside the same transaction and writing code that can support this case is a non trivial thing.

Common but elegant way to make it simple and clear is a realizing "Unit of Work" pattern. Most of ORM has own realization of this pattern, but XData using this mechanic on right level of business logic - to operate with related conglomerate of business objects. Using of Unit of Work pattern with XData is pure declarative: no custom business logic classes is needed to use it.

Lets describe this by example:

Initialization of new UoW container for new root object is described using static [Add](#) method of the [Work](#) class:

```
var rep = dataScope.GetRepository<Model.Patient>(Instance);
var instance = rep.New();
/*
    Using static class Work we init UoW container with new root object and describe
    collections of related objects. Because of new root object we use Work.Empty
    method to initialize that collections without fetching any data.
*/
using (var work = Work.Add(instance,
    Work.Empty<Model.Patient, Model.PatientDiagnosis>()
    .Empty<Model.Patient, Model.PatientOper>()))
{
    if (new PatientWindow { DataContext = instance, Owner = window }
        .ShowDialog() == true)
        work.Submit(); // submitting UoW container changes
}
```

Initialization of UoW container for existing object (update):

```
/*
    Using static class Work we init UoW container over existed root object and describe
    collections of related objects. Because of root object is exists collections will
    be filled from DB using Work.Fill method.
*/
using (var work = Work.Add(patient,
    Work.Fill<Model.Patient, Model.PatientDiagnosis>()
    .Fill<Model.Patient, Model.PatientOper>()))
{
    if (new PatientWindow { DataContext = patient, Owner = window }
        .ShowDialog() != true) return;
    work.Submit(); // submitting UoW container changes
}
```

Initialization of single related object inside of UoW container (one-to-one relationship) is described by [Add](#) method, that analog to [Empty](#) and [Fill](#) methods we use before.

UoW container can be initialized with a not only single root object, but using collection of root objects with overloaded [Add](#) method that accept an collection (*ICollection*) of root objects.

To select a single object form UoW container use [Get](#) method. This method can accept predicate as parameter to specify unique condition of object selection. After acquiring a single object we can change it (or delete it as explained in followed example):

```
_patient = work.Get();
_patient.SetDeleted(True);
```

To select multiple objects from UoW container can be made using [Select](#) method:

```
var newOperations = work.Child<Model.PatientOper>()
    .Select(x => x.Date > startDate);
```

To acquire detail UoW container we can use [Child](#) method. This method can accept predicate to select root object when root container has initialized by collection of objects. New related object can be added to detail UoW container using [New](#) method we discussed above:

```
work.Child<Model.PatientOper>()
    .New(x => x.SomeField = 42, x => x.OtherField = "some");
```

❏ Important

Unit of Work realization is not internal part of XData. It can be attached to Your project when this functionality is really needed using separate NuGet package [XData WorkSet \(UnitOfWork\)](#) package.

In addition, the UoW container supports exporting hierarchically constructed objects using the static [Export](#) method. At the same time, it is possible to export any selected part of the contents of the UoW container, starting either from the root node or from any container node.

To export a node, it must contain a property of type *Guid* for storing the object key in the UoW container and another property of type *Guid* for storing the data layer of the UoW container. These properties should not be included in the mapping.

The [GetKey](#) (or it's overload version) and [SelectKeys](#) methods are used to get the value of the keys of UoW container objects.

To export an object with child objects, it is necessary to define a property to which it will be used to store an array of child objects (or a child instance for one-to-one relation). Such properties should also not be included in the mapping.

In addition, the *bool* property can be defined in the child objects to pass the delete flag. Such properties should also not be included in the mapping.

To determine the structure of the exported part of the UoW container, a LINQ-style descriptor is used, the use of which is clear from the example below. Using the descriptor, it is possible not only to determine which child objects should be exported together with the root element, but also what service properties of objects at each level, what information will be filled.

```
var expInvoice = Work.Export(work.DbLayer, key.Value,
    () => Work.Root<Invoice>(z => z.Layer, /* Guid Data scope layer property */
        z => z.Key) /* Guid Work set key property */
    .With(z => z.InvoiceSpec, /* InvoiceSpec[] child entities array property */
        z => z.Key, /* child entity: Guid Work set key property */
        z => z.Deleted)); /* child entity: bool delete flag property */
```

After exporting, this object (and its child objects transferred with it) can be modified somewhere outside the system (for example, on a JS-based Web client) and then applied using the [Apply](#) method. In addition to the object itself, the *Apply* method accepts a descriptor built on the principle similar to that described above.

```
Work.Apply(inv, () => Work.Root<Invoice>(z => z.Layer, z => z.Key)
    .With(z => z.InvoiceSpec, z => z.Key, z => z.Deleted));
```

❏ Warning

In order to apply changes to the UoW container, the data layer (data scope) on which it is built must not be destroyed like the UoW container itself. XData has internal caching mechanisms that allow them to store and retrieve their instances by layer ID using the [GetDataScope](#) and [GetWorkSet](#) methods.

❏ Important

Data layers are cached independently of UoW containers and you need to destroy the data layer after removing the UoW container separately!

Security issues

XData is able the application to take a full control over data access rights and rights to execute any data aware business logic. Inside the XData has designed a complete data security solution for control access to some properties of [data objects](#), access to data objects corresponds some rules, access to standard (CRUD), or [custom](#) data aware business logic.

For using this ability is required to implement interface [ISecuritySession](#) and use it as parameter of [GetRepository](#) method of [data scope](#) interface to get security sensible [repository](#).

XData is not specify the way to authentication and authorization. Your application can use any existing technologies or products on the market. Here the synthetic but simple and demonstrative example of *ISecuritySession* implementation:

```

public class MySecuritySession : SecuritySession
{
    private const string Manager = "Manager";
    private const string Guest = "Guest";
    private const string Chief = "Chief";
    private const string Clerk = "Clerk";

    public static MySecuritySession CreateSecuritySession
        (string userName, string password)
    {
        if (userName == "Admin" && password == "123")
            return new MySecuritySession(userName, new string[0], true);
        if (userName == "Manager" && password == "111")
            return new MySecuritySession(userName, new[] { Manager, Clerk });
        if (userName == "Guest" && password == "222")
            return new MySecuritySession(userName, new[] { Guest });
        if (userName == "Chief" && password == "333")
            return new MySecuritySession(userName, new[] { Chief, Manager, Clerk });
        if (userName == "Clerk" && password == "444")
            return new MySecuritySession(userName, new[] { Clerk });
        RiseUnauthorizedAccessException();
        return null;
    }

    /// <exception cref="XDataSecurityException"></exception>
    [DebuggerStepThrough]
    [DebuggerNonUserCode]
    private static void RiseUnauthorizedAccessException()
    {
        throw new XDataSecurityException("XDataObjectTest system");
    }

    public MySecuritySession
        (string userName, string[] roles, bool isSupervisor = false)
        : base(userName, roles, isSupervisor) { }

    protected override void InitializeSession()
    {
        //Hide some columns from Guests
        RegisterRestrictedProperties("", typeof(Invoice),
            new[] { Guest.SetValue(new[] {
                Property<Invoice>(x => x.DocAmount),
                Property<Invoice>(x => x.DocLastChange),
                Property<Invoice>(x => x.Scan),
                Property<Invoice>(x => x.Source),
                Property<Invoice>(x => x.CustomerTo),
                Property<Invoice>(x => x.DeliveryType)
            }) });
        RegisterGrantedActions("", typeof(Invoice), new[] {
            //Grant all actions to Chief
            Chief.SetValue(new [] { Actions.All }),
            //Grant all CRUD but TestCustomLogic is denied for Manager
            Manager.SetValue(new [] { Actions.CRUD }),
            //Grant update only for Clerk
            Clerk.SetValue(new [] { Actions.Update }),
            //Grant execute TestCustomLogic only for Guest
            Guest.SetValue(new[] { Action(() => Invoice.TestCustomLogic) })
        });
        //Hide non ACTIVE invoices from Guests
        RegisterSecurityFilters("", typeof(Invoice), new[] {
            Guest.SetValue<string, Expression>(
                (Expression<Func<Invoice, bool>>)(x => x.DocStateCode == "ACTIVE")) });
    }
}

```

To simplify the *ISecuritySession* interface implementation use the NuGet package [XData Security package](#). This package provide an abstract class [SecuritySession](#) implemented base functionality and useful helpers. This functionality is shipped separately because of security session using is optional.

❏ Important

Required to override the constructor [SecuritySession\(string userName, string\[\] roles, bool isSupervisor = false\)](#) used in [three-tier architecture](#) on server side, and override method [InitializeSession](#). Method *InitializeSession* is used to cache access rules profile.

💡 Tip

Caching of all users security rules (as exposed in sample above) is not effective. This for simplify demonstration only. Security session is current user security describer. Recommended to cache rules for current user only using initialized before calling of *InitializeSession* properties: [UserName](#), [Roles](#) and [IsSupervisor](#).

In sample above take a look on using caching methods: [RegisterRestrictedProperties](#), [RegisterGrantedActions](#) and [RegisterSecurityFilters](#), and a helper methods: [Property](#) and [Action](#).

When describing access rules to actions (standard and custom) use constants defined in [Actions](#) static class:

- **All** - all actions over data object, including standard (CRUD) and custom named actions
- **CRUD** - all standard actions: *Create, Update, Delete*
- **Create** - create new data object action
- **Update** - update data object properties
- **Delete** - delete data object

Concurrency resolving

Concurrency can be resolved by [pessimistic or optimistic locking](#) on transaction management level. XData is supported an automatic (or manual) transaction management with both locking types support (when locking is supported on RDBMS level).

Pessimistic locking is RDBMS ability to define lock over one record for one thread of application until transaction is committed. In case of concurrent raise first thread acquired and lock data and other threads will get an exception. XData pessimistic locking is acquired by [repository Lock](#) method call, or an extension method [Lock](#) of [IDataObject](#) interface. In case of reject of locking the exception is handled, but method returns false. To acquire the pessimistic lock the transaction is required. When no transaction is defined then runtime [XDataRuntimeException](#) is raised.

Optimistic locking require a special field inside data table - the concurrency token. Concurrency token is always updated on insert or update of the table using some algorithm. When data changes submitted (update or delete) the value is checked for not changed until request data. If value is changes the [XDataConcurrencyException](#) is raised. To mark property as concurrency token then define mapping use [property flag](#) - [ConcurrencyToken](#) and set one of [default value type](#) to choose one of concurrency value generation algorithm. Concurrency token value checking is provided automatically. There is possible to set [IgnoreConcurrency](#) flag to ignore concurrency when submitting changes using repository [Submit](#) method, or an extension method [Submit](#) of [IDataObject](#) interface. When concurrency is ignored no checks are provided and concurrency is resolved with "last change is applied" rule.

Tip

Concurrency token field can be hidden, but required to be selected from data source table (see [Hidden property](#)).

Important

Concurrency token property mapping required to define default value behavior with extension feature [DefaultFeature.UseOnUpdate](#).

Ways to define concurrency token property mapping is various to [static](#) and [dynamic](#) notation and described in corresponding topics.

Localization

By default all errors and information messages from XData modules use neutral (en-US) locale, but it's possible to translate this using satellite assembly (for .Net 4.0 version) or prepare *resx* files to [Microsoft.Extension.Localization](#) (for ASP.Net Core 2.0 projects used XData .Net Standard 2.0 version). To help developers make localization fast and simple as possible, XData sources completed with some "bat" files.

All settings within files is made for Russian locale (ru-RU), but there are no problem to patch files for any locale.

Prepare text files to translation

1. Patch file `add_locale.bat` with setting path to RESGEN utility ("C:\Program Files\Microsoft SDKs\Windows\v8.0A\bin\NETFX 4.0 Tools"), path to `XData.dll` and locale code.
2. Run `add_locale.bat` and be sure with no errors during making translation file (.restext).
3. Translate localization strings inside .restext file to target locale language. Take care, some string are multi-line, and some strings has format templates structure (see `string.Format`).

Satellite assembly compilation

(for .Net 4.0 version)

1. Patch file `compile.bat` with setting path to AL utility ("C:\Program Files\Microsoft SDKs\Windows\v8.0A\bin\NETFX 4.0 Tools"), path to `XData.dll` and locale code.
2. Run `compile.bat` and be sure and be sure with no errors during making satellite assembly (.dll).

Resource file generation

(for ASP.Net Core 2.0 projects used XData .Net Standard 2.0 version)

1. Patch file `createres.bat` with setting path to RESGEN utility ("C:\Program Files (x86)\Microsoft SDKs\Windows\v10.0A\bin\NETFX 4.6.1 Tools"), path to `XData.dll` and locale code.
2. Run `createres.bat` and be sure and be sure with no errors during making resource file (.resx).

Install satellite assembly

1. *(for .Net 4.0 version)* Copy localization folder named as target locale code to bin folder of Your application (where `XData.dll` is located) or *(for ASP.Net Core 2.0 projects used XData .Net Standard 2.0 version)* copy .resx file for target locale to resources folder of Your ASP.Net Core 2.0 site.
2. Restart Your application or site.

Tip

Recommended materials:

[https://msdn.microsoft.com/en-us/library/f45fce5x\(v=vs.100\).aspx](https://msdn.microsoft.com/en-us/library/f45fce5x(v=vs.100).aspx)

[https://msdn.microsoft.com/en-us/library/ccec7sz1\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/ccec7sz1(v=vs.110).aspx)

[https://msdn.microsoft.com/en-us/library/sb6a8618\(v=vs.100\).aspx](https://msdn.microsoft.com/en-us/library/sb6a8618(v=vs.100).aspx)

<http://www.codeproject.com/Articles/5447/NET-Localization-using-Resource-file>

Logging and debug info

During process of developing and debugging Your application the XData can provide You information to find error or understand context of any data processing activity. XData does not enforce developer with using some logging system. Instead of that XData (version .Net 4.0) provide simple interface [ILogWriter](#) to implement. Implementation of this interface (if not implemented yet) allows to integrate XData log output into any You choose logging subsystem of Your application. As example You can use any log writer plugin from XData sources. Configuration of log writer plugin is described [configuration file section xdata](#) using attribute *log* providing value with assembly qualified type name, where type is implements [ILogWriter](#) interface.

XData (version .Net Standard 2.0) use [Microsoft.Extensions.Logging](#) subsystem.

In the case when the default level (usually the logging level "Information") is not sufficient to diagnose the problem, it can be lowered by means of the basic logging technology (including by setting the logging parameters in the configuration file). The configuration rules of the basic logging subsystem can be found on their websites: [Microsoft.Extensions.Logging](#), [System.Diagnostics.Trace](#), [log4net](#).

* (version .Net Standard 2.0) *Microsoft.Extensions.Logging* category name of XData logger is "XData". Sample logging configuration for debug:

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Debug",
      "XData": "Debug",
      "System": "Information",
      "Microsoft": "Information"
    },
    "Debug": {
      "LogLevel": { "Default": "Debug" }
    },
    "Console": {
      "LogLevel": { "Default": "Information" }
    }
  }
}
```

and in code

```
services.AddLogging(x => x.ClearProviders()
    .AddConfiguration(Configuration.GetSection("Logging"))
    .AddDebug().AddConsole());
```

or in code only

```
services.AddLogging(x => x.AddFilter("XData",
#if DEBUG
    LogLevel.Debug
#else
    LogLevel.Information
#endif
    ).AddDebug().AddConsole());
```

* (version .Net 4.0) *System.Diagnostics.Trace* switch name for XData log messages is "LogLevelSwitch". See [System.Diagnostics.Trace documentation](#) for details.

log4net requires no specific constants to configure log messages severity level. See [log4net documentation](#) to obtain configuration details.

❏ Important

Configuration of logging subsystem to debug mode can reduce application productivity. It's recommended to switch off debug mode when not required. But it's a very powerful tool to help developer with repository mapping correctness check. It's a highly recommended to use it on debug stage! SQL query logging can be also used as source for checking query plan effectiveness. Always check query plans - it's help You to develop high quality products.

Installation and configuration

NuGet configuration

To get XData NuGet packages You will need modify nuget.config as described below and place it in project root folder near Your solution file.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <packageSources>
    <clear />
    <add key="xdata" value="https://nuget.pkg.github.com/mickfierte/index.json" />
    <add key="nuget.org" value="https://api.nuget.org/v3/index.json" />
  </packageSources>
  <packageSourceCredentials>
    <xdata>
      <add key="Username" value="API_KEY" />
      <add key="ClearTextPassword" value="API_KEY" />
    </xdata>
  </packageSourceCredentials>
</configuration>
```

** replace API_KEY with API key provided by author*

Installation

To start use XData basic functionality is required to setup NuGet [XData package](#). XData [plugins or additional modules](#) can be downloaded and used separately.

To work with [related objects](#) (Unit of Work implementation) is required to setup NuGet [XData WorkSet \(UnitOfWork\) package](#).

To implement [security session](#) and use application data access with XData is required to setup NuGet [XData Security package](#).

To use one or more [SQL dialect](#), [SQL adapter](#), [specific .Net 4.0 log writer](#) XData plugins it can be installed like any other NuGet package.

To deploy application in three-tier architecture read [here](#)

Acquiring XData configuration during runtime

.Net Standard 2.0:

XData can be dynamically configured when defining XData service (see [AddXData](#)) in ServiceCollection using optional parameter *options* in LINQ style.

```
.AddSingleton(typeof(IConfigurationRoot), x => null)
.AddXData(x => x.AddDialect("postgresql", XDataAdapter.PostgreSql.Dialect)
    .AddAdapter("postgresql", XDataAdapter.PostgreSql.Adapter)
    .AddContext("test", "postgresql", "postgresql",
        "Server=localhost;Database=XDataTest;User Id=postgres;Password=123456", true))
```

Configuration file in .Net Standard 2.0 version is read by [Microsoft.Extensions.Configuration](#) service registered during application startup.

Tip

[XDataAdapter.PostgreSql.Dialect](#), [XDataAdapter.PostgreSql.Adapter](#) and similar constants for other databases is defined in [adapter NuGet packages](#).

.Net 4.0:

XData can be dynamically configured when acquiring XData engine *XDataManager.InitXData* using optional parameter *options* in

LINQ style.

⚠ Warning

Signature of `XDataManager.InitXData` method is slightly different for .Net 4.0 version! See [versions comparison](#) for details.

```
var dataEngine = XDataManager.InitXData(x =>
    x.AddDialect("postgresql", XData.PostgreSql.Dialect)
      .AddAdapter("postgresql", XData.PostgreSql.Adapters)
      .AddContext("test", "postgresql", "postgresql",
"Server=localhost;Database=XDataTest;User Id=postgres;Password=123456", true));
//or using configuration file
var dataEngine = XDataManager.InitXData(x => x.UseConfiguration(
    System.Web.Configuration.WebConfigurationManager.OpenWebConfiguration("~/")))
```

💡 Tip

When using both versions You can use [IDataOptions](#) methods to store partial non secure connection string only and fill credentials during runtime. Security related information can be obtained separately from some secure source.

Configuration XData using config file

💡 Tip

When using .Net Standard 2.0 version config file can be formatted in any notation but structure of config file need to be correspond XML structure described here

⚠ Important

When using XML formatted config file required to add specific descriptor for XData specific section "xdata"

```
<configuration>
  <configSections>
    <section name="xdata" type="System.Configuration.IgnoreSectionHandler"/>
  </configSections>
  ...
```

⚠ Important

Check *configSections* section is first section inside XML configuration file!

Connection strings for each of [context](#) must be declared in standard section *connectionStrings* and name attribute should equal to context name. Connection strings is required on client side only when two-tier (client-server) architecture used.

```
<connectionStrings>
  <add name="TEST"
    connectionString="Data Source=(local);Initial Catalog=XDataTest;User Id=test;Password=test"/>
</connectionStrings>
```

There are three possible ways to secure connection string in configuration file:

- [Cipher config file section](#) *connectionStrings* section
- Using [partial connection strings](#) during runtime
- Deploy application in [three tier mode](#). In this case connection string is not required to be configured on client side.

XData specific section *xdata* has a structure:

```
<xdata default="TEST_ORACLE_ODP" log="XDataTraceLog.TraceLog, XDataTraceLog">
  <adapter name="ORACLE_ODP" assembly="XOracleODPAdapter"/>
  <dialect name="ORACLE" assembly="XOracleDialect"/>
  <context name="TEST_ORACLE_ODP" adapter="ORACLE_ODP" dialect="ORACLE"
    sequence="XDataObjectTest.MySequenceRule, XDataObjectTest" concurrencyError="-20001"/>
</xdata>
```

The *xdata* section required to contain attribute *default* with [context](#) name used by default. Attribute *log* optional, used for .Net 4.0 version only (.Net Standard 2.0 version used [Microsoft.Extensions.Logging](#) service). Log attribute value is assembly qualified type name for [ILogWriter](#) realization (see [available](#) log plugins or implement any You want log system adapter). Log messages minimal severity level filters can be configured using native tools and abilities provided with chosen log subsystem.

Inside *xdata* section for each of [context](#) required elements:

- **dialect** (required always, can be shared between multiple contexts) - register an unique alias to specific RDBMS SQL dialect support. Attribute *name* contains an alias of dialect. Attribute *assembly* - assembly name of specific dialect plugin. When plugin implementation support additional configuration settings, element *dialect* may have an specifically organized internal structure described in plugin readme file.
- **adapter** (required when two-tier architecture is used only, can be shared between multiple contexts) - register unique alias to adapter for specific ADO.Net provider. Attribute *name* contains an alias of adapter. Attribute *assembly* - assembly name of specific adapter plugin. When plugin implementation support additional configuration settings, element *adapter* may have an specifically organized internal structure described in plugin readme file.
- **context** (required always) - contains [context](#) settings. Attribute *name* contains context name. Attribute *adapter* (required when two-tier architecture is used only) contains adapter alias. Attribute *dialect* - dialect alias. In case of using sequences required to specify attribute *sequence* contains assembly qualified type name of [ISequenceNameRule](#) implementation. When [optimistic concurrency](#) error code need to have a special value for some reason (see specific adapter readme file) set *concurrencyError* attribute to specific error code value.
- **proxy** (required when three-tier architecture is used only) - declare WCF client endpoint name for context named same as proxy attribute *name* value. WCF client endpoint name is stored in *endpoint* attribute value. Same WCF endpoint can be used for multiple contexts. Contexts names on client and server must to have same names. Deploying and configuration process for three tier architecture is described [here](#).

Deploying application in three three-tier architecture

Project organization rules and assembly deployment principles to deploy application using three-tier architecture described [here](#).

□ Note

Three-tier architecture can be applied over .Net 4.0 version only! See next versions to use three-tier feature on .Net Standard 2.0 version.

Install and configure server

Server is implemented as WCF service [XData.Server.dll](#). Base engine XData.dll library is required to server functionality. As well as log plugins, dialects and adapters to connect database directly. To run Your application business logic on server side server logic and data mapping modules are required on server side two.

Limits to choose protocol can be used as transport between client and server:

- Duplex support
- Reenter-ability inside one session support

It's recommended to host XData.Server on IIS. By default IIS is not configured to activate service over net.tcp, and if You want to use this protocol, see [here](#) to configure it.

Changes on client side

When three-tier architecture is used, required to obtain special module [XData.Remote](#). Application folder must contain client logic and data mapping modules, as well as used by Your application log modules, SQL dialect modules **BUT** now adapters not required to contexts used in three-tier mode.

In configuration file, is required to configure WCF client according Your server settings. Section *xdata* must to be configured as described above with defining of *proxy* elements for every context that used three-tier architecture. Remote contexts must to have same names on client and server. Sample three-tier configuration can be obtained from XData test project.

💡 Tip

You are able to combine local connections to some databases with one or more three-tier servers. In the same time You can use multiple dedicated servers, each for one or more contexts.

💡 Tip

You can fast and simple switch from three-tier mode to client server mode and then back to three-tier by commenting/uncommenting *proxy* descriptions. It's a very useful feature to debug server logic code ;)

Quick start

Contents

Introduction

Lesson I. ASP .Net Core project

[Registration XData](#)

[First mapping](#)

[WebAPI Controller](#)

[More complex mapping](#)

[Advanced mapping](#)

[Hierarchical object](#)

[Trigger logic](#)

Lesson II. Using variables

Lesson III. Using calculated properties

[Calculate over data in other properties](#)

[Calculate using SQL expressions mapping](#)

[Calculate using logic module](#)

Introduction

This section is organized a little differently compared to the rest of the documentation, which is organized as a reference and answers the question "What is implemented in XData?". This section is divided into lessons and answers the question "How to use XData in a particular case?".

The number of lessons will gradually grow and topics of future lessons depend on the people who order them. Suggest topics that interest you and as far as possible they will be covered in new lessons...

A few words about how the lessons will be built and what will be in them and what will not be in them:

- The lessons will focus solely on the use of XData and will assume that the reader learns about using other packages and technologies from other sources and during the lessons it will be assumed that he will already know how to use them
- Theoretical questions regarding XData will be given in the form of links to documentation. Here we will consider just a concrete example without thinking about "why so" and "how otherwise". Answers to such questions are described as far as possible in the main part of documentation

[> back to the Quick start contents](#)

Lesson I. ASP .Net Core project

Registration XData

1) Create a ASP.Net Core project

2) Set up the [NuGet configuration](#) and place the configuration file in the solution (or project) folder

3) Install the [XData NuGet packages](#): main package and the adapter package (or many of) according to the database you selected

4) Configure the configuration file of your solution in the [xml](#) notation (or json or [yaml](#) or any you like and .Net Core supported).
And add XData initialization in your *startup.cs* file...

```
services.AddXData(x => x.UseConfiguration(
    System.Web.Configuration.WebConfigurationManager.OpenWebConfiguration("~/")))
```

Optionally you can define configuration [during runtime](#) like that...

```
// You can use something like that class to define constants
public static class Constants
{
    // Database context name (any unique name to identify context)
    public const string TestDb = "TestDb";
    // Database dialect name (any unique name to identify dialect)
    public const string DialectName = "MyDbDialect";
    // Database adapter name (any unique name to identify adapter)
    public const string AdapterName = "MyDbAdapter";
    // Connection string configuration path
    public const string TestDbConnectionString = "connectionStrings:TestDb";
}
```

```
public void ConfigureServices(IServiceCollection services)
{
    ...
    // Read connection string
    var testDb = Configuration.GetValue<string>(Constants.TestDbConnectionString, null);
    // Register XData in container
    services.AddXData(x => x.AddDialect(Constants.DialectName,
        XDataAdapter.PostgreSql.Dialect)
        .AddAdapter(Constants.AdapterName, XDataAdapter.PostgreSql.Adapter)
        .AddContext(Constants.TestDb, Constants.DialectName,
            Constants.AdapterName, testDb, true));
    ...
}
```

Tip

You can add multiply dialects, adapters and contexts you need

Tip

You can share dialects and/or adapters between multiple contexts

Warning

Only one context can be marked as default. But if no default context defined an initialization error will occur

See [documentation topic](#) for more details

> [back to the Quick start contents](#)

First mapping

1) Add table *T_DOC_STATE* to database...

t_doc_state
 doc_state_id
code name

2) Add *Model* class *DocState.cs* in a *Models* folder...

```
using XData;

namespace <..Your namespace..>.Models
{
    public partial class DocState
    {
        public long DocStateId => this.GetProperty(x => x.DocStateId);
        public string Code { get; set; }
        public string Name { get; set; }
    }
}
```

Tip

The class is marked as partial because mapping will be added to another part of its description. It's comfortable.

Tip

Notice that the id field is marked as read-only. Note also that when using the XData, this DTO mapping is not required (absolutely!!! See [documentation topic](#) for more details). This allows you to more correctly use the model classes and more closely match the ideology of [DDD](#).

Note

Errors and warnings may occur before adding mapping. Ignore them. While we will not describe the *GetProperty* method, we will return to it later

3) Add class *DocState.xdata.cs* in a *Models* folder and define mapping...

```
using System.Linq.Expressions;
using XData;
using XData.Mapping.Dynamic;

namespace <..Your namespace..>.Models
{
    public partial class DocState : IDataObject
    {
        private static Expression<CustomMapping<DocState>> _ =
            () => XDataMapping.CustomMapping<DocState>()
                .DataTable("T_DOC_STATE", "S")
                .ReadOnlyProperty(x => x.DocStateId,
                    x => x.Field<long>("S", string.Empty, z => z.Key(),
                        z => z.Default(DefaultType.AutoIncrement)))
                .Map(x => new DocState {
                    Code = x.Field<string>("S", string.Empty),
                    Name = x.Field<string>("S", string.Empty)
                }).SetBaseTable("S");
    }
}
```

4) Now errors have been gone... Let's look at the mapping structure...

5) The table is described by the [DataTable](#) method. The table name and the alias of the data source are passed as parameters. The alias of the data source will also be used as the alias of the table in the final SQL. See [documentation topic](#) for more details

6) Description of the read-only property is done using the [ReadOnlyProperty](#) method. The description of the *DocStateId* property and its mapping described by the [Field](#) method are transmitted to it. See [documentation topic](#) for more details

Field method parameters:

- The data source alias

- *string.Empty* is passed as the field name because the field name corresponds to the default conversion of the property name
- a set of mapping modifiers:
 - attribute of the key field of the data object [Key](#)
 - default value generation method [Default](#) (the auto-increment database value is used here [DefaultType.AutoIncrement](#))

7) Mapping the usual properties available for both reading and writing is described by the [Map](#) method. Mapping of each field is described using the familiar [Field](#) method. See [documentation topic](#) for more details

8) The mapping is completed by the [SetBaseTable](#) method, which determines the [base table](#) of the [data object](#) by the alias of the data source. See [documentation topic](#) for more details

9) We return to the description of the structure of the object itself and to the [GetProperty](#) method with which we get the *DocStateId* read-only property value. See [documentation topic](#) for more details

> [back to the Quick start contents](#)

WebAPI Controller

1) To write a WebAPI controller that serves the model class we wrote earlier, it is recommended to make some base controller class that has the implementation of all the basic actions on the object. This will allow you to reuse this basic logic of it for most typical controllers...

```
using System;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.ModelBinding;
using XData;
using XData.Async;

namespace <..Your namespace..>
{
    /// <summary>
    /// Extensions
    /// </summary>
    public static class Extensions
    {
        /// <summary>
        /// Fill model state
        /// </summary>
        /// <param name="exception"></param>
        /// <param name="modelState"></param>
        public static void FillModelState(this Exception exception, ModelStateDictionary modelState)
        {
            modelState.AddModelError("", exception.Message);
            modelState.AddModelError("at", exception.StackTrace);
            var inner = exception.InnerException;
            while (inner != null)
            {
                modelState.AddModelError("innerException", inner.Message);
                modelState.AddModelError("at", inner.Message);
                inner = inner.InnerException;
            }
        }
    }

    [Route("api/[controller]/[action]")]
    [ApiController]
    public class DataObjectController<T> : ControllerBase
    {
        where T : class, IDataObject, new()

        {
            private protected IDataEngine _dataEngine;
```

```

public DataObjectController(IDataEngine dataEngine)
{
    _dataEngine = dataEngine;
}

[HttpGet]
public virtual async Task<ActionResult<T[]>> All(CancellationTokens cancellationTokens)
{
    try
    {
        using (var dataScope = _dataEngine.NewDataScope(cancellationTokens))
        {
            return await dataScope.GetRepository<T>().ToArrayAsync();
        }
    }
    catch (Exception e)
    {
        e.FillModelState(ModelState);
        throw;
    }
}

[HttpDelete]
public virtual async Task<ActionResult<bool>> Delete([FromBody] T obj, CancellationTokens
cancellationTokens)
{
    try
    {
        using (var dataScope = _dataEngine.NewDataScope(cancellationTokens))
        {
            var found = dataScope.GetRepository<T>().Find(obj);
            if (found == null) return NotFound();
            return await found.SetDeleted(true).SubmitAsync();
        }
    }
    catch (Exception e)
    {
        e.FillModelState(ModelState);
        throw;
    }
}

[HttpPut]
public virtual async Task<ActionResult<T>> Save([FromBody] T obj, CancellationTokens
cancellationTokens)
{
    try
    {
        using (var dataScope = _dataEngine.NewDataScope(cancellationTokens))
        {
            return await dataScope.GetRepository<T>()
                .Apply(obj).SubmitAndReturnAsync();
        }
    }
    catch (Exception e)
    {
        e.FillModelState(ModelState);
        throw;
    }
}
}

```

2) The *FillModelState* helper extension method is used to fill out error information...

3) The base class of the controller receives from the container the implementation of the interface [IDataEngine](#) as a constructor parameter and stores it in a private variable. This object is already initialized with the settings specified during the initialization of the XData described earlier in the article about registering a XData.

4) The *All* method is used to get ALL registered in the database objects served by the controller type. We get the *data_scope* object using [NewDataScope](#) in the default context and call the [GetRepository<T>](#) method from it, which returns the *IQueryable<T>* interface typed by our object type. At the end, we call the asynchronous method wrapper [ToArrayAsync](#) to asynchronously return the collection received from the repository.

5) To delete an object, used the *Delete* method. In addition to the methods discussed above, the [Find](#) method is also used which looks for an object in the repository by the values of key fields. It should be noted that there can be several key fields when using several fields as a composite key of a data object. In this case, the key fields of the data object may not coincide with the key fields of the table since the data object may contain several data sources such as tables and subqueries. After the record is successfully found, we mark it as deleted using the [SetDeleted](#) method and apply the changes using the asynchronous [SubmitAsync](#) method. The *SubmitAsync* method returns a boolean sign of success of the operation.

6) In order to apply object changes made on the web client side (both creating a new object and changing an existing one), the *Save* controller method is used. In addition to the methods already familiar to us, the [Apply](#) method is used here, which analyzes the key fields of the data object and if they (at least one of the key fields) have empty values or by their combination the object is not found in the database, then the object must be created otherwise a new update will be applied to found by key information object. The [SubmitAndReturnAsync](#) method applies changes to the database and returns, if the operation succeeds, the changed (or created) object in asynchronous mode.

7) Based on the base class that we created, we will create a *DocStateController* controller (in *Controllers\API* folder) to work specifically with the *DocState* class described earlier...

```
using XData;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using <..Your namespace..>.Models;

namespace <..Your namespace..>.Controllers.API
{
    public class DocStateController : DataObjectController<DocState>
    {
        public DocStateController(IDataEngine dataEngine) : base(dataEngine) { }

        [HttpGet]
        public async Task<ActionResult<DocState>> Get(long id, CancellationToken cancellationToken)
        {
            try
            {
                using (var dataScope = _dataEngine.NewDataScope(cancellationToken))
                {
                    return await dataScope.GetRepository<DocState>()
                        .SingleOrDefaultAsync(x => x.DocStateId == id);
                }
            }
            catch (Exception e)
            {
                e.FillModelState(ModelState);
                throw;
            }
        }
    }
}
```

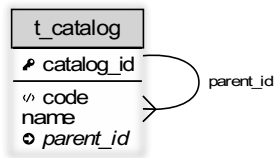
8) Here we define the *Get* method for obtaining data on the primary key, and the new method for us here is the [SingleOrDefaultAsync](#) method, which in asynchronous mode returns the object found by the primary key or *null* value.

See documentation about [asynchronous extensions](#), basic methods of [manipulating data](#) and serialization into [JSON format](#) to get more detailed information...

[> back to the Quick start contents](#)

More complex mapping

1) Now consider a slightly more complex mapping using the directory object as an example. Add a table with a recursive link to ourselves ...



2) Add *Model* class *Catalog.cs* in a *Models* folder...

```
using XData;
using XData.Mapping;

namespace <..Your namespace..>.Models
{
    public partial class Catalog
    {
        public long CatalogId => this.GetProperty(x => x.CatalogId);
        public string Code { get; set; }
        public string Name { get; set; }
        public Link<string, Catalog> Parent { get; set; }
    }
}
```

3) Here, the type of the property *Parent* will be new to us. It is used in XData to describe an external reference to a dictionary or other entity. And here is a slightly more specific case - a tree reference to oneself. The description type of such a link is [Link<TVal,TSource>](#) and it is typed by the type of value that will be in the [Value](#) property of *Link* and the type of entity to which we refer (in this case, our type - *Catalog*). In our case *Value* of *Link* is the unique *Code* of the parent entity. See [documentation topic](#) for more details

4) Add class *Catalog.xdata.cs* in a *Models* folder and define mapping...

```

using System.Linq.Expressions;
using XData;
using XData.Mapping.Dynamic;

namespace <..Your namespace..>.Models
{
    public partial class Catalog : IDataObject
    {
        // ReSharper disable once UnusedMember.Local
        private static Expression<CustomMapping<Catalog>> _ =
            () => XDataMapping.CustomMapping<Catalog>()
                .DataTable("T_CATALOG", "C")
                .DataTable("T_CATALOG", "P",
                    x => x.Link("C", "parent_id", "catalog_id")
                        .SetOperation(FilterOperation.OuterJoin))
                .Column("ParentId", x => x.Field<long?>("C", string.Empty))
                .ReadOnlyProperty(x => x.CatalogId,
                    x => x.Field<long>("C", string.Empty, z => z.Key(),
                        z => z.Default(DefaultType.AutoIncrement)))
                .Map(x => new Catalog
                {
                    Code = x.Field<string>("C", string.Empty),
                    Name = x.Field<string>("C", string.Empty),
                    Parent = x.Link<string, Catalog>("P", "code", z => z.OuterFlag(),
                        z => z.LinkProperty((Catalog y) => y.Code),
                        z => z.LinkProperty<Catalog>(y => y.CatalogId,
                            y => y.GetProperty<long?>("ParentId")))
                }).SetBaseTable("C");
    }
}

```

5) The first mention of the [DataTable](#) method is similar to what we have already seen, but the second is more interesting, we describe the same table under a different alias "P" and also describe its relationship with the first table. This is indicated by the alias "C" as the first parameter of the [Link](#) method, the second parameter denotes the field name of the linked table "C" and the third field name of the current table (with the "P" alias). If the names of the related fields are the same, another overload of this method is used which allows you to specify the field name once. See [documentation topic](#) for more details

6) By default, the connection has the operation "equal", but in this case we need to link these tables by the outer join join and to indicate this we "refine" the link using the [SetOperation](#) method and specify the operation [FilterOperation.OuterJoin](#). See [documentation topic](#) for more details

7) For now, skip the *Column* method and return to it later...

8) The next new element for us will be a description of the mapping of the "Parent" link field. It is described by the [Link<TVal,TSource>](#) method typed as the type of the property in the main description of the object. The first parameter is the data source alias ("P") of the field that will be assigned in link *Value*, the second parameter designates the field that will be assigned in *Value*. The third parameter contains the lambda-modifiers assigned to the link field ...

9) The first modifier [OuterFlag](#) shows that this field is an indicator of whether or not to insert an entry into the table connected by the outer join (when inserting). If value is null inserting to this table (and their child tables in updatable tables hierarchy) will be skipped. When updating to an empty value, it signals the need to delete the record from the table optionally connected and their child tables in updatable tables hierarchy. See [documentation topic](#) for more details

10) The rest of the modifiers describe which fields will be copied when assigning the link from the source object. There are few overloads of the [LinkProperty](#) method:

- with one lambda parameter that takes a link type parameter (in our case, the first occurrence). It determines which property of the link object will be copied to *Value*
- with no parameters (in our case this does not occur). It determines property of the link object will be copied to *Value* named the same as link property

- with one lambda parameter that takes a parameter of the type of the object being edited (in our case this does not occur). It means copying to the specified property from the property of the link object with the same name
- with two parameters (second occurrence). The first parameter is the copy source, the second is the target...

11) The last parameter needs to be considered especially. It is specified using the [GetProperty](#) method with the string parameter "ParentId". Note that this coincides with the first parameter of the [Column](#) method we skipped. Using this method, a hidden field is described that is necessary only for mapping and not displayed on the properties of the described object. And therefore, the *CatalogId* property of the source will be copied to a hidden property named "ParentId". See [documentation topic](#) for more details

12) Web API controller for this object will look like...

```

using XData;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using <..Your namespace..>.Models;

namespace <..Your namespace..>.Controllers.API
{
    public class CatalogController : DataObjectController<Catalog>
    {
        public CatalogController(IDataEngine dataEngine) : base(dataEngine) { }

        [HttpGet]
        public async Task<ActionResult<Catalog>> Get(long id, CancellationToken cancellationToken)
        {
            try
            {
                using (var dataScope = _dataEngine.NewDataScope(cancellationToken))
                {
                    return await dataScope.GetRepository<Catalog>()
                        .SingleOrDefaultAsync(x => x.CatalogId == id);
                }
            }
            catch (Exception e)
            {
                e.FillModelState(ModelState);
                throw;
            }
        }

        [HttpGet]
        public async Task<ActionResult<Catalog[]>> GetChilds(long id, CancellationToken cancellationToken)
        {
            try
            {
                using (var dataScope = _dataEngine.NewDataScope(cancellationToken))
                {
                    if(id == 0) //return root catalogs
                        return await dataScope.GetRepository<Catalog>()
                            .Where(x => x.GetProperty<object>("ParentId") == null)
                            .ToArrayAsync();
                    return await dataScope.GetRepository<Catalog>()
                        .Where(x => x.GetProperty<long>("ParentId") == id)
                        .ToArrayAsync();
                }
            }
            catch (Exception e)
            {
                e.FillModelState(ModelState);
                throw;
            }
        }
    }
}

```

Tip

An attentive reader will certainly notice that this implementation does not have protection against directory tree loops. This can be done using XData, but this will unnecessarily complicate the example of this lesson.

[> back to the Quick start contents](#)

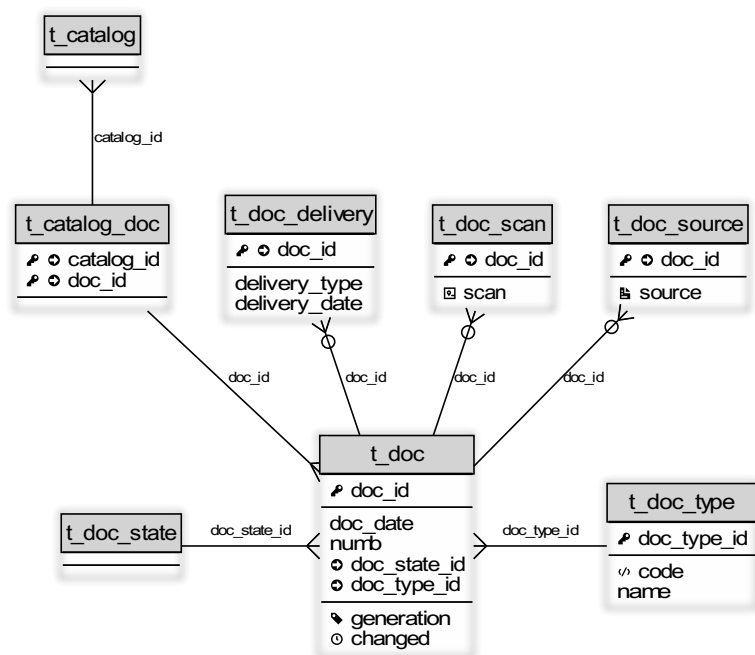
Advanced mapping

1) Let's write something really complicated! Let's start as usual with the preparation of the database. We already have the tables *t_catalog* and *t_doc_state*, we will create the rest and link them as shown in the diagram.

The fields deserve special mention:

- *t_doc.generation* - guid
- *t_doc_delivery.delivery_type* - a short integer (we will store the enum value there)
- *t_doc_scan.scan* - binary
- *t_doc_source.source* - xml

Dictionary *t_doc_type* must contain at least one entry with *code* = "INVOICE"



2) Add *Model* class *Invoice.cs* in a *Models* folder...

```

public enum DeliveryTypeEnum
{
    [Display(Description = "PickUp")]
    Pickup,
    [Display(Description = "Courier")]
    Courier,
    [Display(Description = "Mail")]
    Mail
}

public partial class Invoice
{
    public long DocId => this.GetProperty(x => x.DocId);
    public string DocStateCode => this.GetProperty(x => x.DocStateCode);
    public Link<string, DocState> DocState { get; set; }
    public long DocCatalogId { get; set; }
    public string Numb { get; set; }
    public DateTime? DocDate { get; set; }
    public Guid Generation => this.GetProperty(x => x.Generation);
    public DateTime Changed => this.GetProperty(x => x.Changed);
    public Lob Scan { get; set; }
    public Xml Source { get; set; }
    public DeliveryTypeEnum DeliveryType { get; set; }
    public DateTime DeliveryDate { get; set; }
}

```

3) Add class *Invoice.xdata.cs* in a *Models* folder and define mapping...

```

public partial class Invoice: IDataObject
{
    private static Expression<CustomMapping<Invoice>> _ = (
        () => XDataMapping.CustomMapping<Invoice>()
        .DataTable("T_DOC", "D",
            x => x.DictFilter("T_DOC_TYPE", "doc_type_id", "code", "INVOICE"))
        .DataTable("T_DOC_STATE", "S", x => x.Link("D", "doc_state_id"))
        .DataTable("T_DOC_CATALOG", "DC", "D", x => x.Link("D", "doc_id"))
        .DataTable("T_DOC_SCAN", "SC", "D", x => x.Link("D", "doc_id"))
        .SetOperation(FilterOperation.OuterJoin))
        .DataTable("T_DOC_SOURCE", "SR", "D", x => x.Link("D", "doc_id"))
        .SetOperation(FilterOperation.OuterJoin))
        .DataTable("T_DOC_DELIVERY", "DD", "D", x => x.Link("D", "doc_id"))
        .ReadOnlyProperty(x => x.DocId, x => x.Field<long>("D", string.Empty,
            z => z.Key(),
            z => z.Default(DefaultType.AutoIncrement)))
        .Column("DocStateId", x => x.Field<long>("S", string.Empty))
        .ReadOnlyProperty(x => x.DocStateCode, x => x.Field<string>("S", "code"))
        .ReadOnlyProperty(x => x.Generation, x => x.Field<Guid>("D", string.Empty,
            z => z.ConcurrencyToken(), z => z.Default(DefaultType.NewGuid)))
        .ReadOnlyProperty(x => x.Changed, x => x.Field<DateTime>("D", string.Empty,
            z => z.Default(DefaultType.CurrentDateTime, DefaultFeature.UseOnUpdate)))
        .Map(x => new Invoice {
            DocCatalogId => x.Field<long>("DC", "catalog_id"),
            DocState = x.Link<string, DocState>("S", "name",
                z => z.LinkProperty<DocState>(y => y.Name),
                z => z.LinkProperty<DocState>((Invoice y) => y.DocStateId),
                z => z.LinkProperty<DocState>(y => y.Code, y => y.DocStateCode)),
            Numb = x.Field<string>("D", string.Empty),
            DocDate = x.Field<DateTime?>("D", string.Empty,
                z => z.Default(DefaultType.CurrentDate)),
            Scan = x.Lob("SC", z => z.OuterFlag()),
            Source = x.Xml("SR", z => z.OuterFlag()),
            DeliveryType = x.Field<DeliveryTypeEnum>("DD", string.Empty),
            DeliveryDate = x.Field<DateTime>("DD", string.Empty,
                z => z.Default(DefaultType.CurrentDate))
        }).SetBaseTable("D")
    );
}

```

4) The first unfamiliar thing for us is [DictFilter](#). This method allows you to filter the data object by reference to the dictionary in which there is a unique string field *Code* with which we can not rely on the dictionary identifiers at the level of mapping and business logic. In fact, it degenerates into a filter on a subquery (in our case it will be *D.doc_type_id = (select doc_type_id from T_DOC_TYPE where code = 'INVOICE')*). See [documentation topic](#) for more details

5) A not very noticeable, but extremely important detail is that for some tables a new overload of the [DataTable](#) method is used with the *parent* parameter, which defines the alias of the parent table in [updatable tables hierarchy](#). The introduction of this concept allows applying the change not to one table, but to any number of tables in which information about an instance of a data object is stored.

❏ Important

Do not forget to specify the parent table in the *updatable table tree* for those tables that should be updated when the data object changes. At the same time, tables containing reference information are not included in the *updatable table tree* and do not participate in the update process. For such tables, an overload of the *DataTable* method is used, which we used earlier.

6) The next interesting thing is the [ConcurrencyToken](#) modifier, which is responsible for [optimistic lock](#) support. This modifier works in tandem with the familiar *Default* modifier, which is initialized with the [DefaultType.NewGuid](#) value, which means generating a new unique key (GUID). See [documentation topic](#) for more details

Note that the competition token is described in mapping and does not require any other code! It can even be mapped into a read-only field (as in this case).

7) The next interesting detail is the automatic completion of the *Changed* property by change date. Using the *Default* modifier with two parameters, the first [DefaultType.CurrentDateTime](#) sets the mechanism for obtaining the default field value by storing the date and time of the change in the local time zone, and the second [DefaultFeature.UseOnUpdate](#) allows you to save the change time not only during insertion but also during update. See [documentation topic](#) for more details

8) The *DocDate* and *DeliveryDate* properties are modified with the default value [DefaultType.CurrentDate](#) which means that by default, the current date will be written to these properties.

9) The *Scan* property has a specific type of [Lob](#) and its mapping is described using the [Lob](#) method. Similarly, the *Source* property has a specific [Xml](#) type and its mapping is described using the [Xml](#) method. The work with these types is different, but it is important that by default the value of such fields is loaded only if the value of a large object is immediately addressed. There are also features when serializing them in JSON (we will consider this below). There are also features when serializing them in JSON (we will consider this below). This behavior allows you to significantly accelerate the work with objects that contain information in large fields. See documentation topic about [binary properties](#) and [Xml properties](#) for more details

10) Web API controller for this object will look like...

```
using XData;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using <..Your namespace..>.Models;

namespace <..Your namespace..>.Controllers.API
{
    public class InvoiceController : DataObjectController<Invoice>
    {
        public InvoiceController(IDataEngine dataEngine) : base(dataEngine) { }

        [HttpGet]
        public async Task<ActionResult<Invoice>> Get(long id, CancellationToken cancellationToken)
        {
            try
            {
                using (var dataScope = _dataEngine.NewDataScope(cancellationToken))
                {
                    return await dataScope.GetRepository<Invoice>()
                        .SingleOrDefaultAsync(x => x.DocId == id);
                }
            }
            catch (Exception e)
            {
                e.FillModelState(ModelState);
                throw;
            }
        }

        [HttpGet]
        public async Task<ActionResult<Invoice[]>> GetByCatalog(long id, CancellationToken cancellationToken)
        {
            try
            {
                using (var dataScope = _dataEngine.NewDataScope(cancellationToken))
                {
                    return await dataScope.GetRepository<Invoice>()
                        .ToArrayAsync(x => x.CatalogId == id);
                }
            }
            catch (Exception e)
            {
                e.FillModelState(ModelState);
                throw;
            }
        }
    }
}
```



```

        {
            e.FillModelState(ModelState);
            throw;
        }
    }

    [HttpGet]
    public async Task<ActionResult<Invoice>> Export(long id, CancellationToken cancellationToken)
    {
        try
        {
            using (var dataScope = _dataEngine.NewDataScope(cancellationToken))
            {
                return await dataScope.GetRepository<Invoice>()
                    .SingleOrDefaultAsync(x => x.DocId == id,
                        x => x.LoadLob());
            }
        }
        catch (Exception e)
        {
            e.FillModelState(ModelState);
            throw;
        }
    }

    [HttpGet]
    public async Task<ActionResult<byte[]>> GetScan(long id, CancellationToken cancellationToken)
    {
        try
        {
            using (var dataScope = _dataEngine.NewDataScope(cancellationToken))
            {
                return await dataScope.GetRepository<Invoice>()
                    .Where(x => x.DocId == id)
                    .Scan.Value;
            }
        }
        catch (Exception e)
        {
            e.FillModelState(ModelState);
            throw;
        }
    }

    [HttpPut]
    public async Task<ActionResult<Invoice[]>> FindBySource([FromBody]string xpath, string value,
        CancellationToken cancellationToken)
    {
        try
        {
            using (var dataScope = _dataEngine.NewDataScope(cancellationToken))
            {
                return await dataScope.GetRepository<Invoice>()
                    .Where(x => x.Source.Extract(xpath) == value)
                    .ToArrayAsync();
            }
        }
        catch (Exception e)
        {
            e.FillModelState(ModelState);
            throw;
        }
    }
}

```

11) The *Export* method shows how you can get an object with large fields filled. This is available due to the use of the [LoadLob](#) modifier. Most asynchronous XData methods have overloads to describe serialization parameters. For a complete description, refer to the corresponding [documentation article](#).

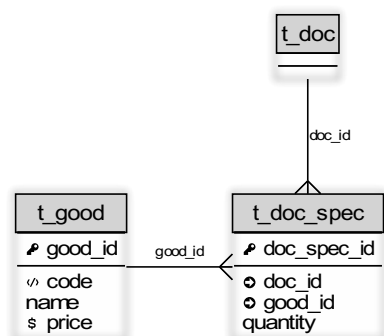
12) Getting the value of a binary object is shown in the *GetScan* method. To get the value, use the [Value](#) property of the *Lob* object.

13) The *Xml* type is interesting, among other things, because its contents can be used to construct XPath queries and use them to filter data. To do this, use the [Extract](#) method of type *Xml* as shown in the *FindBySource* method.

> [back to the Quick start contents](#)

Hierarchical object

1) Now let's build a hierarchically constructed object. To do this, we'll complete the database again ...



2) *Goods* dictionary and specification of a document itself contain mostly familiar to us elements...

```
public partial class Good
{
    public long GoodId => this.GetProperty(x => x.GoodId);
    public string Code { get; set; }
    public string Name { get; set; }
    public decimal Price { get; set; }
}
```

```
public partial class Good : IDataObject
{
    private static Expression<CustomMapping<Good>> _ =
        () => XDataMapping.CustomMapping<Good>()
            .DataTable("T_GOOD", "G")
            .ReadOnlyProperty(x => x.GoodId,
                x => x.Field<long>("G", string.Empty, z => z.Key(),
                    z => z.Default(DefaultType.AutoIncrement)))
            .Map(x => new Good {
                Code = x.Field<string>("G", string.Empty),
                Name = x.Field<string>("G", string.Empty),
                Price = x.Field<decimal>("G", string.Empty)
            }).SetBaseTable("G");
}
```

```

public partial class DocSpec
{
    public long DocSpecId => this.GetProperty(x => x.DocSpecId);
    public long DocId => this.GetProperty(x => x.DocId);
    public string GoodCode => this.GetProperty(x => x.GoodCode);
    public Link<string, Good> Good { get; set; }
    public decimal Price => this.GetProperty(x => x.Price);
    public decimal Quantity { get; set; }
    public decimal Amount => this.GetProperty(x => x.Amount);
    public bool Deleted { get; set; }
}

```

```

public partial class DocSpec : IDataObject
{
    private static Expression<CustomMapping<DocSpec>> _ =
        () => XDataMapping.CustomMapping<DocSpec>()
            .DataTable("T_DOC_SPEC", "S")
            .DataTable("T_GOOD", "G",
                x => x.Link("S", "good_id"))
            .ReadOnlyProperty(x => x.DocSpecId,
                x => x.Field<long>("S", string.Empty, z => z.Key(),
                    z => z.Default(DefaultType.AutoIncrement)))
            .ReadOnlyProperty(x => x.DocId,
                x => x.Field<long>("S", string.Empty))
            .ReadOnlyProperty(x => x.GoodCode,
                x => x.Field<string>("G", "code"))
            .ReadOnlyProperty(x => x.Price,
                x => x.Field<decimal>("G", string.Empty))
            .ReadOnlyProperty(x => x.Amount,
                x => x.Expr<decimal>("amount",
                    z => z.Field<decimal>("G", "price") *
                        z.Field<decimal>("S", "quantity"),
                    DbType.Decimal, z => z.Size(17, 2)))
            .Column("GoodId", x => x.Field<long>("G", string.Empty))
            .Map(x => new DocSpec {
                Quantity = x.Field<decimal>("S", string.Empty),
                Good = x.Link<string, Good>("S", "name",
                    z => z.LinkProperty<Good>(y => y.Name),
                    z => z.LinkProperty<Good>((Invoice y) => y.GoodId),
                    z => z.LinkProperty<Good>(y => y.Code, y => y.GoodCode))
            }).SetBaseTable("S");
}

```

3) What is new to us here is only the [Expr](#) method, which allows us to use SQL expressions in mapping. In this case, we calculated the position amount by multiplying the quantity by the goods price. The first parameter is the name of the expression field, the second describes the expression itself, the third is the type of expression (DbType.Decimal), [Size](#) is used as a modifier, which allows us to specify the size and accuracy of the expression. See [documentation topic](#) for more details

4) And also note that in the *DocSpec* object there is a boolean flag *Deleted* not reflected on the database. But we'll talk about it a little later...

5) Create a *GoodController* controller (in Controllers\API folder)...

```

using XData;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using <..Your namespace..>.Models;

namespace <..Your namespace..>.Controllers.API
{
    public class GoodController : DataObjectController<Good>
    {
        public DocStateController(IDataEngine dataEngine) : base(dataEngine) { }

        [HttpGet]
        public async Task<ActionResult<Good>> Get(long id, CancellationToken cancellationToken)
        {
            try
            {
                using (var dataScope = _dataEngine.NewDataScope(cancellationToken))
                {
                    return await dataScope.GetRepository<Good>()
                        .SingleOrDefaultAsync(x => x.GoodId == id);
                }
            }
            catch (Exception e)
            {
                e.FillModelState(ModelState);
                throw;
            }
        }
    }
}

```

6) And here, instead of the sequence of actions we already know, we modify the object from the previous part (Advanced mapping) of the lesson in order to learn how to work with hierarchical objects. First, we will show the changes in the code and then in detail we will analyze all the new techniques for us that will be used here...

```

public partial class Invoice
{
    public long DocId => this.GetProperty(x => x.DocId);
    public string DocStateCode => this.GetProperty(x => x.DocStateCode);
    public Link<string, DocState> DocState { get; set; }
    public long DocCatalogId { get; set; }
    public string Numb { get; set; }
    public DateTime? DocDate { get; set; }
    public Guid Generation => this.GetProperty(x => x.Generation);
    public DateTime Changed => this.GetProperty(x => x.Changed);
    public Lob Scan { get; set; }
    public Xml Source { get; set; }
    public DeliveryTypeEnum DeliveryType { get; set; }
    public DateTime DeliveryDate { get; set; }
    // Add an array for the entries of the child repository
    public DocSpec[] Specification { get; set; }
    // And the calculated amount for the document as a whole
    public decimal DocAmount => this.GetProperty(x => x.DocAmount);
}

```

```

public partial class Invoice: IDataObject
{
    private static Expression<CustomMapping<Invoice>> _ = (
        () => XDataMapping.CustomMapping<Invoice>()
            .DataTable("T_DOC", "D",
                x => x.DictFilter("T_DOC_TYPE", "doc_type_id", "code", "INVOICE"))
            .DataTable("T_DOC_STATE", "S", x => x.Link("D", "doc_state_id"))
            .DataTable("T_DOC_CATALOG", "DC", "D", x => x.Link("D", "doc_id"))
            .DataTable("T_DOC_SCAN", "SC", "D", x => x.Link("D", "doc_id"))
            .SetOperation(FilterOperation.OuterJoin))
            .DataTable("T_DOC_SOURCE", "SR", "D", x => x.Link("D", "doc_id"))
            .SetOperation(FilterOperation.OuterJoin))
            .DataTable("T_DOC_DELIVERY", "DD", "D", x => x.Link("D", "doc_id"))
            .ReadOnlyProperty(x => x.DocId, x => x.Field<long>("D", string.Empty,
                z => z.Key(),
                z => z.Default(DefaultType.AutoIncrement)))
            .Column("DocStateId", x => x.Field<long>("S", string.Empty))
            .ReadOnlyProperty(x => x.DocStateCode, x => x.Field<string>("S", "code"))
            .ReadOnlyProperty(x => x.Generation, x => x.Field<Guid>("D", string.Empty,
                z => z.ConcurrencyToken(), z => z.Default(DefaultType.NewGuid)))
            .ReadOnlyProperty(x => x.Changed, x => x.Field<DateTime>("D", string.Empty,
                z => z.Default(DefaultType.CurrentDateTime, DefaultFeature.UseOnUpdate)))
            // Describe the subquery for calculating the amount of the document
            .Subquery("A", typeof(DocSpecAmount), "Amount", DataGrouping.Sum,
                x => x.SubqueryLink(x => x.DocId))
            //And mapping properties for the amount of the document
            .ReadOnlyProperty(x => x.DocAmount,
                x => x.Expr<decimal?>(null, DataExpressionType.SubQuery,
                    "A", DbType.Decimal, z => z.Size(17, 5)))
            .Map(x => new Invoice {
                DocCatalogId => x.Field<long>("DC", "catalog_id"),
                DocState = x.Link<string, DocState>("S", "name",
                    z => z.LinkProperty<DocState>(y => y.Name),
                    z => z.LinkProperty<DocState>((Invoice y) => y.DocStateId),
                    z => z.LinkProperty<DocState>(y => y.Code, y => y.DocStateCode)),
                Numb = x.Field<string>("D", string.Empty),
                DocDate = x.Field<DateTime?>("D", string.Empty,
                    z => z.Default(DefaultType.CurrentDate)),
                Scan = x.Lob("SC", z => z.OuterFlag()),
                Source = x.Xml("SR", z => z.OuterFlag()),
                DeliveryType = x.Field<DeliveryTypeEnum>("DD", string.Empty),
                DeliveryDate = x.Field<DateTime>("DD", string.Empty,
                    z => z.Default(DefaultType.CurrentDate))
            },
            // Define a rule for link to a DocSpec child repository
            x => x.ExternalLink<DocSpec>(x => x.DocId)
            ).SetBaseTable("D")
    );
}

```

7) We describe the subquery as a separate mapping. In this case, we don't need the object itself, but a separate mapping calculating the amount for any type of documents will come in handy...

```

public class DocSpecAmount: ISqlObject
{
    private static Expression<CustomMapping<DocSpecAmount>> _ = (
        () => XDataMapping.CustomMapping<DocSpecAmount>()
            .DataTable("T_DOC_SPEC", "S")
            .DataTable("T_GOOD", "G",
                x => x.Link("S", "good_id"))
            .Column("DocId", x => x.Field<long>("S", string.Empty,
                z => z.Hidden()))
            .Column("Amount", x => x.Expr<decimal>("amount",
                z => z.Field<decimal>("G", "price") *
                    z.Field<decimal>("S", "quantity"),
                DbType.Decimal, z => z.Hidden(), z => z.Size(17, 2)))
            .Map().SetBaseTable("S"));
}

```

8) Now we will deal with the new that is used in the presented pieces of code...

9) First, we added an array *Specification* for the objects of the *DocSpec* type described earlier to the main declaration of the *Invoice* object. But we note that he does not participate directly in mapping, and nevertheless, there will be child records in it. But first things first...

10) A subquery is described using the [Subquery](#) method with parameters - the alias of the subquery, the type of mapping description of this subquery, the property that will be selected from it, the group operation that will be applied to this property (note that we are talking here not only about the usual properties of the object, but also about hidden properties, as in this example). The last parameter describes [SubqueryLink](#) - the filter which the subquery link with an external query. The property by which the subquery will be linked will be *DocId*, and since in the subquery the communication property is named the same, additional information is not required. See [documentation topic](#) for more details

11) The description of the SQL expression when using a subquery is somewhat different and uses a different overload of the [Expr](#) method. The first parameter, as before, is the name of the field - expression, but here it is possible to specify *null* to use the field name according to the naming rules depending on the name of the property (here it will be *doc_amount*). The second parameter determines the type of SQL expression and [DataExpressionType.SubQuery](#) is used here. The third parameter in this case (since the SQL type of the expression is *DataExpressionType.SubQuery*) should be equal to the subquery alias "A". See [documentation topic](#) for more details

12) To describe the link rule with the child repository *DocSpec*, the [ExternalLink](#) modifier of the *Map* method is used. The *DocId* property is specified as its first parameter, and since the associated property has the same name, this is enough. See [documentation topic](#) for more details

13) The description of the mapping of the subquery implements the [ISqlObject](#) interface and since we will not use it as an independent data object, we have enough. Quite a few extensions are assigned to the *IDataObject* interface and the use of a more general interface is architecturally justified here.

14) The *Column* method in our example for both properties is marked with the [Hidden](#) modifier. This modifier hides this field from the SQL construct *Select* and for group operations (and we define this subquery exactly for this only), it is often justified. Since we described the name of the "Amount" property and the group operation [DataGrouping.Sum](#) in the description of the use of the subquery, this information will be enough to build the correct SQL construct without further limiting the possible reused description in the future.

15) And now we change *InvoiceController* WebAPI controller...

```

[HttpGet]
public async Task<ActionResult<Invoice>> Get(long id, CancellationToken cancellationToken)
{
    try
    {
        using (var dataScope = _dataEngine.NewDataScope(cancellationToken))
        {
            return await dataScope.GetRepository<Invoice>()
                // Initialize the child repository
                .Do(x => x.Instance.GetChild<DocSpec>(z => z.Specification))
                .SingleOrDefaultAsync(x => x.DocId == id);
        }
    }
    catch (Exception e)
    {
        e.FillModelState(ModelState);
        throw;
    }
}

```

```

[HttpDelete]
public override async Task<ActionResult<bool>> Delete([FromBody] Invoice obj, CancellationToken cancellationToken)
{
    try
    {
        using (var dataScope = _dataEngine.NewDataScope(cancellationToken))
        {
            var found = dataScope.GetRepository<Invoice>()
                // Initialize the child repository
                .Do(x => x.Instance.GetChild<DocSpec>())
                .Find(obj);
            if (found == null) return NotFound();
            return await found.SetDeleted(true).SubmitAsync();
        }
    }
    catch (Exception e)
    {
        e.FillModelState(ModelState);
        throw;
    }
}

```

```
[HttpPut]
public override async Task<ActionResult<Invoice>> Save([FromBody] Invoice obj, CancellationToken
cancellationTokentoken)
{
    try
    {
        using (var dataScope = _dataEngine.NewDataScope(cancellationTokentoken))
        {
            return await dataScope.GetRepository<Invoice>()
                // Initialize the child repository
                .Do(x => x.Instance.GetChild<DocSpec>(z => z.Specification),
                // Specify DeleteFlag
                x => obj.WithJsonSettings(
                    z => z.DeleteFlag(
                        y => y.Specification, y => y.Deleted)))
                .Apply(obj).SubmitAndReturnAsync();
        }
    }
    catch (Exception e)
    {
        e.FillModelState(ModelState);
        throw;
    }
}
```

16) A call was added to the *Get* method. [Do](#) is a helper that allows you to add processing to some object, and after the actions of this processing, we need to continue actions on this object, regardless of what methods that are used in the processing steps are returned. For example, here we use *Do* processing to initialize communication with a child repository. To do this, the repository that is accessible through the processing *Instance* property calls the *GetChild* method typed by *DocSpec* and specifies the *Specification* property where the child entities will be placed. See [documentation topic](#) for more details

□ Important

It is important to note that we do not always use binding with the child repository, but only when we really need to and specify the property where the child entities will be unloaded even less often. Such an approach allows us to significantly increase productivity due to the fact that we access the child repository only when it is really necessary and fill the array of child entities even less often.

17) In the *Delete* method, we initialize the connection with the child repository so that cascading deletion occurs, but (as noted above), we do not specify a property for storing child entities, since this is not required here.

18) In the *Save* method, we not only initialize the connection with the child repository for applying changes to the child objects, but also indicate that these child objects have an additional *Deleted* property not mapped to the database (now we can understand what it for) with which we we can tell the XData system that we need to delete this child entity. To do this, use the *WithJsonSettings* method, which is applied to the disconnected object received from the web. Here, the *DeleteFlag* method with parameters is used as a modifier — the property to which the child repository is configured and the property of the child object that is used as the delete flag. See [documentation topic](#) for more details

💡 Tip

It is important to note that this mechanism can be used for hierarchical construction of objects with several child repositories (and even with several differently filtered repositories of the same type), as well as for organizing hierarchically organized objects of unlimited nesting. It is only necessary to take into account the overhead affecting performance.

> [back to the Quick start contents](#)

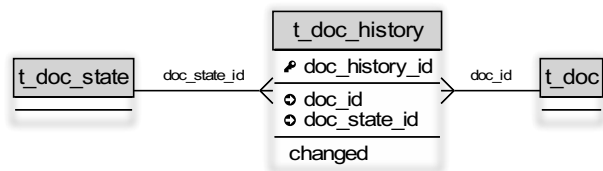
Trigger logic

When a task arises of maintaining information that is not directly related to the business logic of the application, for example, an audit task, it is convenient to take out the code leading the tracking of such information and store it separately from the main

application code. Moreover, it usually does not require user input. Nevertheless, it is often necessary to conduct an audit. The task given here is somewhat degenerate, but nevertheless it is simple and explains the essence of the approach, which is important for training.

In addition, in the course of a closer acquaintance with this mechanism, you will be able to discover not obvious aspects of its use and will be able to evaluate its significance in a new way.

1) Let's start as always with the database...



2) We describe the new mapping of *DocHistory* the and a class of logic. It is recommended to describe the logic and related class mappings in a separate assembly. This will allow loading module as plugin...

```
public partial class DocHistory
{
    public long DocId => this.GetProperty(x => x.DocId);
    public long DocStateId => this.GetProperty(x => x.DocStateId);
    public DateTime Changed { get; set; }
}
```

```
public partial class DocHistory : IDataObject
{
    private static Expression<CustomMapping<DocHistory>> _ =
        () => XDataMapping.CustomMapping<DocHistory>()
            .DataTable("T_DOC_HISTORY", "H")
            .ReadOnlyProperty(x => x.DocId,
                x => x.Field<long>("H", string.Empty))
            .ReadOnlyProperty(x => x.DocStateId,
                x => x.Field<long>("H", string.Empty))
            .Column("DocHistoryId",
                x => x.Field<long>("H", string.Empty, z => z.Key()))
            .Map(x => new DocHistory {
                Changed = x.Field<DateTime>("H", string.Empty)
            }).SetBaseTable("H");
}
```

```

public class InvoiceLogic : XDataLogic<Invoice>
{
    [Action(DataActionType.AfterInsert), Action(DataActionType.AfterUpdate)]
    public Trigger<Invoice> UpdateHistory =>
        ((ref Invoice invoice, ref DataTriggerFlag flag) =>
        {
            if (!invoice.CheckState(DataObjectState.New)
                && !invoice.IsChanged(x => x.DocState)) return true;
            var rep = invoice.GetRepository();
            using (var hist = GetRepository<DocHistory>(rep.Layer, context: rep.Context))
            {
                return hist.Do(x => x.Instance.Reset()
                    .SetFilterValue<DocHistory>(x => x.DocId, invoice.DocId)
                    .SetFilterValue<DocHistory>(x => x.DocStateId,
                        invoice.GetProperty<long>("DocStateId")))
                    .New(x => x.Changed = DateTime.Now)
                    .Submit());
            }
        });

    [Action(DataActionType.BeforeDelete)]
    public Trigger<Invoice> ClearHistory =>
        ((ref Invoice invoice, ref DataTriggerFlag flag) =>
        {
            var i = invoice;
            using (var hist = GetRepository<DocHistory>(
                i.GetLayer(), context: i.GetContext()))
            {
                return hist.Reset().Clear(x => x.DocId == i.DocId);
            }
        });

    [Action(DataActionType.BeforeClear)]
    public RepositoryTrigger<Invoice> ClearHistoryBatch =>
        ((IRepository<Invoice> invoiceRepository, ref DataTriggerFlag flag) =>
        {
            using (var hist = GetRepository<DocHistory>(invoiceRepository.Layer,
                context: invoiceRepository.Context))
            {
                return hist.Reset().Clear(x =>
                    invoiceRepository.Any(z => x.DocId == z.DocId));
            }
        });
}

```

- 3) The type of logic module must be inherited from the [XDataLogic<T>](#) class typed by the type of the parent data object (*Invoice*).
- 4) Inside the logic module, the properties of the delegate type are described which will be called during the processing of entities of the base type (*Invoice*). The moment of the logic call is set using the [Action](#) attribute to which [DataActionType](#), the type of handler, is passed as a parameter. In this case, the same handler can be assigned to several actions (this is shown by the example of the *UpdateHistory* handler). The types of delegates are different for [Trigger<T>](#) record-level handlers, which are triggered when changes are applied to the same object and the repository level - [RepositoryTrigger<T>](#), which are triggered by mass actions on repository objects.
- 5) It should be noted that a different [GetRepository](#) method is used here. It takes as parameters the identifier of the layer and the database context in which the base object (and its repository) are located. This allows us to get into the context of making changes to the base object.
- 6) In addition, the [Reset](#) method is used here, which clears the filters assigned to the repository instance and thereby frees us from the influence of its previous use.
- 7) Our new method will be [Clear](#). It is used to delete all repository objects or objects that satisfy a certain condition specified when

this method is called. In addition, you can configure the logic module to call a special handler before or after deleting objects using this method (as demonstrated by the *ClearHistoryBatch* delegate).

❏ Important

It is important to understand the difference between database level triggers and these XData level triggers. A database trigger is called during changes made to the records of the table, whereas XData logic module processes the changes in the level of the data object. It must be remembered that a data object can include several database tables on the one hand (the same *Invoice* object can serve as an example here) and, on the other hand, a part of information regarding other objects can be stored in one table (for example, the *T_DOC* table will contain data not only related to the *Invoice*). And from this point of view, the value of such a mechanism inside the XData can hardly be overestimated!

8) We register our logic module in the IoC container. In this case, we will provide the registration code directly during the initialization of the container, but you can use more complex usage scenarios if necessary.

```
public void ConfigureServices(IServiceCollection services)
{
    ...
    services.AddSingleton<IDataLogic<Invoice>, InvoiceLogic>();
    ...
}
```

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    ...
    app.UseXData();
    ...
}
```

9) In order to use the registration of logic modules in the IoC container, you need to call the [UseXData](#) method from the service provider.

❏ Note

Here we are considering the use of XData in ASP .Net Core. Registration of logic modules with the IoC container when .Net 4.0 version used the additional NuGet package [XData net 4.0 Inversion of Control container package](#) is required.

💡 Tip

Note that we use the `IDataLogic<T>` interface typed by our data object type *Invoice* as the registration interface in the IoC container

[> back to the Quick start contents](#)

Lesson II. Using variables

Use variable in mapping

Some filters can use [variables](#) as values. This allows more flexible use of the same mapping in different situations. Having described such a filter during mapping, you can specify various values of variables when using the repository.

- Use variables in Constant filters

```
csharp x => x.ConstFilter("ParamCode", "param_code", FilterConstantType.Variable, "ParamCode")
```

Here, the [ConstFilter](#) method is used to describe the filter. The *constantType* parameter in this case has the value [FilterConstantType.Variable](#) and the parameter value *constantValue* will already be interpreted not as a filter value, but as a variable name, the value of which will be used as a filter value.

- Use variables in Dictionary filters

```
csharp x => x.DictFilter("T_DOC_TYPE", "doc_type_id", "code", "DocType", true)
```

Now, to describe the filter, the [DictFilter](#) method overload is used with the *bool* parameter *valuesVariable*. And if it is set to *true*, the value of the *dictValue* parameter will be interpreted as the name of the variable, the value of which will be used as the filter value.

- Use variables as SQL procedure parameter

If in the description of *data object* the [SQL procedure or function](#) is used, then the values of their parameters are also transmitted as variables.

```
.Procedure("ORG", "bids.get_org_list_by_period_id", ProcedureType.Function)
.Parameter<long>("ORG", 1, "p_period_id", DbType.Int64)
.Return("ORG")
```

To call the [data object](#) with that SQL function used, we must set parameter *p_period_id* value with one of followed methods...

Set data object variable value

To set the values of variables *data object*, when receiving an instance of the repository using the [GetRepository](#) method, we can specify the values of the variables that are used in the mapping of this object. To do this, use the *variables* parameter.

```
dataScope.GetRepository<Doc>("DocType".SetVar("INVOICE"));
```

⚠ Warning

Variable name is case sensitive!

When describing an object being reused as a subquery (or inner view), it may often be necessary to use variables to filter data within the subquery. The value of the variable can be set for the final date of the object.

But if the value of a subquery variable is dictated by the logic of its use in external mapping or in external mapping, does such a subquery participate several times? In this case, it is possible to use an overload of the [Subquery](#) (or [InnerView](#)) method with the *variables* parameter:

```
//subquery define
public class DocPropertyValue : ISqlObject
{
    [UsedImplicitly]
    private static Expression<CustomMapping<DocPropertyValue>> _ =
    () => XDataMapping.CustomMapping<DocPropertyValue>()
        .DataTable("public.t_param", "P",
            x => x.ConstFilter("ParamCode", "param_code",
                FilterConstantType.Variable, "ParamCode")) //<= look here
        .DataTable("bids.t_doc_param", "V",
            x => x.Link("P", "param_id").SetOperation(FilterOperation.OuterJoin),
            x => x.ConstFilter("ParamActual", "version_state", VersionState.Actual))
        .Column("DocId", x => x.Field<long>("V", string.Empty, z => z.Hidden()))
        .Column("ParamValue", x => x.Field<string>("V", string.Empty, z => z.Hidden()))
        .Map().SetBaseTable("P");
}
...
.Subquery<DocPropertyValue>("EN",
    x => x.GetProperty("ParamValue"), DataGrouping.None,
    //set variable for EN subquery
    new[] { "ParamCode".SetVar("EXEC_FIO") },
    x => x.SubqueryLink("DocId"))
.Subquery<DocPropertyValue>("ET",
    x => x.GetProperty("ParamValue"), DataGrouping.None,
    //set variable for ET subquery
    new[] { "ParamCode".SetVar("EXEC_TEL") },
    x => x.SubqueryLink("DocId"))
```

> [back to the Quick start contents](#)

Lesson III. Using calculated properties

Calculate over data in other properties

Before considering methods for calculating property values based on complex algorithms. But in fact, most often, enough elementary calculations based on the values of other properties. The fact is that XData only works with properties that are described in mapping, but does not limit us in any way to adding properties to this object.

```
public string FirstName { get; set; }
public string LastName { get; set; }
public string MiddleName { get; set; }
public string Name => $"{FirstName} {(string.IsNullOrEmpty(MiddleName) ? string.Empty : $"
{MiddleName[0].ToString()}. ")}{LastName}";
```

Do not forget about [KISS](#) principle ;) and first of all consider this method of calculating the value.

[> back to the Quick start contents](#)

Calculate using SQL expressions mapping

XData supports several mapping methods for SQL expressions: mapping to [one field subquery result](#), to [LINQ-style expression](#) mapped to SQL, to [SQL plain text](#). Consider each of these methods and remove their advantages in various situations...

Here we will consider options when a field value can be obtained using SQL language tools. We will consider the option when we need some algorithmic logic to calculate the field value in the next part of the lesson...

Calculate using subquery SQL expressions

Yes This method is suitable if the data for the calculation can be obtained using a subquery.

No It is too complicated for calculations based on scalar functions, including those built into the DBMS (consider other methods from this section).

No It does not work if a subquery returns more than one field or more than one row (consider using an [inner view](#)).

No It does not work for table valued functions (consider using an [table valued function](#)).

We used this method several times earlier in previous lessons. To calculate a field value based on a subquery:

1) Add a read-only property to the object

```
public string ExecName => this.GetProperty(x => x.ExecName);
```

2) Describe the subquery, while we can describe the subquery in a separate class and then we can also use the described subquery in another mapping using [Subquery](#) method...

```
//subquery define
public class DocPropertyValue : ISqlObject
{
    [UsedImplicitly]
    private static Expression<CustomMapping<DocPropertyValue>> _ =
    () => XDataMapping.CustomMapping<DocPropertyValue>()
        .DataTable("public.t_param", "P",
            x => x.ConstFilter("ParamCode", "param_code",
                FilterConstantType.Variable, "ParamCode"))
        .DataTable("bids.t_doc_param", "V",
            x => x.Link("P", "param_id").SetOperation(FilterOperation.OuterJoin),
            x => x.ConstFilter("ParamActual", "version_state", VersionState.Actual))
        .Column("DocId", x => x.Field<long>("V", string.Empty, z => z.Hidden()))
        .Column("ParamValue", x => x.Field<string>("V", string.Empty, z => z.Hidden()))
        .Map().SetBaseTable("P");
}
...
//define subquery
.Subquery<DocPropertyValue>("EN",
    x => x.GetProperty("ParamValue"), DataGrouping.None,
    new[] { "ParamCode".SetVar("EXEC_FIO") },
    x => x.SubqueryLink("DocId"))
```

...or describe it directly in the mapping of the main [data object](#).

```
//define subquery
.Subquery("EN", XDataMapping.GetStructure("P")
    .DataTable("public.t_param", "P",
        x => x.ConstFilter("ParamCode", "param_code",
            FilterConstantType.Variable, "EXEC_FIO"))
    .DataTable("bids.t_doc_param", "V",
        x => x.Link("P", "param_id").SetOperation(FilterOperation.OuterJoin),
        x => x.ConstFilter("ParamActual", "version_state", VersionState.Actual))
    .Column("DocId", x => x.Field<long>("V", string.Empty, z => z.Hidden()))
    .Map(x => {
        ParamValue = x.Field<string>("V", string.Empty)
    })),
"ParamValue", DataGrouping.None,
x => x.SubqueryLink("DocId"))
```

3) Describe a property using a subquery described previously.

```
.ReadOnlyProperty(x => x.ExecName,
    x => x.Expr<string>(null, DataExpressionType.SubQuery,
        "EN", DbType.String, z => z.Size(1024)))
```

This uses the overload of the [Expr](#) method.

- The first parameter is used to indicate the field name in the resulting query. Note that the table is not used for expressions (more precisely, a [virtual data source](#) is used). Since the expression has the name of a property matching in the mapping, we can omit the field name by explicitly specifying *null* or *string.Empty* as the value. In this case, the name of the field will be substituted according to the rule used by default (*exec_name*).
- The second parameter indicates the type of SQL expression description. In this example, we are using [DataExpressionType.SubQuery](#) - which means using a subquery expression as the SQL source. Other types of descriptions will be discussed below.
- The next parameter is the actual text of the expression, and in the case of a subquery, the value of this parameter must be an alias of the subquery.
- Next is the value type of the expression field (in this example, this is a string).
- And then comes the parameter - an array of modifiers with the help of which the mapping-expression-field details are specified. The example uses the [Size](#) modifier with which we specify the maximum length of the expression string. Other

modifiers for expressions can be found [here](#) and [here](#).

Calculate using LINQ-style SQL expressions

Yes This method is suitable if the data for the calculation can be obtained using scalar functions, including those built into the DBMS.

Yes It is suitable if the expression is a selection based on the data of the current fetch.

No It does not work for table valued functions (consider using an *table valued function*).

No The method does not allow the use of specific language constructs that are not supported by XData.

1) SQL expression based on scalar function call...

```
...
coalesce(DOC.state, 0) as state
...
```

mapped as...

```
.ReadOnlyProperty(x => x.State, x => x.Expr(null,
    z => z.SqlFn<Common.DocState>("coalesce",
        a => a.CastTo<Common.DocState>().Field<int?>("DOC", "state"),
        a => Common.DocState.Absent), DbType.Int32))
```

Here we use the other overload of the [Expr](#) method.

- The first parameter is the same (field name)
- The second parameter is a description of the expression. In this example, the [SqlFn](#) method is used as the basis for describing the SQL expression, with which you can describe the call of a scalar function built into the DBMS or declared in the database by the developer. The typing parameter of the *SqlFn* method indicates the type of the result of the expression (in this example, an *DocState* enum). The first parameter is the name of the SQL function. The rest are the values that will be passed to it as parameters. An interesting point here is to use the [CastTo](#) method to cast the type of source information to the resulting type at the time of mapping transformation.
- The third parameter is the type of the expression field (*int*).

2) SQL expression based on switch case expression...

```
...
case DOC.state when null then null when 1 then null else DOC.version_time end as state_date
...
```

mapped as...

```
.ReadOnlyProperty(x => x.StateDate, x => x.Expr(null,
    z => z.Case(y => y.Field<int?>("DOC", "state"),
        a => a.Field<DateTime?>("DOC", "version_time"),
        new int?().SetExpression(a => new DateTime?()),
        new int?(1).SetExpression(a => new DateTime?())), DbType.DateTime))
```

Here you need to pay attention to the [Case](#) method used to describe switch case expression. And especially on the order of parameters.

- The first parameter is the field / expression used as the option selection key.
- But the second parameter is the value if the value does not meet any condition. The fact is that there can be several parameters describing the options of values and it is convenient to describe them in the form of an *params* array. And the value of *else* is one.
- The last parameter is a description of the value options. Each variant is described using the [SetExpression](#) method and the

object is the value of the variant (in the example, there is an empty *null* value and a constant), and the parameter is the result of switch case expression.

Similarly, there are methods to facilitate the description of SQL statements: [DateDiff](#) and [HasFlag](#). Their study is recommended for self-study. There are no specifics regarding the options already considered in their use.

3) Interpretation of simple expressions in SQL. All basic (embedded in the csharp language) operations and functions (*) of working with dates and strings are supported. Based on the examples discussed earlier, the following should not cause difficulties in understanding ...

```
.ReadOnlyProperty(x => x.Bids,
    x => x.Expr("bids", a => a.Field<string>("B", "app_type") != null,
        DbType.Boolean))
```

```
.ReadOnlyProperty(x => x.Fio,
    x => x.Expr("Fio", o => o.Field<string>("C", "surname") + " " +
        o.Field<string>("C", "name").Substring(0, 1) + "." +
        o.Field<string>("C", "patronymic").Substring(0, 1) + ".", DbType.String))
```

* - With the exception of *string.Format* - at the moment, differences in the way text information is formatted in various SQL dialects do not allow interpretation of this function. Maybe in the next versions...

Calculate using plain SQL expressions

Yes This method is suitable when XData cannot construct an expression, but it can nevertheless be expressed using SQL.

Yes We can afford to support one particular dialect of SQL

No This method is the "refuge of last hope" and it is not recommended to use this method if it is possible to use other methods from this article.

This method allows using plain SQL inserts when describing mapping of properties to specific SQL expressions. To do this, when calling the *Expr* method, [DataExpressionType.PlainSql](#) is specified as the second parameter, and plain SQL expression as the third.

```
.ReadOnlyProperty(x => x.RowNum,
    x => x.Expr<int>("RowNum", DataExpressionType.PlainSql,
        "row_number() over (order by o.sort_order, c.surname)",
        DbType.Int32))
```

In difficult cases when you need to assemble an SQL expression from parts, you can use the [SqlExpr](#) method which allows you to use a format string and substitution from the names of the fields of query data sources and the values of variables of both ordinary and data object variables (using [GetVariable](#) method).

```
.ReadOnlyProperty(x => x.Something, x => x.Expr(null,
    z => z.SqlExpr<int>("{0} something {1} {2}",
        a => a.Field<string>("C", "name")
        a => someVariable,
        a => a.GetVariable<int>("VarName"))
```

[> back to the Quick start contents](#)

Calculate using logic module

In the most difficult cases, when filling out data, it is necessary to analyze data from external data relative to a particular object using some algorithms. In some DBMSs, for this it is possible to describe calls to external libraries as function calls. A more common solution is post data processing. In this case, after receiving the primary data, an additional pass is made with the filling of additional calculated data.

XData has a mechanism for the [logic modules](#) of a [data object](#). We came across them when considering the trigger logic of data objects. But to calculate the additional data, another type of logic [DataActionType.InitObject](#) is used, it is performed during the

object transformation from relational data.

Suppose that we have some reporting system. When requesting report parameters, we get their descriptions and default values from the database, but the value of the date / time parameter must be calculated. Since the database does not store ready-made values, but only data for their calculation (for example, the last Sunday 00:00).

1) First of all, we need to add property that are not involved in the mapping to the description of our *data object*:

```
public partial class ParamDefaultDescription
{
    ...
    public DateTime? DateVal { get; set; }
}
```

2) When requesting report parameters, we fill in the data object variable with the moment the input arrives at the report generation. We will need these data for calculation.

```
var date = DateTime.Now;
var v = new[] {nameof(userLogin).SetVar(userLogin), nameof(date).SetVar(date)};
using (var dataScope = _dataEngine.NewDataScope(Startup.ReportDb))
{
    return await dataScope.GetRepository<ParamDescription>(v)
        .Do(x => x.Instance.GetChild(z => z.ParamDefault, variables: v),
            x => x.Instance.GetChild(z => z.ParamRule, variables: v))
        .Where(x => templateName == x.TemplateName && x.IsPublic == true)
        .ToArrayAsync();
}
```

3) We describe the logic module of the data object.

```
public class ParamDefaultDescriptionLogic : XDataLogic<ParamDefaultDescription>
{
    [Action(DataActionType.InitObject)]
    public InitObject<ParamDefaultDescription> InitObject =>
        ((ref ParamDefaultDescription def) =>
        {
            if (def.Type.HasFlag(ParamDefaultType.Date))
            {
                var d = def;
                var id = d.Id;
                using (var list = GetRepository<ParamDefault>(d.GetLayer(), context: d.GetContext()))
                {
                    var obj = list.SingleOrDefault(x => x.Id == id);
                    if (obj?.ParamDefaultDateType != null)
                    {
                        var date = (DateTime)def.GetRepository().Variables["date"];
                        def.DateVal = ReportBuilder.GetDate(date, obj.ParamDefaultDateType.Value,
                            obj.ParamDefaultDateAddType, obj.ParamDefaultDateAdd);
                    }
                }
            }
        });
}
```

The data module must be an [XDataLogic](#) descendant with the typing of our data object. Inside should be a property of type [InitObject](#) typed by our type. And this property should be marked with an [Action](#) attribute with the [DataActionType.InitObject](#) parameter.

Inside the object initialization event handler, there should be logic to fill in the value of our property if necessary.

> [back to the Quick start contents](#)

Tips & Tricks

Contents

Configuration

YAML configuration file

Adapter and dialect specific configuration

Set cache lifetime limit

Mapping

Mapping view

Optional filters

Outer joined table filtration specifics

Tree organized repository child specifics

Property update source

Default value features

Cast to type

Using EXISTS expressions

Use some LINQ analogs in mapping

Using

Get connection settings in runtime

Using DI container

Large object loading

JSON serialization

Conditional filters

Asynchronous wrapped operations

Using dictionaries

Caching repository data

Copy data

Using events

Dynamic query

Reuse mapping in dynamic query

Dynamic query external data sources

Data aware logic

[Trigger business logic](#)

[Execute custom business logic](#)

[Using business logic callbacks](#)

[Business logic calculated fields](#)

Testing

[Virtual data and attached handlers](#)

SQL

[Transactions](#)

[SQL hints](#)

[Call custom SQL code](#)

Configuration

YAML configuration file

(for .Net Standard 2.0 only!)

Since .Net Core and .Net Standard 2.0 supported various configuration file formats it's possible to configure XData with, as a sample, YAML file.

```
connectionStrings:
  add:
    - name: "DICT"
      connectionString: "Server=server;Database=test;User Id=postgres;Password=123456;Search
Path=dictionaries"
    - name: "CONTACTS"
      connectionString: "Server=server;Database=test;User Id=postgres;Password=123456;Search
Path=contacts"
xdata:
  adapter:
    - name: "DICT_POSTGRESQL"
      assembly: "XData.PostgreSql.Npgsql.Adapter"
      schema:
        dictionaries:
    - name: "CONTACTS_POSTGRESQL"
      assembly: "XData.PostgreSql.Npgsql.Adapter"
      schema:
        contacts:
  dialect:
    - name: "POSTGRESQL"
      assembly: "XData.PostgreSql.Dialect"
  context:
    - name: "DICT"
      adapter: "DICT_POSTGRESQL"
      dialect: "POSTGRESQL"
    - name: "CONTACTS"
      adapter: "CONTACTS_POSTGRESQL"
      dialect: "POSTGRESQL"
  default: "CONTACTS"
  debug: true
```

Tip

Note, the PostgreSQL connection strings can specify schema name in a *Search Path* key! That is required to work with tables from

different then *public* schema without specify schema name in table name definitions.

Tip

Names of adapters, that use various connection strings must to be unique! Names of contexts must correspond connection strings names.

Tip

Be aware with PostgreSQL specific adapter configuration section *schema* required to specify home schema name for an adapter. Adapter and dialect specific configuration sections has described [here](#).

> [back to the Tips & Tricks contents](#)

Adapter and dialect specific configuration

Some of adapters and dialects has their own specific configuration settings to setup unique features and behaviors. That specific settings can be defined in configuration file or dynamically with [AddAdapter](#) or [AddDialect](#) methods *options* parameter.

Tip

Function signatures is slightly different for .Net 4.0 version.

XData.PostgreSql.Npgsql.Adapter:

... when XML configuration file is used:

```
<adapter name="POSTGRESQL_NPG" assembly="XData.PostgreSql.Npgsql.Adapter">
  <schema name="public"/>
</adapter>
```

... when YAML configuration file is used:

```
adapter:
  - name: "DICT_POSTGRESQL"
    assembly: "XData.PostgreSql.Npgsql.Adapter"
    schema:
      dictionaries:
```

... when configuration is provided in runtime:

```
// replace someschemaname to real schema name
var source = new MemoryConfigurationSource { InitialData = new[] {
    new KeyValuePair<string, string>("schema:someschemaname", null) }};
var conf = new ConfigurationBuilder().Add(source).Build();
...
.AddAdapter("postgresql", XDataAdapter.PostgreSql.Adapter, conf)
```

XData.MsSqlServer.Dialect:

... when XML configuration file is used (for modern versions of MSSQL):

```
<dialect name="MSSQL" assembly="XData.MsSqlServer.Dialect"/>
```

or if you have to use MSSQL 2005/2008

```
<dialect name="MSSQL" assembly="XData.MsSqlServer.Dialect">
  <useOldOrderBy value="true"/>
</dialect>
```

... when YAML configuration file is used (for MSSQL 2005/2008):

```
dialect:
  - name: "DICTION_MSSQL"
    assembly: "XData.MsSqlServer.Dialect"
    useOldOrderBy:
      value: true
```

... when configuration is provided in runtime (for MSSQL 2005/2008):

```
var source = new MemoryConfigurationSource { InitialData = new[] {
    new KeyValuePair<string, string>("useOldOrderBy:value", "true") }};
var conf = new ConfigurationBuilder().Add(source).Build();
...
.AddDialect("mssql", XDataAdapter.MsSqlServer.Dialect, conf)
```

[> back to the Tips & Tricks contents](#)

Set cache lifetime limit

In Web applications, one of the problems with long-lived objects is that when you close a page or browser, the server may not receive information that you need to clear certain objects.

A typical solution is to set the maximum lifetime of cached information. In the case of XData, such long-lived objects can be the [IDataScope](#) data layer and Unit of Work container [IWorkSet<T>](#).

If XData is initialized, the code uses the [SetLifetimeLimit](#) modifier when calling the [AddXData](#) method.

If XData is initialized, the *lifetime* attribute must be specified in the *xdata* section in the configuration file and the value of the maximum lifetime of the cached information in minutes must be specified.

⚠ Warning

To work with XData version .Net 4.0 additional package [XData Cache package](#) must be used (required full Net 4.0 Framework version). Net Standard 2.0 version does not require additional packages.

[> back to the Tips & Tricks contents](#)

Mapping

Mapping view

To map [data objects](#) to SQL views used same techniques and commands as data table, but then mapping non read-only [repository](#) to table some fields of [updatable table](#) can be mapped automatically by XData engine using meta-data information from source database. When mapping is described over view, all fields must to be described explicitly. Same rule is worked when source of data is text represented SQL expression or virtual record-set defined as union of records (see [ToDataSet](#) as example, similar result can be reached using string SQL expression as table name). All fields must to be described explicitly.

⚠ Important

Repository mapping is required to marked as [ReadOnly](#) when view is a base table!

[> back to the Tips & Tricks contents](#)

Optional filters

Optional filters is the mechanic to simplify query structure when some filters are not set. For example, we can map data structure to some class and to allow filter objects over some property of dictionary linked to main data structure. When filter applied this link to dictionary table is used, but when not we ineffectively rise query complexity.

To avoid unplanned query complexity XData has *optional filter* feature. Some filters marked as *primary* - when this filter has value, query has full form. When filter value has cleared or set to null, data source of filtered field (table or view) will be skipped when SQL query composed. When data source has been skipped then links between already skipped and other data sources are

analyzed. And when found link marked as *primary* linked data source will also skipped. When [base table](#) of subquery has been skipped - all subquery will be skipped and subquery links analysis for *primary* links will be provided over outer query. Until no *primary* filters found.

This allows to compose complex query relations over multiple tables and subqueries that reflect on real query only when it really necessary.

Let's explain that on example...

```
public partial class TestOptional: IDataObject
{
    private static Expression<CustomMapping<TestOptional>> _mapping = (
        () => XDataMapping.CustomMapping<TestOptional>()
            .DataTable("T_TEST", "T",
                x => x.SubqueryFilter(string.Empty, "A")
                    .SetOperation(FilterOperation.Exists)
            ).Subquery("A", XDataMapping.GetStructure("H")
                .DataTable("T_TEST_OPT", "H")
                .DataTable("T_OPT", "P",
                    // Link between H and P table marked as Primary filter
                    x => x.Link("H", "opt_id").AsPrimary(),
                    // ConstFilter "OptCode" marked as Primary filter
                    x => x.ConstFilter("OptCode", "code", null).AsPrimary())
                .Select(x => new {
                    TestId = x.Field<long>("H", string.Empty)
                }, "TestId", DataGrouping.None, x => x.SubqueryLink("DocId"))
            ).Column("TestId", x => x.Field<long>("T", string.Empty))
            .Map(x => new TestOptional {
                Name = x.Field<decimal>("T", string.Empty)
            }).SetBaseTable("T"));
}
```

When repository accessed like that...

```
dataScope.GetRepository<TestOptional>().ToArray();
```

XData generate a simple SQL...

```
select T.test_id as T_test_id, T.name as T_name from T_TEST T
```

Because...

1. ConstFilter "OptCode" has null value and marked as primary => T_OPT table skipped
2. T_OPT table skipped and Link to T_TEST_OPT marked as primary => T_TEST_OPT table skipped
3. T_TEST_OPT table skipped and marked as base table => subquery is skipped and SubqueryFilter skipped as well, but because it not marked as primary => table T_TEST is not skipped

And when filter is filled...

```
dataScope.GetRepository<TestOptional>()
    .SetFilterValue(new FilterName("OptCode"), "TEST").ToArray();
```

Now XData generate SQL with optional parts...

```
select T.test_id as T_test_id, T.name as T_name from T_TEST T
where exists(select 1
    from T_TEST_OPT H join T_OPT P on H.opt_id = P.opt_id
    where P.code = "TEST")
```

[> back to the Tips & Tricks contents](#)

Outer joined table filtration specifics

```
XDataMapping.CustomMapping<UserList>()
.DataTable("t_user", "U")
// *t_user_claim* has inner joined to *t_user* by *user_id* field
.DataTable("t_user_claim", "UC", z => z.Link("U", "user_id"),
    // constant filter use default equal operation (see SQL result below)
    z => z.ConstFilter("ClaimNickName", "claim_type", "nickname"))
// *t_user_app* has outer joined to *t_user* by *user_id* field
.DataTable("t_user_app", "A",
    x => x.Link("U", "user_id").SetOperation(FilterOperation.OuterJoin),
    // IMOPRTANT! all filters for outer joined table must use *OuterJoin* operation to apply correct
    filtration
    x => x.ConstFilter("FilterByAsubs", "app_type", (int)
AppType.Asubs).SetOperation(FilterOperation.OuterJoin),
    x => x.ConstFilter("FilterByAsubsEnabled", "enabled", 1).SetOperation(FilterOperation.OuterJoin))
```

```
select ... from t_user U
join t_user_claim UC on UC.user_id = U.user_id
outer join t_user_app A on A.user_id = U.user_id and A.app_type = 1 and A.enabled = 1
```

Constant filters are moved to outer join condition not in where statement conditions and query will return correct results.

Similar rules is applied to static mapping and dynamic query.

[> back to the Tips & Tricks contents](#)

Tree organized repository child specifics

When master repository has hierarchy structure, than detail objects can be filtered by two methods to choose detail repository data: filter data linked with selected tree node only ([direct links](#)), and linked to subtree from selected node and below by hierarchy. XData has functional to choose one of this methods in runtime. To provide this ability mapping of master objects must describe [external links](#) marked as [direct links](#). Than master repository flag [ShowSubtree](#) can be used as a switch between this two methods.

Let's explain that on example...

Mapping master tree organized data object...

```
private static Expression<CustomMapping<Catalog>> _unused = (
    () => XDataMapping.CustomMapping<Catalog>()
    .DataTable("R", "T_CATALOG")
    .Map(x => new Catalog{
        CatalogId = x.Field<long>("R", string.Empty, z => z.Key()),
        Code = x.Field<string>("R", string.Empty),
        Name = x.Field<string>("R", string.Empty),
        ParentId = x.Field<long?>("R", string.Empty) },
    // Direct link
    x => x.ExternalLink<Document>(x => x.CatalogId, x => x.DirectLink()),
    // Subquery link
    x => x.ExternalLink<Document>(x => x.CatalogId,
        z => z.Filter(y => new FilterName("FilterByRoot"))))
    .SetBaseTable("R").SetFlags(DataStructureFlag.Tree)
);
```

Master data object SQL...

```
SELECT R.catalog_id AS R_catalog_id,
    R.code AS R_code,
    R.name AS R_name,
    R.parent_id AS R_parent_id
FROM T_CATALOG R
```

Mapping detail data object...

```
private static Expression<CustomMapping<Document>> _unused = (
    () => XDataMapping.CustomMapping<Document>()
    .DataTable("D", "T_DOC")
    .DataTable("DC", "T_DOC_CATALOG", x => x.Link("D", "doc_id"),
        x => x.SubqueryFilter("catalog_id", "UT")
        .SetOperation(FilterOperation.In))
    .Subquery<CatalogTree>("UT", x => x.CatalogId, DataGrouping.None)
    .Map(x => new Document{
        DocId = x.Field<long>("D", string.Empty, z => z.Key()),
        DocDate = x.Field<DateTime>("D", string.Empty),
        CatalogId = x.Field<long>("DC", string.Empty) })
    .SetBaseTable("D")
);

...
private static Expression<CustomMapping<CatalogTree>> _unused = (
    () => XDataMapping.CustomMapping<CatalogTree>()
    .With("CAT", XDataMapping.GetStructure("R")
        .DataTable("T_CATALOG", "R",
            x => x.ConstFilter("FilterByRoot", "parent_id",
                FilterConstantType.Const, null,
                FilterNullable.NullsCompared))
    .Select(x => new {
        CatalogId = x.Field<long>("R", string.Empty, z => z.Key()),
        ParentId = x.Field<long?>("R", string.Empty),
    }, x => x.Properties(z => z.CatalogId, z => z.ParentId),
    x => x.WithRecursive("S", XDataMapping.GetStructure("S")
        .DataTable("T_CATALOG", "S")
        .Select(y => new {
            CatalogId = y.Field<long>("S", string.Empty, z => z.Key()),
            ParentId = y.Field<long?>("S", string.Empty)
        })),
        z => z.SubqueryLink(y => y.ParentId, y => y.CatalogId)))
    .ReadOnlyProperty(x => x.CatalogId,
        x => x.Field<long>("CAT", string.Empty, z => z.Key()))
    .ReadOnlyProperty(x => x.ParentId,
        x => x.Field<long?>("CAT", string.Empty))
    .Map().SetFlags(DataStructureFlag.ReadOnly)
);
```

Link and using data objects with *ShowSubtree* off...

```
dataScope.GetRepository<Catalog>()
    .Do(x => x.Instance.GetChild<Document>())
    .ToArray();
```

Detail data object SQL...


```

with recursive T (catalog_id, code, name, parent_id)
as ((SELECT R.catalog_id AS R_catalog_id,
      R.parent_id AS R_parent_id
    FROM T_CATALOG R
    WHERE R.catalog_id = @FilterByRoot)
  UNION ALL (SELECT S.catalog_id AS S_catalog_id,
      S.parent_id AS S_parent_id
    FROM T_CATALOG S, T T
    WHERE S.parent_id = (T.catalog_id)))
SELECT D.doc_date AS D_doc_date /* DocDate */,
      D.doc_id AS D_doc_id /* DocId */,
      DC.catalog_id AS DC_catalog_id /* CatalogId */
FROM T_DOC D
  JOIN T_DOC_CATALOGUE DC ON DC.doc_id = D.doc_id
WHERE DC.catalog_id in (SELECT T.catalog_id AS T_catalog_id FROM T T)
  and DC.catalog_id = @FilterByCatalogId

```

Direct link is filled and detail data object data is restricted by single catalogue_id from master data object.

Link and using data objects with *ShowSubtree* on...

```

dataScope.GetRepository<Catalog>()
    .Do(x => x.Instance.GetChild<Document>()),
    x => x.Instance.ShowSubtree = true)
    .ToArray();

```

Detail data object SQL...

```

with recursive T (catalog_id, code, name, parent_id)
as ((SELECT R.catalog_id AS R_catalog_id,
      R.parent_id AS R_parent_id
    FROM T_CATALOG R
    WHERE R.catalog_id = @FilterByRoot)
  UNION ALL (SELECT S.catalog_id AS S_catalog_id,
      S.parent_id AS S_parent_id
    FROM T_CATALOG S, T T
    WHERE S.parent_id = (T.catalog_id)))
SELECT D.doc_date AS D_doc_date /* DocDate */,
      D.doc_id AS D_doc_id /* DocId */,
      DC.catalog_id AS DC_catalog_id /* CatalogId */
FROM T_DOC D
  JOIN T_DOC_CATALOGUE DC ON DC.doc_id = D.doc_id
WHERE DC.catalog_id in (SELECT T.catalog_id AS T_catalog_id FROM T T)

```

Direct link not filled and data not restricted by single catalogue_id from master data object. Resulted detail data will match master data object current record subtree.

[> back to the Tips & Tricks contents](#)

Property update source

Data object properties (or **hidden properties**) values can depend on other properties values or calculated with some expression. XData allow to describe this expression or dependency as part of mapping, to skip that property initialization when inserting or updating data object. Property value update source is used when data object is inserted or updated. When property **default value** and update source are defined, default value is used on insert and update source on update.

Property value update source expression when using static mapping is defined by **PropertyUpdateWithAttribute** (or **ColumnUpdateWithAttribute**) with same rules as PlainSQL or LinqExpression **SQL expression properties**:

```
// for PlainSQL
[PropertyUpdateWith("case P.is_vip when 1 then 10 else 0 end")]
// or for LinqExpression
[PropertyUpdateWith("AllowedDiscount", DataExpressionType.LinqExpression)]
...
[SqlExpression]
private static Calculate<int> AllowedDiscount = x => x.Case<Product, int>(
    z => z.Field<bool>("is_vip"), z => 0, 1.SetExpression(z => 10));
```

When using dynamic mapping it is defined by [Field's](#) mapping attribute [UpdateWith](#) overloads:

- [UpdateWith\(DataExpressionType type, string exprText\)](#)
- [UpdateWith\(Expression<Func<IQueryStructureAdapter, TValue>> expr\)](#)

```
csharp DeliveryDate = x.Field<DateTime?>("DD", string.Empty, z => z.UpdateWith(y =>
DateTime.Today.AddDays(1)))
```

Property value update source dependency when using static mapping is defined by [PropertyUpdateWithAttribute](#) (or [ColumnUpdateWithAttribute](#)) with same rules as Subquery [SQL expression properties](#):

```
[Subquery("A" /* subquery alias */,
    typeof(DocSpecAmounts) /* subquery mapped type */,
    "Amount" /* subquery result property name */,
    Grouping = DataGrouping.Sum /* result property aggregation type */)
...
[PropertyUpdateWith("A", DataExpressionType.SubQuery)]
```

When using dynamic mapping it is defined by [Field's](#) mapping attribute [UpdateWith](#) overloads:

- [UpdateWith\(string alias, Type dobjType, string property, DataGrouping grouping = DataGrouping.None, params Expression<Func<ISubqueryLinkAdapter, IStructureFlag>>\[\] attributes\)](#)
- [UpdateWith\(string alias, IQueryDescription<TObj> sub, string property, DataGrouping grouping = DataGrouping.None, params Expression<Func<ISubqueryLinkAdapter, IStructureFlag>>\[\] attributes\)](#)
- [UpdateWith\(string alias, IQueryDescription<TObj> sub, Expression<Func<TObj, TValue>> property, DataGrouping grouping = DataGrouping.None, params Expression<Func<ISubqueryLinkAdapter, IStructureFlag>>\[\] attributes\)](#)

```
csharp DeliveryType = x.Field<DeliveryTypeEnum>("DD", string.Empty, z => z.UpdateWith("DT",
XDataMapping.GetStructure().Select(y => new { DeliveryType = y.Expr("delivery_type", a =>
DeliveryTypeEnum.Courier, DbType.Int32)}), y => y.DeliveryType, DataGrouping.None)),
```

[> back to the Tips & Tricks contents](#)

Default value features

[Data object](#) properties (or [hidden properties](#)) can be assigned by default values described by mapping.

Using [DefaultFeature](#) enumeration it's possible to extend basic functionality of default value assignment with...

- **UseOnUpdate** - use default value not only on insert, but on update also.
- **SkipWhenAssigned** - skip insert into table when PK value assigned explicitly.
- **UpdateWhenAssigned** - change insert operation over table to update when PK value assigned explicitly.

Last two of features is usable when data object 's primary table is linked with another table in data object mapping as one-to-one, and insert into base table can be skipped (or replaced with update statement) when it's primary key is assigned explicitly in application code.

This features helps logically combine mappings to hide technological entities, which don't reflect business object model of application.

Default features can be combined. For example:

```
DefaultFeature.UseOnUpdate | DefaultFeature.SkipWhenAssigned
```

[> back to the Tips & Tricks contents](#)

Cast to type

To cast type of *Field* or *Expression* while mapping declaration can be reached using [CastTo](#) method.

```
...
OrgSubType = x.Expr("org_subtype",
    z => z.CastTo<OrgSubType>().Field<string>("S", "param_value"), DbType.String)
...
```

[> back to the Tips & Tricks contents](#)

Using EXISTS expressions

in filters...

```
...
where exists(select 1 from...)
...
```

...described when *Static mapping* used as...

```
...
[Subquery("A", typeof(AttachmentsExists), "DocId")]
[SubqueryLink("A", "DocId")]
...
[SubqueryFilter("A", "", "DocId", Operation=FilterOperation.Exists)]
...
```

* If it is necessary to describe the expression NOT EXISTS, the *Operation* parameter must be set to [FilterOperation.NotExists](#)

and when *Dynamic mapping* used as...

```
...
.Subquery<AttachmentsExists>("A", x => x.GetProperty("DocId"), DataGrouping.None,
    x => x.SubqueryLink("DocId"))
...
x => x.SubqueryFilter(null, "A").SetOperation(FilterOperation.Exists)
...
```

* If it is necessary to describe the expression NOT EXISTS, the modifier [SetOperation](#) must be called with the parameter [FilterOperation.NotExists](#)

in fields...

```
...
exists(select 1 from...) as attach_exists
...
```

...described when *Static mapping* used as...

```
...
[Subquery("A", typeof(AttachmentsExists), "DocId")]
[SubqueryLink("A", "DocId")]
...
[Property, PropertyExpression("A", DataExpressionType.SubQuery, DbType.Boolean,
Exists=ExistsOperation.NotExists)]
public bool AttachExists => this.GetParameter(x => x.AttachExists);
...
```

* If it is necessary to describe the expression NOT EXISTS, the *Exists* parameter must be set to [ExistsOperation.NotExists](#)

and when *Dynamic mapping* used as...

```
...  
.Subquery<AttachmentsExists>("A", x => x.GetProperty("DocId"), DataGrouping.None,  
    x => x.SubqueryLink("DocId"))  
...  
.ReadOnlyProperty(x => x.AttachExists,  
    x => x.Expr<bool>(null, DataExpressionType.SubQuery, "A", DbType.Boolean,  
        z => z.Exists()))  
...
```

* If it is necessary to describe the expression NOT EXISTS, the modifier [Exists](#) must be called with the parameter [ExistsOperation.NotExists](#)

[> back to the Tips & Tricks contents](#)

Use some LINQ analogs in mapping

⚠ Warning

Usually the use of LINQ analogues of operations in mapping is redundant and does not allow the most complete reuse of mapping results. But in the case when it is necessary to apply operations not to the final query but to its part (subquery or inner view), there are no options...

Distinct

when *Static mapping* used as...

```
...  
[DataObject("S", Distinct = true)]
```

and when *Dynamic mapping* used as...

```
...  
.Distinct()  
...
```

Skip

when *Static mapping* used as...

```
...  
[DataObject("S", Skip = 10)]
```

and when *Dynamic mapping* used as...

```
...  
.Skip(10)  
...
```

Take

when *Static mapping* used as...

```
...  
[DataObject("S", Limit = 1)]
```

and when *Dynamic mapping* used as...

```
...  
.Limit(1)  
...
```

OrderBy...

Sorting options can be specified when mapping fields...

when *Static mapping* used as...

```
...
[Column("DocStateId", typeof(long?), "S", Flags = DataPropertyFlag.Id,
    OrderBy = 1, /* set order by position for mapped field */
    OrderByDesc = true)] /* (optional) set descent sort order */
...
[Property(OrderBy = 1, /* set order by position for mapped field */
    OrderByDesc = true)] /* (optional) set descent sort order */
...
[Reference(OrderBy = 1, /* set order by position for mapped field */
    OrderByDesc = true)] /* (optional) set descent sort order */
```

and when *Dynamic mapping* used as...

```
...
x.Field<string>("S", String.Empty, z => z.OrderBy(1))
...or for descent sort order
x.Field<string>("S", String.Empty, z => z.OrderBy(1, true))
```

[> back to the Tips & Tricks contents](#)

Using

Get connection settings in runtime

Connection settings can be obtained from initialized context using code below:

```
var ctx = dataScope.GetContext(s);
Console.WriteLine("*** Testing context: {0} (DB type: {1})", ctx.Name, ctx.DatabaseAdapter.ConnectType);
Console.WriteLine("*** Server: {0}, Schema: {1}", ctx.ConnectionInfo.Server, ctx.ConnectionInfo.Schema);
```

[> back to the Tips & Tricks contents](#)

Using DI container

XData can be effectively used with any DI containers You prefer.

When using the .Net Standard 2.0 version of XData, the .Net Core Dependency Abstraction technology is applied internally and can be implemented with any DI container. The [IDataEngine](#) interface is injected as dependency by calling [AddXData](#) method during *ConfigureServices* method of *Startup* module. Any [trigger logic modules](#) can be registered as singleton services in DI container as well.

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddXData(x => x.AddDialect(Constants.DialectName,
        XDataAdapter.PostgreSQL.Dialect)
        .AddAdapter(Constants.AdapterName, XDataAdapter.PostgreSQL.Adapter)
        .AddContext(Constants.TestDb, Constants.DialectName,
            Constants.AdapterName, testDb, true));
    ...
    services.AddSingleton<IDataLogic<Invoice>, InvoiceLogic>();
    ...
}
```

■ Note

The registration of *logic modules* must be performed using the [IDataLogic<T>](#) interface

Tip

The number of logic modules for each data object is not limited, but it must be kept in mind that the order in which various handlers are called is not guaranteed

If any *logic module* are injected through DI container [UseXData](#) method required to call during *Configure* method of *Startup* module.

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    ...
    app.UseXData();
    ...
}
```

When using .Net 4.0 XData version [IDataEngine](#) interface can be obtained from container similarly. In this case, you must use an additional NuGet package [XData net 4.0 Inversion of Control container package](#). [IDataEngine](#) must be registered as singleton with *XDataManager.InitXData* method call. If any *logic module* are injected through DI container *UseXDataDependencies* extension method similar to previously reviewed before version for .Net Standard 2.0 but assigned to *IDependencyResolver* interface is required to call. Here is an example using Autofac but you can use any DI container implementation analogically...

```
var builder = new ContainerBuilder();
builder.Register<IDataLogic<Invoice>>(x => new InvoiceLogic());
builder.Register(x => XDataManager.InitXData(z =>
    z.UseConfiguration(ConfigurationManager
        .OpenExeConfiguration(ConfigurationUserLevel.None))););
var container = builder.Build();
using (var resolver =
    new AutofacApiDependencyResolver(container).UseXDataDependencies())
{
    using (var dataEngine = (IDataEngine)resolver.GetService(typeof(IDataEngine)))
    {
        ...
    }
}
```

Any IoC container with [IDependencyResolver](#) interface implementation is supported. If the selected container DI implementation does not support this interface, it is simple and can easily be implemented independently.

[> back to the Tips & Tricks contents](#)

Large object loading

Large objects specific properties [Lob](#) and [Xml](#) specially used to lazy access to potentially big sized data stored in database.

When [data object](#) accessed using XData, large objects data not queried, but replaced with null values to create full structure of object.

To get or set value of *Lob* or *Xml* [Lob.Value](#) and [Xml.Document](#) properties are used. Size of *Lob* can be checked using [GetSize](#) method.

To simplify value assigning overridden operation `+=` can be used as described in example below:

```
obj.Source += new XDocument(new XElement("invoice", new XAttribute("number", number), new XAttribute("state",
z.DocStateCode)));
obj.Scan += Encoding.UTF8.GetBytes(z.Source.Document.ToString());
```

To modify value of large objects in *LINQ styled sequence of command* helper methods [Modify\(Xml, Action<XDocument>\)](#) and [Modify\(Lob, Action<Byte\[\]>\)](#) can be used like in example below:

```
invoice.Modify(
    z => z.DocState += dataScope.GetDictionaryValue<DocState>(x => x.Code == newStateCode),
    z => z.Source.Modify(doc => doc.Element("invoice").Attribute("state").Value = newStateCode))
.Submit();
```

⚠ Warning

When object is [serialized](#) large objects properties are serialized with null values by default! When actual value of large object is required it must to be requested separately. To fill object large properties before serialization can be used one of [LoadLob](#) or [LoadLobAsync](#) helper method overloads.

> [back to the Tips & Tricks contents](#)

JSON serialization

When used [Json.NET](#) XData [data objects](#) are serialized and deserialized automatically.

There are some serialization specifics of data objects:

- By default [Read only properties](#) and [Hidden properties](#) are serialized and deserialized as well as writable properties, but have a some restrictions when copied to object to apply into database (see *Apply* method description below).
- [Large objects](#)) are serialized to null value by default. They are designed for lazy data loading. To serialize data with actual values one of [LoadLob](#) helper method overloads can be used.
- By default [Links](#) are serialized as limited representation of [link source](#) object. Only properties listed as [LinkProperties](#) are included in serialized object. Additionally an "*linkPropertyName.Value*" JSON property is serialized with the value of [Link.Value](#) property.

Default serialization rules can be overridden with *ICollection<T>* and *T WithJsonSettings* extension methods.

JsonSettings<T> has followed methods to specify serializing options:

- [Hide](#) - allow to specify properties to skip on serialization
- [LoadLob](#) - allow to specify large object properties to load data before serialize (*LoadLob* method alternative)
- [LinkMode](#) - allow to specify serialization rules for multiple (or all) link properties:
 - Default - default rules will be applied (see above)
 - Value - only value will be serialized
 - Source - only source will be serialized
- [DeleteFlag](#) - allow to delete child entities in database using unmapped boolean property value in JSON serialized child object (see below). Used to deserialized (detached) objects only!

```
return dataScope.GetRepository<Some>().ToArray()
.WithJsonSettings(s => s.Hide(JsonHideFlags.Columns) //hide all columns
.Hide(x => x.SomeReadOnlyProperty) //hide SomeReadOnlyProperty
.LoadLob() //load all large data properties
.LinkMode(LinkMode.Default,
    x => SomeLinkProperty, x => AnotherLinkProperty) //set default rules to 2 properties
.LinkMode(LinkMode.Value)); //set other link properties serialize Value property only
```

⚠ Warning

Deserialized objects are detached from repository and had limited ways of using!

- Deserialized object can be used as a filter to find data object in database (for example to delete) with [Find](#) method:

```
var found = dataScope.GetRepository<Some>().Find(detachedObj);
if(found == null) throw new DbConcurrencyException();
return found.SetDeleted(true).Submit();
```

- Deserialized object can be used as a source to copy some data to new object with [Copy](#) method:

```
return dataScope.GetRepository<Some>().New().Copy(detachedObj).Submit();
```

- Deserialized object can be used as to apply changes to database with [Apply](#) method:

```
dataScope.GetRepository<Some>().Apply(detachedObj).Submit();
```

Apply is a combination of *Find* and *Copy* methods with followed algorithm:

- 1) if *detached object* has all keys assigned, corresponded object will be searched in database using keys and correlation token (if described in mapping) values (*)
- 2) if no data found *XDataConcurrencyException* will be trowed (**)
- 3) when some key properties are not assigned new entity has been created
- 4) *detached object's* data will be copied to found (or new) object (***)

(*) Using of correlation token value with *Find* method can be escaped with optional *ignoreConcurrency* parameter. Same result can be reached with [ApplyFlag.IgnoreConcurrency](#) flag has been passed in *Apply* method call.

(**) [ApplyFlag.AddWhenNotFound](#) flag can be passed in *Apply* method call to add new entity when not found in database.

(***) *Copy* and *Apply* has parameter *properties* to specify witch property values will be copied. [ApplyFlag.ExceptProperties](#) flag can be used to SKIP listed *properties* on copy. [ApplyFlag.CopyReadOnlyProperties](#) flag can be used to allow copy [Read only properties](#) and [Hidden properties](#) disabled by default.

You can define child repositories and associate them using the *property* parameter of the [GetChild](#) method with an unmatched property with the type of the child repository (for one-to-one relationships) or with a typed array of the child repository (for one-to-many relationships) . These properties will be populated during data collection, and the values will be automatically serialized to JSON and deserialized from JSON to a detached object. These values can be used to cascade changes to the database. Inserting and updating child objects is supported without further action. But for deleting entities, it is required that the developer define a delete flag - it the unmapped property used as a flag to mark the child entity as removed. To specify which property will be used as the delete flag for the property associated with the child repository, the [WithJsonSettings](#) method is used with the [DeleteFlag](#) combination applied to the detached object.

```
// set delete flag to child repository entities to unmapped boolean property
dataScope.GetRepository<Some>()
    .Do(x => x.Instance.GetChild<Child>(z => z.Childs),
        x => detachedObj.WithJsonSettings(
            z => z.DeleteFlag(y => y.Childs, y => y.Deleted)))
    .Apply(detachedObj).Submit();
```

See [Master-detail relations](#) to get more information about linking objects with Master-detail relations.

[> back to the Tips & Tricks contents](#)

Conditional filters

XData realization of LINQ "Where" method has provide ability to use runtime condition evaluation during predicate compilation into SQL statements to analyze runtime data and conditionally omit non valued data (0, empty or null) filters.

```
// string author, int rating, decimal? lowerPrice and decimal? higherPrice is an optional filter values
return dataScope.GetRepository<Book>()
    .Where(x => (string.IsNullOrEmpty(author) || x.Author == author)
        && (rating <= 0 || x.Rating >= rating)
        && (lowerPrice.HasValue || x.Price >= lowerPrice)
        && (higherPrice.HasValue || x.Price <= higherPrice))
    .ToArray();
```

When runtime condition is true and LINQ expression is *OrElse* whole expression is omitted while predicate compilation into SQL

statements.

[> back to the Tips & Tricks contents](#)

Asynchronous wrapped operations

⚠ Warning

Only .Net Core Standard 2.0 version supported asynchronous wrapped operations

To access data with XData inside async methods it's handy to use asynchronous wrappers declared inside [XData.Async extension methods](#) as illustrated in followed example:

```
using XData.Async;
...
IDataEngine _dataEngine;
...
internal async Task<DocType[]> TestArrayAsync(CancellationTokentoken cancellationToken)
{
    using (var dataScope = dataEngine.NewDataScope("TEST_POSTGRE_NPG", cancellationToken))
    {
        return await dataScope.GetRepository<DocType>().ToArrayAsync();
    }
}
```

Using asynchronously disposable objects requires an additional package [XData net standard 2.0 Asynchronous Disposable support package](#) and supporting of .Net Standard 2.1 specification

To get asynchronous disposable version of...

- *IDataScope* use *NewDataScopeAsync* or *GetDataScopeAsync* extension methods of *IDataEngine*.
- *ITransaction* use *BeginTransactionAsync* extension methods of *IDataScope*.

```
using XData.Async;
using XData.AsyncDisposable;
...
IDataEngine _dataEngine;
...
internal async Task<DocType[]> TestArrayAsync(CancellationTokentoken cancellationToken)
{
    await using var dataScope = _dataEngine.NewDataScopeAsync("TEST_POSTGRE_NPG", cancellationToken);
    return await dataScope.GetRepository<DocType>().ToArrayAsync();
}
```

[> back to the Tips & Tricks contents](#)

Using dictionaries

Dictionaries can be used to reduce data re-query count when rarely changed data of dictionary will linked multiple times through [Link<T,TSource>](#) data type properties over small amount of time (one business process). Dictionaries is cached as single mapped objects (not a data cahe as described in [Caching repository data](#) topic).

[> back to the Tips & Tricks contents](#)

Caching repository data

Some data is changed often, some is rarely, but some data will never changed until application has upgraded. To reduce re-query of this static data, XData has mechanism to cache requested data (not a mapped objects) in XData. Repository property [FixedQuery](#) can be assigned to true when caching is required. All LINQ queries will be translated to this cache without re-query data from database. When this property resets to false, the cache will erased and repository returns to common algorithms of database requests. Similar feature is used to access rarely changed [dictionary](#) data over multiple related operative data changes.

[> back to the Tips & Tricks contents](#)

Copy data

Data can be copied from one repository to another using [IRepository<T>.Add<TFrom>](#) method as illustrated in followed example:

```
// Query source data (statically, or dynamically) into sourceQuery variable
var rep = dataScope.GetRepository<Target>();
using(var tr = dataScope.BeginTransaction())
{
    rep.Add<Source>(sourceQuery,
        x => new Target{
            TargetSome = x.Some,
            TargetAnother = x.Another });
    tr.Commit();
}
```

[> back to the Tips & Tricks contents](#)

Using events

[Custom logic](#) call and [trigger logic](#) is executed on the middle-ware layer, deployed on the application server or not. But some reaction on data change events may be required on client side. XData has followed events to subscribe changes of data: [InitRepository<T> OnInit](#) - event acquired when repository has been initialized [InitObject<T> OnInitObject](#) - event acquired when object has been initialized by database or default values [InvalidateRepository<T> OnInvalidate](#) - repository data has been re-queried from database [InvalidateObject<T> OnInvalidateObject](#) - object has been changed and re-queried from database [CurrentObjectChanging<T> OnCurrentObjectChanging](#) - current object is changing [CurrentObjectChanged<T> OnCurrentObjectChanged](#) - current object has been changed

[> back to the Tips & Tricks contents](#)

Dynamic query

Reuse mapping in dynamic query

Already declared mappings can be reused in [dynamic query](#) construction with generic [GetStructure](#) method overload.

See the example below to understand how it works:

```

using (var dataScope = DataEngine.NewDataScope())
{
    var user = dataScope.GetRepository<User>().SingleOrDefault(x => x.UserId.ToString() == userId);
    if (user == null)
        throw new Exception("User not found!");

    // prepare enum values expression array to *Case* expression based on AppType enum
    var enumValues = typeof(AppType).GetEnumKeyValuePairs()
        .Select(i => i.Key.SetExpression(x => i.Value))
        .ToArray();

    // reuse *UserRole* mapping adding filter by *user_id* column and redefine select output
    var roleClaims = XDataMapping.GetStructure<UserRole>().AddFilters("UR",
        x => x.ConstFilter("FilterByUserId", "user_id", userId.ToString()))
        .Column("UserId", x => x.Field<Guid>("UR", string.Empty))
        .Select(x => new UserClaim
        {
            // set negative id to preserve key uniqueness after applying *Union* operation using expression
            field
            Id = x.Expr("id", z => -1 * z.Field<long>("R", "role_id"), DbType.Int64),
            Type = x.Expr("role", z => "role", DbType.String),
            // construct role name based by *app_type* enum field and role name using SQL *Case* expression
            Value = x.Expr("val", z => z.Case(y => y.Field<long>("R", "app_type"), y => "", enumValues)
                + "." + x.Field<string>("R", "role"), DbType.String)
        }).AsQuery(dataScope);

    user.Applications = apps.SetFilterValue(x => UserId, userId).ToArray();
    user.Roles = dataScope.GetRepository<UserRole>().SetFilterValue(x => UserId, userId).ToArray();
    // Combine simple claims and synthetic role claims
    user.Claims = dataScope.GetRepository<UserClaim>()
        .SetFilterValue(x => UserId, userId)
        .Union(roleClaims).ToArray();

    return user;
}

```

[> back to the Tips & Tricks contents](#)

Dynamic query external data sources

XData [dynamic query](#) can use external data sources to operate with runtime tabular or xml data together with data stored inside database.

⚠ Warning

This feature is not supported by all of databases! XML data sources can be used with Ms SQL Server and PostgreSQL only. Local temporary tables is not supported by Oracle database. As You can see, Oracle not supported both of features. Use PL/SQL functions, packages or predefined global temporary tables instead.

XML data source

```
// prepare XML document
var doc = new XDocument(new XElement("root",
    new XElement("test", new XAttribute("doc_state_id", 1), new XAttribute("test", "one")),
    new XElement("test", new XAttribute("doc_state_id", 2), new XAttribute("test", "two"))
)).ToString();
var query = XDataMapping.GetStructure("S")
    .DataTable("T_DOC_STATE", "S")
    // use XmlSource definition providing XPath expression for data source
    .XmlSource("Test", "T", "root/test", x => x.Link("S", "doc_state_id"))
    .Select(x => new {
        // use XmlField definition with XPath expression as a field source
        DocStateId = x.XmlField<int>("T", string.Empty, "@doc_state_id", DbType.Int32, z => z.Key()),
        Code = x.Field<string>("S", string.Empty),
        // note, string and number (decimal) xml fields need size defined, like an expression fields
        Test = x.XmlField<string>("T", string.Empty, "@test", DbType.String, z => z.Size(20))
    });
// set XML string to variable with the same name as XmlSource
}).AsQuery(dataScope, "Test".SetVar(doc));
```

Temporary table

```
// prepare tabular data
var table = new DataTable();
table.Columns.AddRange(new [] { new DataColumn("doc_state_id", typeof(long)),
    new DataColumn("test", typeof(string)) });
table.Rows.Add(1, "one");
table.Rows.Add(2, "two");
table.AcceptChanges();

var query = XDataMapping.GetStructure("S")
    .DataTable("T_DOC_STATE", "S")
    // use temporary table definition
    .TempTable("Test", "T", x => x.Link("S", "doc_state_id"))
    .Select(x => new {
        // use Field method overload with dbType parameter to define temporary table field
        DocStateId = x.Field<long>("T", string.Empty, DbType.Int64, z => z.Key()),
        Code = x.Field<string>("S", string.Empty),
        // note, string and number (decimal) xml fields need size defined, like an expression fields
        Test = x.Field<string>("T", string.Empty, DbType.String, z => z.Size(20))
    });
// set DataTable to variable with the same name as TempTable
}).AsQuery(dataScope, "Test".SetVar(table));
```

[> back to the Tips & Tricks contents](#)

Data aware logic

Trigger business logic

❏ Important

It is important to understand the difference between database level triggers and these XData level triggers. A database trigger is called during changes made to the records of the table, whereas XData logic module processes the changes in the level of the data object. It must be remembered that a data object can include several database tables on the one hand and, on the other hand, a part of information regarding other objects can be stored in one table. And from this point of view, the value of such a mechanism inside the XData can hardly be overestimated!

Custom logic called over **data object** explicitly. But XData can call trigger logic over business valuable **data object** (not a single table!) on middle-ware level (no matter extracted this logic to dedicated application server or not). This logic will be called automatically when corresponded changes are applied:

- *InitRepository* - middle-ware trigger executed when Repository has been initialized (trigger delegate type `-InitRepository<T>`)
- *InvalidateRepository* - middle-ware trigger executed when Repository data need to be reseted (trigger delegate type

-[InvalidateRepository<T>](#))

- *InvalidateObject* - middle-ware trigger executed when Repository object need to be reloaded (trigger delegate type -[InvalidateObject<T>](#))
- *InitObject* - middle-ware trigger executed when Repository object has been initialized (trigger delegate type -[InitObject<T>](#))
- *BeforeInsert* - middle-ware trigger executed before object inserted into Repository (trigger delegate type -[Trigger<T>](#))
- *BeforeUpdate* - middle-ware trigger executed before Repository object has been updated (trigger delegate type -[Trigger<T>](#))
- *BeforeDelete* - middle-ware trigger executed before object has been deleted from Repository (trigger delegate type -[Trigger<T>](#))
- *AfterInsert* - middle-ware trigger executed after object inserted into Repository (trigger delegate type -[Trigger<T>](#)). [Skip](#) flag is not applicable!
- *AfterUpdate* - middle-ware trigger executed after Repository object has been updated (trigger delegate type -[Trigger<T>](#)). [Skip](#) flag is not applicable!
- *AfterDelete* - middle-ware trigger executed after object has been deleted from Repository (trigger delegate type -[Trigger<T>](#)). [Skip](#) flag is not applicable!
- *BeforeClear* - middle-ware trigger executed before Repository has been cleared (trigger delegate type -[RepositoryTrigger<T>](#))
- *AfterClear* - middle-ware trigger executed after Repository has been cleared (trigger delegate type -[RepositoryTrigger<T>](#)). [Skip](#) flag is not applicable!

Trigger delegate type - it's a method signature for middle-ware trigger handler.

```

public class InvoiceLogic : XDataLogic<Invoice>
{
    [Action(DataActionType.AfterInsert), Action(DataActionType.AfterUpdate)]
    public Trigger<Invoice> UpdateHistory =>
        ((ref Invoice invoice, ref DataTriggerFlag flag) =>
        {
            if (!invoice.CheckState(DataObjectState.New)
                && !invoice.IsChanged(x => x.DocState)) return true;
            var rep = invoice.GetRepository();
            using (var hist =
                GetRepository<DocHistory>(rep.Layer, context: rep.Context))
            {
                hist.Reset()
                    .SetFilterValue(DocHistory.FilterByDocId,
                        invoice.GetProperty<long>("DocId"))
                    .SetFilterValue(DocHistory.FilterByDocStateId,
                        invoice.DocState.Key);
                var newHist = hist.New();
                newHist.HistoryDate = DateTime.UtcNow;
                return hist.Submit(ref newHist);
            }
        });

    [Action(DataActionType.BeforeDelete)]
    public Trigger<Invoice> ClearHistory =>
        ((ref Invoice invoice, ref DataTriggerFlag flag) =>
        {
            var i = invoice;
            using (var hist = GetRepository<DocHistory>(i.GetLayer(),
                context: i.GetContext()))
            {
                return hist.Reset()
                    .Clear(x => x.GetProperty<long>("DocId")
                        == i.GetProperty<long>("DocId"));
            }
        });

    [Action(DataActionType.BeforeClear)]
    public RepositoryTrigger<Invoice> ClearHistoryBatch =>
        ((IRepository<Invoice> invoiceRepository, ref DataTriggerFlag flag) =>
        {
            using (var hist = GetRepository<DocHistory>(invoiceRepository.Layer,
                context: invoiceRepository.Context))
            {
                return hist.Reset().Clear(x => invoiceRepository
                    .Any(z => x.GetProperty<long>("DocId")
                        == z.GetProperty<long>("DocId")));
            }
        });
}

```

Trigger and RepositoryTrigger delegates has a reference parameter of type [DataTriggerFlag](#) to specify behavior of data processing after trigger executed. There are three possible behaviors defined:

- *None* - Submit using default algorithm, representation layer is already refreshed
- *Skip* - Data submitting have completed in trigger logic or not applicable. No standard updates will called. It's a kind of "instead of" trigger logic flag.
- *Refresh* - Default value. Representation layer manual data refreshing is needed.

Trigger logic defined as attributed by [Action attribute](#) read only property of [XDataLogic<T>](#) class descendant. One property can be attributed as a handler for multiple triggers. Property type must to be [Trigger<T>](#) except repository based triggers (*BeforeClear*, *AfterClear*, *InitRepository* or *InvalidateRepository*) - [RepositoryTrigger<T>](#).

[> back to the Tips & Tricks contents](#)

Execute custom business logic

XData has many features targeting this technology as not only ORM, but a full featured Data Access Layer. This is one of this features.

Custom logic defined as public property of `XDataLogic<T>` class descendant with type of `CustomLogic<T>`.

```
public class InvoiceLogic : XDataLogic<Invoice>
{
    public CustomLogic<Invoice> TestCustomLogic => (objects => {
        Log.Write(MessageType.Information,
            () => $"TestCustomLogic called with {objects.Length} objects");
        foreach (var invoice in objects)
        {
            var i = invoice;
            invoice.PostData("testPost",
                () => Encoding.UTF8.GetBytes(i.DocNumb));
            var p = Encoding.UTF8.GetBytes(i.DocNumb);
            var r = i.Callback("testCall", ref p);
            Log.Write(MessageType.Information, () =>
                @$"Call for \"{i.DocNumb}\" returned \"{r}\""
                with data \"{(p == null ? null : Encoding.UTF8.GetString(p))}\"");
        }
        return true;
    });
}

...

//Call example logic
var random = new Random();
return dataScope.GetRepository<Invoice>().ToArray().Execute(() => "TestCustomLogic",
    "testPost".SetValue((Action<byte[]>)(data =>
        Console.WriteLine("Post message received (data=\"{0}\")",
            data == null ? null : Encoding.UTF8.GetString(data))))
        .AsEnum().ToDictionary(),
    "testCall".SetValue((Func<byte[], byte[]>)(data => {
        Console.WriteLine("Call received \"{0}\")",
            data == null ? null : Encoding.UTF8.GetString(data));
        return random.NextDouble() >= 0.5 ? null : Encoding.UTF8.GetBytes(
            $"reply for \"{(data == null ? null : Encoding.UTF8.GetString(data))}\"");
    })).AsEnum().ToDictionary());
```

Programmer can develop data centric logic over some of mapped objects with ability to execute this logic in the middle ware layer of three-tier architecture on a dedicated server (or cluster). And this code can be simple and representable debugged in two-tier environment. Architecture can be switched by configuration! No code modification, no side effects... This feature is linked with [Trigger logic](#) and reached by [Callbacks](#).

When code is organized using data mapping modules, data logic modules and client (or Web site) it can help programmer to keep module relations clean and get ready to changes, with no matter of middle-ware layer existence. And this rules are not dictated, You can use it or not. You can write this logic anywhere You architect to do. Nothing will be breaks, except of ability to use XData three-tier architecture. But if You really don't need it, what is wrong?

[> back to the Tips & Tricks contents](#)

Using business logic callbacks

XData allows to use business logic callbacks to interact back-end modules with front-end using synchronous [Callback](#) and asynchronous [PostData](#) models. Synchronous calls can return result value. Callbacks can be used with [three-tier-architecture](#) as well as with client-server model. Callbacks has provided business logic clear, complete view and allows debug same code in two-

tier environment as code will be executed in three-tier.

```
public class InvoiceLogic : XDataLogic<Invoice>
{
    public CustomLogic<Invoice> TestCustomLogic => (objects => {
        Log.Write(MessageType.Information,
            () => $"TestCustomLogic called with {objects.Length} objects");
        foreach (var invoice in objects)
        {
            var i = invoice;
            invoice.PostData("testPost", () => Encoding.UTF8.GetBytes(i.DocNumb));
            var p = Encoding.UTF8.GetBytes(i.DocNumb);
            var r = i.Callback("testCall", ref p);
            Log.Write(MessageType.Information, () =>
                @$"Call for \"{i.DocNumb}\" returned \"{r}\" with data
                \"{(p == null ? null : Encoding.UTF8.GetString(p))}\"");
        }
        return true;
    });
}
```

[> back to the Tips & Tricks contents](#)

Business logic calculated fields

XData allowed to calculate fields not only using [SQL expressions](#). You can subscribe [OnInvalidateObject](#) event and calculate some fields on client side. More detailed events discussed [below](#).

[> back to the Tips & Tricks contents](#)

Testing

Virtual data and attached handlers

You can use XData as database mock engine. XData has feature to replace database interactions with attached handlers call. Virtual objects is attached to repository using [Attach](#) method of [IRepository<T>](#) interface. To detach virtual objects used [Detach](#) method. Multiple virtual entities can be attached to repository as new objects using [AttachNewObjects](#) method.

Tip

This method ([AttachNewObjects](#)) can be used to add multiple detached objects to real repository.

Every virtual entity can override handlers attached to repository with their own realization using parameters of [AttachNewObjects](#) method or [SetAttachedHandlers](#) method call. To set repository level handlers use special overload of [SetAttachedHandlers](#). To clear attached objects You can use [ClearAttachedObjects](#) method. To help wrap handlers to abstract [AttachedHandler<T>](#) class and unwrap it back to specialized handlers XData has a wrapper class [RepositoryAttachedHandler](#).

Attached handlers can have followed types (see [AttachedHandlerType](#)):

- [OnClear](#) - ([ClearAttachedHandler](#)) handler called instead of [Clear](#)
- [OnExecute](#) - ([ExecuteAttachedHandler<T>](#)) handler called instead of [Execute](#)
- [OnGetLob](#) - ([GetLobAttachedHandler](#)) handler called instead of requesting LOB data
- [OnGetLobSize](#) - ([GetLobSizeAttachedHandler](#)) handler called instead of requesting LOB size
- [OnGetXml](#) - ([GetXmlAttachedHandler](#)) handler called instead of requesting XML data
- [OnLock](#) - ([LockAttachedHandler<T>](#)) handler called instead of [Lock](#)
- [OnRefresh](#) - ([RefreshAttachedHandler<T>](#)) handler called instead of refreshing data
- [OnSetLob](#) - ([SetLobAttachedHandler](#)) handler called instead of saving LOB data
- [OnSetXml](#) - ([SetXmlAttachedHandler](#)) handler called instead of saving XML data
- [OnSubmit](#) - ([SubmitAttachedHandler<T>](#)) handler called instead of [Submit](#) multiple objects
- [OnSubmitObject](#) - ([SubmitObjectAttachedHandler<T>](#)) handler called instead of submitting single object

- *OnSubmitQuery* - ([SubmitQueryAttachedHandler<T>](#)) handler called instead of submitting query generation

[> back to the Tips & Tricks contents](#)

SQL

Transactions

XData use automatic transactions handling by default. But allows to manage transactions manually using *IDataScope* interface method [BeginTransaction](#) returned transaction state interface [ITransaction](#). Transaction state is a *IDisposable* object. Transaction is committed when method [Commit](#) is called. When transaction state is disposed without call [Commit](#) method, transaction is rolled back. Transaction can be started as read-only or read/write. This depends on parameter *read* of [BeginTransaction](#) method. XData can simulate wrap transactions one over another. First non read only transaction is related to real database transaction, and all wrapped transactions is related to transaction labels. When wrapped transaction is rolled back real transaction is rolled back to label only. Only rolling back the "main" transaction calls ROLLBACK statement over database.

[> back to the Tips & Tricks contents](#)

SQL hints

⚠ Warning

Before using SQL hints, be sure to use all other query optimization features. This is a "double-edged weapon." You need to understand very well why you are using what hints and how it will affect performance, and also make all the necessary tests confirming the effectiveness of the changes you made!

XData supports using SQL hints to improve SQL query productivity. All databases supports this feature using various mechanics and can change various parameters of SQL execution...

Types of hints supported by XData for RDBMS...

MS SQL Server

[Query hints](#) [Table hints](#)

Oracle

[Query hints](#)

MySQL

[Query hints](#)

PostgreSQL

PostgreSQL does not support hints for queries, but [procedures that returns set of record](#) required declare columns definition on call (see example below). Of course, this is not a SQL hint actually, but...

```
create or replace function GetRows(text) returns set of record as
$
declare
r record;
begin
for r in EXECUTE ''select * from '' || $1 loop
return next r;
end loop;
return;
end
$
language 'plpgsql';

select * from GetRows('Department') as dept(deptid int, deptname text);
```

SQLite

SQLite does not support hints.

Using hints in static mapping

Query hints:

```
// hint for empty string alias is applied to query
// possible to apply different hints on
// select, insert, update, delete statements or their combination
[DataObject("T"),
 Hint("", "SQL_NO_CACHE", HintType.Select),
 Hint("", "HIGH_PRIORITY", HintType.Select | HintType.Insert),
 ...
```

Table hints (MS SQL Server only):

```
[DataObject("T"),
 DataTable("Test", "T"),
 Hint("T", "nolog"),
 ...
```

Procedure hints (PostgreSQL only):

```
[DataObject("T"),
 Procedure("T", "TestFunction9", ProcedureType.Function),
 Hint("T", "Id integer, Name varchar(20)"),
 Parameter("T", 1, "p_id1", typeof(int), DbType.Int32),
 Parameter("T", 1, "p_name1", typeof(string), DbType.String),
 Parameter("T", 1, "p_id2", typeof(int), DbType.Int32),
 Parameter("T", 1, "p_name2", typeof(string), DbType.String),
 ResultSet("T")]
public class PostgreSqlTestFunction9 : IDataObject
{
    [Property("T", "Id", Flags = DataPropertyFlag.Id)]
    public int Id { get; set; }
    [Property("T", "Name")]
    public string Name { get; set; }
}
```

Using hints in dynamic mapping and dynamic queries

Query hints:

```
// hint for empty string alias is applied to query
// possible to apply different hints on
// select, insert, update, delete statements or their combination
XDataMapping.GetStructure("T")
    .Hint("", "SQL_NO_CACHE", HintType.Select)
    .Hint("", "HIGH_PRIORITY", HintType.Select | HintType.Insert)...
```

Table hints (MS SQL Server only):

```
XDataMapping.GetStructure("T")
    .DataTable("Test", "T")
    .Hint("T", "nolog")...
```

Procedure hints (PostgreSQL only):

```
XDataMapping.GetStructure()  
    .Procedure("T", "TestFunction8", ProcedureType.Function)  
    .Hint("T", "Id integer, Name varchar(20)")  
    .Parameter<Classifier[]>("T", 1, "p_tab", DbType.Object,  
        z => z.UdtDataType(null, "classifier_rec"), z => z.Array())  
    .Select(x => new Classifier {  
        Id = x.Field<int>("T", "Id", z => z.Key()),  
        Name = x.Field<string>("T", "Name") })  
    .AsQuery(DataScope, "p_tab".SetVar(param));
```

[> back to the Tips & Tricks contents](#)

Call custom SQL code

XData can help the programmer to obtain data and process it with many ways. But also we hold in mind XData is just a common tool with limited quantity of features, and some very specific (but handy and powerful) SQL dialect constructions may be out of XData possibilities.

To call dialect specific SQL statements used [IDataScope interface](#) with methods: [FillTable](#), [ExecuteScalar](#) and [ExecuteNonQuery](#).

```

const string getCoordComResourceStateChangesForCaseCommand = @"
select x.ResourceCode, x.ReportedDateTime, x.UserName, x.MissionStatusCode,
x.ResourceStatusCode
from (select h.ResourceCode, h.ReportedDateTime, h.MissionStatusCode,
h.ResourceStatusCode,
isnull((select top(1) log.Creator from cse_CaseFolderLog_tab log with (nolock)
where log.CallCenterId = h.CallCenterId
and log.CaseFolderId = h.CaseFolderId and log.CaseId = h.CaseId
and log.Created >= h.ReportedDateTime
and datediff(s, h.ReportedDateTime, log.Created) <= 10
and (charindex(case h.MissionStatusCode when '---' then 'ready'
else h.MissionStatusCode end,
log.LogText COLLATE Latin1_General_CS_AS) > 0
or charindex(case h.MissionStatusCode when '---' then 'disconnected'
else h.MissionStatusCode end,
log.LogText COLLATE Latin1_General_CS_AS) > 0)
and charindex(h.ResourceCode,
log.LogText COLLATE Latin1_General_CS_AS) > 0
and charindex('|' + log.Creator + '|', @IntegrationUsers) = 0
order by log.Created )
,
(select top(1) log.Creator from cse_CaseFolderLogFinished_tab log with (nolock)
where log.CallCenterId = h.CallCenterId
and log.CaseFolderId = h.CaseFolderId and log.CaseId = h.CaseId
and log.Created >= h.ReportedDateTime
and datediff(s,h.ReportedDateTime, log.Created) <= 10
and (charindex(case h.MissionStatusCode when '---' then 'ready'
else h.MissionStatusCode end,
log.LogText COLLATE Latin1_General_CS_AS) > 0
or charindex(case h.MissionStatusCode when '---' then 'disconnected'
else h.MissionStatusCode end,
log.LogText COLLATE Latin1_General_CS_AS) > 0)
and charindex(h.ResourceCode,
log.LogText COLLATE Latin1_General_CS_AS) > 0
and charindex('|' + log.Creator + '|', @IntegrationUsers) = 0
order by log.Created )) as UserName
from res_ResourceHistory_tab h
where h.CallCenterId = @CallCenterId and h.CaseFolderId = @CaseFolderId
and h.CaseId = @CaseId
and not exists(select 1 from sph_ResourceHistory_tab sh
where sh.CallCenterId = h.CallCenterId and sh.CaseFolderId = h.CaseFolderId
and sh.CaseId = h.CaseId
and sh.ResourceCode = h.ResourceCode
and (sh.MissionStatusCode = h.MissionStatusCode
or (sh.MissionStatusCode = '---'
and sh.ResourceStatusCode = @ClosedStatusCode))
and sh.ReportedDateTime = h.ReportedDateTime)) x
where x.UserName is not null";

using (var dataScope = DataEngine.Value.NewDataScope(CoordComDb))
{
return dataScope.FillTable(
new Query(getCoordComResourceStateChangesForCaseCommand, CommandType.Text,
new QueryParam("@CaseFolderId", caseIdentity.CaseFolderId, DbType.Int32),
new QueryParam("@CallCenterId", caseIdentity.CallCenterId, DbType.Int32),
new QueryParam("@CaseId", caseIdentity.CaseId, DbType.Int32),
new QueryParam("@IntegrationUsers", $"|{string.Join("|", integrationUsers)}|"),
new QueryParam("@ClosedStatusCode", closedStatusCode)
), CommandBehavior.SingleResult).Rows.OfType<DataRow>()
.Select(x => new Tuple<string, DateTime, string, string>(
x.Field<string>(0), x.Field<DateTime>(1), x.Field<string>(2),
x.Field<string>(3) == "---" ? x.Field<string>(4) : x.Field<string>(3))).ToList();
}

```

Use this mechanism with maximum care! This eliminates cross dialect approach of XData development, but we are in a real world with real programmers tasks... and we are never seen real tasks are correspond the rules of ideal program system architecture ;) Anyway this *bad* feature is a back-door we must provide while XData is not support all features of all specific SQL dialects. Is this really possible?. We think that not, but we will try to provide most of necessary features inside XData as soon as possible. We are need Your help to choose the order of features implementation and ideas to do it best!

❏ Important

DDL instructions is not implemented by design. We believe DDL in big complicated real business application is not a thing we can delegate to "dumb-metal-doll". This is why XData has no mechanics to "CodeFirst"... If You're task can use CodeFirst approach - use EF. This way is historically reserved by EF, and no reason to fight for it. Calling custom SQL code is a legal way to apply DDL in XData context. Programmer or database administrator write this SQL script. This code is produced by human, real (not artificial) intellect of professional... XData only has ability to run it over database.

[> back to the Tips & Tricks contents](#)

Useful little things...

During the development of XData, some helpers were developed that may be useful to You too...

Single value IEnumerable

When it is necessary to call the method that requires *IEnumerable<T>* and You have a single value, the [AsEnum](#) method will be useful.

Extensions for collections

To clear *List* with a preliminary call of some logic to free resources associated with a sheet element, the [Clear](#) method will help.

The [GetOrAdd](#) method for *List* will also be useful.

When you need to compare two *Dictionary* instances, the [HasSameValuesAs](#) method can help.

For *ConcurrentDictionary*, the [RemoveAll](#) method may come in handy.

Outer join

For those who are familiar with SQL, it is often convenient to use the concept of *OuterJoin* in the union of sets, and how it is implemented in LINQ is not comfortable to use. XData has a built-in mechanism for combining sets using methods:

- [FullOuterJoin](#)
- [LeftOuterJoin](#)
- [RightOuterJoin](#)

💡 Tip

It is necessary to take into account that these procedures use the Linq2Object interface *IEnumerable* and for the effective use of combining data in the database, you must use [mapping](#) or [dynamic queries](#).

Processing

To pre-process an object before using it (for example, to initialize the properties of a repository before using it as a LINQ data source), it is useful to use the [Do](#) extension:

```
var obj = dataScope.GetRepository<Invoice>()
    .Do(x => x.Instance.GetChild(z => z.InvoiceSpec))
    .First();
```

To access source object use property *Instance* of action argument and properties of dynamic property *State* of action argument to store some values between actions.

Appendix I. API Documentation

Namespace XData

Classes

Actions

Standard actions used by security

AttachedHandler<T>

Data object level attached handler wrapper

DataObjectExtensions

IDataObject extension functions

Log

Log helper

ParameterIndexer

SQL procedure call out parameters indexer

RepositoryAttachedHandler<T>

Repository level attached handler wrapper

RepositoryIndexer

Data object links (parents and childs) indexer

ResultSetIndexer

SQL procedure call out result sets indexer

RuntimeAssemblyLoader

SerializationExtensions

Serialization extensions

SerializedData

Serialized data

SerializedValue

Serialized data value

Variable

Data object repository variable

VariableIndexer

Data object variables indexer

Work

Work set helper class

XDataLogic<T>

XDataManager

Data manager Defines user session context across (possible) multiple data sources

[XDataPureLinqMethodAttribute](#)

Attribute to mark XData pure LINQ methods with no realization

Interfaces

[IDataEngine](#)

XData engine interface

[IDataLogic<T>](#)

Data logic interface

[IDataObject](#)

Data object interface

[IDataOptions](#)

XData options interface

[IDataQuery](#)

Read only data source basic interface

[IDataQuery<T>](#)

Specialized read only data source interface

[IDataScope](#)

XData operational scope

[IRepository](#)

Data repository (data object collection) basic interface

[IRepository<T>](#)

Data repository (data object collection) type specific interface

[ISqlObject](#)

SQL (sub)query part interface

[IWorkScope<T>](#)

Work set export/apply child object structure descriptor interface

[IWorkSet](#)

Basic work set interface

[IWorkSet<T>](#)

Work set interface

Enums

[ApplyFlag](#)

Apply detached object properties flags

[AttachedHandlerType](#)

Attached event types

Combination

Filter combination logical composition

CredentialsType

Database credentials type

DataActionType

Action type

DataCombination

Data set combination modes

DataExpressionType

Expression type

DataGrouping

Data aggregation function

DataObjectState

Data object instance state

DataPropertyFlag

Data object property flags

DataSourceType

Data source type

DataStructureFlag

Data object structure flags

DataSubmitFlag

Data submit options

DataTriggerFlag

Data trigger result flags

DataVersion

Version of data

DateDiffKind

Date difference kind

DefaultFeature

Default value features

DefaultType

Default value type

DialectOptions

SQL dialect options

[ExistsOperation](#)

Exists subquery operation

[FilterConstantType](#)

ConstantFilter value mode

[FilterNullable](#)

Filter null value assignment mode

[FilterOperation](#)

Filter operation

[FilterType](#)

Filter type

[MessageType](#)

Log message type

[ProcedureType](#)

SQL stored procedure type

[WithRecursiveType](#)

Common-Table-Expression (WITH) Subquery type

[WorkState](#)

Work set state

Delegates

[Calculate<T>](#)

XData LINQ-styled SQL expression delegate type

[ClearAttachedHandler](#)

Clear attached handler delegate

[CurrentObjectChanged<T>](#)

Object assigned as current event delegate type

[CurrentObjectChanging<T>](#)

Assigning object as current event delegate type

[CustomLogic<T>](#)

Custom server logic delegate type

[CustomMapping<T>](#)

Custom mapping delegate type

[ExecuteAttachedHandler<T>](#)

Execute attached handler delegate

[GetLobAttachedHandler](#)

Get Lob attached handler delegate

[GetLobSizeAttachedHandler](#)

Get Lob size attached handler delegate

[GetXmlAttachedHandler](#)

Get Xml attached handler delegate

[InitObject<T>](#)

Object instance initialization delegate type

[InitRepository<T>](#)

Repository initialization delegate type

[InvalidateObject<T>](#)

Refresh one object event delegate type

[InvalidateRepository<T>](#)

Refresh repository event delegate type

[LockAttachedHandler<T>](#)

Lock object attached handler delegate

[Mapping<TFrom, T>](#)

Mapping logic delegate type

[RefreshAttachedHandler<T>](#)

Refresh attached handler delegate

[RepositoryTrigger<T>](#)

RepositoryTrigger logic delegate type

[SetLobAttachedHandler](#)

Set Lob attached handler delegate

[SetXmlAttachedHandler](#)

Set Xml attached handler delegate

[SubmitAttachedHandler<T>](#)

Submit attached handler delegate

[SubmitObjectAttachedHandler<T>](#)

Submit object attached handler delegate

[SubmitQueryAttachedHandler<T>](#)

Lock object attached handler delegate

[Trigger<T>](#)

Trigger logic delegate type

Class Actions

Standard actions used by security

Inheritance

System.Object

Actions

Inherited Members

System.Object.Equals(System.Object)

System.Object.Equals(System.Object, System.Object)

System.Object.GetHashCode()

System.Object.GetType()

System.Object.MemberwiseClone()

System.Object.ReferenceEquals(System.Object, System.Object)

System.Object.ToString()

Namespace: [XData](#)

Assembly: XData.docfx.dll

Syntax

```
public static class Actions
```

Fields

All

All actions (CRUD and custom actions)

Declaration

```
public const string All = "`ALL`"
```

Field Value

TYPE	DESCRIPTION
System.String	

Create

Create action

Declaration

```
public const string Create = "`CREATE`"
```

Field Value

TYPE	DESCRIPTION
System.String	

CRUD

CRUD actions

Declaration

```
public const string CRUD = "`CRUD`"
```

Field Value

TYPE	DESCRIPTION
System.String	

Delete

Delete action

Declaration

```
public const string Delete = "`DELETE`"
```

Field Value

TYPE	DESCRIPTION
System.String	

Update

Update action

Declaration

```
public const string Update = "`UPDATE`"
```

Field Value

TYPE	DESCRIPTION
System.String	

Enum ApplyFlag

Apply detached object properties flags

Namespace: [XData](#)

Assembly: XData.docfx.dll

Syntax

```
[Flags]  
public enum ApplyFlag
```

Fields

NAME	DESCRIPTION
AddWhenNotFound	Add new entity when not found in DB
CopyReadOnlyProperties	Copy readonly properties
Default	Default apply values behavior Add a new entity to repository when keys has empty values Or try to find object in DB by key and concurrency token values and throw concurrency exception when not found Then copy all object properties to found or a new object, or properties restricted by optional properties parameter
ExceptProperties	Copy all properties EXCEPT listed in optional properties parameter
IgnoreConcurrency	Ignore concurrency token on find

Extension Methods

[SerializationExtensions.ToXml<ApplyFlag>\(\)](#)

[CollectionsExtensions.SetValue<ApplyFlag, T>\(T\)](#)

[CollectionsExtensions.AsEnum<ApplyFlag>\(\)](#)

[DataObjectQueryableExtensions.CompareVariable\(FilterOperation, String\)](#)

[Processing.Do<ApplyFlag>\(Action<IProcess<ApplyFlag>> >\[\]\)](#)

[TypeExtensions.ChangeType<T>\(\)](#)

[TypeExtensions.ChangeType\(Type\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Type\[\], Expression<Action>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Expression<Action>, BindingFlags\)](#)

[TypeExtensions.ConvertTo<T>\(\)](#)

[TypeExtensions.EqualValue\(Enum\)](#)

[SqlBlockExtensions.SetExpression<ApplyFlag, TResult>\(Expression<Func<IBlockQueryAdapter, TResult>> >\)](#)

[QueryDescriptionExtensions.SetExpression<ApplyFlag, TResult>\(Expression<Func<IQueryStructureAdapter, TResult>> >\)](#)

Class AttachedHandler<T>

Data object level attached handler wrapper

Inheritance

System.Object
AttachedHandler<T>
[RepositoryAttachedHandler<T>](#)

Inherited Members

System.Object.Equals(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.GetHashCode()
System.Object.GetType()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.ToString()

Namespace: [XData](#)
Assembly: XData.docfx.dll

Syntax

```
public class AttachedHandler<T>
    where T : class, IDataObject
```

Type Parameters

NAME	DESCRIPTION
T	

Constructors

AttachedHandler(AttachedHandlerType, Object)

DataObjectAttachedHandler constructor

Declaration

```
protected AttachedHandler(AttachedHandlerType type, object handler)
```

Parameters

TYPE	NAME	DESCRIPTION
AttachedHandlerType	type	
System.Object	handler	

Properties

Handler

Attached handler

Declaration

```
protected object Handler { get; }
```

Property Value

TYPE	DESCRIPTION
System.Object	

Type

Attached handler type

Declaration

```
public AttachedHandlerType Type { get; protected set; }
```

Property Value

TYPE	DESCRIPTION
AttachedHandlerType	

Operators

Explicit(AttachedHandler<T> to GetLobAttachedHandler)

Explicitly unwrap get Lob handler

Declaration

```
public static explicit operator GetLobAttachedHandler(AttachedHandler<T> handler)
```

Parameters

TYPE	NAME	DESCRIPTION
AttachedHandler<T>	handler	Wrapped handler

Returns

TYPE	DESCRIPTION
GetLobAttachedHandler	Unwrapped attached handler

Explicit(AttachedHandler<T> to GetLobSizeAttachedHandler)

Explicitly unwrap get Lob size handler

Declaration

```
public static explicit operator GetLobSizeAttachedHandler(AttachedHandler<T> handler)
```

Parameters

TYPE	NAME	DESCRIPTION
AttachedHandler<T>	handler	Wrapped handler

Returns

TYPE	DESCRIPTION
GetLobSizeAttachedHandler	Unwrapped attached handler

Explicit(AttachedHandler<T> to GetXmlAttachedHandler)

Explicitly unwrap get Xml handler

Declaration

```
public static explicit operator GetXmlAttachedHandler(AttachedHandler<T> handler)
```

Parameters

TYPE	NAME	DESCRIPTION
AttachedHandler<T>	handler	Wrapped handler

Returns

TYPE	DESCRIPTION
GetXmlAttachedHandler	Unwrapped attached handler

Explicit(AttachedHandler<T> to LockAttachedHandler<T>)

Explicitly unwrap lock object handler

Declaration

```
public static explicit operator LockAttachedHandler<T>(AttachedHandler<T> handler)
```

Parameters

TYPE	NAME	DESCRIPTION
AttachedHandler<T>	handler	Wrapped handler

Returns

TYPE	DESCRIPTION
LockAttachedHandler<T>	Unwrapped attached handler

Explicit(AttachedHandler<T> to RefreshAttachedHandler<T>)

Explicitly unwrap refresh object handler

Declaration

```
public static explicit operator RefreshAttachedHandler<T>(AttachedHandler<T> handler)
```

Parameters

TYPE	NAME	DESCRIPTION
AttachedHandler<T>	handler	Wrapped handler

Returns

TYPE	DESCRIPTION
RefreshAttachedHandler<T>	Unwrapped attached handler

Explicit(AttachedHandler<T> to SetLobAttachedHandler)

Explicitly unwrap set Lob handler

Declaration

```
public static explicit operator SetLobAttachedHandler(AttachedHandler<T> handler)
```

Parameters

TYPE	NAME	DESCRIPTION
AttachedHandler<T>	handler	Wrapped handler

Returns

TYPE	DESCRIPTION
SetLobAttachedHandler	Unwrapped attached handler

Explicit(AttachedHandler<T> to SetXmlAttachedHandler)

Explicitly unwrap set Xml handler

Declaration

```
public static explicit operator SetXmlAttachedHandler(AttachedHandler<T> handler)
```

Parameters

TYPE	NAME	DESCRIPTION
AttachedHandler<T>	handler	Wrapped handler

Returns

TYPE	DESCRIPTION
SetXmlAttachedHandler	Unwrapped attached handler

Explicit(AttachedHandler<T> to SubmitObjectAttachedHandler<T>)

Explicitly unwrap submit object handler

Declaration

```
public static explicit operator SubmitObjectAttachedHandler<T>(AttachedHandler<T> handler)
```

Parameters

TYPE	NAME	DESCRIPTION
AttachedHandler<T>	handler	Wrapped handler

Returns

TYPE	DESCRIPTION
SubmitObjectAttachedHandler<T>	Unwrapped attached handler

Explicit(AttachedHandler<T> to SubmitQueryAttachedHandler<T>)

Explicitly unwrap submit query handler

Declaration

```
public static explicit operator SubmitQueryAttachedHandler<T>(AttachedHandler<T> handler)
```

Parameters

TYPE	NAME	DESCRIPTION
AttachedHandler<T>	handler	Wrapped handler

Returns

TYPE	DESCRIPTION
SubmitQueryAttachedHandler<T>	Unwrapped attached handler

Implicit(GetLobAttachedHandler to AttachedHandler<T>)

Implicitly wrap get Lob handler

Declaration

```
public static implicit operator AttachedHandler<T>(GetLobAttachedHandler handler)
```

Parameters

TYPE	NAME	DESCRIPTION
GetLobAttachedHandler	handler	Get Lob handler

Returns

TYPE	DESCRIPTION
AttachedHandler<T>	Data object attached handler

Implicit(GetLobSizeAttachedHandler to AttachedHandler<T>)

Implicitly wrap get Lob size handler

Declaration

```
public static implicit operator AttachedHandler<T>(GetLobSizeAttachedHandler handler)
```

Parameters

TYPE	NAME	DESCRIPTION
GetLobSizeAttachedHandler	handler	Get Lob handler

Returns

TYPE	DESCRIPTION
AttachedHandler<T>	Data object attached handler

Implicit(GetXmlAttachedHandler to AttachedHandler<T>)

Implicitly wrap get Xml handler

Declaration

```
public static implicit operator AttachedHandler<T>(GetXmlAttachedHandler handler)
```

Parameters

TYPE	NAME	DESCRIPTION
GetXmlAttachedHandler	handler	Get Xml handler

Returns

TYPE	DESCRIPTION
AttachedHandler<T>	Data object attached handler

Implicit(LockAttachedHandler<T> to AttachedHandler<T>)

Implicitly wrap lock handler

Declaration

```
public static implicit operator AttachedHandler<T>(LockAttachedHandler<T> handler)
```

Parameters

TYPE	NAME	DESCRIPTION
LockAttachedHandler<T>	handler	Lock handler

Returns

TYPE	DESCRIPTION
AttachedHandler<T>	Data object attached handler

Implicit(RefreshAttachedHandler<T> to AttachedHandler<T>)

Implicitly wrap refresh object handler

Declaration

```
public static implicit operator AttachedHandler<T>(RefreshAttachedHandler<T> handler)
```

Parameters

TYPE	NAME	DESCRIPTION
RefreshAttachedHandler<T>	handler	Refresh handler

Returns

TYPE	DESCRIPTION
AttachedHandler<T>	Data object attached handler

Implicit(SetLobAttachedHandler to AttachedHandler<T>)

Implicitly wrap set Lob handler

Declaration

```
public static implicit operator AttachedHandler<T>(SetLobAttachedHandler handler)
```

Parameters

TYPE	NAME	DESCRIPTION
SetLobAttachedHandler	handler	Set Lob handler

Returns

TYPE	DESCRIPTION
AttachedHandler<T>	Data object attached handler

Implicit(SetXmlAttachedHandler to AttachedHandler<T>)

Implicitly wrap set Xml handler

Declaration

```
public static implicit operator AttachedHandler<T>(SetXmlAttachedHandler handler)
```

Parameters

TYPE	NAME	DESCRIPTION
SetXmlAttachedHandler	handler	Set Xml handler

Returns

TYPE	DESCRIPTION
AttachedHandler <T>	Data object attached handler

Implicit(SubmitObjectAttachedHandler<T> to AttachedHandler<T>)

Implicitly wrap submit object handler

Declaration

```
public static implicit operator AttachedHandler<T>(SubmitObjectAttachedHandler<T> handler)
```

Parameters

TYPE	NAME	DESCRIPTION
SubmitObjectAttachedHandler <T>	handler	Submit handler

Returns

TYPE	DESCRIPTION
AttachedHandler <T>	Data object attached handler

Implicit(SubmitQueryAttachedHandler<T> to AttachedHandler<T>)

Implicitly wrap submit query handler

Declaration

```
public static implicit operator AttachedHandler<T>(SubmitQueryAttachedHandler<T> handler)
```

Parameters

TYPE	NAME	DESCRIPTION
SubmitQueryAttachedHandler <T>	handler	Submit query handler

Returns

TYPE	DESCRIPTION
AttachedHandler<T>	Data object attached handler

Extension Methods

[SerializationExtensions.ToXml<T>\(T\)](#)

[CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)

[CollectionsExtensions.AsEnum<T>\(T\)](#)

[DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)

[Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)

[TypeExtensions.ChangeType<T>\(Object\)](#)

[TypeExtensions.ChangeType\(Object, Type\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)

[SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)

[QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Enum AttachedHandlerType

Attached event types

Namespace: [XData](#)

Assembly: XData.docfx.dll

Syntax

```
public enum AttachedHandlerType
```

Fields

NAME	DESCRIPTION
OnClear	OnClear
OnExecute	OnExecute
OnGetLob	OnGetLob
OnGetLobSize	OnGetLobSize
OnGetXml	OnGetXml
OnLock	OnLock
OnRefresh	OnRefresh
OnSetLob	OnSetLob
OnSetXml	OnSetXml
OnSubmit	OnSubmit
OnSubmitObject	OnSubmitObject
OnSubmitQuery	OnSubmitQuery

Extension Methods

[SerializationExtensions.ToXml<AttachedHandlerType>\(\)](#)

CollectionsExtensions.SetValue<AttachedHandlerType, T>(T)
CollectionsExtensions.AsEnum<AttachedHandlerType>()
DataObjectQueryableExtensions.CompareVariable(FilterOperation, String)
Processing.Do<AttachedHandlerType>(Action<IProcess<AttachedHandlerType>>[])
TypeExtensions.ChangeType<T>()
TypeExtensions.ChangeType(Type)
TypeExtensions.InvokeMethod<TRet>(Type[], Expression<Func<TRet>>, BindingFlags)
TypeExtensions.InvokeMethod(Type[], Expression<Action>, BindingFlags)
TypeExtensions.InvokeMethod<TRet>(Expression<Func<TRet>>, BindingFlags)
TypeExtensions.InvokeMethod(Expression<Action>, BindingFlags)
TypeExtensions.ConvertTo<T>()
TypeExtensions.EqualValue(Enum)
SqlBlockExtensions.SetExpression<AttachedHandlerType, TResult>(Expression<Func<IBlockQueryAdapter, TResult>>)
QueryDescriptionExtensions.SetExpression<AttachedHandlerType, TResult>(Expression<Func<IQueryStructureAdapter, TResult>>)

Delegate Calculate<T>

XData LINQ-styled SQL expression delegate type

Namespace: [XData](#)
Assembly: XData.docfx.dll

Syntax

```
public delegate T Calculate<out T>(IQueryStructureAdapter adapter);
```

Parameters

TYPE	NAME	DESCRIPTION
IQueryStructureAdapter	adapter	Query structure adapter

Returns

TYPE	DESCRIPTION
T	

Type Parameters

NAME	DESCRIPTION
T	Result type

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

See Also

- [SqlExpressionAttribute](#)
- [ExpressionFilterAttribute](#)
- [PropertyExpressionAttribute](#)

Delegate ClearAttachedHandler

Clear attached handler delegate

Namespace: [XData](#)
Assembly: XData.docfx.dll

Syntax

```
public delegate bool ClearAttachedHandler(DataSubmitFlag flag = DataSubmitFlag.None);
```

Parameters

TYPE	NAME	DESCRIPTION
DataSubmitFlag	flag	Submit flag

Returns

TYPE	DESCRIPTION
System.Boolean	

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Enum Combination

Filter combination logical composition

Namespace: [XData](#)
Assembly: XData.docfx.dll

Syntax

```
[DataContract]  
public enum Combination
```

Fields

NAME	DESCRIPTION
And	AND
Or	OR

See Also

- [Combination](#)
- [And](#)
- [Or](#)

Extension Methods

- [SerializationExtensions.ToXml<Combination>\(\)](#)
- [CollectionsExtensions.SetValue<Combination, T>\(T\)](#)
- [CollectionsExtensions.AsEnum<Combination>\(\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(FilterOperation, String\)](#)
- [Processing.Do<Combination>\(Action<IProcess<Combination>> > \[\]\)](#)
- [TypeExtensions.ChangeType<T>\(\)](#)
- [TypeExtensions.ChangeType\(Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.ConvertTo<T>\(\)](#)
- [TypeExtensions.EqualValue\(Enum\)](#)
- [SqlBlockExtensions.SetExpression<Combination, TResult>\(Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<Combination, TResult>\(Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Enum CredentialsType

Database credentials type

Namespace: [XData](#)

Assembly: [XData.docfx.dll](#)

Syntax

```
public enum CredentialsType
```

Fields

NAME	DESCRIPTION
DbIntegratedSecurity	Database integrated security
DbLogin	Database login
DbPassword	Database login password
DbSchema	Database schema name
DbServer	Database server name

Extension Methods

- [SerializationExtensions.ToXml<CredentialsType>\(\)](#)
- [CollectionsExtensions.SetValue<CredentialsType, T>\(T\)](#)
- [CollectionsExtensions.AsEnum<CredentialsType>\(\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(FilterOperation, String\)](#)
- [Processing.Do<CredentialsType>\(Action<IProcess<CredentialsType>> \[\]\)](#)
- [TypeExtensions.ChangeType<T>\(\)](#)
- [TypeExtensions.ChangeType\(Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.ConvertTo<T>\(\)](#)
- [TypeExtensions.EqualValue\(Enum\)](#)
- [SqlBlockExtensions.SetExpression<CredentialsType, TResult>\(Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<CredentialsType, TResult>\(Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Delegate CurrentObjectChanged<T>

Object assigned as current event delegate type

Namespace: [XData](#)
Assembly: XData.docfx.dll

Syntax

```
public delegate void CurrentObjectChanged<in T>(T obj)
    where T : class, IDataObject;
```

Parameters

TYPE	NAME	DESCRIPTION
T	obj	Changing is allowed

Type Parameters

NAME	DESCRIPTION
T	Object type

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

See Also

[OnCurrentObjectChanged](#)

Delegate CurrentObjectChanging<T>

Assigning object as current event delegate type

Namespace: [XData](#)
Assembly: XData.docfx.dll

Syntax

```
public delegate bool CurrentObjectChanging<in T>(T obj)
    where T : class, IDataObject;
```

Parameters

TYPE	NAME	DESCRIPTION
T	obj	Changing is allowed

Returns

TYPE	DESCRIPTION
System.Boolean	

Type Parameters

NAME	DESCRIPTION
T	Object type

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

See Also

[OnCurrentObjectChanging](#)

Delegate CustomLogic<T>

Custom server logic delegate type

Namespace: [XData](#)
Assembly: XData.docfx.dll

Syntax

```
public delegate bool CustomLogic<in T>(params T[] objects)
    where T : class, IDataObject;
```

Parameters

TYPE	NAME	DESCRIPTION
T[]	objects	Object instances to process

Returns

TYPE	DESCRIPTION
System.Boolean	Success flag

Type Parameters

NAME	DESCRIPTION
T	Object type

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Delegate CustomMapping<T>

Custom mapping delegate type

Namespace: [XData](#)
Assembly: XData.docfx.dll

Syntax

```
public delegate IRepositoryDescription<T> CustomMapping<out T>()  
    where T : class, ISqlObject, new();
```

Returns

TYPE	DESCRIPTION
IRepositoryDescription<T>	Success flag

Type Parameters

NAME	DESCRIPTION
T	Object type

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Enum DataActionType

Action type

Namespace: [XData](#)
Assembly: XData.docfx.dll

Syntax

```
public enum DataActionType
```

Fields

NAME	DESCRIPTION
AfterClear	After clear trigger logic
AfterDelete	After delete trigger logic
AfterInsert	After insert trigger logic
AfterUpdate	After update trigger logic
BeforeClear	Before clear trigger logic
BeforeDelete	Before delete trigger logic
BeforeInsert	Before insert trigger logic
BeforeUpdate	Before update trigger logic
InitObject	Object instance initialization code
InitRepository	Repository initialization code
InvalidateObject	Refresh object event code
InvalidateRepository	Refresh repository event code

See Also
[ActionType](#)

InitRepository
InvalidateRepository
InvalidateObject
InitObject
BeforeInsert
BeforeUpdate
BeforeDelete
AfterInsert
AfterUpdate
AfterDelete
BeforeClear
AfterClear

Extension Methods

SerializationExtensions.ToXml<DataActionType>()
CollectionsExtensions.SetValue<DataActionType, T>(T)
CollectionsExtensions.AsEnum<DataActionType>()
DataObjectQueryableExtensions.CompareVariable(FilterOperation, String)
Processing.Do<DataActionType>(Action<IProcess<DataActionType>> >())
TypeExtensions.ChangeType<T>()
TypeExtensions.ChangeType(Type)
TypeExtensions.InvokeMethod<TRet>(Type[], Expression<Func<TRet>>, BindingFlags)
TypeExtensions.InvokeMethod(Type[], Expression<Action>, BindingFlags)
TypeExtensions.InvokeMethod<TRet>(Expression<Func<TRet>>, BindingFlags)
TypeExtensions.InvokeMethod(Expression<Action>, BindingFlags)
TypeExtensions.ConvertTo<T>()
TypeExtensions.EqualValue(Enum)
SqlBlockExtensions.SetExpression<DataActionType, TResult>(Expression<Func<IBlockQueryAdapter, TResult>> >)
QueryDescriptionExtensions.SetExpression<DataActionType, TResult>(Expression<Func<IQueryStructureAdapter, TResult>> >)

Enum DataCombination

Data set combination modes

Namespace: [XData](#)

Assembly: XData.docfx.dll

Syntax

```
[DataContract]
public enum DataCombination
```

Fields

NAME	DESCRIPTION
Except	EXCEPT(MINUS)
Intersect	INTERSECT
None	Data object not use combination
Union	UNION
UnionAll	UNION ALL

See Also

- [None](#)
- [Union](#)
- [UnionAll](#)
- [Except](#)
- [Intersect](#)

Extension Methods

- [SerializationExtensions.ToXml<DataCombination>\(\)](#)
- [CollectionsExtensions.SetValue<DataCombination, T>\(T\)](#)
- [CollectionsExtensions.AsEnum<DataCombination>\(\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(FilterOperation, String\)](#)
- [Processing.Do<DataCombination>\(Action<IProcess<DataCombination>>\[\]\)](#)
- [TypeExtensions.ChangeType<T>\(\)](#)
- [TypeExtensions.ChangeType\(Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.ConvertTo<T>\(\)](#)
- [TypeExtensions.EqualValue\(Enum\)](#)
- [SqlBlockExtensions.SetExpression<DataCombination, TResult>\(Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<DataCombination, TResult>\(Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Enum DataExpressionType

Expression type

Namespace: [XData](#)
Assembly: XData.docfx.dll

Syntax

```
[DataContract]  
public enum DataExpressionType
```

Fields

NAME	DESCRIPTION
LinqExpression	LINQ-style defined expression
PlainSql	Plain SQL expression
SubQuery	Subquery expression

See Also

- [ExprType](#)
- [ExprType](#)
- [PlainSql](#)
- [SubQuery](#)

Extension Methods

- [SerializationExtensions.ToXml<DataExpressionType>\(\)](#)
- [CollectionsExtensions.SetValue<DataExpressionType, T>\(T\)](#)
- [CollectionsExtensions.AsEnum<DataExpressionType>\(\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(FilterOperation, String\)](#)
- [Processing.Do<DataExpressionType>\(Action<IProcess<DataExpressionType>> >\[\]\)](#)
- [TypeExtensions.ChangeType<T>\(\)](#)
- [TypeExtensions.ChangeType\(Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.ConvertTo<T>\(\)](#)
- [TypeExtensions.EqualValue\(Enum\)](#)
- [SqlBlockExtensions.SetExpression<DataExpressionType, TResult>\(Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<DataExpressionType, TResult>\(Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Enum DataGrouping

Data aggregation function

Namespace: [XData](#)

Assembly: XData.docfx.dll

Syntax

```
[DataContract]
public enum DataGrouping
```

Fields

NAME	DESCRIPTION
Avg	Avg()
Count	Count()
Max	Max()
Min	Min()
None	No aggregate function applied
Sum	Sum()

See Also

- [Grouping](#)
- [Grouping](#)
- [Grouping](#)
- [None](#)
- [Count](#)
- [Sum](#)
- [Max](#)
- [Min](#)
- [Avg](#)

Extension Methods

- [SerializationExtensions.ToXml<DataGrouping>\(\)](#)
- [CollectionsExtensions.SetValue<DataGrouping, T>\(T\)](#)
- [CollectionsExtensions.AsEnum<DataGrouping>\(\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(FilterOperation, String\)](#)
- [Processing.Do<DataGrouping>\(Action<IProcess<DataGrouping>>\(\)\)](#)
- [TypeExtensions.ChangeType<T>\(\)](#)
- [TypeExtensions.ChangeType\(Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)

TypeExtensions.InvokeMethod(Type[], Expression<Action>, BindingFlags)
TypeExtensions.InvokeMethod<TRet>(Expression<Func<TRet>>, BindingFlags)
TypeExtensions.InvokeMethod(Expression<Action>, BindingFlags)
TypeExtensions.ConvertTo<T>()
TypeExtensions.EqualValue(Enum)
SqlBlockExtensions.SetExpression<DataGrouping, TResult>(Expression<Func<IBlockQueryAdapter, TResult>>)
QueryDescriptionExtensions.SetExpression<DataGrouping, TResult>(Expression<Func<IQueryStructureAdapter, TResult>>)

Class DataObjectExtensions

IDataObject extension functions

Inheritance

System.Object

DataObjectExtensions

Inherited Members

System.Object.Equals(System.Object)

System.Object.Equals(System.Object, System.Object)

System.Object.GetHashCode()

System.Object.GetType()

System.Object.MemberwiseClone()

System.Object.ReferenceEquals(System.Object, System.Object)

System.Object.ToString()

Namespace: [XData](#)

Assembly: XData.docfx.dll

Syntax

```
public static class DataObjectExtensions
```

Fields

SubquerySources

Data sources types interpretable as sub queries

Declaration

```
public static readonly DataSourceType[] SubquerySources
```

Field Value

TYPE	DESCRIPTION
DataSourceType[]	

Methods

Apply<T>(IRepository<T>, T, String[])

Find object and apply changes from detached object

Declaration

```
public static T Apply<T>(this IRepository<T> repository, T source, params string[] properties)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
IRepository<T>	repository	Repository
T	source	Source detached object

TYPE	NAME	DESCRIPTION
System.String[]	properties	Properties to copy

Returns

TYPE	DESCRIPTION
T	Changed target object

Type Parameters

NAME	DESCRIPTION
T	Object type

Apply<T>(IRepository<T>, T, ApplyFlag, String[])

Find object and apply changes from detached object

Declaration

```
public static T Apply<T>(this IRepository<T> repository, T source, ApplyFlag flags, params string[] properties)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
IRepository<T>	repository	Repository
T	source	Source detached object
ApplyFlag	flags	Apply behaviour flags
System.String[]	properties	Properties to copy

Returns

TYPE	DESCRIPTION
T	Changed target object

Type Parameters

NAME	DESCRIPTION
T	Object type

AssignProperty<T, TValue>(T, String, TValue)

Assign property value

Declaration

```
public static bool AssignProperty<T, TValue>(this T obj, string prop, TValue value)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
T	obj	Data object instance
System.String	prop	Property name
TValue	value	Property value

Returns

TYPE	DESCRIPTION
System.Boolean	Property value changed

Type Parameters

NAME	DESCRIPTION
T	Data object type
TValue	Data property type

Exceptions

TYPE	CONDITION
XDataRuntimeException	Property name can not be null or empty

AsString<T>(IQueryable<T>)

Repository select query text

Declaration

```
public static string AsString<T>(this IQueryable<T> queryable)
    where T : class, IDataObject
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	queryable	Queryable source of data object

Returns

TYPE	DESCRIPTION
System.String	Repository

Type Parameters

NAME	DESCRIPTION
T	Data object type

Remarks

WARNING! High resource utilization! Use it for debug only!

Callback<T>(T, String, ref Byte[])

Callback to client

Declaration

```
public static bool Callback<T>(this T obj, string message, ref byte[] data)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
T	obj	Object context
System.String	message	Message name
Byte[]	data	Data to post

Returns

TYPE	DESCRIPTION
System.Boolean	

Type Parameters

NAME	DESCRIPTION
T	Data object type

CheckState<T>(T, DataObjectState)

Checks object for state

Declaration

```
public static bool CheckState<T>(this T obj, DataObjectState state)
    where T : class, IDataObject
```

Parameters

TYPE	NAME	DESCRIPTION
T	obj	Data object instance
DataObjectState	state	State

Returns

TYPE	DESCRIPTION
System.Boolean	Object has desired state flag

Type Parameters

NAME	DESCRIPTION
T	Data object type

Exceptions

TYPE	CONDITION
XDataRuntimeExpection	No state attached to object

See Also

- [GetState<T>\(T\)](#)
- [SetDeleted<T>\(T, Boolean\)](#)

Copy<T>(T, T, String[])

Copy detached object info

Declaration

```
public static T Copy<T>(this T target, T source, params string[] properties)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
T	target	Target object attached to repository

TYPE	NAME	DESCRIPTION
T	source	Source detached object
System.String[]	properties	Properties to copy

Returns

TYPE	DESCRIPTION
T	Changed target object

Type Parameters

NAME	DESCRIPTION
T	Object type

Copy<T>(T, T, ApplyFlag, String[])

Copy detached object info

Declaration

```
public static T Copy<T>(this T target, T source, ApplyFlag flags, params string[] properties)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
T	target	Target object attached to repository
T	source	Source detached object
ApplyFlag	flags	Apply behaviour flags
System.String[]	properties	Properties to copy

Returns

TYPE	DESCRIPTION
T	Changed target object

Type Parameters

NAME	DESCRIPTION
T	Object type

Execute<T>(T, Expression<Func<CustomLogic<T>>>, IDictionary<String, Action<Byte[]>>, IDictionary<String, Func<Byte[], Byte[]>>)

Run custom logic over single object

Declaration

```
public static bool Execute<T>(this T obj, Expression<Func<CustomLogic<T>>> routine, IDictionary<string,
Action<byte[]>> post = null, IDictionary<string, Func<byte[], byte[]>> callback = null)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
T	obj	Data object instance
System.Linq.Expressions.Expression<System.Func<CustomLogic<T>>>	routine	Action field name expression Example: () => SomeCustomActionField
System.Collections.Generic.IDictionary<System.String, System.Action<Byte[]>>	post	Post data handler
System.Collections.Generic.IDictionary<System.String, System.Func<Byte[], Byte[]>>	callback	Callback handler

Returns

TYPE	DESCRIPTION
System.Boolean	Success flag

Type Parameters

NAME	DESCRIPTION
T	Data object type

Remarks

Hint: When using XData application server custom logic will executed on server side

Exceptions

TYPE	CONDITION
XDataRuntimeException	Action field expression is null

TYPE	CONDITION
XDataRuntimeException	CustomActionAttribute not found for expression specified field

See Also

[Execute<T>\(ICollection<T>, Expression<Func<CustomLogic<T>>>, IDictionary<String, Action<Byte\[\]>>>, IDictionary<String, Func<Byte\[\], Byte\[\]>>>\)](#)

[Execute<T>\(T, String, IDictionary<String, Action<Byte\[\]>>>, IDictionary<String, Func<Byte\[\], Byte\[\]>>>\)](#)

Run custom logic over single object

Declaration

```
public static bool Execute<T>(this T obj, string key, IDictionary<string, Action<byte[]>> post = null,
IDictionary<string, Func<byte[], byte[]>> callback = null)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
T	obj	Data object instance
System.String	key	Action key
System.Collections.Generic.IDictionary<System.String, System.Action< Byte[] >>	post	Post data handler
System.Collections.Generic.IDictionary<System.String, System.Func< Byte[] , Byte[] >>	callback	Callback handler

Returns

TYPE	DESCRIPTION
System.Boolean	Success flag

Type Parameters

NAME	DESCRIPTION
T	Data object type

Remarks

Hint: When using XData application server custom logic will executed on server side

Exceptions

TYPE	CONDITION

TYPE	CONDITION
XDataRuntimeException	Object repository is detached

See Also

[Execute<T>\(T, Expression<Func<CustomLogic<T>>>, IDictionary<String, Action<Byte\[\]>>, IDictionary<String, Func<Byte\[\], Byte\[\]>>>\)](#)

[Execute<T>\(ICollection<T>, Expression<Func<CustomLogic<T>>>, IDictionary<String, Action<Byte\[\]>>, IDictionary<String, Func<Byte\[\], Byte\[\]>>>\)](#)

Run custom logic over array of objects

Declaration

```
public static bool Execute<T>(this ICollection<T> objects, Expression<Func<CustomLogic<T>>> routine,
    IDictionary<string, Action<byte[]>> post = null, IDictionary<string, Func<byte[], byte[]>> callback = null)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
System.Collections.Generic.ICollection<T>	objects	Collection of objects
System.Linq.Expressions.Expression<System.Func< CustomLogic <T>>>	routine	Action field name expression Example: () => SomeCustomActionField
System.Collections.Generic.IDictionary<System.String, System.Action< Byte []>>	post	Post data handler
System.Collections.Generic.IDictionary<System.String, System.Func< Byte [], Byte []>>	callback	Callback handler

Returns

TYPE	DESCRIPTION
System.Boolean	Success flag

Type Parameters

NAME	DESCRIPTION
T	Data object type

Remarks

Hint: When using XData application server custom logic will executed on server side

Exceptions

TYPE	CONDITION
XDataRuntimeException	Action field name expression is null
XDataRuntimeException	CustomActionAttribute not found for expression specified field

See Also

[Execute<T>\(T, Expression<Func<CustomLogic<T>>>, IDictionary<String, Action<Byte\[\]>>, IDictionary<String, Func<Byte\[\], Byte\[\]>>>\)](#)

[Execute<T>\(ICollection<T>, String, IDictionary<String, Action<Byte\[\]>>, IDictionary<String, Func<Byte\[\], Byte\[\]>>\)](#)

Run custom logic over array of objects

Declaration

```
public static bool Execute<T>(this ICollection<T> objects, string key, IDictionary<string, Action<byte[]>> post = null, IDictionary<string, Func<byte[], byte[]>> callback = null)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
System.Collections.Generic.ICollection<T>	objects	Collection of objects
System.String	key	Action key
System.Collections.Generic.IDictionary<System.String, System.Action< Byte[] >>	post	Post data handler
System.Collections.Generic.IDictionary<System.String, System.Func< Byte[] , Byte[] >>	callback	Callback handler

Returns

TYPE	DESCRIPTION
System.Boolean	Success flag

Type Parameters

NAME	DESCRIPTION
T	Data object type

Remarks

Hint: When using XData application server custom logic will executed on server side

Exceptions

TYPE	CONDITION
XDataRuntimeException	Object repository is detached

See Also

[Execute<T>\(ICollection<T>, Expression<Func<CustomLogic<T>>>, IDictionary<String, Action<Byte\[\]>>>, IDictionary<String, Func<Byte\[\], Byte\[\]>>>\)](#)

Find<T>(IRepository<T>, T, Boolean)

Find detached object in repository

Declaration

```
public static T Find<T>(this IRepository<T> repository, T source, bool ignoreConcurrency = false)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
IRepository<T>	repository	Repository
T	source	Detached object
System.Boolean	ignoreConcurrency	Find ignore concurrency token value (optional) default = false

Returns

TYPE	DESCRIPTION
T	Found object

Type Parameters

NAME	DESCRIPTION
T	Object type

FindMemberExpression(Expression)

Property name expression parse helper

Declaration

```
public static MemberExpression FindMemberExpression(Expression e)
```

Parameters

TYPE	NAME	DESCRIPTION

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression	e	Property name expression: x => x.SomePropertyName

Returns

TYPE	DESCRIPTION
System.Linq.Expressions.MemberExpression	Property name

Exceptions

TYPE	CONDITION
XDataRuntimeException	Property name expression can not be null
XDataRuntimeException	Expression has a wrong expression type

GetContext(IDataObject)

Returns context name for object

Declaration

```
public static string GetContext(this IDataObject obj)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataObject	obj	Data object

Returns

TYPE	DESCRIPTION
System.String	Context name

Exceptions

TYPE	CONDITION
XDataRuntimeException	

GetLayer(IDataObject)

Returns repository layer for object

Declaration

```
public static Guid GetLayer(this IDataObject obj)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataObject	obj	Data object

Returns

TYPE	DESCRIPTION
System.Guid	Repository layer

Exceptions

TYPE	CONDITION
XDataRuntimeException	

GetParameter<T>(IDataQuery, String)

Returns SQL procedure/function out parameter value

Declaration

<pre>public static T GetParameter<T>(this IDataQuery query, string param)</pre>

Parameters

TYPE	NAME	DESCRIPTION
IDataQuery	query	Data query
System.String	param	Parameter name

Returns

TYPE	DESCRIPTION
T	Parameter value

Type Parameters

NAME	DESCRIPTION
T	

Exceptions

TYPE	CONDITION
XDataRuntimeException	Parameter name can not be null or empty

See Also

[GetParameter<T, TRet>\(IDataQuery<T>, Expression<Func<T, TRet>>\)](#)

GetParameter<T, TRet>(T, Expression<Func<T, TRet>>)

Returns SQL procedure/function out parameter value

Declaration

```
public static TRet GetParameter<T, TRet>(this T obj, Expression<Func<T, TRet>> param)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
T	obj	Data object instance
System.Linq.Expressions.Expression<System.Func<T, TRet>>	param	Parameter name expression using same name property (case insensitive) Example: x => x.SomePropertyName

Returns

TYPE	DESCRIPTION
TRet	Parameter value

Type Parameters

NAME	DESCRIPTION
T	Data object type
TRet	Parameter type

Exceptions

TYPE	CONDITION
XDataRuntimeExpection	Parameter name expression can not be null

GetParameter<T, TRet>(T, String)

Returns SQL procedure/function out parameter value

Declaration

```
public static TRet GetParameter<T, TRet>(this T obj, string param)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
T	obj	Data object instance

TYPE	NAME	DESCRIPTION
System.String	param	Parameter name

Returns

TYPE	DESCRIPTION
TRet	Parameter value

Type Parameters

NAME	DESCRIPTION
T	Data object type
TRet	Parameter type

Exceptions

TYPE	CONDITION
XDataRuntimeExpection	Parameter name can not be null or empty

See Also

[GetParameter<T, TRet>\(T, Expression<Func<T, TRet>>\)](#)

GetParameter<T, TRet>(IDataQuery<T>, Expression<Func<T, TRet>>)

Returns SQL procedure/function out parameter value

Declaration

```
public static TRet GetParameter<T, TRet>(this IDataQuery<T> query, Expression<Func<T, TRet>> param)
    where T : class, new()
```

Parameters

TYPE	NAME	DESCRIPTION
IDataQuery<T>	query	Data query
System.Linq.Expressions.Expression<System.Func<T, TRet>>	param	Parameter name expression using same name property (case insensitive) Example: x => x.SomePropertyName

Returns

TYPE	DESCRIPTION

TYPE	DESCRIPTION
TRet	Parameter value

Type Parameters

NAME	DESCRIPTION
T	Data object type
TRet	Parameter type

Exceptions

TYPE	CONDITION
XDataRuntimeException	Parameter name expression can not be null

GetProperties<T>(T, IEnumerable<String>, DataVersion)

Returns properties values

Declaration

```
public static IEnumerable<KeyValuePair<string, object>> GetProperties<T>(this T obj, IEnumerable<string> props,
DataVersion version = DataVersion.Actual)
    where T : class, IDataObject
```

Parameters

TYPE	NAME	DESCRIPTION
T	obj	Data object instance
System.Collections.Generic.IEnumerable<System.String>	props	Properties sequence
DataVersion	version	Data version

Returns

TYPE	DESCRIPTION
System.Collections.Generic.IEnumerable<System.Collections.Generic.KeyValuePair<System.String, System.Object>>	Sequence of name value pairs

Type Parameters

NAME	DESCRIPTION

NAME	DESCRIPTION
T	Data object type

Remarks

Hint: To get a single property value use [GetProperty<T, TRet>\(T, Expression<Func<T, TRet>>, DataVersion\)](#) function.

Exceptions

TYPE	CONDITION
XDataRuntimeException	Properties sequence is null

GetProperties<T>(T, String[])

Returns properties values

Hint: To get a single property value use [GetProperty](#) function.

Declaration

```
public static IEnumerable<KeyValuePair<string, object>> GetProperties<T>(this T obj, params string[] props)
    where T : class, IDataObject
```

Parameters

TYPE	NAME	DESCRIPTION
T	obj	Data object instance
System.String[]	props	Properties array

Returns

TYPE	DESCRIPTION
System.Collections.Generic.IEnumerable<System.Collections.Generic.KeyValuePair<System.String, System.Object>>	Sequence of name value pairs

Type Parameters

NAME	DESCRIPTION
T	Data object type

Remarks

Hint: To get a single property value use [GetProperty<T, TRet>\(T, Expression<Func<T, TRet>>, DataVersion\)](#) function.

GetProperties<T>(T, DataVersion, String[])

Returns properties values

Declaration

```
public static IEnumerable<KeyValuePair<string, object>> GetProperties<T>(this T obj, DataVersion version =
DataVersion.Actual, params string[] props)
    where T : class, IDataObject
```

Parameters

TYPE	NAME	DESCRIPTION
T	obj	Data object instance
DataVersion	version	Data version
System.String[]	props	Properties array

Returns

TYPE	DESCRIPTION
System.Collections.Generic.IEnumerable<System.Collections.Generic.KeyValuePair<System.String, System.Object>>	Sequence of name value pairs

Type Parameters

NAME	DESCRIPTION
T	Data object type

Remarks

Hint: To get a single property value use [GetProperty<T, TRet>\(T, Expression<Func<T, TRet>>, DataVersion\)](#) function.

GetProperty<TRet>(IDataObject, String)

Returns single property value

Declaration

```
public static TRet GetProperty<TRet>(this IDataObject obj, string prop)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataObject	obj	Data object instance
System.String	prop	Property name

Returns

TYPE	DESCRIPTION

TYPE	DESCRIPTION
TRet	Property value

Type Parameters

NAME	DESCRIPTION
TRet	Property type

Exceptions

TYPE	CONDITION
XDataRuntimeException	Property name can not be null or empty

See Also

[GetProperty<T, TRet>\(T, Expression<Func<T, TRet>>, DataVersion\)](#)

GetProperty<T, TRet>(T, Expression<Func<T, TRet>>)

Returns property value

Declaration

```
public static TRet GetProperty<T, TRet>(this T obj, Expression<Func<T, TRet>> prop)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
T	obj	Data object instance
System.Linq.Expressions.Expression<System.Func<T, TRet>>	prop	Property name expression: x => x.SomePropertyName

Returns

TYPE	DESCRIPTION
TRet	Property value

Type Parameters

NAME	DESCRIPTION
T	Data object type
TRet	Property type

Exceptions

TYPE	CONDITION
XDataRuntimeException	Property name expression can not be null

GetProperty<T, TRet>(T, Expression<Func<T, TRet>>, DataVersion)

Returns property value

Declaration

<pre>public static TRet GetProperty<T, TRet>(this T obj, Expression<Func<T, TRet>> prop, DataVersion version) where T : class, IDataObject, new()</pre>

Parameters

TYPE	NAME	DESCRIPTION
T	obj	Data object instance
System.Linq.Expressions.Expression<System.Func<T, TRet>>	prop	Property name expression: x => x.SomePropertyName
DataVersion	version	Data version

Returns

TYPE	DESCRIPTION
TRet	Property value

Type Parameters

NAME	DESCRIPTION
T	Data object type
TRet	Property type

Exceptions

TYPE	CONDITION
XDataRuntimeException	Property name expression can not be null

GetRepository<T>(T)

Returns object repository

Declaration

```
public static IRepository<T> GetRepository<T>(this T obj)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
T	obj	Data object instance

Returns

TYPE	DESCRIPTION
IRepository<T>	Object repository

Type Parameters

NAME	DESCRIPTION
T	Data object type

Exceptions

TYPE	CONDITION
XDataRuntimeException	Object repository is detached

GetRepositoryAlias(IDataObject)

Returns object's repository alias

Declaration

```
public static string GetRepositoryAlias(this IDataObject obj)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataObject	obj	Data object

Returns

TYPE	DESCRIPTION
System.String	Repository alias

Exceptions

TYPE	CONDITION
XDataRuntimeException	

GetResultSet<TRes>(IDataQuery, Expression<Func<IEnumerable<TRes>>>)

Returns SQL procedure/function result set

Declaration

```
public static IEnumerable<TRes> GetResultSet<TRes>(this IDataQuery query, Expression<Func<IEnumerable<TRes>>>
resultSetNames)
    where TRes : class
```

Parameters

TYPE	NAME	DESCRIPTION
IDataQuery	query	Data query
System.Linq.Expressions.Expression<System.Func<System.Collections.Generic.IEnumerable<TRes>>>>	resultSetNames	Result set name expression using same name property (case insensitive) Example: x => x.SomePropertyName

Returns

TYPE	DESCRIPTION
System.Collections.Generic.IEnumerable<TRes>	Result set object sequence

Type Parameters

NAME	DESCRIPTION
TRes	Result set object type

Exceptions

TYPE	CONDITION
XDataRuntimeException	Result set name expression can not be null

See Also

[GetResultSet<TRes>\(IDataQuery, String\)](#)

GetResultSet<TRes>(IDataQuery, String)

Returns SQL procedure/function result set

Declaration

```
public static IEnumerable<TRes> GetResultSet<TRes>(this IDataQuery query, string resultSetNames)
    where TRes : class
```

Parameters

TYPE	NAME	DESCRIPTION
IDataQuery	query	Data query
System.String	resultSetNames	Result set name name

Returns

TYPE	DESCRIPTION
System.Collections.Generic.IEnumerable<TRes>	Parameter value

Type Parameters

NAME	DESCRIPTION
TRes	Result set object type

Exceptions

TYPE	CONDITION
XDataRuntimeException	Result set name can not be null or empty

See Also

[GetResultSet<TRes>\(IDataQuery, Expression<Func<IEnumerable<TRes>>>\)](#)

[GetResultSet<T, TRes>\(T, Expression<Func<T, IEnumerable<TRes>>>\)](#)

Returns SQL procedure/function result set

Declaration

```
public static IEnumerable<TRes> GetResultSet<T, TRes>(this T obj, Expression<Func<T, IEnumerable<TRes>>>
resultSetNames)
    where T : class, IDataObject, new()
    where TRes : class
```

Parameters

TYPE	NAME	DESCRIPTION
T	obj	Data object instance
System.Linq.Expressions.Expression<System.Func<T, System.Collections.Generic.IEnumerable<TRes>>>>	resultSetNames	Result set name expression using same name property (case insensitive) Example: x => x.SomePropertyName

Returns

TYPE	DESCRIPTION
System.Collections.Generic.IEnumerable<TRes>	Result set object sequence

Type Parameters

NAME	DESCRIPTION
T	Data object type
TRes	Result set object type

Exceptions

TYPE	CONDITION
XDataRuntimeException	Result set name expression can not be null

GetResultSet<T, TRes>(IRepository<T>, Expression<Func<T, IEnumerable<TRes>>>)

Returns SQL procedure/function result set

Declaration

```
public static IEnumerable<TRes> GetResultSet<T, TRes>(this IRepository<T> repository, Expression<Func<T,
IEnumerable<TRes>>> resultSetName)
    where T : class, IDataObject, new()
    where TRes : class
```

Parameters

TYPE	NAME	DESCRIPTION
IRepository<T>	repository	Data object repository
System.Linq.Expressions.Expression<System.Func<T, System.Collections.Generic.IEnumerable<TRes>>>	resultSetNames	Result set name expression using same name property (case insensitive) Example: x => x.SomePropertyName

Returns

TYPE	DESCRIPTION
System.Collections.Generic.IEnumerable<TRes>	Result set object sequence

Type Parameters

NAME	DESCRIPTION
T	Object type

NAME	DESCRIPTION
TRes	Result set object type

Exceptions

TYPE	CONDITION
XDataRuntimeException	Result set name expression can not be null

See Also

[GetResultSet<TRes>\(IDataQuery, String\)](#)

GetState<T>(T)

Returns state of object

Declaration

```
public static DataObjectState GetState<T>(this T obj)
    where T : class, IDataObject
```

Parameters

TYPE	NAME	DESCRIPTION
T	obj	Data object instance

Returns

TYPE	DESCRIPTION
DataObjectState	Object state flags combination(see DataObjectState)

Type Parameters

NAME	DESCRIPTION
T	Data object type

Exceptions

TYPE	CONDITION
XDataRuntimeException	No state attached to object

See Also

[CheckState<T>\(T, DataObjectState\)](#)

[SetDeleted<T>\(T, Boolean\)](#)

IsAttached(IDataObject, Type)

Check data object is attached

Declaration

```
public static bool IsAttached(this IDataObject obj, Type type)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataObject	obj	Data object instance
System.Type	type	Data object type

Returns

TYPE	DESCRIPTION
System.Boolean	Object is attached

Exceptions

TYPE	CONDITION
XDataRuntimeException	Object is detached

IsChanged<T>(T, Expression<Func<T, Object>>[])

Test property value is assigned

Declaration

```
public static bool IsChanged<T>(this T obj, params Expression<Func<T, object>>[] prop)
    where T : class, IDataObject
```

Parameters

TYPE	NAME	DESCRIPTION
T	obj	Data object instance
System.Linq.Expressions.Expression<System.Func<T, System.Object>>[]	prop	Property name expression Example: x => x.SomePropertyName

Returns

TYPE	DESCRIPTION
System.Boolean	Property is changed

Type Parameters

NAME	DESCRIPTION
T	Data object type

Exceptions

TYPE	CONDITION
XDataRuntimeException	Property name expression can not be null

IsChanged<T>(T, String)

Test property value is assigned

Declaration

```
public static bool IsChanged<T>(this T obj, string prop)
    where T : class, IDataObject
```

Parameters

TYPE	NAME	DESCRIPTION
T	obj	Data object instance
System.String	prop	Property name

Returns

TYPE	DESCRIPTION
System.Boolean	Property is changed

Type Parameters

NAME	DESCRIPTION
T	Data object type

Exceptions

TYPE	CONDITION
XDataRuntimeException	Property name can not be null or empty

IsCleared<T>(T, String)

Test property value is cleared

Declaration

```
public static bool IsCleared<T>(this T obj, string prop)
    where T : class, IDataObject
```

Parameters

TYPE	NAME	DESCRIPTION
T	obj	Data object instance
System.String	prop	Property name

Returns

TYPE	DESCRIPTION
System.Boolean	Property is cleared

Type Parameters

NAME	DESCRIPTION
T	Data object type

Exceptions

TYPE	CONDITION
XDataRuntimeException	Property name can not be null or empty

IsEmpty<T>(T, Expression<Func<T, Object>>)

Test property value is empty

Declaration

```
public static bool IsEmpty<T>(this T obj, Expression<Func<T, object>> prop)
    where T : class, IDataObject
```

Parameters

TYPE	NAME	DESCRIPTION
T	obj	Data object instance
System.Linq.Expressions.Expression<System.Func<T, System.Object>>	prop	Property name expression Example: x => x.SomePropertyName

Returns

TYPE	DESCRIPTION

TYPE	DESCRIPTION
System.Boolean	Property is empty

Type Parameters

NAME	DESCRIPTION
T	Data object type

Exceptions

TYPE	CONDITION
XDataRuntimeException	Property name expression can not be null

IsEmpty<T>(T, String)

Test property value is empty

Declaration

```
public static bool IsEmpty<T>(this T obj, string prop)
    where T : class, IDataObject
```

Parameters

TYPE	NAME	DESCRIPTION
T	obj	Data object instance
System.String	prop	Property name

Returns

TYPE	DESCRIPTION
System.Boolean	Property is empty

Type Parameters

NAME	DESCRIPTION
T	Data object type

Exceptions

TYPE	CONDITION
XDataRuntimeException	Property name can not be null or empty

LoadLob<T>(T)

Load Lob and Xml field values of data object

Declaration

```
public static T LoadLob<T>(this T obj)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
T	obj	Object

Returns

TYPE	DESCRIPTION
T	Enriched object

Type Parameters

NAME	DESCRIPTION
T	Object type

LoadLob<T>(T, Expression<Func<T, Object>>[])

Load Lob and Xml field values of data object

Declaration

```
public static T LoadLob<T>(this T obj, params Expression<Func<T, object>>[] properties)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
T	obj	Object
System.Linq.Expressions.Expression<System.Func<T, System.Object>>[]	properties	Properties expressions

Returns

TYPE	DESCRIPTION
T	Enriched object

Type Parameters

NAME	DESCRIPTION
T	Object type

LoadLob<T>(T, String[])

Load Lob and Xml field values of data object

Declaration

```
public static T LoadLob<T>(this T obj, params string[] properties)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
T	obj	Object
System.String[]	properties	Properties extensions

Returns

TYPE	DESCRIPTION
T	Enriched object

Type Parameters

NAME	DESCRIPTION
T	Object type

Lock<T>(T)

Lock object in database

Declaration

```
public static bool Lock<T>(this T obj)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
T	obj	Data object instance

Returns

TYPE	DESCRIPTION

TYPE	DESCRIPTION
System.Boolean	Lock success

Type Parameters

NAME	DESCRIPTION
T	Data object type

See Also

[Lock\(T\)](#)

Modify(Lob, Action<Byte[]>)

Lob field modify

Declaration

```
public static Lob Modify(this Lob source, Action<byte[]> action)
```

Parameters

TYPE	NAME	DESCRIPTION
Lob	source	Source Lob
System.Action< Byte []>	action	Modify action

Returns

TYPE	DESCRIPTION
Lob	Modified Lob

Exceptions

TYPE	CONDITION
XDataRuntimeException	Modify logic error

See Also

[Lob](#)

[Modify\(Xml, Action<XDocument>\)](#)

Modify(Xml, Action<XDocument>)

Xml field modify

Declaration

```
public static Xml Modify(this Xml source, Action<XDocument> action)
```

Parameters

TYPE	NAME	DESCRIPTION
Xml	source	Source document
System.Action<System.Xml.Linq.XDocument>	action	Modify action

Returns

TYPE	DESCRIPTION
Xml	Modified xml

Examples

```
invoice.Source.Modify(doc => doc.Element("invoice").Attribute("state").Value = "ACTIVE");
```

Exceptions

TYPE	CONDITION
XDataRuntimeExpection	Modify logic error

See Also

[Xml](#)
[Modify\(Lob, Action<Byte\[\]>\)](#)

Modify<T>(T, Action<T>[])

Data object modification wrapper

Declaration

```
public static T Modify<T>(this T obj, params Action<T>[] actions)
    where T : class, IDataObject
```

Parameters

TYPE	NAME	DESCRIPTION
T	obj	Data object
System.Action<T>[]	actions	Modification actions

Returns

TYPE	DESCRIPTION
T	Modified data object

Type Parameters

NAME	DESCRIPTION
T	Data object type

PostData<T>(T, String, Func<Byte[]>)

Post data to client

Declaration

```
public static void PostData<T>(this T obj, string message, Func<byte[]> data)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
T	obj	Object context
System.String	message	Message name
System.Func<Byte[]>	data	Data to post

Type Parameters

NAME	DESCRIPTION
T	Data object type

Refresh<T>(T)

Refresh single data object from database

Declaration

```
public static bool Refresh<T>(this T obj)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
T	obj	Data object instance

Returns

TYPE	DESCRIPTION
System.Boolean	Success flag

Type Parameters

NAME	DESCRIPTION
T	Data object type

Refresh<T>(IRepository<T>)

Refresh repository data

Declaration

```
public static IRepository<T> Refresh<T>(this IRepository<T> source)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
IRepository<T>	source	Repository

Returns

TYPE	DESCRIPTION
IRepository<T>	Repository

Type Parameters

NAME	DESCRIPTION
T	Data object type

SetAttachedHandlers<T>(T, AttachedHandler<T>[])

Attach extra handlers to object instance

Declaration

```
public static void SetAttachedHandlers<T>(this T obj, params AttachedHandler<T>[] handlers)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
T	obj	Data object instance
AttachedHandler<T>[]	handlers	Handlers

Type Parameters

NAME	DESCRIPTION

NAME	DESCRIPTION
T	Data object type

SetCurrent<T>(T)

Set object as repository current

Declaration

```
public static void SetCurrent<T>(this T obj)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
T	obj	Data object instance

Type Parameters

NAME	DESCRIPTION
T	Data object type

See Also

[Current](#)

SetDeleted<T>(T, Boolean)

Set deleted state to object

Declaration

```
public static T SetDeleted<T>(this T obj, bool setState)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
T	obj	Data object instance
System.Boolean	setState	Set or remove DataObjectState.Deleted state flag

Returns

TYPE	DESCRIPTION
T	Object marked as deleted/restored

Type Parameters

NAME	DESCRIPTION
T	Data object type

Exceptions

TYPE	CONDITION
XDataRuntimeException	No state attached to object

See Also

[CheckState<T>\(T, DataObjectState\)](#)

[GetState<T>\(T\)](#)

[SetDeleted<T>\(ICollection<T>, Boolean\)](#)

Set deleted state to each object in collection

Declaration

```
public static ICollection<T> SetDeleted<T>(this ICollection<T> collection, bool setState)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
System.Collections.Generic.ICollection<T>	collection	Data object collection
System.Boolean	setState	Set or remove DataObjectState.Deleted state flag

Returns

TYPE	DESCRIPTION
System.Collections.Generic.ICollection<T>	Collection of objects marked as deleted/restored

Type Parameters

NAME	DESCRIPTION
T	Data object type

Exceptions

TYPE	CONDITION
XDataRuntimeException	No state attached to object

See Also

[CheckState<T>\(T, DataObjectState\)](#)

GetState<T>(T)

SetFilterValue<T, TRes>(IRepository<T>, Expression<Func<T, TRes>>, TRes)

Set value to dynamically created runtime filter

Declaration

```
public static IRepository<T> SetFilterValue<T, TRes>(this IRepository<T> repository, Expression<Func<T, TRes>>
propertyExpression, TRes filterValue)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
IRepository<T>	repository	Repository
System.Linq.Expressions.Expression<System.Func<T, TRes>>	propertyExpression	Property name expression
TRes	filterValue	Filter value

Returns

TYPE	DESCRIPTION
IRepository<T>	Repository

Type Parameters

NAME	DESCRIPTION
T	Repository data object type
TRes	Property type

SetVar<T>(String, T)

Set variable value

Declaration

```
public static Variable SetVar<T>(this string name, T value)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Variable name
T	value	Variable value

Returns

TYPE	DESCRIPTION
Variable	Variable

Type Parameters

NAME	DESCRIPTION
T	Value type

Submit<T>(T, out T, DataSubmitFlag)

Submit data object changes

Declaration

```
public static bool Submit<T>(this T obj, out T res, DataSubmitFlag flag = DataSubmitFlag.None)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
T	obj	Data object instance
T	res	Submitted object
DataSubmitFlag	flag	Submit flag

Returns

TYPE	DESCRIPTION
System.Boolean	Submit success

Type Parameters

NAME	DESCRIPTION
T	Data object type

See Also

[Submit\(ref T, DataSubmitFlag\)](#)

Submit<T>(T, Func<JsonSettings<T>, JsonSettings<T>>, out T, DataSubmitFlag)

Submit data object changes

Declaration

```
public static bool Submit<T>(this T obj, Func<JsonSettings<T>, JsonSettings<T>> settings, out T res,
DataSubmitFlag flag = DataSubmitFlag.None)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
T	obj	Data object instance
System.Func< JsonSettings<T> , JsonSettings<T> >	settings	JSON serialization settings
T	res	Submitted object
DataSubmitFlag	flag	Submit flag

Returns

TYPE	DESCRIPTION
System.Boolean	Submit success

Type Parameters

NAME	DESCRIPTION
T	Data object type

See Also

[Submit\(ref T, DataSubmitFlag\)](#)

[Submit<T>\(T, DataSubmitFlag\)](#)

[Submit data object changes](#)

Declaration

```
public static bool Submit<T>(this T obj, DataSubmitFlag flag = DataSubmitFlag.None)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
T	obj	Data object instance
DataSubmitFlag	flag	Submit flag

Returns

TYPE	DESCRIPTION
System.Boolean	Submit success

Type Parameters

NAME	DESCRIPTION
T	Data object type

See Also

[Submit\(ref T, DataSubmitFlag\)](#)

ToArray<T>(IEnumerable<T>, Func<JsonSettings<T>, JsonSettings<T>>)

Returns array of data objects with JSON serialization settings applied

Declaration

```
public static T[] ToArray<T>(this IEnumerable<T> sourceEnumerable, Func<JsonSettings<T>, JsonSettings<T>> settings)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
System.Collections.Generic.IEnumerable<T>	sourceEnumerable	Source object sequence
System.Func< JsonSettings <T>, JsonSettings <T>>	settings	JSON serialization settings

Returns

TYPE	DESCRIPTION
T[]	Array of data objects with JSON serialization settings applied

Type Parameters

NAME	DESCRIPTION
T	Object type

WithJsonSettings<T>(T, Func<JsonSettings<T>, JsonSettings<T>>)

Apply custom JSON serialization settings to single object

Declaration

```
public static T WithJsonSettings<T>(this T obj, Func<JsonSettings<T>, JsonSettings<T>> settings)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
T	obj	Object to apply serialization settings
System.Func<JsonSettings<T>, JsonSettings<T>>	settings	Serialization settings fill routine

Returns

TYPE	DESCRIPTION
T	Object with custom serialization settings

Type Parameters

NAME	DESCRIPTION
T	ObjectType

WithJsonSettings<T>(ICollection<T>, Func<JsonSettings<T>, JsonSettings<T>>)

Apply custom JSON serialization settings to collection

Declaration

```
public static ICollection<T> WithJsonSettings<T>(this ICollection<T> collection, Func<JsonSettings<T>,
JsonSettings<T>> settings)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
System.Collections.Generic.ICollection<T>	collection	Collection to apply serialization settings
System.Func<JsonSettings<T>, JsonSettings<T>>	settings	Serialization settings fill routine

Returns

TYPE	DESCRIPTION
System.Collections.Generic.ICollection<T>	Collection with custom serialization settings

Type Parameters

NAME	DESCRIPTION
T	ObjectType

Enum DataObjectState

Data object instance state

Namespace: [XData](#)
Assembly: XData.docfx.dll

Syntax

```
[Flags]  
[DataContract]  
public enum DataObjectState
```

Fields

NAME	DESCRIPTION
Changed	Original data values are changed
Deleted	Instance is marked for delete
Ghost	Instance is deleted
New	New instance flag
Original	Original data values has no changes

See Also

- [GetState<T>\(T\)](#)
- [CheckState<T>\(T, DataObjectState\)](#)
- [SetDeleted<T>\(T, System.Boolean\)](#)
- [Original](#)
- [New](#)
- [Changed](#)
- [Deleted](#)
- [Ghost](#)

Extension Methods

- [SerializationExtensions.ToXml<DataObjectState>\(\)](#)
- [CollectionsExtensions.SetValue<DataObjectState, T>\(T\)](#)
- [CollectionsExtensions.AsEnum<DataObjectState>\(\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(FilterOperation, String\)](#)
- [Processing.Do<DataObjectState>\(Action<IProcess<DataObjectState>> > \[\]\)](#)
- [TypeExtensions.ChangeType<T>\(\)](#)
- [TypeExtensions.ChangeType\(Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Expression<Action>, BindingFlags\)](#)

TypeExtensions.ConvertTo<T>()

TypeExtensions.EqualValue(Enum)

SqlBlockExtensions.SetExpression<DataObjectState, TResult>(Expression<Func<IBlockQueryAdapter, TResult>>)

QueryDescriptionExtensions.SetExpression<DataObjectState, TResult>(Expression<Func<IQueryStructureAdapter, TResult>>)

Enum DataPropertyFlag

Data object property flags

Namespace: [XData](#)
Assembly: XData.docfx.dll

Syntax

```
[Flags]  
[DataContract]  
public enum DataPropertyFlag
```

Fields

NAME	DESCRIPTION
ConcurrencyToken	Data object concurrency token field flag
Id	Data object unique key part
None	Common data property
OuterFlag	Outer joined information filled flag

See Also

- [Flags](#)
- [Flags](#)
- [None](#)
- [Id](#)
- [OuterFlag](#)

Extension Methods

- [SerializationExtensions.ToXml<DataPropertyFlag>\(\)](#)
- [CollectionsExtensions.SetValue<DataPropertyFlag, T>\(T\)](#)
- [CollectionsExtensions.AsEnum<DataPropertyFlag>\(\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(FilterOperation, String\)](#)
- [Processing.Do<DataPropertyFlag>\(Action<IProcess<DataPropertyFlag>> > \[\]\)](#)
- [TypeExtensions.ChangeType<T>\(\)](#)
- [TypeExtensions.ChangeType\(Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.ConvertTo<T>\(\)](#)
- [TypeExtensions.EqualValue\(Enum\)](#)
- [SqlBlockExtensions.SetExpression<DataPropertyFlag, TResult>\(Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<DataPropertyFlag, TResult>\(Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Enum DataSourceType

Data source type

Namespace: [XData](#)
Assembly: XData.docfx.dll

Syntax

```
[DataContract]  
public enum DataSourceType
```

Fields

NAME	DESCRIPTION
InnerView	Inner view
Procedure	SQL Stored procedure or Function
Subquery	Subquery
Table	Table
TempTable	Temporary table
WithSubquery	Common-Table-Expression (WITH) Subquery
XmlSource	XML data source

See Also

- [Table](#)
- [Subquery](#)
- [InnerView](#)
- [Procedure](#)

Extension Methods

- [SerializationExtensions.ToXml<DataSourceType>\(\)](#)
- [CollectionsExtensions.SetValue<DataSourceType, T>\(T\)](#)
- [CollectionsExtensions.AsEnum<DataSourceType>\(\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(FilterOperation, String\)](#)
- [Processing.Do<DataSourceType>\(Action<IProcess<DataSourceType>> >\[\]\)](#)
- [TypeExtensions.ChangeType<T>\(\)](#)
- [TypeExtensions.ChangeType\(Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Expression<Func<TRet>>, BindingFlags\)](#)

TypeExtensions.InvokeMethod(Expression<Action>, BindingFlags)

TypeExtensions.ConvertTo<T>()

TypeExtensions.EqualValue(Enum)

SqlBlockExtensions.SetExpression<DataSourceType, TResult>(Expression<Func<IBlockQueryAdapter, TResult>>)

QueryDescriptionExtensions.SetExpression<DataSourceType, TResult>(Expression<Func<IQueryStructureAdapter, TResult>>)

Enum DataStructureFlag

Data object structure flags

Namespace: [XData](#)
Assembly: XData.docfx.dll

Syntax

```
[Flags]
public enum DataStructureFlag
```

Fields

NAME	DESCRIPTION
Grouping	Grouped Data object
None	Common editable Data object
ReadOnly	Read only Data object
Tree	Tree organized Data object

See Also

- [Flags](#)
- [None](#)
- [Tree](#)
- [Grouping](#)
- [ReadOnly](#)

Extension Methods

- [SerializationExtensions.ToXml<DataStructureFlag>\(\)](#)
- [CollectionsExtensions.SetValue<DataStructureFlag, T>\(T\)](#)
- [CollectionsExtensions.AsEnum<DataStructureFlag>\(\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(FilterOperation, String\)](#)
- [Processing.Do<DataStructureFlag>\(Action<IProcess<DataStructureFlag>> \[\]\)](#)
- [TypeExtensions.ChangeType<T>\(\)](#)
- [TypeExtensions.ChangeType\(Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.ConvertTo<T>\(\)](#)
- [TypeExtensions.EqualValue\(Enum\)](#)
- [SqlBlockExtensions.SetExpression<DataStructureFlag, TResult>\(Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<DataStructureFlag, TResult>\(Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Enum DataSubmitFlag

Data submit options

Namespace: [XData](#)
Assembly: XData.docfx.dll

Syntax

```
[Flags]  
[DataContract]  
public enum DataSubmitFlag
```

Fields

NAME	DESCRIPTION
IgnoreConcurrency	Ignore concurrency conflicts
None	Standard submit
Silent	Silent submit

See Also

- [Submit\(ref T, DataSubmitFlag\)](#)
- [Submit\(IEnumerable<T>, DataSubmitFlag\)](#)
- [None](#)
- [Silent](#)
- [IgnoreConcurrency](#)

Extension Methods

- [SerializationExtensions.ToXml<DataSubmitFlag>\(\)](#)
- [CollectionsExtensions.SetValue<DataSubmitFlag, T>\(T\)](#)
- [CollectionsExtensions.AsEnum<DataSubmitFlag>\(\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(FilterOperation, String\)](#)
- [Processing.Do<DataSubmitFlag>\(Action<IProcess<DataSubmitFlag>> \[\]\)](#)
- [TypeExtensions.ChangeType<T>\(\)](#)
- [TypeExtensions.ChangeType\(Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.ConvertTo<T>\(\)](#)
- [TypeExtensions.EqualValue\(Enum\)](#)
- [SqlBlockExtensions.SetExpression<DataSubmitFlag, TResult>\(Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<DataSubmitFlag, TResult>\(Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Enum DataTriggerFlag

Data trigger result flags

Namespace: [XData](#)
Assembly: XData.docfx.dll

Syntax

```
[Flags]
public enum DataTriggerFlag
```

Fields

NAME	DESCRIPTION
None	Standard trigger result
Refresh	Refresh representation layer flag
Skip	Skip standard submitting

See Also

- [Trigger<T>](#)
- [RepositoryTrigger<T>](#)
- [None](#)
- [Skip](#)
- [Refresh](#)

Extension Methods

- [SerializationExtensions.ToXml<DataTriggerFlag>\(\)](#)
- [CollectionsExtensions.SetValue<DataTriggerFlag, T>\(T\)](#)
- [CollectionsExtensions.AsEnum<DataTriggerFlag>\(\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(FilterOperation, String\)](#)
- [Processing.Do<DataTriggerFlag>\(Action<IProcess<DataTriggerFlag>> \[\]\)](#)
- [TypeExtensions.ChangeType<T>\(\)](#)
- [TypeExtensions.ChangeType\(Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.ConvertTo<T>\(\)](#)
- [TypeExtensions.EqualValue\(Enum\)](#)
- [SqlBlockExtensions.SetExpression<DataTriggerFlag, TResult>\(Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<DataTriggerFlag, TResult>\(Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Enum DataVersion

Version of data

Namespace: [XData](#)
Assembly: XData.docfx.dll

Syntax

```
public enum DataVersion
```

Fields

NAME	DESCRIPTION
Actual	Actual data version
Original	Original data version

See Also

- [GetProperties<T>\(T, DataVersion, System.String\[\]\)](#)
- [GetProperties<T>\(T, System.Collections.Generic.IEnumerable<System.String>, DataVersion\)](#)
- [GetProperty<T, TRet>\(T, System.Linq.Expressions.Expression<System.Func<T, TRet>>, DataVersion\)](#)
- [XData.DataObjectExtensions.GetProperty`1\(`0,System.String,XData.DataVersion\)](#)
- [Actual](#)
- [Original](#)

Extension Methods

- [SerializationExtensions.ToXml<DataVersion>\(\)](#)
- [CollectionsExtensions.SetValue<DataVersion, T>\(T\)](#)
- [CollectionsExtensions.AsEnum<DataVersion>\(\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(FilterOperation, String\)](#)
- [Processing.Do<DataVersion>\(Action<IProcess<DataVersion>>\[\]\)](#)
- [TypeExtensions.ChangeType<T>\(\)](#)
- [TypeExtensions.ChangeType\(Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.ConvertTo<T>\(\)](#)
- [TypeExtensions.EqualValue\(Enum\)](#)
- [SqlBlockExtensions.SetExpression<DataVersion, TResult>\(Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<DataVersion, TResult>\(Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Enum DateDiffKind

Date difference kind

Namespace: [XData](#)
Assembly: XData.docfx.dll

Syntax

```
public enum DateDiffKind
```

Fields

NAME	DESCRIPTION
Days	Date difference in month days
DaysTotal	Date difference in days
Hours	Date difference in hours
Minutes	Date difference in minutes
Months	Date difference in months
Seconds	Date difference in seconds
Years	Date difference in years

Extension Methods

```
SerializationExtensions.ToXml<DateDiffKind>()
CollectionsExtensions.SetValue<DateDiffKind, T>(T)
CollectionsExtensions.AsEnum<DateDiffKind>()
DataObjectQueryableExtensions.CompareVariable(FilterOperation, String)
Processing.Do<DateDiffKind>(Action<IProcess<DateDiffKind>>[])
TypeExtensions.ChangeType<T>()
TypeExtensions.ChangeType(Type)
TypeExtensions.InvokeMethod<TRet>(Type[], Expression<Func<TRet>>, BindingFlags)
TypeExtensions.InvokeMethod(Type[], Expression<Action>, BindingFlags)
TypeExtensions.InvokeMethod<TRet>(Expression<Func<TRet>>, BindingFlags)
TypeExtensions.InvokeMethod(Expression<Action>, BindingFlags)
TypeExtensions.ConvertTo<T>()
TypeExtensions.EqualValue(Enum)
SqlBlockExtensions.SetExpression<DateDiffKind, TResult>(Expression<Func<IBlockQueryAdapter, TResult>>)
QueryDescriptionExtensions.SetExpression<DateDiffKind, TResult>(Expression<Func<IQueryStructureAdapter, TResult>>)
```

Enum DefaultFeature

Default value features

Namespace: [XData](#)
Assembly: XData.docfx.dll

Syntax

```
[Flags]
public enum DefaultFeature
```

Fields

NAME	DESCRIPTION
SkipWhenAssigned	Skip insert into table when PK value assigned explicitly
UpdateWhenAssigned	Change insert operation over table to update when PK value assigned explicitly
UseOnInsert	Use default value on insert (by default, always applied)
UseOnUpdate	Use default value on update

Extension Methods

- [SerializationExtensions.ToXml<DefaultFeature>\(\)](#)
- [CollectionsExtensions.SetValue<DefaultFeature, T>\(T\)](#)
- [CollectionsExtensions.AsEnum<DefaultFeature>\(\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(FilterOperation, String\)](#)
- [Processing.Do<DefaultFeature>\(Action<IProcess<DefaultFeature>> > \[\]\)](#)
- [TypeExtensions.ChangeType<T>\(\)](#)
- [TypeExtensions.ChangeType\(Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.ConvertTo<T>\(\)](#)
- [TypeExtensions.EqualValue\(Enum\)](#)
- [SqlBlockExtensions.SetExpression<DefaultFeature, TResult>\(Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<DefaultFeature, TResult>\(Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Enum DefaultType

Default value type

Namespace: [XData](#)
Assembly: XData.docfx.dll

Syntax

```
[DataContract]  
public enum DefaultType
```

Fields

NAME	DESCRIPTION
AutoIncrement	Auto increment value as default
Const	Constant value as default
CurrentDate	Current date
CurrentDateTime	Current date and time
CurrentDateTimeUtc	Current date and time UTC
NewGuid	New GUID as default
None	No default value
UserName	Current user name
Variable	Variable value as default

See Also

- [DefaultSource](#)
- [DefaultSource](#)
- [None](#)
- [CurrentDate](#)
- [CurrentDateTime](#)
- [CurrentDateTimeUtc](#)
- [Variable](#)
- [Const](#)
- [AutoIncrement](#)

Extension Methods

SerializationExtensions.ToXml<DefaultType>()

CollectionsExtensions.SetValue<DefaultType, T>(T)

CollectionsExtensions.AsEnum<DefaultType>()

DataObjectQueryableExtensions.CompareVariable(FilterOperation, String)

Processing.Do<DefaultType>(Action<IProcess<DefaultType>>[])

TypeExtensions.ChangeType<T>()

TypeExtensions.ChangeType(Type)

TypeExtensions.InvokeMethod<TRet>(Type[], Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Type[], Expression<Action>, BindingFlags)

TypeExtensions.InvokeMethod<TRet>(Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Expression<Action>, BindingFlags)

TypeExtensions.ConvertTo<T>()

TypeExtensions.EqualValue(Enum)

SqlBlockExtensions.SetExpression<DefaultType, TResult>(Expression<Func<IBlockQueryAdapter, TResult>>)

QueryDescriptionExtensions.SetExpression<DefaultType, TResult>(Expression<Func<IQueryStructureAdapter, TResult>>)

Enum DialectOptions

SQL dialect options

Namespace: [XData](#)
Assembly: XData.docfx.dll

Syntax

```
[Flags]  
public enum DialectOptions : ulong
```

Fields

NAME	DESCRIPTION
CheckConcurrencySupport	Concurrency check supported
ClearByRowIdArrays	Clear statement used row id arrays
InternalErrorCatch	Concurrency SQL errors are handled and jammed in provider
MultipleRecordsetProceduresSupported	Multiple record set stored procedures are supported
None	No specific SQL dialect options declared
ObjectParametersSupported	User defined objects (UDT) are supported as procedure/function parameter types
RegisterRecordsetRequired	Output record set registering is required
ReturningInResult	Insert returned identity value as query result
RowIteratorSupported	Row iterator is supported
SafeTypeConvert	Safe type conversion required
ScalarFunctionsOutParametersAllowed	Scalar-valued SQL functions OUT parameters are supported
ScalarFunctionsSupported	Scalar-valued SQL functions are supported

NAME	DESCRIPTION
SequenceRequired	Sequence is required to auto increment fields
SequenceSupported	Sequences are supported
SqlBlockOuterTableVariables	Sql block can operate with table variables out of block only
StoredProcedureResultAsExecuteScalar	SQL function result value returned as Execute scalar result
StoredProcedureResultFetch	Stored procedure results fetched by multiple reader execute
StoredProceduresSupported	SQL stored procedures are supported
TableFunctionsSupported	Table-valued SQL functions are supported
TableOutParametersAllowed	Table types are supported as procedure/function OUT parameter types
TableParametersSupported	Table types are supported as procedure/function parameter types
TempTableNotSupported	Temp Table is not supported
UdtAsReturnValueSupported	SQL function can return UDT or Arrays
UpdateSubqueryFields	Update can use subquery fields as update source values and target fields
UpdateTableNotInFrom	Update table is part of update (or delete) statement and not listed in from statement
UpdateTableNotUseAlias	Update and delete statements only for one table and not use table alias
UpdateUseTableNames	Update not use table aliases but lower case table names
WithInsideSubqueryNotSupported	Common-Table-Expression (WITH) queries not supported inside subqueries

NAME	DESCRIPTION
WithSupported	Common-Table-Expression (WITH) queries are supported
WithUnionNotSupported	Common-Table-Expression (WITH) recursive queries combination with UNION operation not supported
XmlSourceSupported	XML data source is supported

Extension Methods

[SerializationExtensions.ToXml<DialectOptions>\(\)](#)

[CollectionsExtensions.SetValue<DialectOptions, T>\(T\)](#)

[CollectionsExtensions.AsEnum<DialectOptions>\(\)](#)

[DataObjectQueryableExtensions.CompareVariable\(FilterOperation, String\)](#)

[Processing.Do<DialectOptions>\(Action<IProcess<DialectOptions>> > \[\]\)](#)

[TypeExtensions.ChangeType<T>\(\)](#)

[TypeExtensions.ChangeType\(Type\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Type\[\], Expression<Action>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Expression<Action>, BindingFlags\)](#)

[TypeExtensions.ConvertTo<T>\(\)](#)

[TypeExtensions.EqualValue\(Enum\)](#)

[SqlBlockExtensions.SetExpression<DialectOptions, TResult>\(Expression<Func<IBlockQueryAdapter, TResult>>\)](#)

[QueryDescriptionExtensions.SetExpression<DialectOptions, TResult>\(Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Delegate ExecuteAttachedHandler<T>

Execute attached handler delegate

Namespace: [XData](#)
Assembly: XData.docfx.dll

Syntax

```
public delegate bool ExecuteAttachedHandler<in T>(string key, IEnumerable<T> objects)
    where T : class, IDataObject;
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	key	Custom logic key
System.Collections.Generic.IEnumerable<T>	objects	Object sequence

Returns

TYPE	DESCRIPTION
System.Boolean	

Type Parameters

NAME	DESCRIPTION
T	Object type

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Enum ExistsOperation

Exists subquery operation

Namespace: [XData](#)
Assembly: XData.docfx.dll

Syntax

```
public enum ExistsOperation
```

Fields

NAME	DESCRIPTION
Exists	Exists
None	Operation other
NotExists	Not exists

Extension Methods

- [SerializationExtensions.ToXml<ExistsOperation>\(\)](#)
- [CollectionsExtensions.SetValue<ExistsOperation, T>\(T\)](#)
- [CollectionsExtensions.AsEnum<ExistsOperation>\(\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(FilterOperation, String\)](#)
- [Processing.Do<ExistsOperation>\(Action<IProcess<ExistsOperation>>\[\]\)](#)
- [TypeExtensions.ChangeType<T>\(\)](#)
- [TypeExtensions.ChangeType\(Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.ConvertTo<T>\(\)](#)
- [TypeExtensions.EqualValue\(Enum\)](#)
- [SqlBlockExtensions.SetExpression<ExistsOperation, TResult>\(Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<ExistsOperation, TResult>\(Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Enum FilterConstantType

ConstantFilter value mode

Namespace: [XData](#)
Assembly: XData.docfx.dll

Syntax

```
[DataContract]
public enum FilterConstantType
```

Fields

NAME	DESCRIPTION
Const	Constant value
CurrentDate	Current date value
CurrentDateTime	Current date and time value
CurrentDateTimeUtc	Current date and time UTC value
Variable	Repository variable value by name

See Also

- [ConstantType](#)
- [Const](#)
- [Variable](#)
- [CurrentDate](#)
- [CurrentDateTime](#)
- [CurrentDateTimeUtc](#)

Extension Methods

- [SerializationExtensions.ToXml<FilterConstantType>\(\)](#)
- [CollectionsExtensions.SetValue<FilterConstantType, T>\(T\)](#)
- [CollectionsExtensions.AsEnum<FilterConstantType>\(\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(FilterOperation, String\)](#)
- [Processing.Do<FilterConstantType>\(Action<IProcess<FilterConstantType>> > \[\]\)](#)
- [TypeExtensions.ChangeType<T>\(\)](#)
- [TypeExtensions.ChangeType\(Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.ConvertTo<T>\(\)](#)
- [TypeExtensions.EqualValue\(Enum\)](#)
- [SqlBlockExtensions.SetExpression<FilterConstantType, TResult>\(Expression<Func<IBlockQueryAdapter, TResult>> >\)](#)

QueryDescriptionExtensions.SetExpression<FilterConstantType, TResult>(Expression<Func<IQueryStructureAdapter, TResult>>)

Enum FilterNullable

Filter null value assignment mode

Namespace: [XData](#)
Assembly: XData.docfx.dll

Syntax

```
[DataContract]
public enum FilterNullable
```

Fields

NAME	DESCRIPTION
Nullable	Null value omit the filter (filter condition will not equated)
NullsAllowed	Null value is compared separately
NullsCompared	Null values is compared only
NullsNotAllowed	Null value is set filter condition to false

See Also

- [Nullable](#)
- [Nullable](#)
- [Nullable](#)
- [NullsAllowed](#)
- [NullsNotAllowed](#)

Extension Methods

- [SerializationExtensions.ToXml<FilterNullable>\(\)](#)
- [CollectionsExtensions.SetValue<FilterNullable, T>\(T\)](#)
- [CollectionsExtensions.AsEnum<FilterNullable>\(\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(FilterOperation, String\)](#)
- [Processing.Do<FilterNullable>\(Action<IProcess<FilterNullable>>\[\]\)](#)
- [TypeExtensions.ChangeType<T>\(\)](#)
- [TypeExtensions.ChangeType\(Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.ConvertTo<T>\(\)](#)
- [TypeExtensions.EqualValue\(Enum\)](#)
- [SqlBlockExtensions.SetExpression<FilterNullable, TResult>\(Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<FilterNullable, TResult>\(Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Enum FilterOperation

Filter operation

Namespace: [XData](#)
Assembly: XData.docfx.dll

Syntax

```
public enum FilterOperation
```

Fields

NAME	DESCRIPTION
Contains	Contains (contains) fulltext filter operation
Equal	Equal (=) filter operation
Exists	Exists (exists (...)) filter operation
GreaterThan	Greater Then (>) filter operation
GreaterThanOrEqual	Greater Then or Equal (>=) filter operation
In	In sequence (in (...)) filter operation
LessThan	Less Then (<) filter operation
LessThanOrEqual	Less Then or Equal (<=) filter operation
Like	Like (like) filter operation
NotContains	Not Contains (not contains) fulltext filter operation
NotEqual	Not Equal (!=) filter operation
NotExists	Not Exists (not exists (...)) filter operation
NotIn	Not In sequence (not in (...)) filter operation

NAME	DESCRIPTION
NotLike	Not Like (not like) filter operation
OuterJoin	Outer join (outer join) filter operation

See Also

- [Operation](#)
- [Operation](#)
- [Equal](#)
- [NotEqual](#)
- [LessThan](#)
- [GreaterThan](#)
- [LessThanOrEqual](#)
- [GreaterThanOrEqual](#)
- [In](#)
- [NotIn](#)
- [Exists](#)
- [NotExists](#)
- [Like](#)
- [NotLike](#)
- [OuterJoin](#)
- [Contains](#)
- [NotContains](#)

Extension Methods

- [SerializationExtensions.ToXml<FilterOperation>\(\)](#)
- [CollectionsExtensions.SetValue<FilterOperation, T>\(T\)](#)
- [CollectionsExtensions.AsEnum<FilterOperation>\(\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(FilterOperation, String\)](#)
- [Processing.Do<FilterOperation>\(Action<IProcess<FilterOperation>> \[\]\)](#)
- [TypeExtensions.ChangeType<T>\(\)](#)
- [TypeExtensions.ChangeType\(Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.ConvertTo<T>\(\)](#)
- [TypeExtensions.EqualValue\(Enum\)](#)
- [SqlBlockExtensions.SetExpression<FilterOperation, TResult>\(Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<FilterOperation, TResult>\(Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Enum FilterType

Filter type

Namespace: [XData](#)
Assembly: XData.docfx.dll

Syntax

```
[DataContract]  
public enum FilterType
```

Fields

NAME	DESCRIPTION
Constant	Constant value filter
Dictionary	Dictionary link
Expression	Expression filter
InnerViewLink	Inner view link
Link	Inner link
Range	Range filter
Runtime	Runtime value filter
Subquery	Subquery filter
SubqueryLink	Subquery link

See Also

- [Link](#)
- [SubqueryLink](#)
- [InnerViewLink](#)
- [Dictionary](#)
- [Constant](#)
- [Subquery](#)
- [Expression](#)
- [Range](#)
- [Runtime](#)

Extension Methods

SerializationExtensions.ToXml<FilterType>()

CollectionsExtensions.SetValue<FilterType, T>(T)

CollectionsExtensions.AsEnum<FilterType>()

DataObjectQueryableExtensions.CompareVariable(FilterOperation, String)

Processing.Do<FilterType>(Action<IProcess<FilterType>>[])

TypeExtensions.ChangeType<T>()

TypeExtensions.ChangeType(Type)

TypeExtensions.InvokeMethod<TRet>(Type[], Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Type[], Expression<Action>, BindingFlags)

TypeExtensions.InvokeMethod<TRet>(Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Expression<Action>, BindingFlags)

TypeExtensions.ConvertTo<T>()

TypeExtensions.EqualValue(Enum)

SqlBlockExtensions.SetExpression<FilterType, TResult>(Expression<Func<IBlockQueryAdapter, TResult>>)

QueryDescriptionExtensions.SetExpression<FilterType, TResult>(Expression<Func<IQueryStructureAdapter, TResult>>)

Delegate GetLobAttachedHandler

Get Lob attached handler delegate

Namespace: [XData](#)
Assembly: XData.docfx.dll

Syntax

```
public delegate bool GetLobAttachedHandler(string property, IDataObject obj, out byte[] body);
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	property	Lob property name
IDataObject	obj	Data object
Byte[]	body	Lob body

Returns

TYPE	DESCRIPTION
System.Boolean	

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Delegate GetLobSizeAttachedHandler

Get Lob size attached handler delegate

Namespace: [XData](#)
Assembly: XData.docfx.dll

Syntax

```
public delegate bool GetLobSizeAttachedHandler(string property, IDataObject obj, out long size);
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	property	Lob property name
IDataObject	obj	Data object
System.Int64	size	Lob body size

Returns

TYPE	DESCRIPTION
System.Boolean	

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Delegate GetXmlAttachedHandler

Get Xml attached handler delegate

Namespace: [XData](#)
Assembly: XData.docfx.dll

Syntax

```
public delegate bool GetXmlAttachedHandler(string property, IDataObject obj, out string xml);
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	property	Xml property name
IDataObject	obj	Data object
System.String	xml	Xml string

Returns

TYPE	DESCRIPTION
System.Boolean	

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Interface IDataEngine

XData engine interface

Namespace: [XData](#)
Assembly: XData.docfx.dll

Syntax

```
public interface IDataEngine
```

Properties

Configuration

XData configuration object

Declaration

```
IConfiguration Configuration { get; }
```

Property Value

TYPE	DESCRIPTION
IConfiguration	

See Also

[IConfiguration](#)

Methods

GetDataScope(Guid, Boolean)

Request existing or create data scope

Declaration

```
IDataScope GetDataScope(Guid layer, bool throwIfNotFound = false)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Guid	layer	Data scope layer
System.Boolean	throwIfNotFound	Throw an exception when data scope is not found

Returns

TYPE	DESCRIPTION
IDataScope	Data scope

See Also

[IDataScope](#)

GetDataScope(Guid, String, CancellationToken, Boolean)

Request existing or create data scope

Declaration

```
IDataScope GetDataScope(Guid layer, string defaultContext, CancellationToken token, bool throwIfNotFound = false)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Guid	layer	Data scope layer
System.String	defaultContext	Default context of scope (if not found)
System.Threading.CancellationToken	token	Cancellation token
System.Boolean	throwIfNotFound	Throw an exception when data scope is not found

Returns

TYPE	DESCRIPTION
IDataScope	Data scope

See Also

[IDataScope](#)

GetDataScope(Guid, String, ISecuritySession, CancellationToken, Boolean)

Request existing or create data scope

Declaration

```
IDataScope GetDataScope(Guid layer, string defaultContext, ISecuritySession security, CancellationToken token, bool throwIfNotFound = false)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Guid	layer	Data scope layer
System.String	defaultContext	Default context of scope (if not found)
ISecuritySession	security	Security session

TYPE	NAME	DESCRIPTION
System.Threading.CancellationToken	token	Cancellation token
System.Boolean	throwIfNotFound	Throw an exception when data scope is not found

Returns

TYPE	DESCRIPTION
IDataScope	Data scope

See Also

[IDataScope](#)

GetDataScope(Guid, CancellationToken, Boolean)

Request existing or create data scope

Declaration

```
IDataScope GetDataScope(Guid layer, CancellationToken token, bool throwIfNotFound = false)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Guid	layer	Data scope layer
System.Threading.CancellationToken	token	Cancellation token
System.Boolean	throwIfNotFound	Throw an exception when data scope is not found

Returns

TYPE	DESCRIPTION
IDataScope	Data scope

See Also

[IDataScope](#)

GetDataScope(Guid, ISecuritySession, CancellationToken, Boolean)

Request existing or create data scope

Declaration

```
IDataScope GetDataScope(Guid layer, ISecuritySession security, CancellationToken token, bool throwIfNotFound = false)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Guid	layer	Data scope layer
ISecuritySession	security	Security session
System.Threading.CancellationToken	token	Cancellation token
System.Boolean	throwIfNotFound	Throw an exception when data scope is not found

Returns

TYPE	DESCRIPTION
IDataScope	Data scope

See Also

[IDataScope](#)

NewDataScope()

Create new data scope

Declaration

<code>IDataScope NewDataScope()</code>

Returns

TYPE	DESCRIPTION
IDataScope	Data scope

See Also

[IDataScope](#)

NewDataScope(String)

Create new data scope

Declaration

<code>IDataScope NewDataScope(string defaultContext)</code>

Parameters

TYPE	NAME	DESCRIPTION

TYPE	NAME	DESCRIPTION
System.String	defaultContext	Default context of scope

Returns

TYPE	DESCRIPTION
IDataScope	Data scope

See Also

[IDataScope](#)

NewDataScope(String, CancellationToken)

Create new data scope

Declaration

```
IDataScope NewDataScope(string defaultContext, CancellationToken token)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	defaultContext	Default context of scope
System.Threading.CancellationToken	token	Cancellation token

Returns

TYPE	DESCRIPTION
IDataScope	Data scope

See Also

[IDataScope](#)

NewDataScope(String, ISecuritySession)

Create new data scope

Declaration

```
IDataScope NewDataScope(string defaultContext, ISecuritySession security)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	defaultContext	Default context of scope

TYPE	NAME	DESCRIPTION
ISecuritySession	security	Security session

Returns

TYPE	DESCRIPTION
IDataScope	Data scope

See Also

[IDataScope](#)

NewDataScope(String, ISecuritySession, CancellationToken)

Create new data scope

Declaration

```
IDataScope NewDataScope(string defaultContext, ISecuritySession security, CancellationToken token)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	defaultContext	Default context of scope
ISecuritySession	security	Security session
System.Threading.CancellationToken	token	Cancellation token

Returns

TYPE	DESCRIPTION
IDataScope	Data scope

See Also

[IDataScope](#)

NewDataScope(CancellationToken)

Create new data scope

Declaration

```
IDataScope NewDataScope(CancellationToken token)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Threading.CancellationToken	token	Cancellation token

Returns

TYPE	DESCRIPTION
IDataScope	Data scope

See Also

[IDataScope](#)

NewDataScope(ISecuritySession, CancellationToken)

Create new data scope

Declaration

```
IDataScope NewDataScope(ISecuritySession security, CancellationToken token)
```

Parameters

TYPE	NAME	DESCRIPTION
ISecuritySession	security	Security session
System.Threading.CancellationToken	token	Cancellation token

Returns

TYPE	DESCRIPTION
IDataScope	Data scope

See Also

[IDataScope](#)

ResetAllDictionaries()

Reset all dictionaries cache

Declaration

```
void ResetAllDictionaries()
```

See Also

[GetDictionaryValue<T>\(Expression<Func<T, Boolean>>, String, ISecuritySession, Variable\[\]\)](#)

ResetDictionaryCache<T>()

Reset single dictionary cache

Declaration

```
void ResetDictionaryCache<T>()  
    where T : class, IDataObject
```

Type Parameters

NAME	DESCRIPTION
T	Dictionary type

See Also

[GetDictionaryValue<T>\(Expression<Func<T, Boolean>>, String, ISecuritySession, Variable\[\]\)](#)

Extension Methods

[SerializationExtensions.ToXml<T>\(T\)](#)

[CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)

[CollectionsExtensions.AsEnum<T>\(T\)](#)

[DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)

[Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)

[TypeExtensions.ChangeType<T>\(Object\)](#)

[TypeExtensions.ChangeType\(Object, Type\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)

[SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)

[QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Interface IDataLogic<T>

Data logic interface

Namespace: [XData](#)
Assembly: XData.docfx.dll

Syntax

```
public interface IDataLogic<T>
    where T : class, IDataObject
```

Type Parameters

NAME	DESCRIPTION
T	

Remarks

Marking interface for classes contains data aware logic of mapped objects

Used to divide data aware logic from mapping

Hint: Separating mapping and logic to different assemblies allows to run data aware logic on XData application server

Hint: Application can use XData application server or direct connect to database depending of configuration (see [IConfiguration](#))

Examples

```
public abstract class InvoiceLogic : IDataLogic<Invoice> {

    [Action(DataActionType.BeforeClear)]

    public readonly static RepositoryTrigger<Invoice> ClearHistoryBatch =

        ((IRepository<Invoice> invoiceRepository, ref DataTriggerFlag flag) => DoSomething());

    [CustomAction("{ab12d6f0-69d0-4997-bb0b-5f5f7c0581d1}")]

    public static readonly CustomLogic<Invoice> TestCustomLogic = (objects => { return DoSomethingElse(); });

}
```

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)

[Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
[TypeExtensions.ChangeType<T>\(Object\)](#)
[TypeExtensions.ChangeType\(Object, Type\)](#)
[TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
[TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
[TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
[TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
[SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
[QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

See Also

[IDataObject](#)
[InitRepository<T>](#)
[InvalidateRepository<T>](#)
[InitObject<T>](#)
[InvalidateObject<T>](#)
[Trigger<T>](#)
[CustomLogic<T>](#)

Interface IDataObject

Data object interface

Namespace: [XData](#)

Assembly: XData.docfx.dll

Syntax

```
[JsonConverter(typeof(JsonDataObjectConverter<IDataObject>))]  
public interface IDataObject : ISqlObject
```

Remarks

Mapping domain object to database

WARNING! Data object mapping requires [DataObjectAttribute](#)!

Hint: Use [DataTableAttribute](#), [SubqueryAttribute](#) and [InnerViewAttribute](#) to map data object to data sources

Hint: Use [FilterAttribute](#)'s specific successor to link data sources and apply filters to data

Hint: Use [ExternalLinkAttribute](#) to declare master/slave relations between repositories

Hint: Use [Lob](#) to map binary fields

```
[Property("scan", "SC", Flags = DataPropertyFlag.OuterFlag)]
```

```
public Lob Scan { get; set; }
```

Hint: Use [Xml](#) to map Xml fields

```
[Property("source", "SR", Flags = DataPropertyFlag.OuterFlag)]
```

```
public Xml Source { get; set; }
```

Hint: Use [Link<TVal, TSrc>](#) to map data link

```
[Property("name", "S"),
```

```
    DictionaryProperty("Name", "DocState"),
```

```
    DictionaryProperty("Code", "DocStateCode")]
```

```
public Link<string, DocState> DocState { get; set; }
```

Hint: Use [ColumnAttribute](#) to map data without declaring property (for example to declare [Id](#))

```
[DataObject("S"),

    DataTable("T_DOC_STATE", "S"),

    Column("DocStateId", "doc_state_id", typeof(long?), "S", Flags = DataPropertyFlag.Id)]
```

Hint: Use [PropertyExpressionAttribute](#) or [ColumnExpressionAttribute](#) to use SQL expression or subquery as property (or mapped column) data source

```
[Property("doc_amount"),

    PropertyExpression("A", DataExpressionType.SubQuery, DbType.Decimal, ExprSize = 17, ExprScale = 5)]

public double? DocAmount { get { return this.GetProperty(() => DocAmount); } }
```

Hint: Use [PropertyDefaultAttribute](#) or [ColumnDefaultAttribute](#) to set default value generation rule to property (or mapped column)

```
[Property("doc_date", "D"),

    PropertyDefault(DefaultType.CurrentDate)]

public DateTime? DocDate { get; set; }
```

Hint: Use [ReferenceAttribute](#) to use inner view property in parent data object

```
[PropertyLink("HistoryDate", "H")]

public DateTime? DocLastChange { get { return this.GetProperty(() => DocLastChange); } }
```

Hint: Declare readonly fields as

```
[PropertyLink("HistoryDate", "H")]

public DateTime? DocLastChange { get { return this.GetProperty(() => DocLastChange); } }
```

Hint: Declare custom logic functions (see [IDataLogic<T>](#) and [Execute<T>\(ICollection<T>, String, IDictionary<String, Action<Byte\[\]>>, IDictionary<String, Func<Byte\[\], Byte\[\]>>\)\)](#) as

```
//generate unique action key and use it for mark it realization in IDataLogic<Invoice>

[CustomAction("{ab12d6f0-69d0-4997-bb0b-5f5f7c0581d1}")]

public static CustomLogic<Invoice> TestCustomLogic;

//using

obj.Execute(() => Invoice.TestCustomLogic);
```

Hint: [Variables](#) can be used as temporary values storage

```
// store

obj.GetRepository().Variables.Add("SomeName", "SomeValue");

// and somewhere you can get it

var something = obj.GetRepository().Variables["SomeName"];
```

Hint: [HierarchyAttribute](#) can be used to fill and maintain hierarchy tables automatically

```

// Table T_TEST has parent_id field tree linked to test_id (T_TEST's pk)

// Table L_TEST has two fields parent_id and child_id all linked to T_TEST.test_id

// Table L_TEST contains all subtree relations inside T_TEST

// For example: for T_TEST (test_id, parent_id): (1, null) : (2, 1) : (3, 2)

// L_TEST will contain (parent_id, child_id): (1, 1) : (1, 2) : (1, 3) : (2, 2) : (2, 3) : (3, 3)

[DataObject("T"),

    DataTable("T_TEST", "T"),

    Hierarchy("T", "L_TEST", "parent_id", "parent_id", "child_id"),

    Column("TestId", "test_id", typeof(long?), "T", Flags = DataPropertyFlag.Id)]

```

Examples

```

[DataObject("T"),

    DataTable("T_DOC_TYPE", "T"),

    Column("DocTypeId", "doc_type_id", typeof(long?), "T", Flags = DataPropertyFlag.Id)]

public class DocType : IDataObject {

    [Property("name", "T")]

    public string Name { get; set; }

    [Property("code", "T")]

    public string Code { get; set; }

}

```

Extension Methods

[DataObjectExtensions.Modify<T>\(T, Action<T>\[\]\)](#)
[DataObjectExtensions.GetProperties<T>\(T, IEnumerable<String>, DataVersion\)](#)
[DataObjectExtensions.GetProperties<T>\(T, String\[\]\)](#)
[DataObjectExtensions.GetProperties<T>\(T, DataVersion, String\[\]\)](#)
[DataObjectExtensions.IsCleared<T>\(T, String\)](#)
[DataObjectExtensions.IsChanged<T>\(T, String\)](#)
[DataObjectExtensions.IsChanged<T>\(T, Expression<Func<T, Object>>\[\]\)](#)
[DataObjectExtensions.IsEmpty<T>\(T, String\)](#)
[DataObjectExtensions.IsEmpty<T>\(T, Expression<Func<T, Object>>\)](#)
[DataObjectExtensions.GetProperty<TRet>\(IDataObject, String\)](#)
[DataObjectExtensions.GetContext\(IDataObject\)](#)
[DataObjectExtensions.GetLayer\(IDataObject\)](#)
[DataObjectExtensions.GetRepositoryAlias\(IDataObject\)](#)
[DataObjectExtensions.IsAttached\(IDataObject, Type\)](#)
[DataObjectExtensions.GetState<T>\(T\)](#)
[DataObjectExtensions.CheckState<T>\(T, DataObjectState\)](#)
[SerializationExtensions.ToXml<T>\(T\)](#)
[CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
[CollectionsExtensions.AsEnum<T>\(T\)](#)
[DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
[Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
[TypeExtensions.ChangeType<T>\(Object\)](#)
[TypeExtensions.ChangeType\(Object, Type\)](#)
[TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
[TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
[TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
[TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
[SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
[QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)
[QueryDescriptionExtensions.GetProperty\(ISqlObject, String\)](#)

See Also

[IRepository<T>](#)
[IDataLogic<T>](#)
[CheckState<T>\(T, DataObjectState\)](#)
[SetDeleted<T>\(T, System.Boolean\)](#)
[Execute<T>\(T, System.Linq.Expressions.Expression<System.Func<\[CustomLogic\]\(#\)<T>>>, System.Collections.Generic.IDictionary<System.String, System.Action<\[Byte\]\(#\)\[\]>>, System.Collections.Generic.IDictionary<System.String, System.Func<\[Byte\]\(#\)\[\], \[Byte\]\(#\)\[\]>>\)](#)
[Serialize<T>\(T\)](#)
[Restore<T>\(T, \[SerializedData\]\(#\)\)](#)
[GetProperties<T>\(T, \[DataVersion\]\(#\), System.String\[\]\)](#)
[GetProperty<T, TRet>\(T, System.Linq.Expressions.Expression<System.Func<T, TRet>>, \[DataVersion\]\(#\)\)](#)
[XData.DataObjectExtensions.GetProperty``1\(``0, System.String, XData.DataVersion\)](#)
[GetRepository<T>\(T\)](#)
[IsChanged<T>\(T, System.Linq.Expressions.Expression<System.Func<T, System.Object>>\[\]\)](#)
[IsEmpty<T>\(T, System.Linq.Expressions.Expression<System.Func<T, System.Object>>\)](#)
[IsCleared<T>\(T, System.String\)](#)
[SetCurrent<T>\(T\)](#)

Submit<T>(T, DataSubmitFlag)

Lock<T>(T)

Interface IDataOptions

XData options interface

Namespace: [XData](#)

Assembly: XData.docfx.dll

Syntax

```
public interface IDataOptions
```

Methods

AddAdapter(String, String, IConfiguration)

Add supported SQL adapter description

Declaration

```
IDataOptions AddAdapter(string name, string assemblyName, IConfiguration options = null)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Adapter name
System.String	assemblyName	Assembly name of adapter plugin
Microsoft.Extensions.Configuration.IConfiguration	options	Adapter options (xml) see concrete adapter readme for details

Returns

TYPE	DESCRIPTION
IDataOptions	XData options interface

AddContext(String, String, String, String, Boolean, String, String)

Add context description

Declaration

```
IDataOptions AddContext(string name, string dialect, string adapter, string connectionString, bool isDefault, string sequence = null, string concurrencyErrorId = null)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Context name

TYPE	NAME	DESCRIPTION
System.String	dialect	Dialect name
System.String	adapter	Adapter or proxy name
System.String	connectionString	Connection string
System.Boolean	isDefault	Default context flag
System.String	sequence	(optional) Assembly qualified type name implements ISequenceNameRule
System.String	concurrencyErrorId	(optional) Concurrency error id

Returns

TYPE	DESCRIPTION
IDataOptions	XData options interface

AddDialect(String, String, IConfiguration)

Add supported SQL dialect description

Declaration

```
IDataOptions AddDialect(string name, string assemblyName, IConfiguration options = null)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Dialect name
System.String	assemblyName	Assembly name of dialect plugin
Microsoft.Extensions.Configuration.IConfiguration	options	Dialect options (xml) see concrete dialect readme for details

Returns

TYPE	DESCRIPTION
IDataOptions	XData options interface

AddProxy(String, String)

Add proxy adapter description

Declaration

```
IDataOptions AddProxy(string name, string endpoint)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Proxy name
System.String	endpoint	WCF client endpoint name

Returns

TYPE	DESCRIPTION
IDataOptions	XData options interface

SetLifeTmeLimit(Int32)

Set XData cache lifetime limit in minutes

Declaration

```
IDataOptions SetLifeTmeLimit(int minutes)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Int32	minutes	XData cache lifetime limit

Returns

TYPE	DESCRIPTION
IDataOptions	XData options interface

SetMonitoringContext(String)

Set monitoring context (optional, manual configuration)

Declaration

```
IDataOptions SetMonitoringContext(string monitoringContext)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	monitoringContext	

Returns

TYPE	DESCRIPTION
IDataOptions	XData options interface

SetSecurityContext(String)

Set security context (optional, manual configuration)

Declaration

```
IDataOptions SetSecurityContext(string securityContext)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	securityContext	Security context

Returns

TYPE	DESCRIPTION
IDataOptions	XData options interface

UseCredentials(String, KeyValuePair<CredentialsType, String>[])

Set database credentials

Declaration

```
IDataOptions UseCredentials(string context, params KeyValuePair<CredentialsType, string>[] credentials)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	context	Context name
System.Collections.Generic.KeyValuePair< CredentialsType , System.String>[]	credentials	Credentials

Returns

TYPE	DESCRIPTION
IDataOptions	XData options interface

Extension Methods

SerializationExtensions.ToXml<T>(T)

CollectionsExtensions.SetValue<TKey, T>(TKey, T)

CollectionsExtensions.AsEnum<T>(T)

DataObjectQueryableExtensions.CompareVariable(Object, FilterOperation, String)

Processing.Do<T>(T, Action<IProcess<T>>[])

TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>)

TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>, TResult)

TypeExtensions.ChangeType<T>(Object)

TypeExtensions.ChangeType(Object, Type)

TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)

TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)

SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)

QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Interface IDataQuery

Read only data source basic interface

Inherited Members

System.Linq.IQueryable.ElementType

System.Linq.IQueryable.Expression

System.Linq.IQueryable.Provider

System.Collections.IEnumerable.GetEnumerator()

System.IDisposable.Dispose()

Namespace: [XData](#)

Assembly: XData.docfx.dll

Syntax

```
public interface IDataQuery : IOrderedQueryable, IQueryable, IEnumerable, IDisposable
```

Properties

Alias

Data repository alias

Declaration

```
string Alias { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Remarks

Default value is data object type name

Context

Data context

Declaration

```
string Context { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Keys

Data query/repository key properties

Declaration

```
string[] Keys { get; }
```

Property Value

TYPE	DESCRIPTION
System.String[]	

Layer

Data repository layer

Declaration

Guid Layer { get; }

Property Value

TYPE	DESCRIPTION
System.Guid	

Remarks

Sharing same layer the data object repositories can interact automatically as parent/child (see [ExternalLinkAttribute](#))

Repositories on different layers but same data object are independent for each other

Repositories on same layer based on same data object is the same repository

Token

Cancellation token

Declaration

CancellationToken Token { get; }

Property Value

TYPE	DESCRIPTION
System.Threading.CancellationToken	

Variables

Data query/repository/call result variables

Declaration

VariableIndexer Variables { get; }

Property Value

TYPE	DESCRIPTION
VariableIndexer	

Remarks

Variable values can be used to set the filter values (query/repository)

Variable values can be used in custom logic (repository)

Variables is stored in query/repository/call result instance

Examples

```
var rep = dataScope.GetRepository<Customer>(variables: new []{"OurCode", "SELF"});

var newInvoice = dataScope.GetRepository<Invoice>().New();

newInvoice.CustomerFrom += rep.First(x => x.Code.CompareVariable(FilterOperation.Equal, "SELF"));
```

See Also

Variable

[GetRepository<T>\(String, ISecuritySession, Variable\[\]\)](#)

[XData.DataObjectExtensions.CompareVariable\(XData.IDataQueryInternal,System.Object,XData.FilterOperation,System.String\)](#)

Extension Methods

[DataObjectExtensions.GetParameter<T>\(IDataQuery, String\)](#)

[DataObjectExtensions.GetResultSet<TRes>\(IDataQuery, String\)](#)

[DataObjectExtensions.GetResultSet<TRes>\(IDataQuery, Expression<Func<IEnumerable<TRes>>>\)](#)

[SerializationExtensions.ToXml<T>\(T\)](#)

[AsyncExtensions.GetParameterAsync<T>\(IDataQuery, String\)](#)

[AsyncExtensions.GetResultSetAsync<TRes>\(IDataQuery, String\)](#)

[AsyncExtensions.GetResultSetAsync<TRes>\(IDataQuery, Expression<Func<IEnumerable<TRes>>>\)](#)

[CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)

[CollectionsExtensions.AsEnum<T>\(T\)](#)

[DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)

[Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)

[TypeExtensions.ChangeType<T>\(Object\)](#)

[TypeExtensions.ChangeType\(Object, Type\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)

[SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)

[QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Interface IDataQuery<T>

Specialized read only data source interface

Inherited Members

- [IDataQuery.Alias](#)
- [IDataQuery.Layer](#)
- [IDataQuery.Context](#)
- [IDataQuery.Variables](#)
- [IDataQuery.Keys](#)
- [IDataQuery.Token](#)

- System.IDisposable.Dispose()
- System.Collections.Generic.IEnumerable<T>.GetEnumerator()
- System.Linq.IQueryable.ElementType
- System.Linq.IQueryable.Expression
- System.Linq.IQueryable.Provider

Namespace: [XData](#)
Assembly: XData.docfx.dll

Syntax

```
public interface IDataQuery<out T> : IDataQuery, IDisposable, IOrderedQueryable<T>, IOrderedQueryable, IQueryable<T>, IEnumerable<T>, IQueryable, IEnumerable
```

Type Parameters

NAME	DESCRIPTION
T	Data source type

Methods

GetFilterValue(RuntimeFilter)

Get metadata defined RuntimeFilter value

Declaration

```
object GetFilterValue(RuntimeFilter filter)
```

Parameters

TYPE	NAME	DESCRIPTION
RuntimeFilter	filter	Filter

Returns

TYPE	DESCRIPTION
System.Object	Filter value

Exceptions

TYPE	CONDITION
XDataRuntimeException	"filter" parameter is null.

See Also

- [RuntimeFilter](#)
- [Reset\(\)](#)
- [SetFilterValue\(RuntimeFilter, Object\)](#)

Reset()

Reset repository structure and filter values

Declaration

```
IDataQuery<T> Reset()
```

Returns

TYPE	DESCRIPTION
IDataQuery<T>	Self repository link

See Also

- [RuntimeFilter](#)
- [GetFilterValue\(RuntimeFilter\)](#)
- [SetFilterValue\(RuntimeFilter, Object\)](#)

SetFilterValue(RuntimeFilter, Object)

Set metadata defined RuntimeFilter value

Declaration

```
IDataQuery<T> SetFilterValue(RuntimeFilter filter, object val)
```

Parameters

TYPE	NAME	DESCRIPTION
RuntimeFilter	filter	Filter
System.Object	val	Filter value

Returns

TYPE	DESCRIPTION
IDataQuery<T>	Self repository link

Remarks

Hint: Set `val = null` to clear filter

Exceptions

TYPE	CONDITION
XDataRuntimeException	"filter" parameter is null.

See Also

[RuntimeFilter](#)

[Reset\(\)](#)

[GetFilterValue\(RuntimeFilter\)](#)

Extension Methods

[DataObjectExtensions.GetParameter<T>\(IDataQuery, String\)](#)

[DataObjectExtensions.GetResultSet<TRes>\(IDataQuery, String\)](#)

[DataObjectExtensions.GetResultSet<TRes>\(IDataQuery, Expression<Func<IEnumerable<TRes>>>\)](#)

[SerializationExtensions.ToXml<T>\(T\)](#)

[AsyncExtensions.AllAsync<T>\(IQueryable<T>, Expression<Func<T, Boolean>>\)](#)

[AsyncExtensions.AllAsync<T>\(IQueryable<T>, Expression<Func<T, Boolean>>, CancellationToken\)](#)

[AsyncExtensions.AllAsync<T>\(IDataQuery<T>, Expression<Func<T, Boolean>>\)](#)

[AsyncExtensions.AnyAsync<T>\(IQueryable<T>\)](#)

[AsyncExtensions.AnyAsync<T>\(IQueryable<T>, Expression<Func<T, Boolean>>\)](#)

[AsyncExtensions.AnyAsync<T>\(IQueryable<T>, Expression<Func<T, Boolean>>, CancellationToken\)](#)

[AsyncExtensions.AnyAsync<T>\(IDataQuery<T>, Expression<Func<T, Boolean>>\)](#)

[AsyncExtensions.AverageAsync<T>\(IQueryable<T>, Expression<Func<T, Single>>\)](#)

[AsyncExtensions.AverageAsync<T>\(IQueryable<T>, Expression<Func<T, Single>>, CancellationToken\)](#)

[AsyncExtensions.AverageAsync<T>\(IDataQuery<T>, Expression<Func<T, Single>>\)](#)

[AsyncExtensions.AverageAsync<T>\(IQueryable<T>, Expression<Func<T, Double>>\)](#)

[AsyncExtensions.AverageAsync<T>\(IQueryable<T>, Expression<Func<T, Double>>, CancellationToken\)](#)

[AsyncExtensions.AverageAsync<T>\(IDataQuery<T>, Expression<Func<T, Double>>\)](#)

[AsyncExtensions.AverageAsync<T>\(IQueryable<T>, Expression<Func<T, Decimal>>\)](#)

[AsyncExtensions.AverageAsync<T>\(IQueryable<T>, Expression<Func<T, Decimal>>, CancellationToken\)](#)

[AsyncExtensions.AverageAsync<T>\(IDataQuery<T>, Expression<Func<T, Decimal>>\)](#)

[AsyncExtensions.AverageAsync<T>\(IQueryable<T>, Expression<Func<T, Int32>>\)](#)

[AsyncExtensions.AverageAsync<T>\(IQueryable<T>, Expression<Func<T, Int32>>, CancellationToken\)](#)

[AsyncExtensions.AverageAsync<T>\(IDataQuery<T>, Expression<Func<T, Int32>>\)](#)

[AsyncExtensions.AverageAsync<T>\(IQueryable<T>, Expression<Func<T, Int64>>\)](#)

[AsyncExtensions.AverageAsync<T>\(IQueryable<T>, Expression<Func<T, Int64>>, CancellationToken\)](#)

[AsyncExtensions.AverageAsync<T>\(IDataQuery<T>, Expression<Func<T, Int64>>\)](#)

[AsyncExtensions.AverageAsync<T>\(IQueryable<T>, Expression<Func<T, Nullable<Single>>>\)](#)

[AsyncExtensions.AverageAsync<T>\(IQueryable<T>, Expression<Func<T, Nullable<Single>>>, CancellationToken\)](#)

[AsyncExtensions.AverageAsync<T>\(IDataQuery<T>, Expression<Func<T, Nullable<Single>>>\)](#)

[AsyncExtensions.AverageAsync<T>\(IQueryable<T>, Expression<Func<T, Nullable<Double>>>\)](#)

[AsyncExtensions.AverageAsync<T>\(IQueryable<T>, Expression<Func<T, Nullable<Double>>>, CancellationToken\)](#)

[AsyncExtensions.AverageAsync<T>\(IDataQuery<T>, Expression<Func<T, Nullable<Double>>>\)](#)

[AsyncExtensions.AverageAsync<T>\(IQueryable<T>, Expression<Func<T, Nullable<Decimal>>>\)](#)

[AsyncExtensions.AverageAsync<T>\(IQueryable<T>, Expression<Func<T, Nullable<Decimal>>>, CancellationToken\)](#)

[AsyncExtensions.AverageAsync<T>\(IDataQuery<T>, Expression<Func<T, Nullable<Decimal>>>\)](#)

[AsyncExtensions.AverageAsync<T>\(IQueryable<T>, Expression<Func<T, Nullable<Int32>>>\)](#)

[AsyncExtensions.AverageAsync<T>\(IQueryable<T>, Expression<Func<T, Nullable<Int32>>>, CancellationToken\)](#)

[AsyncExtensions.AverageAsync<T>\(IDataQuery<T>, Expression<Func<T, Nullable<Int32>>>\)](#)

[AsyncExtensions.AverageAsync<T>\(IQueryable<T>, Expression<Func<T, Nullable<Int64>>>\)](#)

[AsyncExtensions.AverageAsync<T>\(IQueryable<T>, Expression<Func<T, Nullable<Int64>>>, CancellationToken\)](#)

AsyncExtensions.AverageAsync<T>(IDataQuery<T>, Expression<Func<T, Nullable<Int64>>>)

AsyncExtensions.ContainsAsync<T>(IQueryable<T>, T, Func<T, Object>[])

AsyncExtensions.ContainsAsync<T>(IQueryable<T>, T, IEqualityComparer<T>)

AsyncExtensions.ContainsAsync<T>(IQueryable<T>, T, IEqualityComparer<T>, CancellationToken)

AsyncExtensions.ContainsAsync<T>(IDataQuery<T>, T, IEqualityComparer<T>)

AsyncExtensions.ContainsAsync<T>(IDataQuery<T>, T)

AsyncExtensions.CountAsync<T>(IQueryable<T>)

AsyncExtensions.CountAsync<T>(IQueryable<T>, Expression<Func<T, Boolean>>)

AsyncExtensions.CountAsync<T>(IQueryable<T>, Expression<Func<T, Boolean>>, CancellationToken)

AsyncExtensions.CountAsync<T>(IDataQuery<T>)

AsyncExtensions.CountAsync<T>(IDataQuery<T>, Expression<Func<T, Boolean>>)

AsyncExtensions.LongCountAsync<T>(IQueryable<T>)

AsyncExtensions.LongCountAsync<T>(IQueryable<T>, Expression<Func<T, Boolean>>)

AsyncExtensions.LongCountAsync<T>(IQueryable<T>, Expression<Func<T, Boolean>>, CancellationToken)

AsyncExtensions.LongCountAsync<T>(IDataQuery<T>)

AsyncExtensions.LongCountAsync<T>(IDataQuery<T>, Expression<Func<T, Boolean>>)

AsyncExtensions.FirstAsync<T>(IQueryable<T>)

AsyncExtensions.FirstAsync<T>(IQueryable<T>, Expression<Func<T, Boolean>>)

AsyncExtensions.FirstAsync<T>(IQueryable<T>, Expression<Func<T, Boolean>>, CancellationToken)

AsyncExtensions.FirstAsync<T>(IDataQuery<T>)

AsyncExtensions.FirstAsync<T>(IDataQuery<T>, Expression<Func<T, Boolean>>)

AsyncExtensions.FirstOrDefaultAsync<T>(IQueryable<T>)

AsyncExtensions.FirstOrDefaultAsync<T>(IQueryable<T>, Expression<Func<T, Boolean>>)

AsyncExtensions.FirstOrDefaultAsync<T>(IQueryable<T>, Expression<Func<T, Boolean>>, CancellationToken)

AsyncExtensions.FirstOrDefaultAsync<T>(IDataQuery<T>)

AsyncExtensions.FirstOrDefaultAsync<T>(IDataQuery<T>, Expression<Func<T, Boolean>>)

AsyncExtensions.ForEachAsync<T>(IQueryable<T>, Func<T, Task>)

AsyncExtensions.ForEachAsync<T>(IQueryable<T>, Func<T, Task>, CancellationToken)

AsyncExtensions.ForEachAsync<T>(IDataQuery<T>, Func<T, Task>)

AsyncExtensions.ForEachAsync<T>(IQueryable<T>, Action<T>)

AsyncExtensions.ForEachAsync<T>(IQueryable<T>, Action<T>, CancellationToken)

AsyncExtensions.ForEachAsync<T>(IDataQuery<T>, Action<T>)

AsyncExtensions.LastAsync<T>(IQueryable<T>)

AsyncExtensions.LastAsync<T>(IQueryable<T>, Expression<Func<T, Boolean>>)

AsyncExtensions.LastAsync<T>(IQueryable<T>, Expression<Func<T, Boolean>>, CancellationToken)

AsyncExtensions.LastAsync<T>(IDataQuery<T>)

AsyncExtensions.LastAsync<T>(IDataQuery<T>, Expression<Func<T, Boolean>>)

AsyncExtensions.LastOrDefaultAsync<T>(IQueryable<T>)

AsyncExtensions.LastOrDefaultAsync<T>(IQueryable<T>, Expression<Func<T, Boolean>>)

AsyncExtensions.LastOrDefaultAsync<T>(IQueryable<T>, Expression<Func<T, Boolean>>, CancellationToken)

AsyncExtensions.LastOrDefaultAsync<T>(IDataQuery<T>)

AsyncExtensions.LastOrDefaultAsync<T>(IDataQuery<T>, Expression<Func<T, Boolean>>)

AsyncExtensions.MaxAsync<T, TResult>(IQueryable<T>, Expression<Func<T, TResult>>)

AsyncExtensions.MaxAsync<T, TResult>(IQueryable<T>, Expression<Func<T, TResult>>, CancellationToken)

AsyncExtensions.MaxAsync<T, TResult>(IDataQuery<T>, Expression<Func<T, TResult>>)

AsyncExtensions.MinAsync<T, TResult>(IQueryable<T>, Expression<Func<T, TResult>>)

AsyncExtensions.MinAsync<T, TResult>(IQueryable<T>, Expression<Func<T, TResult>>, CancellationToken)

AsyncExtensions.MinAsync<T, TResult>(IDataQuery<T>, Expression<Func<T, TResult>>)

AsyncExtensions.GetParameterAsync<T>(IDataQuery, String)

AsyncExtensions.GetResultSetAsync<TRes>(IDataQuery, String)

AsyncExtensions.GetResultSetAsync<TRes>(IDataQuery, Expression<Func<IEnumerable<TRes>>>)

AsyncExtensions.SingleAsync<T>(IQueryable<T>)

[illegible]

JoinedEnumerable.Inner<TElement>(IEnumerable<TElement>)

JoinedEnumerable.Outer<TElement>(IEnumerable<TElement>)

JoinedEnumerable.LeftOuterJoin<TOuter, TInner, TKey, TResult>(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter, TKey>, Func<TInner, TKey>, Func<TOuter, TInner, TResult>, IEqualityComparer<TKey>)

JoinedEnumerable.RightOuterJoin<TOuter, TInner, TKey, TResult>(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter, TKey>, Func<TInner, TKey>, Func<TOuter, TInner, TResult>, IEqualityComparer<TKey>)

JoinedEnumerable.FullOuterJoin<TOuter, TInner, TKey, TResult>(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter, TKey>, Func<TInner, TKey>, Func<TOuter, TInner, TResult>, IEqualityComparer<TKey>)

Processing.Do<T>(T, Action<IProcess<T>>[])

TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>)

TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>, TResult)

TypeExtensions.ChangeType<T>(Object)

TypeExtensions.ChangeType(Object, Type)

TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)

TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)

SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)

QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

QueryDescriptionExtensions.ToDataSet<T>(IEnumerable<T>, String, ISqlBuilder)

Interface IDataScope

XData operational scope

Inherited Members

System.IDisposable.Dispose()

Namespace: [XData](#)

Assembly: XData.docfx.dll

Syntax

```
public interface IDataScope : IDisposable
```

Properties

DefaultContext

Data scope default context

Declaration

```
string DefaultContext { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Layer

Data scope layer identity

Declaration

```
Guid Layer { get; }
```

Property Value

TYPE	DESCRIPTION
System.Guid	

Security

Security session

Declaration

```
ISecuritySession Security { get; }
```

Property Value

TYPE	DESCRIPTION
ISecuritySession	

Token

Data scope cancellation token

Declaration

```
CancellationToken Token { get; }
```

Property Value

TYPE	DESCRIPTION
System.Threading.CancellationToken	

Methods

BeginTransaction(Boolean, IsolationLevel)

Begins single context transaction

Declaration

```
ITransaction BeginTransaction(bool read = false, IsolationLevel isolation = IsolationLevel.ReadCommitted)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Boolean	read	Read only transaction
System.Data.IsolationLevel	isolation	Transaction isolation level

Returns

TYPE	DESCRIPTION
ITransaction	Transaction wrapper

Exceptions

TYPE	CONDITION
XDataConfigurationException	XData configuration errors.
XDataAdapterException	XData adapter errors.
XDataSqlException	SQL exceptions while beginning transaction.

See Also

[ITransaction](#)

System.Data.IsolationLevel

ExecuteNonQuery(Query)

Execute SQL expression custom query with no result set

Declaration

```
QueryResult ExecuteNonQuery(Query query)
```

Parameters

TYPE	NAME	DESCRIPTION
Query	query	Custom query

Returns

TYPE	DESCRIPTION
QueryResult	Custom query affected rows and out parameters

ExecuteScalar<TResult>(Query)

Execute SQL single value result custom query

Declaration

```
TResult ExecuteScalar<TResult>(Query query)
```

Parameters

TYPE	NAME	DESCRIPTION
Query	query	Custom query

Returns

TYPE	DESCRIPTION
TResult	Custom query single value result

Type Parameters

NAME	DESCRIPTION
TResult	Result value type

FillTable(Query, CommandBehavior)

Fill DataTable with custom query results

Declaration

```
DataTable FillTable(Query query, CommandBehavior behavior = CommandBehavior.Default)
```

Parameters

TYPE	NAME	DESCRIPTION
Query	query	Custom query
System.Data.CommandBehavior	behavior	Query command behavior

Returns

TYPE	DESCRIPTION
System.Data.DataTable	DataTable

GetContext()

Returns context by name

Declaration

XDataContext GetContext()

Returns

TYPE	DESCRIPTION
XDataContext	Context

Exceptions

TYPE	CONDITION
XDataConfigurationException	XData configuration errors.
XDataAdapterException	XData adapter errors.

GetDictionaryValue<T>(Expression<Func<T, Boolean>>, String, ISecuritySession, Variable[])

Returns dictionary value by find expression and cache result

Declaration

<pre>T GetDictionaryValue<T>(Expression<Func<T, bool>> findExpression, string alias = null, ISecuritySession security = null, params Variable[] variables) where T : class, IDataObject, new()</pre>

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func<T, System.Boolean>>	findExpression	Expression to find value

TYPE	NAME	DESCRIPTION
System.String	alias	Repository alias
ISecuritySession	security	Security session
Variable[]	variables	Repository variables

Returns

TYPE	DESCRIPTION
T	Dictionary object

Type Parameters

NAME	DESCRIPTION
T	Dictionary type

Exceptions

TYPE	CONDITION
XDataRuntimeException	
XDataRuntimeException	No original data found for object
XDataRuntimeException	findExpression parameter is null

GetQuery<T>(TextReader, String, ISecuritySession)

Get query

Declaration

```
IDataQuery<T> GetQuery<T>(TextReader reader, string alias = null, ISecuritySession security = null)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
System.IO.TextReader	reader	Serialized data reader
System.String	alias	Base repository alias

TYPE	NAME	DESCRIPTION
ISecuritySession	security	Security session

Returns

TYPE	DESCRIPTION
IDataQuery<T>	Data query

Type Parameters

NAME	DESCRIPTION
T	Data object type

Exceptions

TYPE	CONDITION
XDataConfigurationException	XData configuration errors.
XDataAdapterException	XData adapter errors.
XDataAdapterException	Adapter logic error.
XDataRuntimeException	reader parameter can not be null

GetRepository<T>(TextReader)

Get repository

Declaration

```

IRepository<T> GetRepository<T>(TextReader reader)
    where T : class, IDataObject, new()

```

Parameters

TYPE	NAME	DESCRIPTION
System.IO.TextReader	reader	Serialized data reader

Returns

TYPE	DESCRIPTION

TYPE	DESCRIPTION
IRepository<T>	Repository

Type Parameters

NAME	DESCRIPTION
T	Data object type

Exceptions

TYPE	CONDITION
XDataConfigurationException	XData configuration errors.
XDataAdapterException	XData adapter errors.
XDataAdapterException	Adapter logic error.
XDataRuntimeException	reader parameter can not be null

GetRepository<T>(String, ISecuritySession, Variable[])

Get repository

Declaration

```
IRepository<T> GetRepository<T>(string alias = null, ISecuritySession security = null, params Variable[]
variables)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	alias	Data object alias
ISecuritySession	security	Security session
Variable[]	variables	Data object variables

Returns

TYPE	DESCRIPTION
IRepository<T>	Repository

Type Parameters

NAME	DESCRIPTION
T	Data object type

Exceptions

TYPE	CONDITION
XDataQueryStructureException	no mapping found.
XDataConfigurationException	XData configuration errors.
XDataAdapterException	XData adapter errors.
XDataAdapterException	Custom logic error.

GetRepository<T>(String, Variable[])

Get repository

Declaration

```
IRepository<T> GetRepository<T>(string alias = null, params Variable[] variables)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	alias	Data object alias
Variable[]	variables	Data object variables

Returns

TYPE	DESCRIPTION
IRepository<T>	Repository

Type Parameters

NAME	DESCRIPTION
T	Data object type

Exceptions

TYPE	CONDITION
XDataQueryStructureException	no mapping found.
XDataConfigurationException	XData configuration errors.
XDataAdapterException	XData adapter errors.
XDataAdapterException	Custom logic error.

GetRepository<T>(ISecuritySession, Variable[])

Get repository

Declaration

```

IRepository<T> GetRepository<T>(ISecuritySession security = null, params Variable[] variables)
    where T : class, IDataObject, new()

```

Parameters

TYPE	NAME	DESCRIPTION
ISecuritySession	security	Security session
Variable[]	variables	Data object variables

Returns

TYPE	DESCRIPTION
IRepository<T>	Repository

Type Parameters

NAME	DESCRIPTION
T	Data object type

Exceptions

TYPE	CONDITION
XDataQueryStructureException	no mapping found.
XDataConfigurationException	XData configuration errors.
XDataAdapterException	XData adapter errors.
XDataAdapterException	Custom logic error.

GetRepository<T>(Variable[])

Get repository

Declaration

```

IRepository<T> GetRepository<T>(params Variable[] variables)
    where T : class, IDataObject, new()

```

Parameters

TYPE	NAME	DESCRIPTION
Variable[]	variables	Data object variables

Returns

TYPE	DESCRIPTION
IRepository<T>	Repository

Type Parameters

NAME	DESCRIPTION
T	Data object type

Exceptions

TYPE	CONDITION
XDataQueryStructureException	no mapping found.
XDataConfigurationException	XData configuration errors.

TYPE	CONDITION
XDataAdapterException	XData adapter errors.
XDataAdapterException	Custom logic error.

Reset()

Reset data scope caches

Declaration

<code>void Reset()</code>

Extension Methods

[SerializationExtensions.ToXml<T>\(T\)](#)

[AsyncExtensions.FillTableAsync\(IDataScope, Query, CommandBehavior\)](#)

[AsyncExtensions.ExecuteScalarAsync<TResult>\(IDataScope, Query\)](#)

[AsyncExtensions.ExecuteNonQueryAsync\(IDataScope, Query\)](#)

[AsyncExtensions.ExecuteSqlBlockAsync\(IDataScope, Expression<SqlBlockDefinition>, Expression<Func<ISqlParamAdapter, ISqlParam>> > \[\]\)](#)

[AsyncExtensions.ExecuteSqlBlockAsync\(IDataScope, CancellationToken, Expression<SqlBlockDefinition>, Expression<Func<ISqlParamAdapter, ISqlParam>> > \[\]\)](#)

[AsyncExtensions.GetSqlBlockDataAsync<T>\(IDataScope, Expression<SqlBlockDefinition<T>>, Expression<Func<ISqlParamAdapter, ISqlParam>> > \[\]\)](#)

[AsyncExtensions.GetSqlBlockDataAsync<T>\(IDataScope, Func<JsonSettings<T>, JsonSettings<T>>, Expression<SqlBlockDefinition<T>>, Expression<Func<ISqlParamAdapter, ISqlParam>> > \[\]\)](#)

[CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)

[CollectionsExtensions.AsEnum<T>\(T\)](#)

[DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)

[Processing.Do<T>\(T, Action<IProcess<T>> > \[\]\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)

[TypeExtensions.ChangeType<T>\(Object\)](#)

[TypeExtensions.ChangeType\(Object, Type\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)

[SqlBlockExtensions.ExecuteSqlBlock\(IDataScope, Expression<SqlBlockDefinition>, Expression<Func<ISqlParamAdapter, ISqlParam>> > \[\]\)](#)

[SqlBlockExtensions.GetSqlBlockData<T>\(IDataScope, Expression<SqlBlockDefinition<T>>, Expression<Func<ISqlParamAdapter, ISqlParam>> > \[\]\)](#)

[SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)

[QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Delegate InitObject<T>

Object instance initialization delegate type

Namespace: [XData](#)
Assembly: XData.docfx.dll

Syntax

```
public delegate void InitObject<T>(ref T obj)
    where T : class, IDataObject;
```

Parameters

TYPE	NAME	DESCRIPTION
T	obj	Object instance

Type Parameters

NAME	DESCRIPTION
T	Object type

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

See Also

- [InitObject](#)
- [OnInitObject](#)

Delegate InitRepository<T>

Repository initialization delegate type

Namespace: [XData](#)
Assembly: XData.docfx.dll

Syntax

```
public delegate void InitRepository<T>(IRepository<T> repository)
    where T : class, IDataObject;
```

Parameters

TYPE	NAME	DESCRIPTION
IRepository<T>	repository	Repository instance

Type Parameters

NAME	DESCRIPTION
T	Object type

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

See Also

- [InitRepository](#)
- [OnInit](#)

Delegate InvalidateObject<T>

Refresh one object event delegate type

Namespace: [XData](#)
Assembly: XData.docfx.dll

Syntax

```
public delegate void InvalidateObject<in T>(T obj)
    where T : class, IDataObject;
```

Parameters

TYPE	NAME	DESCRIPTION
T	obj	Object to refresh

Type Parameters

NAME	DESCRIPTION
T	Object type

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

See Also

- [InvalidateObject](#)
- [OnInvalidateObject](#)

Delegate InvalidateRepository<T>

Refresh repository event delegate type

Namespace: [XData](#)
Assembly: XData.docfx.dll

Syntax

```
public delegate void InvalidateRepository<T>(IRepository<T> repository)
    where T : class, IDataObject;
```

Parameters

TYPE	NAME	DESCRIPTION
IRepository<T>	repository	Repository instance

Type Parameters

NAME	DESCRIPTION
T	Object type

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

See Also

- [InvalidateRepository](#)
- [OnInvalidate](#)

Interface IRepository

Data repository (data object collection) basic interface

Inherited Members

[IDataQuery.Alias](#)

[IDataQuery.Layer](#)

[IDataQuery.Context](#)

[IDataQuery.Variables](#)

[IDataQuery.Keys](#)

[IDataQuery.Token](#)

[System.Linq.IQueryable.ElementType](#)

[System.Linq.IQueryable.Expression](#)

[System.Linq.IQueryable.Provider](#)

[System.Collections.IEnumerable.GetEnumerator\(\)](#)

[System.IDisposable.Dispose\(\)](#)

Namespace: [XData](#)

Assembly: XData.docfx.dll

Syntax

```
public interface IRepository : IDataQuery, IOrderedQueryable, IQueryable, IEnumerable, IDisposable
```

Properties

Count

Get repository object count

Declaration

```
long Count { get; }
```

Property Value

TYPE	DESCRIPTION
System.Int64	

Editable

Get editable state of repository objects

Declaration

```
bool Editable { get; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

FixedQuery

Data repository fixed query

Declaration

```
bool FixedQuery { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

Remarks

Used to cache record set once (at first call) and have an access to cached data multiple times

Hint: Useful when needed to read the same data multiple times according dynamic rules and all records count approximately equal or less then requests count

Examples

```
var q = dataScope.GetRepository<Some>().Where(x => x.SomeField == 1);

q.FixedQuery = true;

q.FirstOrDefault(x => x.OtherField == 3); //Queried across the cached data
```

HasCurrentRecord

Check current record is assigned

Declaration

```
bool HasCurrentRecord { get; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

Manual

Manual repository flag

Declaration

```
bool Manual { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

Remarks

Manual repositories can be filled by attached objects only

Hint: Use manual repositories to build logic filled repositories

Hint: Use manual repositories as mock repositories for testing

Hint: Object attaching (see [Attach\(T, DataObjectState, IEnumerable<KeyValuePair<String, Object>>, AttachedHandler<T>\[\]\)](#)) set Manual property to True automatically

WARNING! Detach objects (see [Detach\(Predicate<T>\)](#)) NOT set Manual property to False automatically!

See Also

[Attach\(T, DataObjectState, IEnumerable<KeyValuePair<String, Object>>, AttachedHandler<T>\[\]\)](#)

[Detach\(Predicate<T>\)](#)

[SetAttachedHandlers\(ref T, AttachedHandler<T>\[\]\)](#)

[SetAttachedHandlers\(RepositoryAttachedHandler<T>\[\]\)](#)

ShowSubtree

Data repository tree modifier

Declaration

```
bool ShowSubtree { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

Remarks

Used to show/hide subtree related items in slave view when master is tree

When ShowSubtree = false - slave contains selected node child items only

When ShowSubtree = true - selected node and it's subtree child items

Works with [DirectLink](#): External links marked as DirectLink is filled while ShowSubtree is false and omitted otherwise

See Also

[DirectLink](#)

Methods

Clear(DataSubmitFlag)

Clear repository objects

Declaration

```
bool Clear(DataSubmitFlag flag = DataSubmitFlag.None)
```

Parameters

TYPE	NAME	DESCRIPTION
DataSubmitFlag	flag	

Returns

TYPE	DESCRIPTION
System.Boolean	

Remarks

All objects corresponded current repository limitations will be deleted through single operation

WARNING! Trigger logic [BeforeDelete](#) and [AfterDelete](#) are not used in this case!

Use [BeforeClear](#) and [AfterClear](#) instead!

CloseChilds()

Close child repositories

Declaration

```
void CloseChilds()
```

DetachChild<TChild>(String)

Detach child repository

Declaration

```
void DetachChild<TChild>(string alias = null)
    where TChild : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	alias	

Type Parameters

NAME	DESCRIPTION
TChild	

FindParent<TParent>(out IRepository<TParent>, String)

Find parent repository

Declaration

```
bool FindParent<TParent>(out IRepository<TParent> parent, string alias = null)
    where TParent : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
IRepository <TParent>	parent	Parent repository
System.String	alias	Parent repository alias

Returns

TYPE	DESCRIPTION

TYPE	DESCRIPTION
System.Boolean	Find success

Type Parameters

NAME	DESCRIPTION
TParent	Parent repository type

GetChild<TChild>(String, ISecuritySession, Variable[])

Get a child repository

Declaration

```

IRepository<TChild> GetChild<TChild>(string alias = null, ISecuritySession security = null, params Variable[]
variables)
    where TChild : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	alias	Child alias
ISecuritySession	security	Security session
Variable[]	variables	Child variables

Returns

TYPE	DESCRIPTION
IRepository <TChild>	Repository

Type Parameters

NAME	DESCRIPTION
TChild	Child type

Remarks

If child repository not exists it will be created and linked to base repository as a child

When alias parameter value is not set child object type name used as child repository alias

See Also

[DetachChild<TChild>\(String\)](#)

[CloseChilds\(\)](#)

Refresh()

Refresh repository data cache

Declaration

```
void Refresh()
```

Extension Methods

[DataObjectExtensions.GetParameter<T>\(IDataQuery, String\)](#)

[DataObjectExtensions.GetResultSet<TRes>\(IDataQuery, String\)](#)

[DataObjectExtensions.GetResultSet<TRes>\(IDataQuery, Expression<Func<IEnumerable<TRes>>>\)](#)

[SerializationExtensions.ToXml<T>\(T\)](#)

[AsyncExtensions.GetParameterAsync<T>\(IDataQuery, String\)](#)

[AsyncExtensions.GetResultSetAsync<TRes>\(IDataQuery, String\)](#)

[AsyncExtensions.GetResultSetAsync<TRes>\(IDataQuery, Expression<Func<IEnumerable<TRes>>>\)](#)

[CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)

[CollectionsExtensions.AsEnum<T>\(T\)](#)

[DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)

[Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)

[TypeExtensions.ChangeType<T>\(Object\)](#)

[TypeExtensions.ChangeType\(Object, Type\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)

[SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)

[QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

See Also

[IRepository<T>](#)

Interface IRepository<T>

Data repository (data object collection) type specific interface

Inherited Members

- [IDataQuery<T>.GetFilterValue\(RuntimeFilter\)](#)
- [System.Collections.Generic.IEnumerable<T>.GetEnumerator\(\)](#)
- [IRepository.ShowSubtree](#)
- [IRepository.FixedQuery](#)
- [IRepository.DetachChild<TChild>\(String\)](#)
- [IRepository.Clear\(DataSubmitFlag\)](#)
- [IRepository.CloseChilds\(\)](#)
- [IRepository.Refresh\(\)](#)
- [IRepository.GetChild<TChild>\(String, ISecuritySession, Variable\[\]\)](#)
- [IRepository.FindParent<TParent>\(IRepository<TParent>, String\)](#)
- [IRepository.Count](#)
- [IRepository.Editable](#)
- [IRepository.HasCurrentRecord](#)
- [IRepository.Manual](#)
- [IDataQuery.Alias](#)
- [IDataQuery.Layer](#)
- [IDataQuery.Context](#)
- [IDataQuery.Variables](#)
- [IDataQuery.Keys](#)
- [IDataQuery.Token](#)
- [System.Linq.IQueryable.ElementType](#)
- [System.Linq.IQueryable.Expression](#)
- [System.Linq.IQueryable.Provider](#)
- [System.IDisposable.Dispose\(\)](#)

Namespace: [XData](#)
Assembly: XData.docfx.dll

Syntax

```
public interface IRepository<T> : IDataQuery<T>, IOrderedQueryable<T>, IQueryable<T>, IEnumerable<T>, IRepository, IDataQuery, IOrderedQueryable, IQueryable, IEnumerable, IDisposable where T : class, IDataObject
```

Type Parameters

NAME	DESCRIPTION
T	

Remarks

Hint: Its possible to use any DI container with XData:

```
// example with Autofac syntax

// register

Container.Register(() => dataScope.GetRepository<Invoice>()).As<IRepository<Invoice>>();

Container.Register(() => dataScope.GetRepository<Invoice>().GetChild<InvoiceSpec>
()).As<IRepository<InvoiceSpec>>();

// resolve

var spec = Container.Resolve<IRepository<InvoiceSpec>>(); //spec is linked with Invoice repository as a
slave!
```

Properties

Current

Get/set current data object

Declaration

```
T Current { get; set; }
```

Property Value

TYPE	DESCRIPTION
T	Current data object instance

Exceptions

TYPE	CONDITION
XDataQueryStructureException	object not mapped properly.

Methods

Add<TFrom>(IEnumerable<TFrom>, Mapping<TFrom, T>, DataSubmitFlag)

Copy objects from source repository

Declaration

```
bool Add<TFrom>(IEnumerable<TFrom> source, Mapping<TFrom, T> mapping = null, DataSubmitFlag flag =
DataSubmitFlag.None)
    where TFrom : class
```

Parameters

TYPE	NAME	DESCRIPTION
System.Collections.Generic.IEnumerable<TFrom>	source	Source repository
Mapping <TFrom, T>	mapping	Data mapping
DataSubmitFlag	flag	Submit flag

Returns

TYPE	DESCRIPTION
System.Boolean	Success flag

Type Parameters

NAME	DESCRIPTION
TFrom	

Exceptions

TYPE	CONDITION
XDataRuntimeException	"source" parameter is null.
XDataQueryStructureException	mapping error.

See Also

[Mapping](#)<TFrom, T>

Attach(T, DataObjectState, IEnumerable<KeyValuePair<String, Object>>, AttachedHandler<T>[])

Attach object to repository

Declaration

```

IRepository<T> Attach(T obj, DataObjectState state = DataObjectState.Original,
IEnumerable<KeyValuePair<string, object>> originalValues = null, params AttachedHandler<T>[] handlers)

```

Parameters

TYPE	NAME	DESCRIPTION
T	obj	Object instance
DataObjectState	state	Object state

TYPE	NAME	DESCRIPTION
System.Collections.Generic.IEnumerable<System.Collections.Generic.KeyValuePair<System.String, System.Object>>	originalValues	Original property values
AttachedHandler<T>[]	handlers	Attached handlers

Returns

TYPE	DESCRIPTION
IRepository<T>	Repository

Remarks

- Hint: Object attaching set [Manual](#) property to True automatically
- Hint: The originalValues parameter can contain only necessary not null valued properties
- Hint: Use [SetValue<TKey, T>\(TKey, T\)](#) to fill originalValues parameter
- Hint: Use [AttachedHandler<T>](#) convert operators to attach object specific handlers

```
(SubmitObjectAttachedHandler<Invoice>)((ref Invoice i, DataSubmitFlag f) => true)
```

WARNING! Use repository [SetAttachedHandlers\(RepositoryAttachedHandler<T>\[\]\)](#) method to apply same handler for all objects!

Exceptions

TYPE	CONDITION
XDataQueryStructureException	no DataObjectAttribute found in data object mapping.
XDataRuntimeExcepion	type loading error.

See Also

- [SetValue<TKey, T>\(TKey, T\)](#)
- [Detach\(Predicate<T>\)](#)

AttachNewObjects(IEnumerable<T>, AttachedHandler<T>[])

Attach object to repository

Declaration

```
IEnumerable<T> AttachNewObjects(IEnumerable<T> source, params AttachedHandler<T>[] handlers)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Collections.Generic.IEnumerable<T>	source	Object sequence
AttachedHandler <T>[]	handlers	Attached handlers

Returns

TYPE	DESCRIPTION
System.Collections.Generic.IEnumerable<T>	Repository

Remarks

Hint: Object attaching set [Manual](#) property to True automatically

Hint: The originalValues parameter can contain only necessary not null valued properties

Hint: Use [SetValue<TKey, T>\(TKey, T\)](#) to fill originalValues parameter

Hint: Use [AttachedHandler<T>](#) convert operators to attach object specific handlers

```
(SubmitObjectAttachedHandler<Invoice>)((ref Invoice i, DataSubmitFlag f) => true)
```

WARNING! Use repository [SetAttachedHandlers\(RepositoryAttachedHandler<T>\[\]\)](#) method to apply same handler for all objects!

Exceptions

TYPE	CONDITION
XDataQueryStructureException	no DataObjectAttribute found in data object mapping.
XDataRuntimeException	type loading error.

See Also

[SetValue<TKey, T>\(TKey, T\)](#)

[Detach\(Predicate<T>\)](#)

[Clear\(Expression<Func<T, Boolean>>, DataSubmitFlag\)](#)

Clear repository objects

Declaration

```
bool Clear(Expression<Func<T, bool>> predicate, DataSubmitFlag flag = DataSubmitFlag.None)
```

Parameters

TYPE	NAME	DESCRIPTION

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func<T, System.Boolean>>	predicate	
DataSubmitFlag	flag	

Returns

TYPE	DESCRIPTION
System.Boolean	

Remarks

All objects corresponded current repository limitations will be deleted through single operation

WARNING! Trigger logic [BeforeDelete](#) and [AfterDelete](#) are not used in this case!

Use [BeforeClear](#) and [AfterClear](#) instead!

Exceptions

TYPE	CONDITION
XDataQueryStructureException	no DataObjectAttribute found in data object mapping.
XDataRuntimeException	type loading error.

ClearAttachedObjects()

Clear attached objects

Declaration

<pre> IRepository<T> ClearAttachedObjects()</pre>

Returns

TYPE	DESCRIPTION
IRepository <T>	Repository

Remarks

WARNING! Clear attached objects NOT set [Manual](#) property to False automatically!

Exceptions

TYPE	CONDITION
XDataQueryStructureException	no DataObjectAttribute found in data object mapping.
XDataRuntimeException	type loading error.

Detach(Predicate<T>)

Detach virtual objects from repository

Declaration

```
 IRepository<T> Detach(Predicate<T> predicate)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Predicate<T>	predicate	Object search predicate to detach

Returns

TYPE	DESCRIPTION
IRepository<T>	Repository

Exceptions

TYPE	CONDITION
XDataQueryStructureException	no DataObjectAttribute found in data object mapping.
XDataRuntimeException	type loading error.

See Also

[Attach\(T, DataObjectState, IEnumerable<KeyValuePair<String, Object>>, AttachedHandler<T>\[\]\)](#)
[Detach\(Predicate<T>\)](#)

Execute(String, IEnumerable<T>, IDictionary<String, Action<Byte[]>>, IDictionary<String, Func<Byte[], Byte[]>>)

Run custom logic over object instances

Declaration

```
bool Execute(string key, IEnumerable<T> objects, IDictionary<string, Action<byte[]>> post = null, IDictionary<string, Func<byte[], byte[]>> callback = null)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	key	Logic key
System.Collections.Generic.IEnumerable<T>	objects	Object instances
System.Collections.Generic.IDictionary<System.String, System.Action< Byte []>>	post	Post data handler

TYPE	NAME	DESCRIPTION
System.Collections.Generic.IDictionary<System.String, System.Func<Byte[], Byte[]>>	callback	Callback handler

Returns

TYPE	DESCRIPTION
System.Boolean	Success flag

Exceptions

TYPE	CONDITION
XDataQueryStructureException	no DataObjectAttribute found in data object mapping.
XDataRuntimeException	type loading error.

See Also

[CustomLogic<T>](#)
[Execute<T>](#)(T, System.Linq.Expressions.Expression<System.Func<[CustomLogic<T>](#)>>>, System.Collections.Generic.IDictionary<System.String, System.Action<[Byte\[\]](#)>>>, System.Collections.Generic.IDictionary<System.String, System.Func<[Byte\[\]](#), [Byte\[\]](#)>>>) [Execute<T>](#)(ICollection<T>, Expression<Func<[CustomLogic<T>](#)>>>, IDictionary<String, Action<[Byte\[\]](#)>>>, IDictionary<String, Func<[Byte\[\]](#), [Byte\[\]](#)>>>)

GetChild<TChild>(Expression<Func<T, TChild[]>>>, String, ISecuritySession, Variable[])

Get a child repository

Declaration

```

IRepository<TChild> GetChild<TChild>(Expression<Func<T, TChild[]>> property, string alias = null,
ISecuritySession security = null, params Variable[] variables)
    where TChild : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func<T, TChild[]>>	property	Property to store child entities
System.String	alias	Child alias
ISecuritySession	security	Security session
Variable[]	variables	Child variables

Returns

TYPE	DESCRIPTION
IRepository<TChild>	Repository

Type Parameters

NAME	DESCRIPTION
TChild	Child type

Remarks

If child repository not exists it will be created and linked to base repository as a child

When alias parameter value is not set child object type name used as child repository alias

Exceptions

TYPE	CONDITION
XDataQueryStructureException	no DataObjectAttribute found in data object mapping.
XDataRuntimeExcpetion	type loading error.

See Also

[DetachChild<TChild>\(System.String\)](#)

[CloseChilds\(\)](#)

[GetChild<TChild>\(Expression<Func<T, TChild>>, String, ISecuritySession, Variable\[\]\)](#)

Get a child repository

Declaration

<pre>IRepository<TChild> GetChild<TChild>(Expression<Func<T, TChild>> property, string alias = null, ISecuritySession security = null, params Variable[] variables) where TChild : class, IDataObject, new()</pre>

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func<T, TChild>>	property	Property to store child entity (1-to-1)
System.String	alias	Child alias
ISecuritySession	security	Security session

TYPE	NAME	DESCRIPTION
Variable[]	variables	Child variables

Returns

TYPE	DESCRIPTION
IRepository<TChild>	Repository

Type Parameters

NAME	DESCRIPTION
TChild	Child type

Remarks

If child repository not exists it will be created and linked to base repository as a child

When alias parameter value is not set child object type name used as child repository alias

Exceptions

TYPE	CONDITION
XDataQueryStructureException	no DataObjectAttribute found in data object mapping.
XDataRuntimeException	type loading error.

See Also

[DetachChild<TChild>\(System.String\)](#)

[CloseChilds\(\)](#)

Lock(T)

Lock object in database

Declaration

bool Lock(T obj)

Parameters

TYPE	NAME	DESCRIPTION
T	obj	Data object instance

Returns

TYPE	DESCRIPTION
System.Boolean	Success flag

Exceptions

TYPE	CONDITION
XDataRuntimeException	"obj" parameter is null.

See Also

[Lock<T>\(T\)](#)

New()

Get new instance of data object

Declaration

T New()

Returns

TYPE	DESCRIPTION
T	Data object instance

Exceptions

TYPE	CONDITION
XDataQueryStructureException	object not mapped properly.

New(Action<T>[])

Get new instance of data object

Declaration

T New(params Action<T>[] init)

Parameters

TYPE	NAME	DESCRIPTION
System.Action<T>[]	init	Data object initialization actions

Returns

TYPE	DESCRIPTION
T	Data object instance

Exceptions

TYPE	CONDITION
XDataQueryStructureException	object not mapped properly.

Refresh(ref T)

Refresh object data from database

Declaration

bool Refresh(ref T obj)

Parameters

TYPE	NAME	DESCRIPTION
T	obj	Data object instance

Returns

TYPE	DESCRIPTION
System.Boolean	Success flag

Exceptions

TYPE	CONDITION
XDataRuntimeException	"obj" parameter is null.
XDataQueryStructureException	object not mapped properly.

See Also

[Refresh<T>\(T\)](#)

Reset()

Reset repository structure and filter values

Declaration

IRepository<T> Reset()

Returns

TYPE	DESCRIPTION
IRepository<T>	Self repository link

Exceptions

TYPE	CONDITION
XDataQueryStructureException	no DataObjectAttribute found in data object mapping.
XDataRuntimeException	type loading error.

See Also

[RuntimeFilter](#)
[GetFilterValue\(RuntimeFilter\)](#)
[SetFilterValue\(RuntimeFilter, Object\)](#)

SetAttachedHandlers(ref T, AttachedHandler<T>[])

Attach extra handlers to single object

Declaration

```
void SetAttachedHandlers(ref T obj, params AttachedHandler<T>[] handlers)
```

Parameters

TYPE	NAME	DESCRIPTION
T	obj	Object instance
AttachedHandler<T>[]	handlers	Attached handlers

Remarks

Hint: Use [AttachedHandler<T>](#) convert operators to attach object specific handlers

```
(SubmitObjectAttachedHandler<Invoice>)((ref Invoice i, DataSubmitFlag f) => true)
```

WARNING! Use repository [SetAttachedHandlers\(RepositoryAttachedHandler<T>\[\]\)](#) method to apply same handler for all objects!

Exceptions

TYPE	CONDITION
XDataQueryStructureException	no DataObjectAttribute found in data object mapping.
XDataRuntimeException	type loading error.

See Also

[Attach\(T, DataObjectState, IEnumerable<KeyValuePair<String, Object>>, AttachedHandler<T>\[\]\)](#)
[SetAttachedHandlers\(RepositoryAttachedHandler<T>\[\]\)](#)
[SetAttachedHandlers<T>\(T, AttachedHandler<T>\[\]\)](#)

SetAttachedHandlers(RepositoryAttachedHandler<T>[])

Attach extra handlers to repository

Declaration

```
IRepository<T> SetAttachedHandlers(params RepositoryAttachedHandler<T>[] handlers)
```

Parameters

TYPE	NAME	DESCRIPTION
RepositoryAttachedHandler<T>[]	handlers	Attached handlers

Returns

TYPE	DESCRIPTION
IRepository<T>	Repository

Remarks

Hint: Use [RepositoryAttachedHandler<T>](#) convert operators to attach object specific handlers

```
(SubmitObjectAttachedHandler<Invoice>)((ref Invoice i, DataSubmitFlag f) => true)
```

Exceptions

TYPE	CONDITION
XDataQueryStructureException	no DataObjectAttribute found in data object mapping.
XDataRuntimeException	type loading error.

SetFilterValue(RuntimeFilter, Object)

Set metadata defined RuntimeFilter value

Declaration

```
IRepository<T> SetFilterValue(RuntimeFilter filter, object val)
```

Parameters

TYPE	NAME	DESCRIPTION
RuntimeFilter	filter	Filter
System.Object	val	Filter value

Returns

TYPE	DESCRIPTION
IRepository<T>	Self repository link

Remarks

Hint: Set `val = null` to clear filter

Exceptions

TYPE	CONDITION
XDataRuntimeException	"filter" parameter is null.

See Also

[RuntimeFilter](#)
[Reset\(\)](#)
[GetFilterValue\(RuntimeFilter\)](#)

Submit(ref T, DataSubmitFlag)

Save single data object changes

Declaration

```
bool Submit(ref T obj, DataSubmitFlag flag = DataSubmitFlag.None)
```

Parameters

TYPE	NAME	DESCRIPTION
T	obj	Data object instance
DataSubmitFlag	flag	Submit flag

Returns

TYPE	DESCRIPTION
System.Boolean	Success flag

Exceptions

TYPE	CONDITION
XDataRuntimeException	"objects" is null.

See Also

[Submit<T>\(T, DataSubmitFlag\)](#)

Submit(IEnumerable<T>, DataSubmitFlag)

Save multiple data objects changes

Declaration

```
bool Submit(IEnumerable<T> objects, DataSubmitFlag flag = DataSubmitFlag.None)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Collections.Generic.IEnumerable<T>	objects	Data object sequence
DataSubmitFlag	flag	Submit flag

Returns

TYPE	DESCRIPTION
System.Boolean	Success flag

Exceptions

TYPE	CONDITION
XDataRuntimeException	"objects" is null or empty sequence.

See Also

[DataSubmitFlag](#)

Events

OnCurrentObjectChanged

Event fired when repository current object is changed

Declaration

```
event CurrentObjectChanged<T> OnCurrentObjectChanged
```

Event Type

TYPE	DESCRIPTION
CurrentObjectChanged<T>	

Exceptions

TYPE	CONDITION
XDataRuntimeException	custom logic fires exception.

OnCurrentObjectChanging

Event fired when repository current object changing occurred

Declaration

event CurrentObjectChanging<T> OnCurrentObjectChanging

Event Type

TYPE	DESCRIPTION
CurrentObjectChanging<T>	

Exceptions

TYPE	CONDITION
XDataRuntimeException	custom logic fires exception.

OnInit

Event fired when repository is initialized

Declaration

event InitRepository<T> OnInit

Event Type

TYPE	DESCRIPTION
InitRepository<T>	

Exceptions

TYPE	CONDITION
XDataRuntimeException	custom logic fires exception.

OnInitObject

Event fired when object is initialized

Declaration

event InitObject<T> OnInitObject

Event Type

TYPE	DESCRIPTION
InitObject<T>	

Exceptions

TYPE	CONDITION
XDataRuntimeException	custom logic fires exception.

OnInvalidate

Event fired when repository need to be refreshed

Declaration

```
event InvalidateRepository<T> OnInvalidate
```

Event Type

TYPE	DESCRIPTION
InvalidateRepository<T>	

Exceptions

TYPE	CONDITION
XDataRuntimeException	custom logic fires exception.

OnInvalidateObject

Event fired when object need to be refreshed

Declaration

```
event InvalidateObject<T> OnInvalidateObject
```

Event Type

TYPE	DESCRIPTION
InvalidateObject<T>	

Exceptions

TYPE	CONDITION
XDataRuntimeException	custom logic fires exception.

Extension Methods

- DataObjectExtensions.AsString<T>(IQueryable<T>)
- DataObjectExtensions.GetParameter<T>(IDataQuery, String)
- DataObjectExtensions.GetResultSet<TRes>(IDataQuery, String)
- DataObjectExtensions.GetResultSet<TRes>(IDataQuery, Expression<Func<IEnumerable<TRes>>>)
- SerializationExtensions.ToXml<T>(T)
- AsyncExtensions.AllAsync<T>(IQueryable<T>, Expression<Func<T, Boolean>>)
- AsyncExtensions.AllAsync<T>(IQueryable<T>, Expression<Func<T, Boolean>>, CancellationToken)
- AsyncExtensions.AllAsync<T>(IDataQuery<T>, Expression<Func<T, Boolean>>)
- AsyncExtensions.AnyAsync<T>(IQueryable<T>)
- AsyncExtensions.AnyAsync<T>(IQueryable<T>, Expression<Func<T, Boolean>>)
- AsyncExtensions.AnyAsync<T>(IQueryable<T>, Expression<Func<T, Boolean>>, CancellationToken)
- AsyncExtensions.AnyAsync<T>(IDataQuery<T>, Expression<Func<T, Boolean>>)
- AsyncExtensions.AverageAsync<T>(IQueryable<T>, Expression<Func<T, Single>>)
- AsyncExtensions.AverageAsync<T>(IQueryable<T>, Expression<Func<T, Single>>, CancellationToken)

[illegible]

[illegible]

AsyncExtensions.SumAsync<T>(IQueryable<T>, Expression<Func<T, Nullable<Single>>>, CancellationToken)
AsyncExtensions.SumAsync<T>(IDataQuery<T>, Expression<Func<T, Nullable<Single>>>)
AsyncExtensions.SumAsync<T>(IQueryable<T>, Expression<Func<T, Nullable<Double>>>)
AsyncExtensions.SumAsync<T>(IQueryable<T>, Expression<Func<T, Nullable<Double>>>, CancellationToken)
AsyncExtensions.SumAsync<T>(IDataQuery<T>, Expression<Func<T, Nullable<Double>>>)
AsyncExtensions.SumAsync<T>(IQueryable<T>, Expression<Func<T, Nullable<Decimal>>>)
AsyncExtensions.SumAsync<T>(IQueryable<T>, Expression<Func<T, Nullable<Decimal>>>, CancellationToken)
AsyncExtensions.SumAsync<T>(IDataQuery<T>, Expression<Func<T, Nullable<Decimal>>>)
AsyncExtensions.SumAsync<T>(IQueryable<T>, Expression<Func<T, Nullable<Int32>>>)
AsyncExtensions.SumAsync<T>(IQueryable<T>, Expression<Func<T, Nullable<Int32>>>, CancellationToken)
AsyncExtensions.SumAsync<T>(IDataQuery<T>, Expression<Func<T, Nullable<Int32>>>)
AsyncExtensions.SumAsync<T>(IQueryable<T>, Expression<Func<T, Nullable<Int64>>>)
AsyncExtensions.SumAsync<T>(IQueryable<T>, Expression<Func<T, Nullable<Int64>>>, CancellationToken)
AsyncExtensions.SumAsync<T>(IDataQuery<T>, Expression<Func<T, Nullable<Int64>>>)
AsyncExtensions.ToArrayAsync<T>(IQueryable<T>)
AsyncExtensions.ToArrayAsync<T>(IQueryable<T>, CancellationToken)
AsyncExtensions.ToArrayAsync<T>(IDataQuery<T>)
AsyncExtensions.ToDictionaryAsync<T, TKey, TValue>(IQueryable<T>, Func<T, TKey>, Func<T, TValue>)
AsyncExtensions.ToDictionaryAsync<T, TKey, TValue>(IQueryable<T>, Func<T, TKey>, Func<T, TValue>, CancellationToken)
AsyncExtensions.ToDictionaryAsync<T, TKey, TValue>(IDataQuery<T>, Func<T, TKey>, Func<T, TValue>)
AsyncExtensions.ToListAsync<T>(IQueryable<T>)
AsyncExtensions.ToListAsync<T>(IQueryable<T>, CancellationToken)
AsyncExtensions.ToListAsync<T>(IDataQuery<T>)
CollectionsExtensions.ToSortedSet<T>(IEnumerable<T>, IComparer<T>)
CollectionsExtensions.SetValue<TKey, T>(TKey, T)
CollectionsExtensions.AsEnum<T>(T)
DataObjectQueryableExtensions.CompareVariable(Object, FilterOperation, String)
JoinedEnumerable.Inner<TElement>(IEnumerable<TElement>)
JoinedEnumerable.Outer<TElement>(IEnumerable<TElement>)
JoinedEnumerable.LeftOuterJoin<TOuter, TInner, TKey, TResult>(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter, TKey>, Func<TInner, TKey>, Func<TOuter, TInner, TResult>, IEqualityComparer<TKey>)
JoinedEnumerable.RightOuterJoin<TOuter, TInner, TKey, TResult>(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter, TKey>, Func<TInner, TKey>, Func<TOuter, TInner, TResult>, IEqualityComparer<TKey>)
JoinedEnumerable.FullOuterJoin<TOuter, TInner, TKey, TResult>(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter, TKey>, Func<TInner, TKey>, Func<TOuter, TInner, TResult>, IEqualityComparer<TKey>)
Processing.Do<T>(T, Action<IProcess<T>>[])
TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>)
TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>, TResult)
TypeExtensions.ChangeType<T>(Object)
TypeExtensions.ChangeType(Object, Type)
TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)
TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)
TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)
TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)
SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)
QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)
QueryDescriptionExtensions.ToDataSet<T>(IEnumerable<T>, String, ISqlBuilder)

See Also

[IDataObject](#)

[AsString<T>\(System.Linq.IQueryable<T>\)](#)

[GetRepository<T>\(T\)](#)

GetRepository<T>(String, ISecuritySession, Variable[])

Interface ISqlObject

SQL (sub)query part interface

Namespace: [XData](#)
Assembly: [XData.docfx.dll](#)

Syntax

```
public interface ISqlObject
```

Remarks

Mapping object as subquery

Hint: Use abstract classes to map subquery

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.GetProperty\(ISqlObject, String\)](#)

Interface IWorkScope<T>

Work set export/apply child object structure descriptor interface

Namespace: [XData](#)
Assembly: XData.WorkSet.docfx.dll

Syntax

```
public interface IWorkScope<T>
```

Type Parameters

NAME	DESCRIPTION
T	Parent object type

Methods

With<TChild>(Expression<Func<T, TChild[]>>, Expression<Func<TChild, Guid>>)

Work set export/apply child object structure descriptor

Declaration

```
IWorkScope<T> With<TChild>(Expression<Func<T, TChild[]>> childProperty, Expression<Func<TChild, Guid>> keyProperty)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func<T, TChild[]>>	childProperty	Child array property
System.Linq.Expressions.Expression<System.Func<TChild, System.Guid>>	keyProperty	Child work set key property

Returns

TYPE	DESCRIPTION
IWorkScope<T>	Structure descriptor

Type Parameters

NAME	DESCRIPTION
TChild	Child object type

With<TChild>(Expression<Func<T, TChild[]>>, Expression<Func<TChild, Guid>>, Expression<Func<TChild, Boolean>>)

Work set export/apply child object structure descriptor

Declaration

`IWorkScope<T> With<TChild>(Expression<Func<T, TChild[]>> childProperty, Expression<Func<TChild, Guid>> keyProperty, Expression<Func<TChild, bool>> deletedProperty)`

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func<T, TChild[]>>	childProperty	Child array property
System.Linq.Expressions.Expression<System.Func<TChild, System.Guid>>	keyProperty	Child work set key property
System.Linq.Expressions.Expression<System.Func<TChild, System.Boolean>>	deletedProperty	Child deleted flag property

Returns

TYPE	DESCRIPTION
IWorkScope<T>	Structure descriptor

Type Parameters

NAME	DESCRIPTION
TChild	Child object type

`With<TChild>(Expression<Func<T, TChild[]>>, Expression<Func<TChild, Guid>>, Expression<Func<TChild, Boolean>>, Expression<Func<IWorkScope<TChild>, IWorkScope<TChild>>>)`

Work set export/apply child object structure descriptor

Declaration

`IWorkScope<T> With<TChild>(Expression<Func<T, TChild[]>> childProperty, Expression<Func<TChild, Guid>> keyProperty, Expression<Func<TChild, bool>> deletedProperty, Expression<Func<IWorkScope<TChild>, IWorkScope<TChild>>> childs)`

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func<T, TChild[]>>	childProperty	Child array property
System.Linq.Expressions.Expression<System.Func<TChild, System.Guid>>	keyProperty	Child work set key property
System.Linq.Expressions.Expression<System.Func<TChild, System.Boolean>>	deletedProperty	Child deleted flag property

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func<IWorkScope<TChild>, IWorkScope<TChild>>>	childs	Childs structure descriptor

Returns

TYPE	DESCRIPTION
IWorkScope<T>	Structure descriptor

Type Parameters

NAME	DESCRIPTION
TChild	Child object type

With<TChild>(Expression<Func<T, TChild[]>>, Expression<Func<TChild, Guid>>, Expression<Func<TChild, Boolean>>, String)

Work set export/apply child object structure descriptor

Declaration

```
IWorkScope<T> With<TChild>(Expression<Func<T, TChild[]>> childProperty, Expression<Func<TChild, Guid>>
keyProperty, Expression<Func<TChild, bool>> deletedProperty, string alias)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func<T, TChild[]>>	childProperty	Child array property
System.Linq.Expressions.Expression<System.Func<TChild, System.Guid>>	keyProperty	Child work set key property
System.Linq.Expressions.Expression<System.Func<TChild, System.Boolean>>	deletedProperty	Child deleted flag property
System.String	alias	Child repository alias

Returns

TYPE	DESCRIPTION
IWorkScope<T>	Structure descriptor

Type Parameters

NAME	DESCRIPTION
TChild	Child object type

With<TChild>(Expression<Func<T, TChild[]>>, Expression<Func<TChild, Guid>>, Expression<Func<TChild, Boolean>>, String, Expression<Func<IWorkScope<TChild>, IWorkScope<TChild>>>)

Work set export/apply child object structure descriptor

Declaration

```
IWorkScope<T> With<TChild>(Expression<Func<T, TChild[]>> childProperty, Expression<Func<TChild, Guid>>
keyProperty, Expression<Func<TChild, bool>> deletedProperty, string alias, Expression<Func<IWorkScope<TChild>,
IWorkScope<TChild>>> childs)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func<T, TChild[]> >	childProperty	Child array property
System.Linq.Expressions.Expression<System.Func<TChild, System.Guid> >	keyProperty	Child work set key property
System.Linq.Expressions.Expression<System.Func<TChild, System.Boolean> >	deletedProperty	Child deleted flag property
System.String	alias	Child repository alias
System.Linq.Expressions.Expression<System.Func<IWorkScope<TChild>, IWorkScope<TChild>>>	childs	Childs structure descriptor

Returns

TYPE	DESCRIPTION
IWorkScope<T>	Structure descriptor

Type Parameters

NAME	DESCRIPTION
TChild	Child object type

With<TChild>(Expression<Func<T, TChild[]>>, Expression<Func<TChild, Guid>>, Expression<Func<IWorkScope<TChild>, IWorkScope<TChild>>>)

Work set export/apply child object structure descriptor

Declaration

```
IWorkScope<T> With<TChild>(Expression<Func<T, TChild[]>> childProperty, Expression<Func<TChild, Guid>>
keyProperty, Expression<Func<IWorkScope<TChild>, IWorkScope<TChild>>> childs)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func<T, TChild[]> >	childProperty	Child array property
System.Linq.Expressions.Expression<System.Func<TChild, System.Guid> >	keyProperty	Child work set key property
System.Linq.Expressions.Expression<System.Func<IWorkScope<TChild>, IWorkScope<TChild>>> >	childs	Childs structure descriptor

Returns

TYPE	DESCRIPTION
IWorkScope<T>	Structure descriptor

Type Parameters

NAME	DESCRIPTION
TChild	Child object type

With<TChild>(Expression<Func<T, TChild[]>>, Expression<Func<TChild, Guid>>, String)

Work set export/apply child object structure descriptor

Declaration

```
IWorkScope<T> With<TChild>(Expression<Func<T, TChild[]>> childProperty, Expression<Func<TChild, Guid>>
keyProperty, string alias)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func<T, TChild[]> >	childProperty	Child array property
System.Linq.Expressions.Expression<System.Func<TChild, System.Guid> >	keyProperty	Child work set key property
System.String	alias	Child repository alias

Returns

TYPE	DESCRIPTION
IWorkScope<T>	Structure descriptor

Type Parameters

NAME	DESCRIPTION
TChild	Child object type

With<TChild>(Expression<Func<T, TChild[]>>, Expression<Func<TChild, Guid>>, String, Expression<Func<IWorkScope<TChild>, IWorkScope<TChild>>>)

Work set export/apply child object structure descriptor

Declaration

IWorkScope<T> With<TChild>(Expression<Func<T, TChild[]>> childProperty, Expression<Func<TChild, Guid>> keyProperty, string alias, Expression<Func<IWorkScope<TChild>, IWorkScope<TChild>>> childs)

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func<T, TChild[]>>	childProperty	Child array property
System.Linq.Expressions.Expression<System.Func<TChild, System.Guid>>	keyProperty	Child work set key property
System.String	alias	Child repository alias
System.Linq.Expressions.Expression<System.Func< IWorkScope<TChild> , IWorkScope<TChild> >>>	childs	Childs structure descriptor

Returns

TYPE	DESCRIPTION
IWorkScope<T>	Structure descriptor

Type Parameters

NAME	DESCRIPTION
TChild	Child object type

With<TChild>(Expression<Func<T, TChild>>, Expression<Func<TChild, Guid>>)

Work set export/apply child object structure descriptor

Work set export/apply child object structure descriptor

Declaration

```
IWorkScope<T> With<TChild>(Expression<Func<T, TChild>> childProperty, Expression<Func<TChild, Guid>> keyProperty)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func<T, TChild> >	childProperty	Single child property
System.Linq.Expressions.Expression<System.Func<TChild, System.Guid> >	keyProperty	Child work set key property

Returns

TYPE	DESCRIPTION
IWorkScope<T>	Structure descriptor

Type Parameters

NAME	DESCRIPTION
TChild	Child object type

With<TChild>(Expression<Func<T, TChild>>, Expression<Func<TChild, Guid>>, Expression<Func<TChild, Boolean>>)

Work set export/apply child object structure descriptor

Declaration

```
IWorkScope<T> With<TChild>(Expression<Func<T, TChild>> childProperty, Expression<Func<TChild, Guid>> keyProperty, Expression<Func<TChild, bool>> deletedProperty)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func<T, TChild> >	childProperty	Single child property
System.Linq.Expressions.Expression<System.Func<TChild, System.Guid> >	keyProperty	Child work set key property
System.Linq.Expressions.Expression<System.Func<TChild, System.Boolean> >	deletedProperty	Child deleted flag property

Returns

TYPE	DESCRIPTION
IWorkScope<T>	Structure descriptor

Type Parameters

NAME	DESCRIPTION
TChild	Child object type

With<TChild>(Expression<Func<T, TChild>>, Expression<Func<TChild, Guid>>, Expression<Func<TChild, Boolean>>, Expression<Func<IWorkScope<TChild>, IWorkScope<TChild>>>>)

Work set export/apply child object structure descriptor

Declaration

```
IWorkScope<T> With<TChild>(Expression<Func<T, TChild>> childProperty, Expression<Func<TChild, Guid>>
keyProperty, Expression<Func<TChild, bool>> deletedProperty, Expression<Func<IWorkScope<TChild>,
IWorkScope<TChild>>> childs)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func<T, TChild>>	childProperty	Single child property
System.Linq.Expressions.Expression<System.Func<TChild, System.Guid>>	keyProperty	Child work set key property
System.Linq.Expressions.Expression<System.Func<TChild, System.Boolean>>	deletedProperty	Child deleted flag property
System.Linq.Expressions.Expression<System.Func<IWorkScope<TChild>, IWorkScope<TChild>>>	childs	Childs structure descriptor

Returns

TYPE	DESCRIPTION
IWorkScope<T>	Structure descriptor

Type Parameters

NAME	DESCRIPTION
TChild	Child object type

With<TChild>(Expression<Func<T, TChild>>, Expression<Func<TChild, Guid>>, Expression<Func<TChild, Boolean>>, String)

Work set export/apply child object structure descriptor

Declaration

```
IWorkScope<T> With<TChild>(Expression<Func<T, TChild>> childProperty, Expression<Func<TChild, Guid>>
keyProperty, Expression<Func<TChild, bool>> deletedProperty, string alias)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func<T, TChild>>	childProperty	Single child property
System.Linq.Expressions.Expression<System.Func<TChild, System.Guid>>	keyProperty	Child work set key property
System.Linq.Expressions.Expression<System.Func<TChild, System.Boolean>>	deletedProperty	Child deleted flag property
System.String	alias	Child repository alias

Returns

TYPE	DESCRIPTION
IWorkScope<T>	Structure descriptor

Type Parameters

NAME	DESCRIPTION
TChild	Child object type

With<TChild>(Expression<Func<T, TChild>>, Expression<Func<TChild, Guid>>, Expression<Func<TChild, Boolean>>, String, Expression<Func<IWorkScope<TChild>, IWorkScope<TChild>>>)

Work set export/apply child object structure descriptor

Declaration

```
IWorkScope<T> With<TChild>(Expression<Func<T, TChild>> childProperty, Expression<Func<TChild, Guid>>
keyProperty, Expression<Func<TChild, bool>> deletedProperty, string alias, Expression<Func<IWorkScope<TChild>,
IWorkScope<TChild>>> childs)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func<T, TChild>>	childProperty	Single child property

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func<TChild, System.Guid>>	keyProperty	Child work set key property
System.Linq.Expressions.Expression<System.Func<TChild, System.Boolean>>	deletedProperty	Child deleted flag property
System.String	alias	Child repository alias
System.Linq.Expressions.Expression<System.Func<IWorkScope<TChild>, IWorkScope<TChild>>>	childs	Childs structure descriptor

Returns

TYPE	DESCRIPTION
IWorkScope<T>	Structure descriptor

Type Parameters

NAME	DESCRIPTION
TChild	Child object type

With<TChild>(Expression<Func<T, TChild>>, Expression<Func<TChild, Guid>>, Expression<Func<IWorkScope<TChild>, IWorkScope<TChild>>>)

Work set export/apply child object structure descriptor

Declaration

```
IWorkScope<T> With<TChild>(Expression<Func<T, TChild>> childProperty, Expression<Func<TChild, Guid>> keyProperty, Expression<Func<IWorkScope<TChild>, IWorkScope<TChild>>> childs)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func<T, TChild>>	childProperty	Single child property
System.Linq.Expressions.Expression<System.Func<TChild, System.Guid>>	keyProperty	Child work set key property
System.Linq.Expressions.Expression<System.Func<IWorkScope<TChild>, IWorkScope<TChild>>>	childs	Childs structure descriptor

Returns

TYPE	DESCRIPTION
IWorkScope<T>	Structure descriptor

Type Parameters

NAME	DESCRIPTION
TChild	Child object type

With<TChild>(Expression<Func<T, TChild>>, Expression<Func<TChild, Guid>>, String)

Work set export/apply child object structure descriptor

Declaration

```
IWorkScope<T> With<TChild>(Expression<Func<T, TChild>> childProperty, Expression<Func<TChild, Guid>>
keyProperty, string alias)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func<T, TChild>>	childProperty	Single child property
System.Linq.Expressions.Expression<System.Func<TChild, System.Guid>>	keyProperty	Child work set key property
System.String	alias	Child repository alias

Returns

TYPE	DESCRIPTION
IWorkScope<T>	Structure descriptor

Type Parameters

NAME	DESCRIPTION
TChild	Child object type

With<TChild>(Expression<Func<T, TChild>>, Expression<Func<TChild, Guid>>, String,
Expression<Func<IWorkScope<TChild>, IWorkScope<TChild>>>)

Work set export/apply child object structure descriptor

Declaration

```
IWorkScope<T> With<TChild>(Expression<Func<T, TChild>> childProperty, Expression<Func<TChild, Guid>>
keyProperty, string alias, Expression<Func<IWorkScope<TChild>, IWorkScope<TChild>>> childs)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func<T, TChild> >	childProperty	Single child property
System.Linq.Expressions.Expression<System.Func<TChild, System.Guid> >	keyProperty	Child work set key property
System.String	alias	Child repository alias
System.Linq.Expressions.Expression<System.Func< IWorkScope <TChild>, IWorkScope <TChild>>>	childs	Childs structure descriptor

Returns

TYPE	DESCRIPTION
IWorkScope <T>	Structure descriptor

Type Parameters

NAME	DESCRIPTION
TChild	Child object type

Extension Methods

[SerializationExtensions.ToXml](#)<T>(T)

[CollectionsExtensions.SetValue](#)<TKey, T>(TKey, T)

[CollectionsExtensions.AsEnum](#)<T>(T)

[DataObjectQueryableExtensions.CompareVariable](#)(Object, FilterOperation, String)

[Processing.Do](#)<T>(T, Action<IProcess<T>>[])

[TypeExtensions.TryConvert](#)<T, TResult>(T, Func<T, TResult>)

[TypeExtensions.TryConvert](#)<T, TResult>(T, Func<T, TResult>, TResult)

[TypeExtensions.ChangeType](#)<T>(Object)

[TypeExtensions.ChangeType](#)(Object, Type)

[TypeExtensions.InvokeMethod](#)<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)

[TypeExtensions.InvokeMethod](#)(Object, Type[], Expression<Action>, BindingFlags)

[TypeExtensions.InvokeMethod](#)<TRet>(Object, Expression<Func<TRet>>, BindingFlags)

[TypeExtensions.InvokeMethod](#)(Object, Expression<Action>, BindingFlags)

[SqlBlockExtensions.SetExpression](#)<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)

[QueryDescriptionExtensions.SetExpression](#)<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

See Also

[Root](#)<T>(Expression<Func<T, Guid>>, Expression<Func<T, Guid>>)

Export<TRoot>(Guid, Guid, Expression<Func<IWorkScope<TRoot>>>)&br/>Apply<TRoot>(TRoot, Expression<Func<IWorkScope<TRoot>>>)

Interface IWorkSet

Basic work set interface

Inherited Members

System.IDisposable.Dispose()

Namespace: [XData](#)

Assembly: XData.WorkSet.docfx.dll

Syntax

```
public interface IWorkSet : IDisposable
```

Properties

DbLayer

Data scope layer

Declaration

```
Guid DbLayer { get; }
```

Property Value

TYPE	DESCRIPTION
System.Guid	

Name

Work set name

Declaration

```
string Name { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Type

Work set type

Declaration

```
Type Type { get; }
```

Property Value

TYPE	DESCRIPTION
System.Type	

Methods

Clear()

Clear work set objects

Declaration

```
void Clear()
```

Submit()

Submit work set objects

Declaration

```
bool Submit()
```

Returns

TYPE	DESCRIPTION
System.Boolean	Success flag

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

See Also

[IWorkSet<T>](#)

Interface IWorkSet<T>

Work set interface

Inherited Members

- [IWorkSet.Name](#)
- [IWorkSet.Clear\(\)](#)
- [IWorkSet.Submit\(\)](#)
- [IWorkSet.Type](#)
- [IWorkSet.DbLayer](#)
- [System.IDisposable.Dispose\(\)](#)

Namespace: [XData](#)
Assembly: [XData.WorkSet.docfx.dll](#)

Syntax

```
public interface IWorkSet<out T> : IWorkSet, IDisposable where T : class, IDataObject, new()
```

Type Parameters

NAME	DESCRIPTION
T	Data object type

Remarks

Used to create/modify complex objects

Hint: UnitOfWork template

Methods

[Child<TRet>\(Func<T, Boolean>, String\)](#)

Find depended work set

Declaration

```
IWorkSet<TRet> Child<TRet>(Func<T, bool> predicate, string alias = null)  
    where TRet : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
System.Func<T, System.Boolean>	predicate	Predicate to find parent object
System.String	alias	(optional) Child repository alias

Returns

TYPE	DESCRIPTION
IWorkSet<TRet>	Found work set

Type Parameters

NAME	DESCRIPTION
TRet	Depended object type

Child<TRet>(Func<T, Int32, Boolean>, Func<T, IComparable>[], String)

Find depended work set (using order index)

Declaration

```
IWorkSet<TRet> Child<TRet>(Func<T, int, bool> predicate, Func<T, IComparable>[] order, string alias = null)
    where TRet : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
System.Func<T, System.Int32, System.Boolean>	predicate	Predicate to find parent object
System.Func<T, IComparable>[]	order	Order statements
System.String	alias	(optional) Child repository alias

Returns

TYPE	DESCRIPTION
IWorkSet<TRet>	Found work set

Type Parameters

NAME	DESCRIPTION
TRet	Depended object type

Child<TRet>(String)

Find depended work set

Declaration

```
IWorkSet<TRet> Child<TRet>(string alias = null)
    where TRet : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION

TYPE	NAME	DESCRIPTION
System.String	alias	(optional) Child repository alias

Returns

TYPE	DESCRIPTION
IWorkSet <TRet>	Found work set

Type Parameters

NAME	DESCRIPTION
TRet	Depended object type

Child<TRet, TKey>(Func<T, Int32, Boolean>, Func<T, TKey>, String)

Find depended work set (using order index)

Declaration

```
IWorkSet<TRet> Child<TRet, TKey>(Func<T, int, bool> predicate, Func<T, TKey> order, string alias = null)
    where TRet : class, IDataObject, new()
    where TKey : IComparable
```

Parameters

TYPE	NAME	DESCRIPTION
System.Func<T, System.Int32, System.Boolean>	predicate	Predicate to find parent object
System.Func<T, TKey>	order	Order statements
System.String	alias	(optional) Child repository alias

Returns

TYPE	DESCRIPTION
IWorkSet <TRet>	Found work set

Type Parameters

NAME	DESCRIPTION
TRet	Depended object type

NAME	DESCRIPTION
TKey	Order key type

Get(Func<T, Boolean>, WorkState)

Get single data object by predicate

Declaration

```
T Get(Func<T, bool> predicate = null, WorkState state = WorkState.Active)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Func<T, System.Boolean>	predicate	Predicate to find specific object
WorkState	state	Work set state

Returns

TYPE	DESCRIPTION
T	Data object

Get(Func<T, Int32, Boolean>, Func<T, IComparable>[], WorkState)

Get single data object by predicate (using order index)

Declaration

```
T Get(Func<T, int, bool> predicate, Func<T, IComparable>[] order, WorkState state = WorkState.Active)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Func<T, System.Int32, System.Boolean>	predicate	Predicate to find specific object
System.Func<T, IComparable >[]	order	Order statements
WorkState	state	Work set state

Returns

TYPE	DESCRIPTION

TYPE	DESCRIPTION
T	Data object

Get<TKey>(Func<T, Int32, Boolean>, Func<T, TKey>, WorkState)

Get single data object by predicate (using order index)

Declaration

```
T Get<TKey>(Func<T, int, bool> predicate, Func<T, TKey> order, WorkState state = WorkState.Active)
    where TKey : IComparable
```

Parameters

TYPE	NAME	DESCRIPTION
System.Func<T, System.Int32, System.Boolean>	predicate	Predicate to find specific object
System.Func<T, TKey>	order	Order statement
WorkState	state	Work set state

Returns

TYPE	DESCRIPTION
T	Data object

Type Parameters

NAME	DESCRIPTION
TKey	Order key type

GetKey()

Get single data object key

Declaration

```
Guid? GetKey()
```

Returns

TYPE	DESCRIPTION
System.Nullable<System.Guid>	Data object key

GetKey(Func<T, Boolean>, WorkState)

Get single data object key by predicate

Declaration

```
Guid? GetKey(Func<T, bool> predicate, WorkState state = WorkState.Active)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Func<T, System.Boolean>	predicate	Predicate to find specific object
WorkState	state	Work set state

Returns

TYPE	DESCRIPTION
System.Nullable<System.Guid>	Data object key

GetKey(Func<T, Int32, Boolean>, Func<T, IComparable>[], WorkState)

Get single data object key by predicate (using order index)

Declaration

```
Guid? GetKey(Func<T, int, bool> predicate, Func<T, IComparable>[] order, WorkState state = WorkState.Active)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Func<T, System.Int32, System.Boolean>	predicate	Predicate to find specific object
System.Func<T, IComparable>[]	order	Order statements
WorkState	state	Work set state

Returns

TYPE	DESCRIPTION
System.Nullable<System.Guid>	Data object key

GetKey(WorkState)

Get single data object key

Declaration

```
Guid? GetKey(WorkState state)
```

Parameters

TYPE	NAME	DESCRIPTION
WorkState	state	Work set state

Returns

TYPE	DESCRIPTION
System.Nullable<System.Guid>	Data object key

GetKey<TKey>(Func<T, Int32, Boolean>, Func<T, TKey>, WorkState)

Get single data object key by predicate (using order index)

Declaration

```
Guid? GetKey<TKey>(Func<T, int, bool> predicate, Func<T, TKey> order, WorkState state = WorkState.Active)
    where TKey : IComparable
```

Parameters

TYPE	NAME	DESCRIPTION
System.Func<T, System.Int32, System.Boolean>	predicate	Predicate to find specific object
System.Func<T, TKey>	order	Order statement
WorkState	state	Work set state

Returns

TYPE	DESCRIPTION
System.Nullable<System.Guid>	Data object key

Type Parameters

NAME	DESCRIPTION
TKey	Order key type

Lock(Func<T, Boolean>)

Try to obtain lock on data object

Declaration

```
bool Lock(Func<T, bool> predicate = null)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Func<T, System.Boolean>	predicate	Predicate to find specific object

Returns

TYPE	DESCRIPTION
System.Boolean	Lock success flag

Lock(Func<T, Int32, Boolean>, Func<T, IComparable>[])

Try to obtain lock on data object by predicate (using order index)

Declaration

<pre>bool Lock(Func<T, int, bool> predicate, Func<T, IComparable>[] order)</pre>

Parameters

TYPE	NAME	DESCRIPTION
System.Func<T, System.Int32, System.Boolean>	predicate	Predicate to find specific object
System.Func<T, IComparable>[]	order	Order statements

Returns

TYPE	DESCRIPTION
System.Boolean	Lock success flag

Lock<TKey>(Func<T, Int32, Boolean>, Func<T, TKey>)

Try to obtain lock on data object by predicate (using order index)

Declaration

<pre>bool Lock<TKey>(Func<T, int, bool> predicate, Func<T, TKey> order) where TKey : IComparable</pre>

Parameters

TYPE	NAME	DESCRIPTION
System.Func<T, System.Int32, System.Boolean>	predicate	Predicate to find specific object
System.Func<T, TKey>	order	Order statement

Returns

TYPE	DESCRIPTION
System.Boolean	Lock success flag

Type Parameters

NAME	DESCRIPTION
TKey	Order key type

New(Action<T>[])

Add object to work set collection

Declaration

T New(params Action<T>[] init)

Parameters

TYPE	NAME	DESCRIPTION
System.Action<T>[]	init	Object creator

Returns

TYPE	DESCRIPTION
T	Added object

Select(WorkState)

Retrieve all objects by work set

Declaration

IEnumerable<T> Select(WorkState state = WorkState.Active)

Parameters

TYPE	NAME	DESCRIPTION
WorkState	state	Work set state

Returns

TYPE	DESCRIPTION
System.Collections.Generic.IEnumerable<T>	Data object sequence

Select<TRet>(Func<T, TRet>, WorkState)

Retrieve all objects by work set and apply selector logic to convert data

Declaration

```
IEnumerable<TRet> Select<TRet>(Func<T, TRet> selector, WorkState state = WorkState.Active)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Func<T, TRet>	selector	Selector convert routine
WorkState	state	Work set state

Returns

TYPE	DESCRIPTION
System.Collections.Generic.IEnumerable<TRet>	Result object sequence

Type Parameters

NAME	DESCRIPTION
TRet	

SelectKeys(WorkState)

Retrieve all keys by work set

Declaration

```
IEnumerable<Guid> SelectKeys(WorkState state = WorkState.Active)
```

Parameters

TYPE	NAME	DESCRIPTION
WorkState	state	Work set state

Returns

TYPE	DESCRIPTION
System.Collections.Generic.IEnumerable<System.Guid>	Keys sequence

Extension Methods

- SerializationExtensions.ToXml<T>(T)
- CollectionsExtensions.SetValue<TKey, T>(TKey, T)
- CollectionsExtensions.AsEnum<T>(T)
- DataObjectQueryableExtensions.CompareVariable(Object, FilterOperation, String)

[Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
[TypeExtensions.ChangeType<T>\(Object\)](#)
[TypeExtensions.ChangeType\(Object, Type\)](#)
[TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
[TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
[TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
[TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
[SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
[QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

See Also

[Name](#)

[Clear\(\)](#)

[Submit\(\)](#)

[Lock<TKey>\(Func<T, Int32, Boolean>, Func<T, TKey>\)](#)

[Get\(Func<T, Int32, Boolean>, Func<T, IComparable>\[\], WorkState\)](#)

[Select\(WorkState\)](#)

[Child<TRet>\(String\)](#)

[Child<TRet>\(Func<T, Boolean>, String\)](#)

[Child<TRet>\(Func<T, Int32, Boolean>, Func<T, IComparable>\[\], String\)](#)

[Child<TRet, TKey>\(Func<T, Int32, Boolean>, Func<T, TKey>, String\)](#)

[New\(Action<T>\[\]\)](#)

[GetKey\(\)](#)

[SelectKeys\(WorkState\)](#)

[Work](#)

Delegate LockAttachedHandler<T>

Lock object attached handler delegate

Namespace: [XData](#)
Assembly: XData.docfx.dll

Syntax

```
public delegate bool LockAttachedHandler<in T>(T obj)
    where T : class, IDataObject;
```

Parameters

TYPE	NAME	DESCRIPTION
T	obj	Object instance

Returns

TYPE	DESCRIPTION
System.Boolean	

Type Parameters

NAME	DESCRIPTION
T	Object type

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Class Log

Log helper

Inheritance

System.Object

Log

Inherited Members

- System.Object.Equals(System.Object)
- System.Object.Equals(System.Object, System.Object)
- System.Object.GetHashCode()
- System.Object.GetType()
- System.Object.MemberwiseClone()
- System.Object.ReferenceEquals(System.Object, System.Object)
- System.Object.ToString()

Namespace: [XData](#)
Assembly: XData.docfx.dll

Syntax

```
public static class Log
```

Methods

Write(MessageType, Func<String>, Func<AdditionalInfo>[])

Write message to log

Declaration

```
public static void Write(MessageType msgType, Func<string> getMsgText, params Func<AdditionalInfo>[] getInfo)
```

Parameters

TYPE	NAME	DESCRIPTION
MessageType	msgType	Message type
System.Func<System.String>	getMsgText	Get message text handler
System.Func< AdditionalInfo > []	getInfo	Get additional information handlers

Delegate Mapping<TFrom, T>

Mapping logic delegate type

Namespace: [XData](#)
Assembly: XData.docfx.dll

Syntax

```
public delegate void Mapping<in TFrom, T>(TFrom source, ref T result)
    where TFrom : class where T : class, IDataObject;
```

Parameters

TYPE	NAME	DESCRIPTION
TFrom	source	Source object instance
T	result	Target object instance

Type Parameters

NAME	DESCRIPTION
TFrom	Source object type
T	Target object type

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

See Also

[Add<TFrom>\(IEnumerable<TFrom>, Mapping<TFrom, T>, DataSubmitFlag\)](#)

Enum MessageType

Log message type

Namespace: [XData](#)
Assembly: XData.docfx.dll

Syntax

```
public enum MessageType
```

Fields

NAME	DESCRIPTION
Debug	Debug log entry
Error	Error log entry
Information	Information log entry
Warning	Warning log entry

See Also

- [Error](#)
- [Information](#)
- [Debug](#)

Extension Methods

- [SerializationExtensions.ToXml<MessageType>\(\)](#)
- [CollectionsExtensions.SetValue<MessageType, T>\(T\)](#)
- [CollectionsExtensions.AsEnum<MessageType>\(\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(FilterOperation, String\)](#)
- [Processing.Do<MessageType>\(Action<IProcess<MessageType>>\[\]\)](#)
- [TypeExtensions.ChangeType<T>\(\)](#)
- [TypeExtensions.ChangeType\(Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.ConvertTo<T>\(\)](#)
- [TypeExtensions.EqualValue\(Enum\)](#)
- [SqlBlockExtensions.SetExpression<MessageType, TResult>\(Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<MessageType, TResult>\(Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Class ParameterIndexer

SQL procedure call out parameters indexer

Inheritance

System.Object
ParameterIndexer

Inherited Members

System.Object.Equals(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.GetHashCode()
System.Object.GetType()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.ToString()

Namespace: [XData](#)
Assembly: XData.docfx.dll

Syntax

```
public sealed class ParameterIndexer
```

Constructors

ParameterIndexer(IEnumerable<IParameterDescription>)

Parameter indexer constructor

Declaration

```
public ParameterIndexer(IEnumerable<IParameterDescription> parameters)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Collections.Generic.IEnumerable< IParameterDescription >	parameters	Parameter sequence

Properties

Count

Variables count

Declaration

```
public int Count { get; }
```

Property Value

TYPE	DESCRIPTION
System.Int32	

Item[String]

Parameter indexer

Declaration

```
public IParameterDescription this[string name] { get; }
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Parameter name

Property Value

TYPE	DESCRIPTION
IParameterDescription	Variable

Methods

Add(IParameterDescription)

Add parameter

Declaration

```
public void Add(IParameterDescription parameter)
```

Parameters

TYPE	NAME	DESCRIPTION
IParameterDescription	parameter	Variable

Add(ParameterIndexer)

Add variables to collection

Declaration

```
public void Add(ParameterIndexer variables)
```

Parameters

TYPE	NAME	DESCRIPTION
ParameterIndexer	variables	Variable indexer

Contains(String)

Variables contains name

Declaration

```
public bool Contains(string name)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Variable name

Returns

TYPE	DESCRIPTION
System.Boolean	Contains flag

GetValue(String)

Returns parameter value

Declaration

```
public object GetValue(string name)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Parameter name

Returns

TYPE	DESCRIPTION
System.Object	Parameter value

GetValue<T>(String)

Returns parameter value

Declaration

```
public T GetValue<T>(string name)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Parameter name

Returns

TYPE	DESCRIPTION
T	Parameter value

Type Parameters

NAME	DESCRIPTION
T	Parameter type

Extension Methods

SerializationExtensions.ToXml<T>(T)

CollectionsExtensions.SetValue<TKey, T>(TKey, T)

CollectionsExtensions.AsEnum<T>(T)

DataObjectQueryableExtensions.CompareVariable(Object, FilterOperation, String)

Processing.Do<T>(T, Action<IProcess<T>>[])

TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>)

TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>, TResult)

TypeExtensions.ChangeType<T>(Object)

TypeExtensions.ChangeType(Object, Type)

TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)

TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)

SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)

QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Enum ProcedureType

SQL stored procedure type

Namespace: [XData](#)

Assembly: XData.docfx.dll

Syntax

```
public enum ProcedureType
```

Fields

NAME	DESCRIPTION
Function	Function
Procedure	Stored procedure

See Also

[Procedure](#)

[Function](#)

Extension Methods

- [SerializationExtensions.ToXml<ProcedureType>\(\)](#)
- [CollectionsExtensions.SetValue<ProcedureType, T>\(T\)](#)
- [CollectionsExtensions.AsEnum<ProcedureType>\(\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(FilterOperation, String\)](#)
- [Processing.Do<ProcedureType>\(Action<IProcess<ProcedureType>>\[\]\)](#)
- [TypeExtensions.ChangeType<T>\(\)](#)
- [TypeExtensions.ChangeType\(Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.ConvertTo<T>\(\)](#)
- [TypeExtensions.EqualValue\(Enum\)](#)
- [SqlBlockExtensions.SetExpression<ProcedureType, TResult>\(Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<ProcedureType, TResult>\(Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Delegate RefreshAttachedHandler<T>

Refresh attached handler delegate

Namespace: [XData](#)
Assembly: XData.docfx.dll

Syntax

```
public delegate bool RefreshAttachedHandler<T>(ref T obj)
    where T : class, IDataObject;
```

Parameters

TYPE	NAME	DESCRIPTION
T	obj	Object instance

Returns

TYPE	DESCRIPTION
System.Boolean	

Type Parameters

NAME	DESCRIPTION
T	Object type

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Class RepositoryAttachedHandler<T>

Repository level attached handler wrapper

Inheritance

System.Object

[AttachedHandler<T>](#)

RepositoryAttachedHandler<T>

Inherited Members

[AttachedHandler<T>.Type](#)

[AttachedHandler<T>.Handler](#)

System.Object.Equals(System.Object)

System.Object.Equals(System.Object, System.Object)

System.Object.GetHashCode()

System.Object.GetType()

System.Object.MemberwiseClone()

System.Object.ReferenceEquals(System.Object, System.Object)

System.Object.ToString()

Namespace: [XData](#)

Assembly: XData.docfx.dll

Syntax

```
public sealed class RepositoryAttachedHandler<T> : AttachedHandler<T> where T : class, IDataObject
```

Type Parameters

NAME	DESCRIPTION
T	

Operators

Explicit(RepositoryAttachedHandler<T> to ClearAttachedHandler)

Explicitly unwrap clear handler

Declaration

```
public static explicit operator ClearAttachedHandler(RepositoryAttachedHandler<T> handler)
```

Parameters

TYPE	NAME	DESCRIPTION
RepositoryAttachedHandler<T>	handler	Wrapped handler

Returns

TYPE	DESCRIPTION
ClearAttachedHandler	Unwrapped attached handler

Explicit(RepositoryAttachedHandler<T> to ExecuteAttachedHandler<T>)

Explicitly unwrap execute handler

Explicitly unwrap execute handler

Declaration

```
public static explicit operator ExecuteAttachedHandler<T>(RepositoryAttachedHandler<T> handler)
```

Parameters

TYPE	NAME	DESCRIPTION
RepositoryAttachedHandler<T>	handler	Wrapped handler

Returns

TYPE	DESCRIPTION
ExecuteAttachedHandler<T>	Unwrapped attached handler

Explicit(RepositoryAttachedHandler<T> to SubmitAttachedHandler<T>)

Explicitly unwrap submit handler

Declaration

```
public static explicit operator SubmitAttachedHandler<T>(RepositoryAttachedHandler<T> handler)
```

Parameters

TYPE	NAME	DESCRIPTION
RepositoryAttachedHandler<T>	handler	Wrapped handler

Returns

TYPE	DESCRIPTION
SubmitAttachedHandler<T>	Unwrapped attached handler

Implicit(ClearAttachedHandler to RepositoryAttachedHandler<T>)

Implicitly wrap clear handler

Declaration

```
public static implicit operator RepositoryAttachedHandler<T>(ClearAttachedHandler handler)
```

Parameters

TYPE	NAME	DESCRIPTION
ClearAttachedHandler	handler	Clear handler

Returns

TYPE	DESCRIPTION
RepositoryAttachedHandler<T>	Repository attached handler

Implicit([ExecuteAttachedHandler<T>](#) to [RepositoryAttachedHandler<T>](#))

Implicitly wrap execute handler

Declaration

```
public static implicit operator RepositoryAttachedHandler<T>(ExecuteAttachedHandler<T> handler)
```

Parameters

TYPE	NAME	DESCRIPTION
ExecuteAttachedHandler<T>	handler	Execute handler

Returns

TYPE	DESCRIPTION
RepositoryAttachedHandler<T>	Repository attached handler

Implicit([SubmitAttachedHandler<T>](#) to [RepositoryAttachedHandler<T>](#))

Implicitly wrap submit handler

Declaration

```
public static implicit operator RepositoryAttachedHandler<T>(SubmitAttachedHandler<T> handler)
```

Parameters

TYPE	NAME	DESCRIPTION
SubmitAttachedHandler<T>	handler	Submit handler

Returns

TYPE	DESCRIPTION
RepositoryAttachedHandler<T>	Repository attached handler

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)

TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>, TResult)

TypeExtensions.ChangeType<T>(Object)

TypeExtensions.ChangeType(Object, Type)

TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)

TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)

SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)

QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Class RepositoryIndexer

Data object links (parents and childs) indexer

Inheritance

System.Object
RepositoryIndexer

Implements

System.Collections.Generic.IEnumerable<IRepository>
System.Collections.IEnumerable

Inherited Members

System.Object.Equals(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.GetHashCode()
System.Object.GetType()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.ToString()

Namespace: XData
Assembly: XData.docfx.dll

Syntax

```
public sealed class RepositoryIndexer : IEnumerable<IRepository>, IEnumerable
```

Properties

Count

Indexer count

Declaration

```
public int Count { get; }
```

Property Value

TYPE	DESCRIPTION
System.Int32	

Item[String]

Repository indexer

Declaration

```
public IRepository this[string alias] { get; }
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	alias	Repository alias

Property Value

TYPE	DESCRIPTION
IRepository	

Methods

Contains(String)

Indexer contains repository by alias

Declaration

```
public bool Contains(string alias)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	alias	Repository alias

Returns

TYPE	DESCRIPTION
System.Boolean	Contains flag

GetEnumerator()

Returns an enumerator that iterates through the collection.

Declaration

```
public IEnumerator<IRepository> GetEnumerator()
```

Returns

TYPE	DESCRIPTION
System.Collections.Generic.IEnumerator< IRepository >	A System.Collections.Generic.IEnumerator<T> that can be used to iterate through the collection.

Explicit Interface Implementations

IEnumerable.GetEnumerator()

Returns an enumerator that iterates through a collection.

Declaration

```
IEnumerator IEnumerable.GetEnumerator()
```

Returns

TYPE	DESCRIPTION

TYPE	DESCRIPTION
System.Collections.IEnumerator	An System.Collections.IEnumerator object that can be used to iterate through the collection.

Implements

System.Collections.Generic.IEnumerable<T>
System.Collections.IEnumerable

Extension Methods

SerializationExtensions.ToXml<T>(T)
CollectionsExtensions.ToSortedSet<T>(IEnumerable<T>, IComparer<T>)
CollectionsExtensions.SetValue<TKey, T>(TKey, T)
CollectionsExtensions.AsEnum<T>(T)
DataObjectQueryableExtensions.CompareVariable(Object, FilterOperation, String)
JoinedEnumerable.Inner<TElement>(IEnumerable<TElement>)
JoinedEnumerable.Outer<TElement>(IEnumerable<TElement>)
JoinedEnumerable.LeftOuterJoin<TOuter, TInner, TKey, TResult>(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter, TKey>, Func<TInner, TKey>, Func<TOuter, TInner, TResult>, IEqualityComparer<TKey>)
JoinedEnumerable.RightOuterJoin<TOuter, TInner, TKey, TResult>(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter, TKey>, Func<TInner, TKey>, Func<TOuter, TInner, TResult>, IEqualityComparer<TKey>)
JoinedEnumerable.FullOuterJoin<TOuter, TInner, TKey, TResult>(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter, TKey>, Func<TInner, TKey>, Func<TOuter, TInner, TResult>, IEqualityComparer<TKey>)
Processing.Do<T>(T, Action<IProcess<T>>[])
TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>)
TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>, TResult)
TypeExtensions.ChangeType<T>(Object)
TypeExtensions.ChangeType(Object, Type)
TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)
TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)
TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)
TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)
SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)
QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)
QueryDescriptionExtensions.ToDataSet<T>(IEnumerable<T>, String, ISqlBuilder)

Delegate RepositoryTrigger<T>

RepositoryTrigger logic delegate type

Namespace: [XData](#)
Assembly: XData.docfx.dll

Syntax

```
public delegate bool RepositoryTrigger<T>(IRepository<T> repository, ref DataTriggerFlag flag)
    where T : class, IDataObject;
```

Parameters

TYPE	NAME	DESCRIPTION
IRepository<T>	repository	Repository
DataTriggerFlag	flag	Trigger routing flag

Returns

TYPE	DESCRIPTION
System.Boolean	Success flag

Type Parameters

NAME	DESCRIPTION
T	Object type

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

See Also

- [BeforeClear](#)
- [AfterClear](#)

Class ResultSetIndexer

SQL procedure call out result sets indexer

Inheritance

System.Object
ResultSetIndexer

Inherited Members

System.Object.Equals(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.GetHashCode()
System.Object.GetType()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.ToString()

Namespace: [XData](#)
Assembly: XData.docfx.dll

Syntax

```
public sealed class ResultSetIndexer
```

Properties

Count

Variables count

Declaration

```
public int Count { get; }
```

Property Value

TYPE	DESCRIPTION
System.Int32	

Item[String]

Result set data rows

Declaration

```
public DataTable this[string name] { get; }
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Result set name

Property Value

TYPE	DESCRIPTION
System.Data.DataTable	

Methods

Contains(String)

Variables contains name

Declaration

```
public bool Contains(string name)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Variable name

Returns

TYPE	DESCRIPTION
System.Boolean	Contains flag

GetResultSet<T>(String)

Returns typed result collection

Declaration

```
public IEnumerable<T> GetResultSet<T>(string name)
    where T : class
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Result set name

Returns

TYPE	DESCRIPTION
System.Collections.Generic.IEnumerable<T>	Types result collection

Type Parameters

NAME	DESCRIPTION
T	Result set element mapped type

Extension Methods

[SerializationExtensions.ToXml<T>\(T\)](#)

[CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)

[CollectionsExtensions.AsEnum<T>\(T\)](#)

DataObjectQueryableExtensions.CompareVariable(Object, FilterOperation, String)
Processing.Do<T>(T, Action<IProcess<T>>[])
TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>)
TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>, TResult)
TypeExtensions.ChangeType<T>(Object)
TypeExtensions.ChangeType(Object, Type)
TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)
TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)
TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)
TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)
SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)
QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Class RuntimeAssemblyLoader

Inheritance

System.Object
System.Runtime.Loader.AssemblyLoadContext
RuntimeAssemblyLoader

Inherited Members

System.Runtime.Loader.AssemblyLoadContext.GetAssemblyName(System.String)
System.Runtime.Loader.AssemblyLoadContext.GetLoadContext(System.Reflection.Assembly)
System.Runtime.Loader.AssemblyLoadContext.LoadFromAssemblyName(System.Reflection.AssemblyName)
System.Runtime.Loader.AssemblyLoadContext.LoadFromAssemblyPath(System.String)
System.Runtime.Loader.AssemblyLoadContext.LoadFromNativeImagePath(System.String, System.String)
System.Runtime.Loader.AssemblyLoadContext.LoadFromStream(System.IO.Stream)
System.Runtime.Loader.AssemblyLoadContext.LoadFromStream(System.IO.Stream, System.IO.Stream)
System.Runtime.Loader.AssemblyLoadContext.LoadUnmanagedDllFromPath(System.String)
System.Runtime.Loader.AssemblyLoadContext.SetProfileOptimizationRoot(System.String)
System.Runtime.Loader.AssemblyLoadContext.StartProfileOptimization(System.String)
System.Runtime.Loader.AssemblyLoadContext.Default
System.Runtime.Loader.AssemblyLoadContext.Resolving
System.Runtime.Loader.AssemblyLoadContext.Unloading
System.Object.Equals(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.GetHashCode()
System.Object.GetType()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.ToString()

Namespace: [XData](#)
Assembly: XData.docfx.dll

Syntax

```
public class RuntimeAssemblyLoader : AssemblyLoadContext
```

Constructors

RuntimeAssemblyLoader(String)

Declaration

```
public RuntimeAssemblyLoader(string folderPath)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	folderPath	

Methods

Load(AssemblyName)

Declaration

```
protected override Assembly Load(AssemblyName assemblyName)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Reflection.AssemblyName	assemblyName	

Returns

TYPE	DESCRIPTION
System.Reflection.Assembly	

Overrides

System.Runtime.Loader.AssemblyLoadContext.Load(System.Reflection.AssemblyName)

LoadUnmanagedDll(String)

Declaration

protected override IntPtr LoadUnmanagedDll(string unmanagedDllName)

Parameters

TYPE	NAME	DESCRIPTION
System.String	unmanagedDllName	

Returns

TYPE	DESCRIPTION
System.IntPtr	

Overrides

System.Runtime.Loader.AssemblyLoadContext.LoadUnmanagedDll(System.String)

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Class SerializationExtensions

Serialization extensions

Inheritance

System.Object

SerializationExtensions

Inherited Members

System.Object.Equals(System.Object)

System.Object.Equals(System.Object, System.Object)

System.Object.GetHashCode()

System.Object.GetType()

System.Object.MemberwiseClone()

System.Object.ReferenceEquals(System.Object, System.Object)

System.Object.ToString()

Namespace: [XData](#)

Assembly: XData.docfx.dll

Syntax

```
public static class SerializationExtensions
```

Methods

ObjectToXml(Object)

Serialize object data to Xml

Declaration

```
public static string ObjectToXml(object source)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Object	source	Object to serialize

Returns

TYPE	DESCRIPTION
System.String	Xml string

ReadXml<T>(String)

Deserialize helper

Declaration

```
public static T ReadXml<T>(this string source)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	source	String contained serialized object

Returns

TYPE	DESCRIPTION
T	De-serialized object

Type Parameters

NAME	DESCRIPTION
T	Object type

Restore<T>(T, SerializedData)

Set serialized data to object

Declaration

```
public static void Restore<T>(this T obj, SerializedData data)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
T	obj	Data object instance
SerializedData	data	Serialized data

Type Parameters

NAME	DESCRIPTION
T	Data object type

Exceptions

TYPE	CONDITION
XDataQueryStructureException	serialized data contains not mapped properties.
XDataRuntimeExcepion	serialized data is null.

TYPE	CONDITION
XDataRuntimeException	serialized data have a wrong type.

See Also

[Serialize<T>\(T\)](#)

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

Serialize<T>(T)

Get serialized data from object

Declaration

```
public static SerializedData Serialize<T>(this T obj)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
T	obj	Data object instance

Returns

TYPE	DESCRIPTION
SerializedData	Serialized data

Type Parameters

NAME	DESCRIPTION
T	Data object type

See Also

[Restore<T>\(T, SerializedData\)](#)

Serialize<T>(IEnumerable<T>)

Get serialized data from objects sequence

Declaration

```
public static SerializedData[] Serialize<T>(this IEnumerable<T> objects)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
System.Collections.Generic.IEnumerable<T>	objects	Data objects sequence

Returns

TYPE	DESCRIPTION
SerializedData []	Serialized data

Type Parameters

NAME	DESCRIPTION
T	Data object type

See Also

[Restore<T>\(T, SerializedData\)](#)

Serialize<T>(IDataQuery<T>, TextWriter)

Serialize data query helper

Declaration

```
public static void Serialize<T>(this IDataQuery<T> query, TextWriter writer)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
IDataQuery<T>	query	Query
System.IO.TextWriter	writer	Writer to serialize

Type Parameters

NAME	DESCRIPTION
T	Data object type

Exceptions

TYPE	CONDITION
XDataRuntimeException	Writer is null

See Also

[GetRepository<T>\(TextReader\)](#)

Serialize<T>(IRepository<T>, TextWriter)

Serialize repository helper

Declaration

```
public static void Serialize<T>(this IRepository<T> repository, TextWriter writer)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
IRepository<T>	repository	Repository
System.IO.TextWriter	writer	Writer to serialize

Type Parameters

NAME	DESCRIPTION
T	Data object type

Exceptions

TYPE	CONDITION
XDataRuntimeException	Writer is null

See Also

[GetRepository<T>\(TextReader\)](#)

[ToXml<T>\(T\)](#)

Serialize helper

Declaration

```
public static string ToXml<T>(this T source)
```

Parameters

TYPE	NAME	DESCRIPTION
T	source	Object to serialize

Returns

TYPE	DESCRIPTION
System.String	Serialized collection

Type Parameters

NAME	DESCRIPTION

NAME	DESCRIPTION
T	Object type

Class SerializedData

Serialized data

Inheritance

System.Object
SerializedData

Inherited Members

System.Object.Equals(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.GetHashCode()
System.Object.GetType()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)

Namespace: [XData](#)
Assembly: XData.docfx.dll

Syntax

```
[DataContract]  
[Serializable]  
public class SerializedData
```

Constructors

SerializedData(IRepository, IDataObject)

Serialized data constructor

Declaration

```
public SerializedData(IRepository repository, IDataObject obj)
```

Parameters

TYPE	NAME	DESCRIPTION
IRepository	repository	Repository
IDataObject	obj	Data object to serialize

Properties

Data

Serialized data

Declaration

```
[DataMember]  
public SerializedValue[] Data { get; set; }
```

Property Value

TYPE	DESCRIPTION
SerializedValue[]	

ObjectType

Object type

Declaration

[DataMember] public string ObjectType { get; }

Property Value

TYPE	DESCRIPTION
System.String	

State

DataObject state

Declaration

[DataMember] public DataObjectState State { get; set; }

Property Value

TYPE	DESCRIPTION
DataObjectState	

Methods

GetFieldValue(Type, String)

Returns field value

Declaration

public object GetFieldValue(Type type, string fieldName)

Parameters

TYPE	NAME	DESCRIPTION
System.Type	type	Field type
System.String	fieldName	Field name

Returns

TYPE	DESCRIPTION

TYPE	DESCRIPTION
System.Object	Field value

Exceptions

TYPE	CONDITION
XDataRuntimeException	serialized data has not contains requested field.

HasFieldValue(String)

Serialized data has field value

Declaration

```
public bool HasFieldValue(string fieldName)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	fieldName	Field name

Returns

TYPE	DESCRIPTION
System.Boolean	Field value found

IsFieldNull(String)

Serialized data field value is null

Declaration

```
public bool IsFieldNull(string fieldName)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	fieldName	Field name

Returns

TYPE	DESCRIPTION
System.Boolean	Field value is null

ToString()

Returns a System.String that represents the current System.Object.

Declaration

```
public override string ToString()
```

Returns

TYPE	DESCRIPTION
System.String	A System.String that represents the current System.Object.

Overrides

System.Object.ToString()

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Class SerializedValue

Serialized data value

Inheritance

System.Object
SerializedValue

Inherited Members

System.Object.Equals(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.GetHashCode()
System.Object.GetType()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.ToString()

Namespace: [XData](#)
Assembly: XData.docfx.dll

Syntax

```
[Serializable]  
[DataContract]  
public class SerializedValue
```

Constructors

SerializedValue(String, String, Object, Object)

Serialized value constructor

Declaration

```
public SerializedValue(string propertyName, string fieldName, object original, object value)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	propertyName	Property name
System.String	fieldName	Field name
System.Object	original	Original value
System.Object	value	Value

Properties

FieldName

Field name

Declaration

```
[DataMember]  
public string FieldName { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

IsNotChanged

Not changed flag

Declaration

```
[DataMember]  
public bool IsNotChanged { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

IsNull

Null value flag

Declaration

```
[DataMember]  
public bool IsNull { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

Original

Original value

Declaration

```
[DataMember]  
public string Original { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

PropertyName

Property name

Declaration

```
[DataMember]  
public string PropertyName { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

TypeName

Value type name

Declaration

```
[DataMember]  
public string TypeName { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Value

Original value

Declaration

```
[DataMember]  
public string Value { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Methods

Deserialize(Type, String)

Deserialize value

Declaration

```
public static object Deserialize(Type type, string value)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Type	type	Data type
System.String	value	String value representation

Returns

TYPE	DESCRIPTION
System.Object	Value

GetValue(Type)

Returns serialized value

Declaration

```
public object GetValue(Type type)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Type	type	

Returns

TYPE	DESCRIPTION
System.Object	

Serialize(Object)

Serialize value

Declaration

```
public static string Serialize(object value)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Object	value	Value

Returns

TYPE	DESCRIPTION
System.String	String value representation

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)

TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)

TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)

SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)

QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Delegate SetLobAttachedHandler

Set Lob attached handler delegate

Namespace: [XData](#)
Assembly: XData.docfx.dll

Syntax

```
public delegate void SetLobAttachedHandler(string property, IDataObject obj, byte[] body);
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	property	Lob property name
IDataObject	obj	Data object
Byte[]	body	Lob body

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Delegate SetXmlAttachedHandler

Set Xml attached handler delegate

Namespace: [XData](#)
Assembly: XData.docfx.dll

Syntax

```
public delegate bool SetXmlAttachedHandler(string property, IDataObject obj, string xml);
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	property	Lob property name
IDataObject	obj	Data object
System.String	xml	Xml string

Returns

TYPE	DESCRIPTION
System.Boolean	

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Delegate SubmitAttachedHandler<T>

Submit attached handler delegate

Namespace: [XData](#)
Assembly: XData.docfx.dll

Syntax

```
public delegate bool SubmitAttachedHandler<in T>(IEnumerable<T> objects, DataSubmitFlag flag = DataSubmitFlag.None)
    where T : class, IDataObject;
```

Parameters

TYPE	NAME	DESCRIPTION
System.Collections.Generic.IEnumerable<T>	objects	Object sequence
DataSubmitFlag	flag	Submit flag

Returns

TYPE	DESCRIPTION
System.Boolean	

Type Parameters

NAME	DESCRIPTION
T	Object type

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Delegate SubmitObjectAttachedHandler<T>

Submit object attached handler delegate

Namespace: [XData](#)
Assembly: XData.docfx.dll

Syntax

```
public delegate bool SubmitObjectAttachedHandler<T>(ref T obj, DataSubmitFlag flag)
    where T : class, IDataObject;
```

Parameters

TYPE	NAME	DESCRIPTION
T	obj	Object instance
DataSubmitFlag	flag	Submit flag

Returns

TYPE	DESCRIPTION
System.Boolean	

Type Parameters

NAME	DESCRIPTION
T	Object type

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Delegate SubmitQueryAttachedHandler<T>

Lock object attached handler delegate

Namespace: [XData](#)
Assembly: XData.docfx.dll

Syntax

```
public delegate QueryResult SubmitQueryAttachedHandler<in T>(T obj, Query query)
    where T : class, IDataObject;
```

Parameters

TYPE	NAME	DESCRIPTION
T	obj	Object instance
Query	query	Query

Returns

TYPE	DESCRIPTION
QueryResult	

Type Parameters

NAME	DESCRIPTION
T	Object type

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Delegate Trigger<T>

Trigger logic delegate type

Namespace: [XData](#)
Assembly: XData.docfx.dll

Syntax

```
public delegate bool Trigger<T>(ref T obj, ref DataTriggerFlag flag)
    where T : class, IDataObject;
```

Parameters

TYPE	NAME	DESCRIPTION
T	obj	Object instance
DataTriggerFlag	flag	Trigger routing flag

Returns

TYPE	DESCRIPTION
System.Boolean	Success flag

Type Parameters

NAME	DESCRIPTION
T	Object type

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

See Also

- [BeforeDelete](#)
- [BeforeInsert](#)

BeforeUpdate
AfterDelete
AfterInsert
AfterUpdate

Class Variable

Data object repository variable

Inheritance

System.Object
Variable

Inherited Members

System.Object.Equals(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.GetHashCode()
System.Object.GetType()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)

Namespace: [XData](#)
Assembly: XData.docfx.dll

Syntax

```
public sealed class Variable
```

Constructors

Variable(String, Object)

Data object repository variable constructor

Declaration

```
public Variable(string name, object val)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Variable name
System.Object	val	Variable value

Properties

Name

Variable name

Declaration

```
public string Name { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Value

Variable value

Declaration

```
public object Value { get; }
```

Property Value

TYPE	DESCRIPTION
System.Object	

Methods

ToString()

Returns a string that represents the current object.

Declaration

```
public override string ToString()
```

Returns

TYPE	DESCRIPTION
System.String	A string that represents the current object.

Overrides

System.Object.ToString()

Extension Methods

- SerializationExtensions.ToXml<T>(T)
- CollectionsExtensions.SetValue<TKey, T>(TKey, T)
- CollectionsExtensions.AsEnum<T>(T)
- DataObjectQueryableExtensions.CompareVariable(Object, FilterOperation, String)
- Processing.Do<T>(T, Action<IProcess<T>>[])
- TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>)
- TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>, TResult)
- TypeExtensions.ChangeType<T>(Object)
- TypeExtensions.ChangeType(Object, Type)
- TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)
- TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)
- TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)
- TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)
- SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)
- QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Class VariableIndexer

Data object variables indexer

Inheritance

System.Object
VariableIndexer

Inherited Members

System.Object.Equals(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.GetHashCode()
System.Object.GetType()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.ToString()

Namespace: [XData](#)
Assembly: XData.docfx.dll

Syntax

```
public sealed class VariableIndexer
```

Constructors

VariableIndexer(IEnumerable<Variable>)

Variable indexer constructor

Declaration

```
public VariableIndexer(IEnumerable<Variable> variables)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Collections.Generic.IEnumerable< Variable >	variables	Variables sequence

Properties

Count

Variables count

Declaration

```
public int Count { get; }
```

Property Value

TYPE	DESCRIPTION
System.Int32	

Item[String]

Variable indexer

Declaration

```
public object this[string name] { get; }
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Variable name

Property Value

TYPE	DESCRIPTION
System.Object	Variable

Methods

Contains(String)

Variables contains name

Declaration

```
public bool Contains(string name)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Variable name

Returns

TYPE	DESCRIPTION
System.Boolean	Contains flag

Set(IEnumerable<Variable>)

Set variables values

Declaration

```
public VariableIndexer Set(IEnumerable<Variable> variables)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Collections.Generic.IEnumerable< Variable >	variables	Variables sequence

Returns

TYPE	DESCRIPTION
VariableIndexer	

Set(String, String)

Set variable value

Declaration

```
public VariableIndexer Set(string name, string value)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Variable name
System.String	value	Variable value

Returns

TYPE	DESCRIPTION
VariableIndexer	

Set(Variable)

Set variable value

Declaration

```
public VariableIndexer Set(Variable variable)
```

Parameters

TYPE	NAME	DESCRIPTION
Variable	variable	Variable

Returns

TYPE	DESCRIPTION
VariableIndexer	

Set(VariableIndexer)

Set variables values

Declaration

```
public VariableIndexer Set(VariableIndexer variables)
```

Parameters

TYPE	NAME	DESCRIPTION
VariableIndexer	variables	Variable indexer

Returns

TYPE	DESCRIPTION
VariableIndexer	

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Enum WithRecursiveType

Common-Table-Expression (WITH) Subquery type

Namespace: [XData](#)
Assembly: XData.docfx.dll

Syntax

```
[DataContract]  
public enum WithRecursiveType
```

Fields

NAME	DESCRIPTION
RecursiveUnion	Recursive select with UNION combination
RecursiveUnionAll	Recursive select with ALL combination

Extension Methods

- [SerializationExtensions.ToXml<WithRecursiveType>\(\)](#)
- [CollectionsExtensions.SetValue<WithRecursiveType, T>\(T\)](#)
- [CollectionsExtensions.AsEnum<WithRecursiveType>\(\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(FilterOperation, String\)](#)
- [Processing.Do<WithRecursiveType>\(Action<IProcess<WithRecursiveType>>\[\]\)](#)
- [TypeExtensions.ChangeType<T>\(\)](#)
- [TypeExtensions.ChangeType\(Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.ConvertTo<T>\(\)](#)
- [TypeExtensions.EqualValue\(Enum\)](#)
- [SqlBlockExtensions.SetExpression<WithRecursiveType, TResult>\(Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<WithRecursiveType, TResult>\(Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Class Work

Work set helper class

Inheritance

System.Object

Work

Inherited Members

- System.Object.Equals(System.Object)
- System.Object.Equals(System.Object, System.Object)
- System.Object.GetHashCode()
- System.Object.GetType()
- System.Object.MemberwiseClone()
- System.Object.ReferenceEquals(System.Object, System.Object)
- System.Object.ToString()

Namespace: [XData](#)
Assembly: XData.WorkSet.docfx.dll

Syntax

```
public static class Work
```

Methods

Add<T>(T, Expression<Func<T, IWorkSet>>[])

Create work set based on single object

Declaration

```
public static IWorkSet<T> Add<T>(T data, params Expression<Func<T, IWorkSet>>[] workSets)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
T	data	Data object
System.Linq.Expressions.Expression<System.Func<T, IWorkSet >>[]	workSets	Depended objects/collections initializers

Returns

TYPE	DESCRIPTION
IWorkSet <T>	Work set

Type Parameters

NAME	DESCRIPTION
T	Data object type

See Also

[Empty<T, TRet>\(String, ISecuritySession, Expression<Func<TRet, IWorkSet>>\[\]\)](#)

[Fill<T, TRet>\(String, ISecuritySession, Expression<Func<TRet, IWorkSet>>\[\]\)](#)

Add<T>(ICollection<T>, Expression<Func<T, IWorkSet>>[])

Create work set based on collection

Declaration

```
public static IWorkSet<T> Add<T>(ICollection<T> data, params Expression<Func<T, IWorkSet>>[] workSets)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
System.Collections.Generic.ICollection<T>	data	Data object collection
System.Linq.Expressions.Expression<System.Func<T, IWorkSet>>[]	workSets	Depended objects/collections initializers

Returns

TYPE	DESCRIPTION
IWorkSet<T>	Work set

Type Parameters

NAME	DESCRIPTION
T	Data object type

See Also

[Empty<T, TRet>\(String, ISecuritySession, Expression<Func<TRet, IWorkSet>>\[\]\)](#)

[Fill<T, TRet>\(String, ISecuritySession, Expression<Func<TRet, IWorkSet>>\[\]\)](#)

Apply<TRoot>(TRoot, Expression<Func<IWorkScope<TRoot>>>)

Apply single data object (optionally with some childs) to work set

Declaration

```
public static void Apply<TRoot>(TRoot obj, Expression<Func<IWorkScope<TRoot>>> structure)
    where TRoot : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
TRoot	obj	Detached data object to apply

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func< IWorkScope <TRoot>>>	structure	Childs data description

Type Parameters

NAME	DESCRIPTION
TRoot	Root object type

Empty<T, TRet>(String, ISecuritySession, Expression<Func<TRet, IWorkSet>>[])

Empty single depended object initializer

Declaration

```
public static Expression<Func<T, IWorkSet>> Empty<T, TRet>(string alias = null, ISecuritySession security = null, params Expression<Func<TRet, IWorkSet>>[] workSets)
    where T : class, IDataObject, new()
    where TRet : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	alias	(optional) Child repository alias
ISecuritySession	security	(optional) Security session
System.Linq.Expressions.Expression<System.Func<TRet, IWorkSet >>[]	workSets	Depended objects/collections initializers

Returns

TYPE	DESCRIPTION
System.Linq.Expressions.Expression<System.Func<T, IWorkSet >>	Work set

Type Parameters

NAME	DESCRIPTION
T	Parent data object type
TRet	Depended data object type

See Also

[Fill<T, TRet>\(String, ISecuritySession, Expression<Func<TRet, IWorkSet>>\[\]\)](#)

Export<TRoot>(Guid, Guid, Expression<Func<IWorkScope<TRoot>>>)

Export single data object (optionally with some childs) by key

Declaration

```
public static TRoot Export<TRoot>(Guid layer, Guid key, Expression<Func<IWorkScope<TRoot>>> structure)
    where TRoot : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
System.Guid	layer	Data set layer
System.Guid	key	Work set key
System.Linq.Expressions.Expression<System.Func< IWorkScope <TRoot>>>	structure	Structure description

Returns

TYPE	DESCRIPTION
TRoot	Data object

Type Parameters

NAME	DESCRIPTION
TRoot	Root object type

Fill<T, TRet>(String, ISecuritySession, Expression<Func<TRet, IWorkSet>>[])

Existed depended collection initializer

Declaration

```
public static Expression<Func<T, IWorkSet>> Fill<T, TRet>(string alias = null, ISecuritySession security =
    null, params Expression<Func<TRet, IWorkSet>>[] workSets)
    where T : class, IDataObject, new()
    where TRet : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	alias	(optional) Child repository alias
ISecuritySession	security	(optional) Security session

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func<TRet, IWorkSet>> []	workSets	Depended objects/collections initializers

Returns

TYPE	DESCRIPTION
System.Linq.Expressions.Expression<System.Func<T, IWorkSet>>	Work set

Type Parameters

NAME	DESCRIPTION
T	Parent data object type
TRet	Depended data object type

See Also

[Empty<T, TRet>\(String, ISecuritySession, Expression<Func<TRet, IWorkSet>> \[\]\)](#)

GetWorkSet<T>(Guid)

Returns a cached work set

Declaration

```
public static IWorkSet<T> GetWorkSet<T>(Guid layer)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
System.Guid	layer	Data scope layer

Returns

TYPE	DESCRIPTION
IWorkSet<T>	Work set

Type Parameters

NAME	DESCRIPTION
T	Data object type

Root<T>(Expression<Func<T, Guid>>, Expression<Func<T, Guid>>)

Root structure describer for work set export/apply

Declaration

```
public static IWorkScope<T> Root<T>(Expression<Func<T, Guid>> layerProperty, Expression<Func<T, Guid>> keyProperty)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func<T, System.Guid>>	layerProperty	Data scope layer property
System.Linq.Expressions.Expression<System.Func<T, System.Guid>>	keyProperty	Work set key property

Returns

TYPE	DESCRIPTION
IWorkScope<T>	Structure describer

Type Parameters

NAME	DESCRIPTION
T	Root type

See Also

[Empty<T, TRet>\(String, ISecuritySession, Expression<Func<TRet, IWorkSet>>\[\]\)](#)
[Fill<T, TRet>\(String, ISecuritySession, Expression<Func<TRet, IWorkSet>>\[\]\)](#)

Enum WorkState

Work set state

Namespace: [XData](#)

Assembly: XData.WorkSet.docfx.dll

Syntax

```
public enum WorkState
```

Fields

NAME	DESCRIPTION
Active	Active (not deleted) objects
Changed	Changed objects
Deleted	Deleted objects
New	New objects
Original	Original objects

Extension Methods

- [SerializationExtensions.ToXml<WorkState>\(\)](#)
- [CollectionsExtensions.SetValue<WorkState, T>\(T\)](#)
- [CollectionsExtensions.AsEnum<WorkState>\(\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(FilterOperation, String\)](#)
- [Processing.Do<WorkState>\(Action<IProcess<WorkState>>\[\]\)](#)
- [TypeExtensions.ChangeType<T>\(\)](#)
- [TypeExtensions.ChangeType\(Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.ConvertTo<T>\(\)](#)
- [TypeExtensions.EqualValue\(Enum\)](#)
- [SqlBlockExtensions.SetExpression<WorkState, TResult>\(Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<WorkState, TResult>\(Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Class XDataLogic<T>

Inheritance

System.Object
XDataLogic<T>

Implements

[IDataLogic<T>](#)

Inherited Members

System.Object.Equals(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.GetHashCode()
System.Object.GetType()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.ToString()

Namespace: [XData](#)
Assembly: XData.docfx.dll

Syntax

```
public class XDataLogic<T> : IDataLogic<T> where T : class, IDataObject, new()
```

Type Parameters

NAME	DESCRIPTION
T	

Methods

BeginTransaction(Guid, String, Boolean, IsolationLevel)

Declaration

```
protected ITransaction BeginTransaction(Guid layer, string context = null, bool read = false, IsolationLevel isolation = IsolationLevel.ReadCommitted)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Guid	layer	
System.String	context	
System.Boolean	read	
System.Data.IsolationLevel	isolation	

Returns

TYPE	DESCRIPTION
ITransaction	

GetDictionaryValue<TDict>(Expression<Func<TDict, Boolean>>, Guid, String, String, ISecuritySession, Variable[])

Declaration

```
protected TDict GetDictionaryValue<TDict>(Expression<Func<TDict, bool>> findExpression, Guid layer, string alias = null, string context = null, ISecuritySession security = null, params Variable[] variables)
    where TDict : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func<TDict, System.Boolean>>	findExpression	
System.Guid	layer	
System.String	alias	
System.String	context	
ISecuritySession	security	
Variable[]	variables	

Returns

TYPE	DESCRIPTION
TDict	

Type Parameters

NAME	DESCRIPTION
TDict	

GetRepository<TRep>(Guid, String, String, ISecuritySession, Variable[])

Declaration

```
protected IRepository<TRep> GetRepository<TRep>(Guid layer, string alias = null, string context = null, ISecuritySession security = null, params Variable[] variables)
    where TRep : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
System.Guid	layer	
System.String	alias	
System.String	context	
ISecuritySession	security	
Variable[]	variables	

Returns

TYPE	DESCRIPTION
IRepository<TRep>	

Type Parameters

NAME	DESCRIPTION
TRep	

Implements

[IDataLogic<T>](#)

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Class XDataManager

Data manager Defines user session context across (possible) multiple data sources

Inheritance

System.Object
XDataManager

Inherited Members

System.Object.Equals(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.GetHashCode()
System.Object.GetType()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.ToString()

Namespace: [XData](#)
Assembly: XData.docfx.dll

Syntax

```
public static class XDataManager
```

Fields

Resources

Declaration

```
public static dynamic Resources
```

Field Value

TYPE	DESCRIPTION
System.Object	

Properties

Configuration

Declaration

```
public static IConfiguration Configuration { get; }
```

Property Value

TYPE	DESCRIPTION
IConfiguration	

Credentials

Declaration

```
public static Dictionary<string, Dictionary<CredentialsType, string>> Credentials { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Collections.Generic.Dictionary<System.String, System.Collections.Generic.Dictionary< CredentialsType , System.String>>	

Methods

AddXData(IServiceCollection, Func<IDataOptions, IDataOptions>)

XData initialization

Declaration

```
public static IServiceCollection AddXData(this IServiceCollection services, Func<IDataOptions, IDataOptions> options = null)
```

Parameters

TYPE	NAME	DESCRIPTION
Microsoft.Extensions.DependencyInjection.IServiceCollection	services	Services collection
System.Func< IDataOptions , IDataOptions >	options	XData engine options

Returns

TYPE	DESCRIPTION
Microsoft.Extensions.DependencyInjection.IServiceCollection	XData engine interface

GetConcurrencyErrorId(String)

Returns concurrency error id for context

Declaration

```
public static int GetConcurrencyErrorId(string context)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	context	Context name

Returns

TYPE	DESCRIPTION
System.Int32	Concurrency error id

UseXData(IApplicationBuilder)

XData initialization

Declaration

```
public static IApplicationBuilder UseXData(this IApplicationBuilder builder)
```

Parameters

TYPE	NAME	DESCRIPTION
Microsoft.AspNetCore.Builder.IApplicationBuilder	builder	Application builder interface

Returns

TYPE	DESCRIPTION
Microsoft.AspNetCore.Builder.IApplicationBuilder	Application builder interface

UseXData(IServiceProvider)

XData initialization

Declaration

```
public static IServiceProvider UseXData(this IServiceProvider provider)
```

Parameters

TYPE	NAME	DESCRIPTION
System.IServiceProvider	provider	Service provider interface

Returns

TYPE	DESCRIPTION
System.IServiceProvider	Service provider interface

Class XDataPureLinqMethodAttribute

Attribute to mark XData pure LINQ methods with no realization

Inheritance

System.Object

System.Attribute

XDataPureLinqMethodAttribute

Inherited Members

System.Attribute.Equals(System.Object)

System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Module)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.GetHashCode()

System.Attribute.IsDefaultAttribute()

System.Attribute.IsDefined(System.Reflection.Assembly, System.Type)

System.Attribute.IsDefined(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type)

System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.Module, System.Type)

System.Attribute.IsDefined(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type)

System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.Match(System.Object)

System.Attribute.TypeId

System.Object.Equals(System.Object, System.Object)

System.Object.GetType()

System.Object.MemberwiseClone()

System.Object.ReferenceEquals(System.Object, System.Object)

System.Object.ToString()

Namespace: [XData](#)

Syntax

```
[AttributeUsage(AttributeTargets.Method)]  
public class XDataPureLinqMethodAttribute : Attribute
```

Extension Methods

[SerializationExtensions.ToXml<T>\(T\)](#)

[CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)

[CollectionsExtensions.AsEnum<T>\(T\)](#)

[DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)

[Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)

[TypeExtensions.ChangeType<T>\(Object\)](#)

[TypeExtensions.ChangeType\(Object, Type\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)

[SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)

[QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Namespace XData.Async

Classes

[AsyncExtensions](#)

Class AsyncExtensions

Inheritance

System.Object
AsyncExtensions

Inherited Members

System.Object.Equals(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.GetHashCode()
System.Object.GetType()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.ToString()

Namespace: [XData.Async](#)

Assembly: XData.docfx.dll

Syntax

```
public static class AsyncExtensions
```

Methods

AddAsync<TFrom, T>(IRepository<T>, IEnumerable<TFrom>, DataSubmitFlag)

Asynchronous wrapper over Add

Declaration

```
public static Task<bool> AddAsync<TFrom, T>(this IRepository<T> source, IEnumerable<TFrom> objects,
DataSubmitFlag flag = DataSubmitFlag.None)
    where TFrom : class where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
IRepository<T>	source	Repository to submit
System.Collections.Generic.IEnumerable<TFrom>	objects	Objects to submit
DataSubmitFlag	flag	Data submit flag

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Boolean>	Asynchronous wrapper over Submit

Type Parameters

NAME	DESCRIPTION
TFrom	
T	

AddAsync<TFrom, T>(IRepository<T>, IEnumerable<TFrom>, Mapping<TFrom, T>, DataSubmitFlag)

Asynchronous wrapper over Add

Declaration

```
public static Task<bool> AddAsync<TFrom, T>(this IRepository<T> source, IEnumerable<TFrom> objects,
Mapping<TFrom, T> mapping, DataSubmitFlag flag = DataSubmitFlag.None)
    where TFrom : class where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
IRepository<T>	source	Repository to submit
System.Collections.Generic.IEnumerable<TFrom>	objects	Objects to submit
Mapping<TFrom, T>	mapping	Property mapping
DataSubmitFlag	flag	Data submit flag

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Boolean>	Asynchronous wrapper over Submit

Type Parameters

NAME	DESCRIPTION
TFrom	
T	

AllAsync<T>(IQueryable<T>, Expression<Func<T, Boolean>>)

Asynchronous wrapper over All() LINQ function

Declaration

```
public static Task<bool> AllAsync<T>(this IQueryable<T> source, Expression<Func<T, bool>> predicate)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, System.Boolean>>	predicate	A function to test each element for a condition

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Boolean>	Asynchronous call over All() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

AllAsync<T>(IQueryable<T>, Expression<Func<T, Boolean>>, CancellationToken)

Asynchronous wrapper over All() LINQ function

Declaration

```
public static Task<bool> AllAsync<T>(this IQueryable<T> source, Expression<Func<T, bool>> predicate,
CancellationToken token)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, System.Boolean>>	predicate	A function to test each element for a condition
System.Threading.CancellationToken	token	Cancellation token

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Boolean>	Asynchronous call over All() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

AllAsync<T>(IDataQuery<T>, Expression<Func<T, Boolean>>)

Asynchronous wrapper over All() LINQ function

Declaration

```
public static Task<bool> AllAsync<T>(this IDataQuery<T> source, Expression<Func<T, bool>> predicate)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataQuery<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, System.Boolean>>	predicate	A function to test each element for a condition

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Boolean>	Asynchronous call over All() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

AnyAsync<T>(IQueryable<T>)

Asynchronous wrapper over Any() LINQ function

Declaration

```
public static Task<bool> AnyAsync<T>(this IQueryable<T> source)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Boolean>	Asynchronous call over Any() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

AnyAsync<T>(IQueryable<T>, Expression<Func<T, Boolean>>)

Asynchronous wrapper over Any() LINQ function

Declaration

```
public static Task<bool> AnyAsync<T>(this IQueryable<T> source, Expression<Func<T, bool>> predicate)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, System.Boolean>>	predicate	A function to test each element for a condition

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Boolean>	Asynchronous call over Any() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

AnyAsync<T>(IQueryable<T>, Expression<Func<T, Boolean>>, CancellationToken)

Asynchronous wrapper over Any() LINQ function

Declaration

```
public static Task<bool> AnyAsync<T>(this IQueryable<T> source, Expression<Func<T, bool>> predicate,
CancellationToken token)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, System.Boolean>>	predicate	A function to test each element for a condition
System.Threading.CancellationToken	token	Cancellation token

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Boolean>	Asynchronous call over Any() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

AnyAsync<T>(IDataQuery<T>, Expression<Func<T, Boolean>>)

Asynchronous wrapper over Any() LINQ function

Declaration

```
public static Task<bool> AnyAsync<T>(this IDataQuery<T> source, Expression<Func<T, bool>> predicate)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataQuery <T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, System.Boolean>>	predicate	A function to test each element for a condition

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Boolean>	Asynchronous call over Any() LINQ function

Type Parameters

NAME	DESCRIPTION

NAME	DESCRIPTION
T	Object type

AverageAsync<T>(IQueryable<T>, Expression<Func<T, Decimal>>)

Asynchronous wrapper over Average() LINQ function

Declaration

```
public static Task<decimal> AverageAsync<T>(this IQueryable<T> source, Expression<Func<T, decimal>> selector)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, Decimal >>	selector	A projection function to apply to each element

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task< Decimal >	Asynchronous call over Average() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

AverageAsync<T>(IQueryable<T>, Expression<Func<T, Decimal>>, CancellationToken)

Asynchronous wrapper over Average() LINQ function

Declaration

```
public static Task<decimal> AverageAsync<T>(this IQueryable<T> source, Expression<Func<T, decimal>> selector,
CancellationToken token)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, Decimal >>	selector	A projection function to apply to each element

TYPE	NAME	DESCRIPTION
System.Threading.CancellationToken	token	Cancellation token

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<Decimal>	Asynchronous call over Average() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

AverageAsync<T>(IQueryable<T>, Expression<Func<T, Double>>)

Asynchronous wrapper over Average() LINQ function

Declaration

```
public static Task<double> AverageAsync<T>(this IQueryable<T> source, Expression<Func<T, double>> selector)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, Double>>	selector	A projection function to apply to each element

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<Double>	Asynchronous call over Average() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

AverageAsync<T>(IQueryable<T>, Expression<Func<T, Double>>, CancellationToken)

Asynchronous wrapper over Average() LINQ function

Declaration

```
public static Task<double> AverageAsync<T>(this IQueryable<T> source, Expression<Func<T, double>> selector,
CancellationToken token)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, Double>>	selector	A projection function to apply to each element
System.Threading.CancellationToken	token	Cancellation token

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<Double>	Asynchronous call over Average() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

AverageAsync<T>(IQueryable<T>, Expression<Func<T, Int32>>)

Asynchronous wrapper over Average() LINQ function

Declaration

```
public static Task<double> AverageAsync<T>(this IQueryable<T> source, Expression<Func<T, int>> selector)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, System.Int32>>	selector	A projection function to apply to each element

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<Double>	Asynchronous call over Average() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

AverageAsync<T>(IQueryable<T>, Expression<Func<T, Int32>>, CancellationToken)

Asynchronous wrapper over Average() LINQ function

Declaration

```
public static Task<double> AverageAsync<T>(this IQueryable<T> source, Expression<Func<T, int>> selector,
CancellationToken token)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, System.Int32>>	selector	A projection function to apply to each element
System.Threading.CancellationToken	token	Cancellation token

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<Double>	Asynchronous call over Average() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

AverageAsync<T>(IQueryable<T>, Expression<Func<T, Int64>>)

Asynchronous wrapper over Average() LINQ function

Declaration

```
public static Task<double> AverageAsync<T>(this IQueryable<T> source, Expression<Func<T, long>> selector)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func<T, System.Int64>>	selector	A projection function to apply to each element

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<Double>	Asynchronous call over Average() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

AverageAsync<T>(IQueryable<T>, Expression<Func<T, Int64>>, CancellationToken)

Asynchronous wrapper over Average() LINQ function

Declaration

```
public static Task<double> AverageAsync<T>(this IQueryable<T> source, Expression<Func<T, long>> selector,
CancellationToken token)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, System.Int64>>	selector	A projection function to apply to each element
System.Threading.CancellationToken	token	Cancellation token

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<Double>	Asynchronous call over Average() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

AverageAsync<T>(IQueryable<T>, Expression<Func<T, Nullable<Decimal>>>)

Asynchronous wrapper over Average() LINQ function

Declaration

```
public static Task<decimal?> AverageAsync<T>(this IQueryable<T> source, Expression<Func<T, decimal?>> selector)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, System.Nullable<Decimal>>>	selector	A projection function to apply to each element

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Nullable<Decimal>>	Asynchronous call over Average() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

AverageAsync<T>(IQueryable<T>, Expression<Func<T, Nullable<Decimal>>>, CancellationToken)

Asynchronous wrapper over Average() LINQ function

Declaration

```
public static Task<decimal?> AverageAsync<T>(this IQueryable<T> source, Expression<Func<T, decimal?>> selector, CancellationToken token)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, System.Nullable<Decimal>>>	selector	A projection function to apply to each element
System.Threading.CancellationToken	token	Cancellation token

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Nullable<Decimal>>	Asynchronous call over Average() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

AverageAsync<T>(IQueryable<T>, Expression<Func<T, Nullable<Double>>>)

Asynchronous wrapper over Average() LINQ function

Declaration

```
public static Task<double?> AverageAsync<T>(this IQueryable<T> source, Expression<Func<T, double?>> selector)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, System.Nullable<Double>>>	selector	A projection function to apply to each element

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Nullable<Double>>	Asynchronous call over Average() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

AverageAsync<T>(IQueryable<T>, Expression<Func<T, Nullable<Double>>>, CancellationToken)

Asynchronous wrapper over Average() LINQ function

Declaration

```
public static Task<double?> AverageAsync<T>(this IQueryable<T> source, Expression<Func<T, double?>> selector, CancellationToken token)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, System.Nullable<Double>>>	selector	A projection function to apply to each element
System.Threading.CancellationToken	token	Cancellation token

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Nullable<Double>>	Asynchronous call over Average() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

AverageAsync<T>(IQueryable<T>, Expression<Func<T, Nullable<Int32>>>)

Asynchronous wrapper over Average() LINQ function

Declaration

```
public static Task<double?> AverageAsync<T>(this IQueryable<T> source, Expression<Func<T, int?>> selector)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, System.Nullable<System.Int32>>>	selector	A projection function to apply to each element

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Nullable<Double>>	Asynchronous call over Average() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

AverageAsync<T>(IQueryable<T>, Expression<Func<T, Nullable<Int32>>>, CancellationToken)

Asynchronous wrapper over Average() LINQ function

Declaration

```
public static Task<double?> AverageAsync<T>(this IQueryable<T> source, Expression<Func<T, int?>> selector,
CancellationToken token)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, System.Nullable<System.Int32>>>>	selector	A projection function to apply to each element
System.Threading.CancellationToken	token	Cancellation token

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Nullable<Double>>>	Asynchronous call over Average() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

AverageAsync<T>(IQueryable<T>, Expression<Func<T, Nullable<Int64>>>>)

Asynchronous wrapper over Average() LINQ function

Declaration

```
public static Task<double?> AverageAsync<T>(this IQueryable<T> source, Expression<Func<T, long?>> selector)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func<T, System.Nullable<System.Int64>>>	selector	A projection function to apply to each element

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Nullable<Double>>	Asynchronous call over Average() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

AverageAsync<T>(IQueryable<T>, Expression<Func<T, Nullable<Int64>>>, CancellationToken)

Asynchronous wrapper over Average() LINQ function

Declaration

```
public static Task<double?> AverageAsync<T>(this IQueryable<T> source, Expression<Func<T, long?>> selector,
CancellationToken token)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, System.Nullable<System.Int64>>>	selector	A projection function to apply to each element
System.Threading.CancellationToken	token	Cancellation token

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Nullable<Double>>	Asynchronous call over Average() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

AverageAsync<T>(IQueryable<T>, Expression<Func<T, Nullable<Single>>>)

Asynchronous wrapper over Average() LINQ function

Declaration

```
public static Task<float?> AverageAsync<T>(this IQueryable<T> source, Expression<Func<T, float?>> selector)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, System.Nullable<Single>>>>	selector	A projection function to apply to each element

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Nullable<Single>>>	Asynchronous call over Average() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

AverageAsync<T>(IQueryable<T>, Expression<Func<T, Nullable<Single>>>, CancellationToken)

Asynchronous wrapper over Average() LINQ function

Declaration

```
public static Task<float?> AverageAsync<T>(this IQueryable<T> source, Expression<Func<T, float?>> selector, CancellationToken token)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, System.Nullable<Single>>>>	selector	A projection function to apply to each element
System.Threading.CancellationToken	token	Cancellation token

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Nullable<Single>>	Asynchronous call over Average() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

AverageAsync<T>(IQueryable<T>, Expression<Func<T, Single>>)

Asynchronous wrapper over Average() LINQ function

Declaration

```
public static Task<float> AverageAsync<T>(this IQueryable<T> source, Expression<Func<T, float>> selector)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, Single>>	selector	A projection function to apply to each element

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<Single>	Asynchronous call over Average() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

AverageAsync<T>(IQueryable<T>, Expression<Func<T, Single>>, CancellationToken)

Asynchronous wrapper over Average() LINQ function

Declaration

```
public static Task<float> AverageAsync<T>(this IQueryable<T> source, Expression<Func<T, float>> selector,
CancellationToken token)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, Single >>	selector	A projection function to apply to each element
System.Threading.CancellationToken	token	Cancellation token

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task< Single >	Asynchronous call over Average() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

AverageAsync<T>(IDataQuery<T>, Expression<Func<T, Decimal>>)

Asynchronous wrapper over Average() LINQ function

Declaration

```
public static Task<decimal> AverageAsync<T>(this IDataQuery<T> source, Expression<Func<T, decimal>> selector)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataQuery <T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, Decimal >>	selector	A projection function to apply to each element

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task< Decimal >	Asynchronous call over Average() LINQ function

Type Parameters

NAME	DESCRIPTION

NAME	DESCRIPTION
T	Object type

AverageAsync<T>(IDataQuery<T>, Expression<Func<T, Double>>)

Asynchronous wrapper over Average() LINQ function

Declaration

```
public static Task<double> AverageAsync<T>(this IDataQuery<T> source, Expression<Func<T, double>> selector)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataQuery<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, Double >>	selector	A projection function to apply to each element

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task< Double >	Asynchronous call over Average() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

AverageAsync<T>(IDataQuery<T>, Expression<Func<T, Int32>>)

Asynchronous wrapper over Average() LINQ function

Declaration

```
public static Task<double> AverageAsync<T>(this IDataQuery<T> source, Expression<Func<T, int>> selector)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataQuery<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, System.Int32>>	selector	A projection function to apply to each element

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<Double>	Asynchronous call over Average() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

AverageAsync<T>(IDataQuery<T>, Expression<Func<T, Int64>>)

Asynchronous wrapper over Average() LINQ function

Declaration

```
public static Task<double> AverageAsync<T>(this IDataQuery<T> source, Expression<Func<T, long>> selector)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataQuery<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, System.Int64>>	selector	A projection function to apply to each element

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<Double>	Asynchronous call over Average() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

AverageAsync<T>(IDataQuery<T>, Expression<Func<T, Nullable<Decimal>>>)

Asynchronous wrapper over Average() LINQ function

Declaration

```
public static Task<decimal?> AverageAsync<T>(this IDataQuery<T> source, Expression<Func<T, decimal?>> selector)
```

Parameters

TYPE	NAME	DESCRIPTION
<code>IDataQuery<T></code>	source	Source object sequence
<code>System.Linq.Expressions.Expression<System.Func<T, System.Nullable<Decimal>>></code>	selector	A projection function to apply to each element

Returns

TYPE	DESCRIPTION
<code>System.Threading.Tasks.Task<System.Nullable<Decimal>></code>	Asynchronous call over Average() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

AverageAsync<T>(IDataQuery<T>, Expression<Func<T, Nullable<Double>>>)

Asynchronous wrapper over Average() LINQ function

Declaration

```
public static Task<double?> AverageAsync<T>(this IDataQuery<T> source, Expression<Func<T, double?>> selector)
```

Parameters

TYPE	NAME	DESCRIPTION
<code>IDataQuery<T></code>	source	Source object sequence
<code>System.Linq.Expressions.Expression<System.Func<T, System.Nullable<Double>>></code>	selector	A projection function to apply to each element

Returns

TYPE	DESCRIPTION
<code>System.Threading.Tasks.Task<System.Nullable<Double>></code>	Asynchronous call over Average() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

AverageAsync<T>(IDataQuery<T>, Expression<Func<T, Nullable<Int32>>>)

Asynchronous wrapper over Average() LINQ function

Declaration

```
public static Task<double?> AverageAsync<T>(this IDataQuery<T> source, Expression<Func<T, int?>> selector)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataQuery<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, System.Nullable<System.Int32>>>	selector	A projection function to apply to each element

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Nullable<Double>>	Asynchronous call over Average() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

AverageAsync<T>(IDataQuery<T>, Expression<Func<T, Nullable<Int64>>>)

Asynchronous wrapper over Average() LINQ function

Declaration

```
public static Task<double?> AverageAsync<T>(this IDataQuery<T> source, Expression<Func<T, long?>> selector)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataQuery<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, System.Nullable<System.Int64>>>	selector	A projection function to apply to each element

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Nullable<Double>>	Asynchronous call over Average() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

AverageAsync<T>(IDataQuery<T>, Expression<Func<T, Nullable<Single>>>)

Asynchronous wrapper over Average() LINQ function

Declaration

```
public static Task<float?> AverageAsync<T>(this IDataQuery<T> source, Expression<Func<T, float?>> selector)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataQuery<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, System.Nullable<Single>>>	selector	A projection function to apply to each element

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Nullable<Single>>	Asynchronous call over Average() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

AverageAsync<T>(IDataQuery<T>, Expression<Func<T, Single>>)

Asynchronous wrapper over Average() LINQ function

Declaration

```
public static Task<float> AverageAsync<T>(this IDataQuery<T> source, Expression<Func<T, float>> selector)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataQuery<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, Single>>	selector	A projection function to apply to each element

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task< Single >	Asynchronous call over Average() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

ClearAsync<T>(IRepository<T>, Expression<Func<T, Boolean>>, DataSubmitFlag)

Asynchronous wrapper over Clear

Declaration

```
public static Task<bool> ClearAsync<T>(this IRepository<T> repository, Expression<Func<T, bool>> predicate,
DataSubmitFlag flag = DataSubmitFlag.None)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
IRepository<T>	repository	Repository to submit
System.Linq.Expressions.Expression<System.Func<T, System.Boolean>>	predicate	Predicate to filter repository before clear
DataSubmitFlag	flag	Data submit flag

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Boolean>	Asynchronous wrapper over Submit

Type Parameters

NAME	DESCRIPTION
T	

ClearAsync<T>(IRepository<T>, DataSubmitFlag)

Asynchronous wrapper over Clear

Declaration


```
public static Task<bool> ClearAsync<T>(this IRepository<T> repository, DataSubmitFlag flag =
DataSubmitFlag.None)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
IRepository<T>	repository	Repository to submit
DataSubmitFlag	flag	Data submit flag

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Boolean>	Asynchronous wrapper over Submit

Type Parameters

NAME	DESCRIPTION
T	

ContainsAsync<T>(IQueryable<T>, T, IEqualityComparer<T>)

Asynchronous wrapper over Contains() LINQ function

Declaration

```
public static Task<bool> ContainsAsync<T>(this IQueryable<T> source, T item, IEqualityComparer<T> comparer)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence
T	item	The object to locate in the sequence.
System.Collections.Generic.IEqualityComparer<T>	comparer	Object comparer

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Boolean>	Asynchronous call over Contains() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

ContainsAsync<T>(IQueryable<T>, T, IEqualityComparer<T>, CancellationToken)

Asynchronous wrapper over Contains() LINQ function

Declaration

```
public static Task<bool> ContainsAsync<T>(this IQueryable<T> source, T item, IEqualityComparer<T> comparer,
CancellationToken token)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence
T	item	The object to locate in the sequence.
System.Collections.Generic.IEqualityComparer<T>	comparer	Object comparer
System.Threading.CancellationToken	token	Cancellation token

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Boolean>	Asynchronous call over Contains() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

ContainsAsync<T>(IQueryable<T>, T, Func<T, Object>[])

Asynchronous wrapper over Contains() LINQ function

Declaration

```
public static Task<bool> ContainsAsync<T>(this IQueryable<T> source, T item, params Func<T, object>[]
properties)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence
T	item	The object to locate in the sequence.
System.Func<T, System.Object>[]	properties	Object properties to compare

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Boolean>	Asynchronous call over Contains() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

ContainsAsync<T>(IDataQuery<T>, T)

Asynchronous wrapper over Contains() LINQ function

Declaration

```
public static Task<bool> ContainsAsync<T>(this IDataQuery<T> source, T item)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataQuery<T>	source	Source object sequence
T	item	The object to locate in the sequence.

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Boolean>	Asynchronous call over Contains() LINQ function

Type Parameters

NAME	DESCRIPTION

NAME	DESCRIPTION
T	Object type

ContainsAsync<T>(IDataQuery<T>, T, IEqualityComparer<T>)

Asynchronous wrapper over Contains() LINQ function

Declaration

```
public static Task<bool> ContainsAsync<T>(this IDataQuery<T> source, T item, IEqualityComparer<T> comparer)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataQuery<T>	source	Source object sequence
T	item	The object to locate in the sequence.
System.Collections.Generic.IEqualityComparer<T>	comparer	Object comparer

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Boolean>	Asynchronous call over Contains() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

CountAsync<T>(IQueryable<T>)

Asynchronous wrapper over Count() LINQ function

Declaration

```
public static Task<int> CountAsync<T>(this IQueryable<T> source)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Int32>	Asynchronous call over Count() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

CountAsync<T>(IQueryable<T>, Expression<Func<T, Boolean>>)

Asynchronous wrapper over Count() LINQ function

Declaration

```
public static Task<int> CountAsync<T>(this IQueryable<T> source, Expression<Func<T, bool>> predicate)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, System.Boolean>>	predicate	A function to test each element for a condition

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Int32>	Asynchronous call over Count() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

CountAsync<T>(IQueryable<T>, Expression<Func<T, Boolean>>, CancellationToken)

Asynchronous wrapper over Count() LINQ function

Declaration

```
public static Task<int> CountAsync<T>(this IQueryable<T> source, Expression<Func<T, bool>> predicate,
CancellationToken token)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, System.Boolean>>	predicate	A function to test each element for a condition
System.Threading.CancellationToken	token	Cancellation token

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Int32>	Asynchronous call over Count() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

CountAsync<T>(IDataQuery<T>)

Asynchronous wrapper over Count() LINQ function

Declaration

```
public static Task<int> CountAsync<T>(this IDataQuery<T> source)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataQuery<T>	source	Source object sequence

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Int32>	Asynchronous call over Count() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

CountAsync<T>(IDataQuery<T>, Expression<Func<T, Boolean>>)

Asynchronous wrapper over Count() LINQ function

Declaration

```
public static Task<int> CountAsync<T>(this IQueryable<T> source, Expression<Func<T, bool>> predicate)
```

Parameters

TYPE	NAME	DESCRIPTION
IQueryable<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, System.Boolean>>	predicate	A function to test each element for a condition

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Int32>	Asynchronous call over Count() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

ExecuteAsync<T>(T, Expression<Func<CustomLogic<T>>>, IDictionary<String, Action<Byte[]>>)

Asynchronous wrapper over Execute

Declaration

```
public static Task<bool> ExecuteAsync<T>(this T obj, Expression<Func<CustomLogic<T>>> routine,  
IDictionary<string, Action<byte[]>> post)  
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
T	obj	Data object
System.Linq.Expressions.Expression<System.Func< CustomLogic<T> >>>	routine	Custom logic name
System.Collections.Generic.IDictionary<System.String, System.Action< Byte[] >>	post	Post data handlers

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Boolean>	Asynchronous wrapper over Submit

Type Parameters

NAME	DESCRIPTION
T	

ExecuteAsync<T>(T, Expression<Func<CustomLogic<T>>>, IDictionary<String, Action<Byte[]>>, IDictionary<String, Func<Byte[], Byte[]>>)

Asynchronous wrapper over Execute

Declaration

```
public static Task<bool> ExecuteAsync<T>(this T obj, Expression<Func<CustomLogic<T>>> routine,
IDictionary<string, Action<byte[]>> post, IDictionary<string, Func<byte[], byte[]>> callback)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
T	obj	Data object
System.Linq.Expressions.Expression<System.Func<CustomLogic<T>>>	routine	Custom logic name
System.Collections.Generic.IDictionary<System.String, System.Action<Byte[]>>	post	Post data handlers
System.Collections.Generic.IDictionary<System.String, System.Func<Byte[], Byte[]>>	callback	Callback handlers

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Boolean>	Asynchronous wrapper over Submit

Type Parameters

NAME	DESCRIPTION
T	

ExecuteAsync<T>(T, Expression<Func<CustomLogic<T>>>, IDictionary<String, Func<Byte[], Byte[]>>)

Asynchronous wrapper over Execute

Declaration


```
public static Task<bool> ExecuteAsync<T>(this T obj, Expression<Func<CustomLogic<T>>> routine,
IDictionary<string, Func<byte[], byte[]>> callback)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
T	obj	Data object
System.Linq.Expressions.Expression<System.Func<CustomLogic<T>>>	routine	Custom logic name
System.Collections.Generic.IDictionary<System.String, System.Func<Byte[], Byte[]>>	callback	Callback handlers

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Boolean>	Asynchronous wrapper over Submit

Type Parameters

NAME	DESCRIPTION
T	

ExecuteAsync<T>(T, String, IDictionary<String, Action<Byte[]>>)

Asynchronous wrapper over Execute

Declaration

```
public static Task<bool> ExecuteAsync<T>(this T obj, string key, IDictionary<string, Action<byte[]>> post)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
T	obj	Data object
System.String	key	Custom logic name
System.Collections.Generic.IDictionary<System.String, System.Action<Byte[]>>	post	Post data handlers

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Boolean>	Asynchronous wrapper over Submit

Type Parameters

NAME	DESCRIPTION
T	

ExecuteAsync<T>(T, String, IDictionary<String, Action<Byte[]>>, IDictionary<String, Func<Byte[], Byte[]>>)

Asynchronous wrapper over Execute

Declaration

```
public static Task<bool> ExecuteAsync<T>(this T obj, string key, IDictionary<string, Action<byte[]>> post,
IDictionary<string, Func<byte[], byte[]>> callback)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
T	obj	Data object
System.String	key	Custom logic name
System.Collections.Generic.IDictionary<System.String, System.Action<Byte[]>>	post	Post data handlers
System.Collections.Generic.IDictionary<System.String, System.Func<Byte[], Byte[]>>	callback	Callback handlers

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Boolean>	Asynchronous wrapper over Submit

Type Parameters

NAME	DESCRIPTION
T	

ExecuteAsync<T>(T, String, IDictionary<String, Func<Byte[], Byte[]>>)

Asynchronous wrapper over Execute

Declaration

```
public static Task<bool> ExecuteAsync<T>(this T obj, string key, IDictionary<string, Func<byte[], byte[]>>
callback)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
T	obj	Data object
System.String	key	Custom logic name
System.Collections.Generic.IDictionary<System.String, System.Func<Byte[], Byte[]>>	callback	Callback handlers

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Boolean>	Asynchronous wrapper over Submit

Type Parameters

NAME	DESCRIPTION
T	

ExecuteAsync<T>(ICollection<T>, Expression<Func<CustomLogic<T>>>, IDictionary<String, Action<Byte[]>>)

Asynchronous wrapper over Execute

Declaration

```
public static Task<bool> ExecuteAsync<T>(this ICollection<T> objects, Expression<Func<CustomLogic<T>>>
routine, IDictionary<string, Action<byte[]>> post)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
System.Collections.Generic.ICollection<T>	objects	Data objects collection
System.Linq.Expressions.Expression<System.Func<CustomLogic<T>>>	routine	Custom logic name
System.Collections.Generic.IDictionary<System.String, System.Action<Byte[]>>	post	Post data handlers

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Boolean>	Asynchronous wrapper over Submit

Type Parameters

NAME	DESCRIPTION
T	

ExecuteAsync<T>(ICollection<T>, Expression<Func<CustomLogic<T>>>, IDictionary<String, Action<Byte[]>>, IDictionary<String, Func<Byte[], Byte[]>>)

Asynchronous wrapper over Execute

Declaration

```
public static Task<bool> ExecuteAsync<T>(this ICollection<T> objects, Expression<Func<CustomLogic<T>>>
routine, IDictionary<string, Action<byte[]>> post, IDictionary<string, Func<byte[], byte[]>> callback)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
System.Collections.Generic.ICollection<T>	objects	Data objects collection
System.Linq.Expressions.Expression<System.Func<CustomLogic<T>>>	routine	Custom logic name
System.Collections.Generic.IDictionary<System.String, System.Action<Byte[]>>	post	Post data handlers
System.Collections.Generic.IDictionary<System.String, System.Func<Byte[], Byte[]>>	callback	Callback handlers

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Boolean>	Asynchronous wrapper over Submit

Type Parameters

NAME	DESCRIPTION
T	

ExecuteAsync<T>(ICollection<T>, Expression<Func<CustomLogic<T>>>, IDictionary<String, Func<Byte[], Byte[]>>)

Asynchronous wrapper over Execute

Declaration

```
public static Task<bool> ExecuteAsync<T>(this ICollection<T> objects, Expression<Func<CustomLogic<T>>>
routine, IDictionary<string, Func<byte[], byte[]>> callback)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
System.Collections.Generic.ICollection<T>	objects	Data objects collection
System.Linq.Expressions.Expression<System.Func<CustomLogic<T>>>	routine	Custom logic name
System.Collections.Generic.IDictionary<System.String, System.Func<Byte[], Byte[]>>	callback	Callback handlers

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Boolean>	Asynchronous wrapper over Submit

Type Parameters

NAME	DESCRIPTION
T	

ExecuteAsync<T>(ICollection<T>, String, IDictionary<String, Action<Byte[]>>)

Asynchronous wrapper over Execute

Declaration

```
public static Task<bool> ExecuteAsync<T>(this ICollection<T> objects, string key, IDictionary<string,
Action<byte[]>> post)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
System.Collections.Generic.ICollection<T>	objects	Data objects collection
System.String	key	Custom logic name
System.Collections.Generic.IDictionary<System.String, System.Action<Byte[]>>	post	Post data handlers

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Boolean>	Asynchronous wrapper over Submit

Type Parameters

NAME	DESCRIPTION
T	

ExecuteAsync<T>(ICollection<T>, String, IDictionary<String, Action<Byte[]>>, IDictionary<String, Func<Byte[], Byte[]>>)

Asynchronous wrapper over Execute

Declaration

```
public static Task<bool> ExecuteAsync<T>(this ICollection<T> objects, string key, IDictionary<string,
Action<byte[]>> post, IDictionary<string, Func<byte[], byte[]>> callback)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
System.Collections.Generic.ICollection<T>	objects	Data objects collection
System.String	key	Custom logic name
System.Collections.Generic.IDictionary<System.String, System.Action<Byte[]>>	post	Post data handlers
System.Collections.Generic.IDictionary<System.String, System.Func<Byte[], Byte[]>>	callback	Callback handlers

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Boolean>	Asynchronous wrapper over Submit

Type Parameters

NAME	DESCRIPTION
T	

ExecuteAsync<T>(ICollection<T>, String, IDictionary<String, Func<Byte[], Byte[]>>)

Asynchronous wrapper over Execute

Declaration

```
public static Task<bool> ExecuteAsync<T>(this ICollection<T> objects, string key, IDictionary<string,
Func<byte[], byte[]>> callback)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
System.Collections.Generic.ICollection<T>	objects	Data objects collection
System.String	key	Custom logic name
System.Collections.Generic.IDictionary<System.String, System.Func< Byte[] , Byte[] >>	callback	Callback handlers

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Boolean>	Asynchronous wrapper over Submit

Type Parameters

NAME	DESCRIPTION
T	

ExecuteNonQueryAsync(IDataScope, Query)

Asynchronous wrapper over [ExecuteNonQuery\(Query\)](#) call

Declaration

```
public static Task ExecuteNonQueryAsync(this IDataScope dataScope, Query query)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataScope	dataScope	Data scope
Query	query	Query

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task	Asynchronous call over ExecuteNonQuery

ExecuteScalarAsync<TResult>(IDataScope, Query)

Asynchronous wrapper over [ExecuteScalar<TResult>\(Query\)](#) call

Declaration

```
public static Task<TResult> ExecuteScalarAsync<TResult>(this IDataScope dataScope, Query query)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataScope	dataScope	Data scope
Query	query	Query

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<TResult>	Asynchronous call over ExecuteScalar

Type Parameters

NAME	DESCRIPTION
TResult	

[ExecuteSqlBlockAsync](#)([IDataScope](#), [Expression<SqlBlockDefinition>](#), [Expression<Func<ISqlParamAdapter, ISqlParam>>\[\]](#))

Asynchronous wrapper over SQL block execution

Declaration

```
public static Task ExecuteSqlBlockAsync(this IDataScope dataScope, Expression<SqlBlockDefinition> block, params Expression<Func<ISqlParamAdapter, ISqlParam>>[] parameters)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataScope	dataScope	Data scope
System.Linq.Expressions.Expression< SqlBlockDefinition >	block	Sql block structure
System.Linq.Expressions.Expression<System.Func< ISqlParamAdapter , ISqlParam >>[]	parameters	Sql block parameters

Returns

TYPE	DESCRIPTION

TYPE	DESCRIPTION
System.Threading.Tasks.Task	Asynchronous wrapper over SQL block execution

ExecuteSqlBlockAsync(IDataScope, CancellationToken, Expression<SqlBlockDefinition>, Expression<Func<ISqlParamAdapter, ISqlParam>>[])

Asynchronous wrapper over SQL block execution

Declaration

```
public static Task ExecuteSqlBlockAsync(this IDataScope dataScope, CancellationToken token,
Expression<SqlBlockDefinition> block, params Expression<Func<ISqlParamAdapter, ISqlParam>>[] parameters)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataScope	dataScope	Data scope
System.Threading.CancellationToken	token	Cancellation token
System.Linq.Expressions.Expression< SqlBlockDefinition >	block	Sql block structure
System.Linq.Expressions.Expression<System.Func< ISqlParamAdapter , ISqlParam >>[]	parameters	Sql block parameters

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task	Asynchronous wrapper over SQL block execution

FillTableAsync(IDataScope, Query, CommandBehavior)

Asynchronous wrapper over [FillTable\(Query, CommandBehavior\)](#) call

Declaration

```
public static Task<DataTable> FillTableAsync(this IDataScope dataScope, Query query, CommandBehavior behavior
= CommandBehavior.Default)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataScope	dataScope	Data scope
Query	query	Query

TYPE	NAME	DESCRIPTION
System.Data.CommandBehavior	behavior	Command behavior

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Data.DataTable>	Asynchronous call over FillTable

FirstAsync<T>(IEnumerable<T>)

Asynchronous wrapper over First() LINQ function

Declaration

```
public static Task<T> FirstAsync<T>(this IEnumerable<T> source)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<T>	Asynchronous call over First() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

FirstAsync<T>(IEnumerable<T>, Expression<Func<T, Boolean>>)

Asynchronous wrapper over First() LINQ function

Declaration

```
public static Task<T> FirstAsync<T>(this IEnumerable<T> source, Expression<Func<T, bool>> predicate)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func<T, System.Boolean>>	predicate	A function to test each element for a condition

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<T>	Asynchronous call over First() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

FirstAsync<T>(IQueryable<T>, Expression<Func<T, Boolean>>, CancellationToken)

Asynchronous wrapper over First() LINQ function

Declaration

```
public static Task<T> FirstAsync<T>(this IQueryable<T> source, Expression<Func<T, bool>> predicate,
CancellationToken token)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, System.Boolean>>	predicate	A function to test each element for a condition
System.Threading.CancellationToken	token	Cancellation token

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<T>	Asynchronous call over First() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

FirstAsync<T>(IDataQuery<T>)

Asynchronous wrapper over First() LINQ function

Declaration

```
public static Task<T> FirstAsync<T>(this IQueryable<T> source)
```

Parameters

TYPE	NAME	DESCRIPTION
IQueryable<T>	source	Source object sequence

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<T>	Asynchronous call over First() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

FirstAsync<T>(IQueryable<T>, Func<JsonSettings<T>, JsonSettings<T>>)

Asynchronous wrapper over First() LINQ function

Declaration

```
public static Task<T> FirstAsync<T>(this IQueryable<T> source, Func<JsonSettings<T>, JsonSettings<T>> settings)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
IQueryable<T>	source	Source object sequence
System.Func<JsonSettings<T>, JsonSettings<T>>	settings	JSON serialization settings

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<T>	Asynchronous call over First() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

FirstAsync<T>(IDataQuery<T>, Expression<Func<T, Boolean>>)

Asynchronous wrapper over First() LINQ function

Declaration

```
public static Task<T> FirstAsync<T>(this IDataQuery<T> source, Expression<Func<T, bool>> predicate)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataQuery<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, System.Boolean>>	predicate	A function to test each element for a condition

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<T>	Asynchronous call over First() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

FirstAsync<T>(IDataQuery<T>, Expression<Func<T, Boolean>>, Func<JsonSettings<T>, JsonSettings<T>>)

Asynchronous wrapper over First() LINQ function

Declaration

```
public static Task<T> FirstAsync<T>(this IDataQuery<T> source, Expression<Func<T, bool>> predicate,
Func<JsonSettings<T>, JsonSettings<T>> settings)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
IDataQuery<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, System.Boolean>>	predicate	A function to test each element for a condition

TYPE	NAME	DESCRIPTION
System.Func<JsonSettings<T>, JsonSettings<T>>	settings	JSON serialization settings

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<T>	Asynchronous call over First() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

FirstOrDefaultAsync<T>(IQueryable<T>)

Asynchronous wrapper over FirstOrDefault() LINQ function

Declaration

```
public static Task<T> FirstOrDefaultAsync<T>(this IQueryable<T> source)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<T>	Asynchronous call over FirstOrDefault() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

FirstOrDefaultAsync<T>(IQueryable<T>, Expression<Func<T, Boolean>>)

Asynchronous wrapper over FirstOrDefault() LINQ function

Declaration

```
public static Task<T> FirstOrDefaultAsync<T>(this IQueryable<T> source, Expression<Func<T, bool>> predicate)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, System.Boolean>>	predicate	A function to test each element for a condition

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<T>	Asynchronous call over FirstOrDefault() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

FirstOrDefaultAsync<T>(IQueryable<T>, Expression<Func<T, Boolean>>, CancellationToken)

Asynchronous wrapper over FirstOrDefault() LINQ function

Declaration

```
public static Task<T> FirstOrDefaultAsync<T>(this IQueryable<T> source, Expression<Func<T, bool>> predicate,
CancellationToken token)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, System.Boolean>>	predicate	A function to test each element for a condition
System.Threading.CancellationToken	token	Cancellation token

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<T>	Asynchronous call over FirstOrDefault() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

FirstOrDefaultAsync<T>(IDataQuery<T>)

Asynchronous wrapper over FirstOrDefault() LINQ function

Declaration

```
public static Task<T> FirstOrDefaultAsync<T>(this IDataQuery<T> source)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataQuery<T>	source	Source object sequence

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<T>	Asynchronous call over FirstOrDefault() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

FirstOrDefaultAsync<T>(IDataQuery<T>, Func<JsonSettings<T>, JsonSettings<T>>)

Asynchronous wrapper over FirstOrDefault() LINQ function

Declaration

```
public static Task<T> FirstOrDefaultAsync<T>(this IDataQuery<T> source, Func<JsonSettings<T>, JsonSettings<T>> settings)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
IDataQuery<T>	source	Source object sequence
System.Func< JsonSettings<T> , JsonSettings<T> >	settings	JSON serialization settings

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<T>	Asynchronous call over FirstOrDefault() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

FirstOrDefaultAsync<T>(IDataQuery<T>, Expression<Func<T, Boolean>>)

Asynchronous wrapper over FirstOrDefault() LINQ function

Declaration

```
public static Task<T> FirstOrDefaultAsync<T>(this IDataQuery<T> source, Expression<Func<T, bool>> predicate)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataQuery<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, System.Boolean>>	predicate	A function to test each element for a condition

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<T>	Asynchronous call over FirstOrDefault() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

FirstOrDefaultAsync<T>(IDataQuery<T>, Expression<Func<T, Boolean>>, Func<JsonSettings<T>, JsonSettings<T>>)

Asynchronous wrapper over FirstOrDefault() LINQ function

Declaration

```
public static Task<T> FirstOrDefaultAsync<T>(this IDataQuery<T> source, Expression<Func<T, bool>> predicate,
Func<JsonSettings<T>, JsonSettings<T>> settings)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
IDataQuery<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, System.Boolean>>	predicate	A function to test each element for a condition
System.Func< JsonSettings<T> , JsonSettings<T> >	settings	JSON serialization settings

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<T>	Asynchronous call over FirstOrDefault() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

ForEachAsync<T>(IQueryable<T>, Action<T>)

Asynchronous wrapper over query result iteration

Declaration

```
public static Task ForEachAsync<T>(this IQueryable<T> source, Action<T> body)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence
System.Action<T>	body	Iteration logic

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task	Asynchronous call over query result iteration

Type Parameters

NAME	DESCRIPTION

NAME	DESCRIPTION
T	Object type

ForEachAsync<T>(IEnumerable<T>, Action<T>, CancellationToken)

Asynchronous wrapper over query result iteration

Declaration

```
public static Task ForEachAsync<T>(this IEnumerable<T> source, Action<T> body, CancellationToken token)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence
System.Action<T>	body	Iteration logic
System.Threading.CancellationToken	token	Cancellation token

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task	Asynchronous call over query result iteration

Type Parameters

NAME	DESCRIPTION
T	Object type

ForEachAsync<T>(IEnumerable<T>, Func<T, Task>)

Asynchronous wrapper over query result iteration

Declaration

```
public static Task ForEachAsync<T>(this IEnumerable<T> source, Func<T, Task> body)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence

TYPE	NAME	DESCRIPTION
System.Func<T, System.Threading.Tasks.Task>	body	Iteration logic

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task	Asynchronous call over query result iteration

Type Parameters

NAME	DESCRIPTION
T	Object type

ForEachAsync<T>(IEnumerable<T>, Func<T, Task>, CancellationToken)

Asynchronous wrapper over query result iteration

Declaration

```
public static Task ForEachAsync<T>(this IEnumerable<T> source, Func<T, Task> body, CancellationToken token)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence
System.Func<T, System.Threading.Tasks.Task>	body	Iteration logic
System.Threading.CancellationToken	token	Cancellation token

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task	Asynchronous call over query result iteration

Type Parameters

NAME	DESCRIPTION
T	Object type

ForEachAsync<T>(IDataQuery<T>, Action<T>)

Asynchronous wrapper over query result iteration

Declaration

```
public static Task ForEachAsync<T>(this IDataQuery<T> source, Action<T> body)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataQuery<T>	source	Source object sequence
System.Action<T>	body	Iteration logic

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task	Asynchronous call over query result iteration

Type Parameters

NAME	DESCRIPTION
T	Object type

ForEachAsync<T>(IDataQuery<T>, Func<T, Task>)

Asynchronous wrapper over query result iteration

Declaration

```
public static Task ForEachAsync<T>(this IDataQuery<T> source, Func<T, Task> body)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataQuery<T>	source	Source object sequence
System.Func<T, System.Threading.Tasks.Task>	body	Iteration logic

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task	Asynchronous call over query result iteration

Type Parameters

NAME	DESCRIPTION
T	Object type

GetParameterAsync<T>(IDataQuery, String)

Asynchronous wrapper over GetParameter

Declaration

```
public static Task<T> GetParameterAsync<T>(this IDataQuery source, string param)
    where T : class
```

Parameters

TYPE	NAME	DESCRIPTION
IDataQuery	source	SQL procedure based data source
System.String	param	Parameter name

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<T>	Asynchronous wrapper over Submit

Type Parameters

NAME	DESCRIPTION
T	

GetParameterAsync<T, TRet>(T, Expression<Func<T, TRet>>)

Asynchronous wrapper over GetParameter

Declaration

```
public static Task<TRet> GetParameterAsync<T, TRet>(this T obj, Expression<Func<T, TRet>> param)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
T	obj	SQL procedure based single object result
System.Linq.Expressions.Expression<System.Func<T, TRet>>	param	Parameter name

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<TRet>	Asynchronous wrapper over Submit

Type Parameters

NAME	DESCRIPTION
T	
TRet	

GetParameterAsync<T, TRet>(T, String)

Asynchronous wrapper over GetParameter

Declaration

```
public static Task<TRet> GetParameterAsync<T, TRet>(this T obj, string param)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
T	obj	SQL procedure based single object result
System.String	param	Parameter name

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<TRet>	Asynchronous wrapper over Submit

Type Parameters

NAME	DESCRIPTION
T	
TRet	

GetParameterAsync<T, TRet>(IDataQuery<T>, Expression<Func<T, TRet>>)

Asynchronous wrapper over GetParameter

Declaration

```
public static Task<TRet> GetParameterAsync<T, TRet>(this IDataQuery<T> source, Expression<Func<T, TRet>>
    param)
    where T : class, new()
```

Parameters

TYPE	NAME	DESCRIPTION
IDataQuery<T>	source	SQL procedure based data source
System.Linq.Expressions.Expression<System.Func<T, TRet>>	param	Parameter name

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<TRet>	Asynchronous wrapper over Submit

Type Parameters

NAME	DESCRIPTION
T	
TRet	

GetResultSetAsync<TRes>(IDataQuery, Expression<Func<IEnumerable<TRes>>>)

Asynchronous wrapper over GetResultSet

Declaration

```
public static Task<IEnumerable<TRes>> GetResultSetAsync<TRes>(this IDataQuery source,
Expression<Func<IEnumerable<TRes>>> resultSetName)
    where TRes : class, new()
```

Parameters

TYPE	NAME	DESCRIPTION
IDataQuery	source	SQL procedure based data source
System.Linq.Expressions.Expression<System.Func<System.Collections.Generic.IEnumerable<TRes>>>>	resultSetNames	Resultset name

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Collections.Generic.IEnumerable<TRes>>	Asynchronous wrapper over Submit

Type Parameters

NAME	DESCRIPTION
TRes	

GetResultSetAsync<TRes>(IDataQuery, String)

Asynchronous wrapper over GetResultSet

Declaration

```
public static Task<IEnumerable<TRes>> GetResultSetAsync<TRes>(this IDataQuery source, string resultSetName)
    where TRes : class, new()
```

Parameters

TYPE	NAME	DESCRIPTION
IDataQuery	source	SQL procedure based data source
System.String	resultSetNames	Resultset name

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Collections.Generic.IEnumerable<TRes>>	Asynchronous wrapper over Submit

Type Parameters

NAME	DESCRIPTION
TRes	

GetResultSetAsync<T, TRes>(T, Expression<Func<T, IEnumerable<TRes>>>)

Asynchronous wrapper over GetResultSet

Declaration

```
public static Task<IEnumerable<TRes>> GetResultSetAsync<T, TRes>(this T obj, Expression<Func<T,
IEnumerable<TRes>>> resultSetNames)
    where T : class, IDataObject, new()
    where TRes : class
```

Parameters

TYPE	NAME	DESCRIPTION
T	obj	SQL procedure based single object result
System.Linq.Expressions.Expression<System.Func<T, System.Collections.Generic.IEnumerable<TRes>>>	resultSetNames	Resultset name

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Collections.Generic.IEnumerable<TRes>>	Asynchronous wrapper over Submit

Type Parameters

NAME	DESCRIPTION
T	
TRes	

GetResultSetAsync<T, TRes>(IRepository<T>, Expression<Func<T, IEnumerable<TRes>>>)

Asynchronous wrapper over GetResultSet

Declaration

```
public static Task<IEnumerable<TRes>> GetResultSetAsync<T, TRes>(this IRepository<T> source,
Expression<Func<T, IEnumerable<TRes>>> resultSetName)
    where T : class, IDataObject, new()
    where TRes : class
```

Parameters

TYPE	NAME	DESCRIPTION
IRepository<T>	source	SQL procedure based data source
System.Linq.Expressions.Expression<System.Func<T, System.Collections.Generic.IEnumerable<TRes>>>	resultSetNames	Resultset name

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Collections.Generic.IEnumerable<TRes>>	Asynchronous wrapper over Submit

Type Parameters

NAME	DESCRIPTION
T	
TRes	

GetSqlBlockDataAsync<T>(IDataScope, Func<JsonSettings<T>, JsonSettings<T>>, Expression<SqlBlockDefinition<T>>, Expression<Func<ISqlParamAdapter, ISqlParam>>[])

Asynchronous wrapper over select SQL block data

Declaration

```
public static Task<T[]> GetSqlBlockDataAsync<T>(this IDataScope dataScope, Func<JsonSettings<T>,
JsonSettings<T>> settings, Expression<SqlBlockDefinition<T>> block, params Expression<Func<ISqlParamAdapter,
ISqlParam>>[] parameters)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
IDataScope	dataScope	Data scope layer
System.Func< JsonSettings <T>, JsonSettings <T>>	settings	JSON serialization settings
System.Linq.Expressions.Expression< SqlBlockDefinition <T>>	block	Sql block structure
System.Linq.Expressions.Expression<System.Func< ISqlParamAdapter , ISqlParam >>[]	parameters	Sql block parameters

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<T[]>	Asynchronous wrapper over SQL block data selected and mapped

Type Parameters

NAME	DESCRIPTION
T	

GetSqlBlockDataAsync<T>(IDataScope, Expression<SqlBlockDefinition<T>>, Expression<Func<ISqlParamAdapter, ISqlParam>>[])

Asynchronous wrapper over select SQL block data

Declaration

```
public static Task<T[]> GetSqlBlockDataAsync<T>(this IDataScope dataScope, Expression<SqlBlockDefinition<T>>
block, params Expression<Func<ISqlParamAdapter, ISqlParam>>[] parameters)
    where T : class, new()
```

Parameters

TYPE	NAME	DESCRIPTION
IDataScope	dataScope	Data scope layer
System.Linq.Expressions.Expression< SqlBlockDefinition <T>>	block	Sql block structure

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func<ISqlParamAdapter, ISqlParam>>[]	parameters	Sql block parameters

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<T[]>	Asynchronous wrapper over SQL block data selected and mapped

Type Parameters

NAME	DESCRIPTION
T	

LastAsync<T>(IQueryable<T>)

Asynchronous wrapper over Last() LINQ function

Declaration

```
public static Task<T> LastAsync<T>(this IQueryable<T> source)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<T>	Asynchronous call over Last() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

LastAsync<T>(IQueryable<T>, Expression<Func<T, Boolean>>)

Asynchronous wrapper over Last() LINQ function

Declaration

```
public static Task<T> LastAsync<T>(this IQueryable<T> source, Expression<Func<T, bool>> predicate)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, System.Boolean>>	predicate	A function to test each element for a condition

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<T>	Asynchronous call over Last() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

LastAsync<T>(IQueryable<T>, Expression<Func<T, Boolean>>, CancellationToken)

Asynchronous wrapper over Last() LINQ function

Declaration

```
public static Task<T> LastAsync<T>(this IQueryable<T> source, Expression<Func<T, bool>> predicate,
CancellationToken token)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, System.Boolean>>	predicate	A function to test each element for a condition
System.Threading.CancellationToken	token	Cancellation token

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<T>	Asynchronous call over Last() LINQ function

Type Parameters

NAME	DESCRIPTION

NAME	DESCRIPTION
T	Object type

LastAsync<T>(IDataQuery<T>)

Asynchronous wrapper over Last() LINQ function

Declaration

```
public static Task<T> LastAsync<T>(this IDataQuery<T> source)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataQuery<T>	source	Source object sequence

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<T>	Asynchronous call over Last() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

LastAsync<T>(IDataQuery<T>, Func<JsonSettings<T>, JsonSettings<T>>>)

Asynchronous wrapper over Last() LINQ function

Declaration

```
public static Task<T> LastAsync<T>(this IDataQuery<T> source, Func<JsonSettings<T>, JsonSettings<T>>> settings)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
IDataQuery<T>	source	Source object sequence
System.Func< JsonSettings<T> , JsonSettings<T> >>	settings	JSON serialization settings

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<T>	Asynchronous call over Last() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

LastAsync<T>(IDataQuery<T>, Expression<Func<T, Boolean>>)

Asynchronous wrapper over Last() LINQ function

Declaration

```
public static Task<T> LastAsync<T>(this IDataQuery<T> source, Expression<Func<T, bool>> predicate)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataQuery<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, System.Boolean>>	predicate	A function to test each element for a condition

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<T>	Asynchronous call over Last() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

LastAsync<T>(IDataQuery<T>, Expression<Func<T, Boolean>>, Func<JsonSettings<T>, JsonSettings<T>>)

Asynchronous wrapper over Last() LINQ function

Declaration

```
public static Task<T> LastAsync<T>(this IDataQuery<T> source, Expression<Func<T, bool>> predicate,
Func<JsonSettings<T>, JsonSettings<T>> settings)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
IDataQuery<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, System.Boolean>>	predicate	A function to test each element for a condition
System.Func< JsonSettings<T> , JsonSettings<T> >	settings	JSON serialization settings

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<T>	Asynchronous call over Last() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

LastOrDefaultAsync<T>(IQueryable<T>)

Asynchronous wrapper over LastOrDefault() LINQ function

Declaration

```
public static Task<T> LastOrDefaultAsync<T>(this IQueryable<T> source)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<T>	Asynchronous call over LastOrDefault() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

LastOrDefaultAsync<T>(IQueryable<T>, Expression<Func<T, Boolean>>)

Asynchronous wrapper over LastOrDefault() LINQ function

Declaration

```
public static Task<T> LastOrDefaultAsync<T>(this IQueryable<T> source, Expression<Func<T, bool>> predicate)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, System.Boolean>>	predicate	A function to test each element for a condition

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<T>	Asynchronous call over LastOrDefault() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

LastOrDefaultAsync<T>(IQueryable<T>, Expression<Func<T, Boolean>>, CancellationToken)

Asynchronous wrapper over LastOrDefault() LINQ function

Declaration

```
public static Task<T> LastOrDefaultAsync<T>(this IQueryable<T> source, Expression<Func<T, bool>> predicate, CancellationToken token)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, System.Boolean>>	predicate	A function to test each element for a condition
System.Threading.CancellationToken	token	Cancellation token

Returns

TYPE	DESCRIPTION

TYPE	DESCRIPTION
System.Threading.Tasks.Task<T>	Asynchronous call over LastOrDefault() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

LastOrDefaultAsync<T>(IDataQuery<T>)

Asynchronous wrapper over LastOrDefault() LINQ function

Declaration

```
public static Task<T> LastOrDefaultAsync<T>(this IDataQuery<T> source)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataQuery<T>	source	Source object sequence

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<T>	Asynchronous call over LastOrDefault() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

LastOrDefaultAsync<T>(IDataQuery<T>, Func<JsonSettings<T>, JsonSettings<T>>)

Asynchronous wrapper over LastOrDefault() LINQ function

Declaration

```
public static Task<T> LastOrDefaultAsync<T>(this IDataQuery<T> source, Func<JsonSettings<T>, JsonSettings<T>> settings)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
IDataQuery<T>	source	Source object sequence

TYPE	NAME	DESCRIPTION
System.Func<JsonSettings<T>, JsonSettings<T>>	settings	JSON serialization settings

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<T>	Asynchronous call over LastOrDefault() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

LastOrDefaultAsync<T>(IDataQuery<T>, Expression<Func<T, Boolean>>)

Asynchronous wrapper over LastOrDefault() LINQ function

Declaration

```
public static Task<T> LastOrDefaultAsync<T>(this IDataQuery<T> source, Expression<Func<T, bool>> predicate)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataQuery<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, System.Boolean>>	predicate	A function to test each element for a condition

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<T>	Asynchronous call over LastOrDefault() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

LastOrDefaultAsync<T>(IDataQuery<T>, Expression<Func<T, Boolean>>, Func<JsonSettings<T>, JsonSettings<T>>)

Asynchronous wrapper over LastOrDefault() LINQ function

Declaration

```
public static Task<T> LastOrDefaultAsync<T>(this IDataQuery<T> source, Expression<Func<T, bool>> predicate,
Func<JsonSettings<T>, JsonSettings<T>> settings)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
IDataQuery<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, System.Boolean>>	predicate	A function to test each element for a condition
System.Func< JsonSettings<T> , JsonSettings<T> >	settings	JSON serialization settings

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<T>	Asynchronous call over LastOrDefault() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

LoadLobAsync<T>(T)

Asynchronous wrapper over LoadLob

Declaration

```
public static Task<T> LoadLobAsync<T>(this T obj)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
T	obj	Data object to submit

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<T>	Asynchronous wrapper over LoadLob

Type Parameters

NAME	DESCRIPTION
T	

LoadLobAsync<T>(T, Expression<Func<T, Object>>[])

Asynchronous wrapper over LoadLob

Declaration

```
public static Task<T> LoadLobAsync<T>(this T obj, params Expression<Func<T, object>>[] properties)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
T	obj	Data object to submit
System.Linq.Expressions.Expression<System.Func<T, System.Object>>[]	properties	Large data properties expressions to load

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<T>	Asynchronous wrapper over LoadLob

Type Parameters

NAME	DESCRIPTION
T	

LoadLobAsync<T>(T, String[])

Asynchronous wrapper over LoadLob

Declaration

```
public static Task<T> LoadLobAsync<T>(this T obj, params string[] properties)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
T	obj	Data object to submit
System.String[]	properties	Large data properties to load

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<T>	Asynchronous wrapper over LoadLob

Type Parameters

NAME	DESCRIPTION
T	

LongCountAsync<T>(IEnumerable<T>)

Asynchronous wrapper over LongCount() LINQ function

Declaration

```
public static Task<long> LongCountAsync<T>(this IEnumerable<T> source)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Int64>	Asynchronous call over LongCount() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

LongCountAsync<T>(IEnumerable<T>, Expression<Func<T, Boolean>>)

Asynchronous wrapper over LongCount() LINQ function

Declaration

```
public static Task<long> LongCountAsync<T>(this IEnumerable<T> source, Expression<Func<T, bool>> predicate)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, System.Boolean>>	predicate	A function to test each element for a condition

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Int64>	Asynchronous call over LongCount() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

LongCountAsync<T>(IQueryable<T>, Expression<Func<T, Boolean>>, CancellationToken)

Asynchronous wrapper over LongCount() LINQ function

Declaration

```
public static Task<long> LongCountAsync<T>(this IQueryable<T> source, Expression<Func<T, bool>> predicate,
CancellationToken token)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, System.Boolean>>	predicate	A function to test each element for a condition
System.Threading.CancellationToken	token	Cancellation token

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Int64>	Asynchronous call over LongCount() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

LongCountAsync<T>(IDataQuery<T>)

Asynchronous wrapper over LongCount() LINQ function

Declaration

```
public static Task<long> LongCountAsync<T>(this IDataQuery<T> source)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataQuery<T>	source	Source object sequence

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Int64>	Asynchronous call over LongCount() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

LongCountAsync<T>(IDataQuery<T>, Expression<Func<T, Boolean>>)

Asynchronous wrapper over LongCount() LINQ function

Declaration

public static Task<long> LongCountAsync<T>(this IDataQuery<T> source, Expression<Func<T, bool>> predicate)

Parameters

TYPE	NAME	DESCRIPTION
IDataQuery<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, System.Boolean>>	predicate	A function to test each element for a condition

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Int64>	Asynchronous call over LongCount() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

MaxAsync<T, TResult>(IQueryable<T>, Expression<Func<T, TResult>>)

Asynchronous wrapper over Max() LINQ function

Declaration


```
public static Task<TResult> MaxAsync<T, TResult>(this IQueryable<T> source, Expression<Func<T, TResult>> selector)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, TResult> >	selector	A projection function to apply to each element

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<TResult>	Asynchronous call over Max() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type
TResult	Function applied field type

MaxAsync<T, TResult>(IQueryable<T>, Expression<Func<T, TResult>>, CancellationToken)

Asynchronous wrapper over Max() LINQ function

Declaration

```
public static Task<TResult> MaxAsync<T, TResult>(this IQueryable<T> source, Expression<Func<T, TResult>> selector, CancellationToken token)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, TResult> >	selector	A projection function to apply to each element
System.Threading.CancellationToken	token	Cancellation token

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<TResult>	Asynchronous call over Max() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type
TResult	Function applied field type

MaxAsync<T, TResult>(IDataQuery<T>, Expression<Func<T, TResult>>)

Asynchronous wrapper over Max() LINQ function

Declaration

```
public static Task<TResult> MaxAsync<T, TResult>(this IDataQuery<T> source, Expression<Func<T, TResult>> selector)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataQuery<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, TResult> >	selector	A projection function to apply to each element

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<TResult>	Asynchronous call over Max() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type
TResult	Function applied field type

MinAsync<T, TResult>(IQueryable<T>, Expression<Func<T, TResult>>)

Asynchronous wrapper over Min() LINQ function

Declaration

```
public static Task<TResult> MinAsync<T, TResult>(this IQueryable<T> source, Expression<Func<T, TResult>> selector)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, TResult> >	selector	A projection function to apply to each element

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<TResult>	Asynchronous call over Min() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type
TResult	Function applied field type

MinAsync<T, TResult>(IQueryable<T>, Expression<Func<T, TResult>>, CancellationToken)

Asynchronous wrapper over Min() LINQ function

Declaration

```
public static Task<TResult> MinAsync<T, TResult>(this IQueryable<T> source, Expression<Func<T, TResult>> selector, CancellationToken token)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, TResult> >	selector	A projection function to apply to each element
System.Threading.CancellationToken	token	Cancellation token

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<TResult>	Asynchronous call over Min() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type
TResult	Function applied field type

MinAsync<T, TResult>(IDataQuery<T>, Expression<Func<T, TResult>>)

Asynchronous wrapper over Min() LINQ function

Declaration

```
public static Task<TResult> MinAsync<T, TResult>(this IDataQuery<T> source, Expression<Func<T, TResult>> selector)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataQuery<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, TResult>>	selector	A projection function to apply to each element

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<TResult>	Asynchronous call over Min() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type
TResult	Function applied field type

SingleAsync<T>(IQueryable<T>)

Asynchronous wrapper over Single() LINQ function

Declaration

```
public static Task<T> SingleAsync<T>(this IQueryable<T> source)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<T>	Asynchronous call over Single() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

SingleAsync<T>(IQueryable<T>, Expression<Func<T, Boolean>>)

Asynchronous wrapper over Single() LINQ function

Declaration

```
public static Task<T> SingleAsync<T>(this IQueryable<T> source, Expression<Func<T, bool>> predicate)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, System.Boolean>>	predicate	A function to test each element for a condition

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<T>	Asynchronous call over Single() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

SingleAsync<T>(IQueryable<T>, Expression<Func<T, Boolean>>, CancellationToken)

Asynchronous wrapper over Single() LINQ function

Declaration

```
public static Task<T> SingleAsync<T>(this IQueryable<T> source, Expression<Func<T, bool>> predicate,
Cancellation token)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, System.Boolean>>	predicate	A function to test each element for a condition
System.Threading.CancellationToken	token	Cancellation token

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<T>	Asynchronous call over Single() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

SingleAsync<T>(IDataQuery<T>)

Asynchronous wrapper over Single() LINQ function

Declaration

```
public static Task<T> SingleAsync<T>(this IDataQuery<T> source)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataQuery <T>	source	Source object sequence

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<T>	Asynchronous call over Single() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

SingleAsync<T>(IDataQuery<T>, Func<JsonSettings<T>, JsonSettings<T>>)

Asynchronous wrapper over Single() LINQ function

Declaration

```
public static Task<T> SingleAsync<T>(this IDataQuery<T> source, Func<JsonSettings<T>, JsonSettings<T>> settings)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
IDataQuery<T>	source	Source object sequence
System.Func< JsonSettings<T> , JsonSettings<T> >	settings	JSON serialization settings

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<T>	Asynchronous call over Single() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

SingleAsync<T>(IDataQuery<T>, Expression<Func<T, Boolean>>)

Asynchronous wrapper over Single() LINQ function

Declaration

```
public static Task<T> SingleAsync<T>(this IDataQuery<T> source, Expression<Func<T, bool>> predicate)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataQuery<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, System.Boolean>>	predicate	A function to test each element for a condition

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<T>	Asynchronous call over Single() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

SingleAsync<T>(IDataQuery<T>, Expression<Func<T, Boolean>>, Func<JsonSettings<T>, JsonSettings<T>>>)

Asynchronous wrapper over Single() LINQ function

Declaration

```
public static Task<T> SingleAsync<T>(this IDataQuery<T> source, Expression<Func<T, bool>> predicate,
Func<JsonSettings<T>, JsonSettings<T>> settings)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
IDataQuery<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, System.Boolean>>	predicate	A function to test each element for a condition
System.Func<JsonSettings<T>, JsonSettings<T>>	settings	JSON serialization settings

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<T>	Asynchronous call over Single() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

SingleOrDefaultAsync<T>(IQueryable<T>)

Asynchronous wrapper over SingleOrDefault() LINQ function

Declaration

```
public static Task<T> SingleOrDefaultAsync<T>(this IQueryable<T> source)
```


Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<T>	Asynchronous call over SingleOrDefault() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

SingleOrDefaultAsync<T>(IQueryable<T>, Expression<Func<T, Boolean>>)

Asynchronous wrapper over SingleOrDefault() LINQ function

Declaration

```
public static Task<T> SingleOrDefaultAsync<T>(this IQueryable<T> source, Expression<Func<T, bool>> predicate)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, System.Boolean>>	predicate	A function to test each element for a condition

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<T>	Asynchronous call over SingleOrDefault() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

SingleOrDefaultAsync<T>(IQueryable<T>, Expression<Func<T, Boolean>>, CancellationToken)

Asynchronous wrapper over SingleOrDefault() LINQ function

Declaration

```
public static Task<T> SingleOrDefaultAsync<T>(this IQueryable<T> source, Expression<Func<T, bool>> predicate,
CancellationToken token)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, System.Boolean>>	predicate	A function to test each element for a condition
System.Threading.CancellationToken	token	Cancellation token

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<T>	Asynchronous call over SingleOrDefault() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

SingleOrDefaultAsync<T>(IDataQuery<T>)

Asynchronous wrapper over SingleOrDefault() LINQ function

Declaration

```
public static Task<T> SingleOrDefaultAsync<T>(this IDataQuery<T> source)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataQuery <T>	source	Source object sequence

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<T>	Asynchronous call over SingleOrDefault() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

SingleOrDefaultAsync<T>(IDataQuery<T>, Func<JsonSettings<T>, JsonSettings<T>>>)

Asynchronous wrapper over SingleOrDefault() LINQ function

Declaration

```
public static Task<T> SingleOrDefaultAsync<T>(this IDataQuery<T> source, Func<JsonSettings<T>,
JsonSettings<T>> settings)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
IDataQuery<T>	source	Source object sequence
System.Func< JsonSettings<T> , JsonSettings<T> >	settings	JSON serialization settings

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<T>	Asynchronous call over SingleOrDefault() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

SingleOrDefaultAsync<T>(IDataQuery<T>, Expression<Func<T, Boolean>>)

Asynchronous wrapper over SingleOrDefault() LINQ function

Declaration

```
public static Task<T> SingleOrDefaultAsync<T>(this IDataQuery<T> source, Expression<Func<T, bool>> predicate)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataQuery<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, System.Boolean>>	predicate	A function to test each element for a condition

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<T>	Asynchronous call over SingleOrDefault() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

SingleOrDefaultAsync<T>(IDataQuery<T>, Expression<Func<T, Boolean>>, Func<JsonSettings<T>, JsonSettings<T>>)

Asynchronous wrapper over SingleOrDefault() LINQ function

Declaration

```
public static Task<T> SingleOrDefaultAsync<T>(this IDataQuery<T> source, Expression<Func<T, bool>> predicate,
Func<JsonSettings<T>, JsonSettings<T>> settings)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
IDataQuery<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, System.Boolean>>	predicate	A function to test each element for a condition
System.Func< JsonSettings<T> , JsonSettings<T> >	settings	JSON serialization settings

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<T>	Asynchronous call over SingleOrDefault() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

SubmitAndReturnAsync<T>(T, Func<JsonSettings<T>, JsonSettings<T>>, DataSubmitFlag)

Asynchronous wrapper over Submit

Declaration

```
public static Task<T> SubmitAndReturnAsync<T>(this T obj, Func<JsonSettings<T>, JsonSettings<T>> settings,
DataSubmitFlag flag = DataSubmitFlag.None)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
T	obj	Data object to submit
System.Func<JsonSettings<T>, JsonSettings<T>>	settings	JSON serialization settings
DataSubmitFlag	flag	Data submit flag

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<T>	Asynchronous wrapper over Submit

Type Parameters

NAME	DESCRIPTION
T	

SubmitAndReturnAsync<T>(T, DataSubmitFlag)

Asynchronous wrapper over Submit

Declaration

```
public static Task<T> SubmitAndReturnAsync<T>(this T obj, DataSubmitFlag flag = DataSubmitFlag.None)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
T	obj	Data object to submit
DataSubmitFlag	flag	Data submit flag

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<T>	Asynchronous wrapper over Submit

Type Parameters

NAME	DESCRIPTION
T	

SubmitAsync<T>(T, DataSubmitFlag)

Asynchronous wrapper over Submit

Declaration

```
public static Task<bool> SubmitAsync<T>(this T obj, DataSubmitFlag flag = DataSubmitFlag.None)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
T	obj	Data object to submit
DataSubmitFlag	flag	Data submit flag

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Boolean>	Asynchronous wrapper over Submit

Type Parameters

NAME	DESCRIPTION
T	

SubmitAsync<T>(IRepository<T>, IEnumerable<T>, DataSubmitFlag)

Asynchronous wrapper over Submit

Declaration

```
public static Task<bool> SubmitAsync<T>(this IRepository<T> source, IEnumerable<T> objects, DataSubmitFlag
flag = DataSubmitFlag.None)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
IRepository<T>	source	Repository to submit
System.Collections.Generic.IEnumerable<T>	objects	Objects to submit

TYPE	NAME	DESCRIPTION
DataSubmitFlag	flag	Data submit flag

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Boolean>	Asynchronous wrapper over Submit

Type Parameters

NAME	DESCRIPTION
T	

SumAsync<T>(IQueryable<T>, Expression<Func<T, Decimal>>)

Asynchronous wrapper over Sum() LINQ function

Declaration

```
public static Task<decimal> SumAsync<T>(this IQueryable<T> source, Expression<Func<T, decimal>> selector)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, Decimal >>	selector	A projection function to apply to each element

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task< Decimal >	Asynchronous call over Sum() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

SumAsync<T>(IQueryable<T>, Expression<Func<T, Decimal>>, CancellationToken)

Asynchronous wrapper over Sum() LINQ function

Declaration

```
public static Task<decimal> SumAsync<T>(this IQueryable<T> source, Expression<Func<T, decimal>> selector,
CancellationToken token)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, Decimal >>	selector	A projection function to apply to each element
System.Threading.CancellationToken	token	Cancellation token

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task< Decimal >	Asynchronous call over Sum() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

SumAsync<T>(IQueryable<T>, Expression<Func<T, Double>>)

Asynchronous wrapper over Sum() LINQ function

Declaration

```
public static Task<double> SumAsync<T>(this IQueryable<T> source, Expression<Func<T, double>> selector)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, Double >>	selector	A projection function to apply to each element

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task< Double >	Asynchronous call over Sum() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

SumAsync<T>(IQueryable<T>, Expression<Func<T, Double>>, CancellationToken)

Asynchronous wrapper over Sum() LINQ function

Declaration

```
public static Task<double> SumAsync<T>(this IQueryable<T> source, Expression<Func<T, double>> selector,
CancellationToken token)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, Double>>	selector	A projection function to apply to each element
System.Threading.CancellationToken	token	Cancellation token

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<Double>	Asynchronous call over Sum() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

SumAsync<T>(IQueryable<T>, Expression<Func<T, Int32>>)

Asynchronous wrapper over Sum() LINQ function

Declaration

```
public static Task<int> SumAsync<T>(this IQueryable<T> source, Expression<Func<T, int>> selector)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func<T, System.Int32>>	selector	A projection function to apply to each element

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Int32>	Asynchronous call over Sum() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

SumAsync<T>(IQueryable<T>, Expression<Func<T, Int32>>, CancellationToken)

Asynchronous wrapper over Sum() LINQ function

Declaration

```
public static Task<int> SumAsync<T>(this IQueryable<T> source, Expression<Func<T, int>> selector,
CancellationToken token)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, System.Int32>>	selector	A projection function to apply to each element
System.Threading.CancellationToken	token	Cancellation token

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Int32>	Asynchronous call over Sum() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

SumAsync<T>(IQueryable<T>, Expression<Func<T, Int64>>)

Asynchronous wrapper over Sum() LINQ function

Declaration

```
public static Task<long> SumAsync<T>(this IQueryable<T> source, Expression<Func<T, long>> selector)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, System.Int64>>	selector	A projection function to apply to each element

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Int64>	Asynchronous call over Sum() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

SumAsync<T>(IQueryable<T>, Expression<Func<T, Int64>>, CancellationToken)

Asynchronous wrapper over Sum() LINQ function

Declaration

```
public static Task<long> SumAsync<T>(this IQueryable<T> source, Expression<Func<T, long>> selector, CancellationToken token)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, System.Int64>>	selector	A projection function to apply to each element
System.Threading.CancellationToken	token	Cancellation token

Returns

TYPE	DESCRIPTION

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Int64>	Asynchronous call over Sum() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

SumAsync<T>(IQueryable<T>, Expression<Func<T, Nullable<Decimal>>>)

Asynchronous wrapper over Sum() LINQ function

Declaration

```
public static Task<decimal?> SumAsync<T>(this IQueryable<T> source, Expression<Func<T, decimal?>> selector)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, System.Nullable<Decimal>>>>	selector	A projection function to apply to each element

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Nullable<Decimal>>	Asynchronous call over Average() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

SumAsync<T>(IQueryable<T>, Expression<Func<T, Nullable<Decimal>>>, CancellationToken)

Asynchronous wrapper over Sum() LINQ function

Declaration

```
public static Task<decimal?> SumAsync<T>(this IQueryable<T> source, Expression<Func<T, decimal?>> selector, CancellationToken token)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, System.Nullable<Decimal>>>	selector	A projection function to apply to each element
System.Threading.CancellationToken	token	Cancellation token

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Nullable<Decimal>>	Asynchronous call over Sum() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

SumAsync<T>(IQueryable<T>, Expression<Func<T, Nullable<Double>>>)

Asynchronous wrapper over Sum() LINQ function

Declaration

```
public static Task<double?> SumAsync<T>(this IQueryable<T> source, Expression<Func<T, double?>> selector)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, System.Nullable<Double>>>	selector	A projection function to apply to each element

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Nullable<Double>>	Asynchronous call over Sum() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

SumAsync<T>(IQueryable<T>, Expression<Func<T, Nullable<Double>>>, CancellationToken)

Asynchronous wrapper over Sum() LINQ function

Declaration

```
public static Task<double?> SumAsync<T>(this IQueryable<T> source, Expression<Func<T, double?>> selector,
CancellationToken token)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, System.Nullable<Double>>>>	selector	A projection function to apply to each element
System.Threading.CancellationToken	token	Cancellation token

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Nullable<Double>>	Asynchronous call over Sum() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

SumAsync<T>(IQueryable<T>, Expression<Func<T, Nullable<Int32>>>>)

Asynchronous wrapper over Sum() LINQ function

Declaration

```
public static Task<int?> SumAsync<T>(this IQueryable<T> source, Expression<Func<T, int?>> selector)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func<T, System.Nullable<System.Int32>>>	selector	A projection function to apply to each element

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Nullable<System.Int32>>	Asynchronous call over Sum() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

SumAsync<T>(IQueryable<T>, Expression<Func<T, Nullable<Int32>>>, CancellationToken)

Asynchronous wrapper over Average() LINQ function

Declaration

```
public static Task<int?> SumAsync<T>(this IQueryable<T> source, Expression<Func<T, int?>> selector,
CancellationToken token)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, System.Nullable<System.Int32>>>	selector	A projection function to apply to each element
System.Threading.CancellationToken	token	Cancellation token

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Nullable<System.Int32>>	Asynchronous call over Average() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

SumAsync<T>(IQueryable<T>, Expression<Func<T, Nullable<Int64>>>)

Asynchronous wrapper over Sum() LINQ function

Declaration

```
public static Task<long?> SumAsync<T>(this IQueryable<T> source, Expression<Func<T, long?>> selector)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, System.Nullable<System.Int64>>>>	selector	A projection function to apply to each element

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Nullable<System.Int64>>	Asynchronous call over Sum() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

SumAsync<T>(IQueryable<T>, Expression<Func<T, Nullable<Int64>>>, CancellationToken)

Asynchronous wrapper over Average() LINQ function

Declaration

```
public static Task<long?> SumAsync<T>(this IQueryable<T> source, Expression<Func<T, long?>> selector, CancellationToken token)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, System.Nullable<System.Int64>>>>	selector	A projection function to apply to each element
System.Threading.CancellationToken	token	Cancellation token

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Nullable<System.Int64>>	Asynchronous call over Average() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

SumAsync<T>(IQueryable<T>, Expression<Func<T, Nullable<Single>>>)

Asynchronous wrapper over Sum() LINQ function

Declaration

```
public static Task<float?> SumAsync<T>(this IQueryable<T> source, Expression<Func<T, float?>> selector)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, System.Nullable<Single>>>	selector	A projection function to apply to each element

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Nullable<Single>>	Asynchronous call over Sum() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

SumAsync<T>(IQueryable<T>, Expression<Func<T, Nullable<Single>>>, CancellationToken)

Asynchronous wrapper over Sum() LINQ function

Declaration

```
public static Task<float?> SumAsync<T>(this IQueryable<T> source, Expression<Func<T, float?>> selector, CancellationToken token)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, System.Nullable<Single>>>	selector	A projection function to apply to each element
System.Threading.CancellationToken	token	Cancellation token

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Nullable<Single>>	Asynchronous call over Sum() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

SumAsync<T>(IQueryable<T>, Expression<Func<T, Single>>)

Asynchronous wrapper over Sum() LINQ function

Declaration

```
public static Task<float> SumAsync<T>(this IQueryable<T> source, Expression<Func<T, float>> selector)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, Single>>	selector	A projection function to apply to each element

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<Single>	Asynchronous call over Sum() LINQ function

Type Parameters

NAME	DESCRIPTION

NAME	DESCRIPTION
T	Object type

SumAsync<T>(IQueryable<T>, Expression<Func<T, Single>>, CancellationToken)

Asynchronous wrapper over Sum() LINQ function

Declaration

```
public static Task<float> SumAsync<T>(this IQueryable<T> source, Expression<Func<T, float>> selector,
CancellationToken token)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, Single >>	selector	A projection function to apply to each element
System.Threading.CancellationToken	token	Cancellation token

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task< Single >	Asynchronous call over Sum() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

SumAsync<T>(IDataQuery<T>, Expression<Func<T, Decimal>>)

Asynchronous wrapper over Sum() LINQ function

Declaration

```
public static Task<decimal> SumAsync<T>(this IDataQuery<T> source, Expression<Func<T, decimal>> selector)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataQuery <T>	source	Source object sequence

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func<T, Decimal >>	selector	A projection function to apply to each element

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task< Decimal >	Asynchronous call over Sum() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

SumAsync<T>(IDataQuery<T>, Expression<Func<T, Double>>)

Asynchronous wrapper over Sum() LINQ function

Declaration

```
public static Task<double> SumAsync<T>(this IDataQuery<T> source, Expression<Func<T, double>> selector)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataQuery <T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, Double >>	selector	A projection function to apply to each element

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task< Double >	Asynchronous call over Sum() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

SumAsync<T>(IDataQuery<T>, Expression<Func<T, Int32>>)

Asynchronous wrapper over Sum() LINQ function

Declaration

```
public static Task<int> SumAsync<T>(this IDataQuery<T> source, Expression<Func<T, int>> selector)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataQuery <T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, System.Int32>>	selector	A projection function to apply to each element

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Int32>	Asynchronous call over Sum() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

SumAsync<T>(IDataQuery<T>, Expression<Func<T, Int64>>)

Asynchronous wrapper over Sum() LINQ function

Declaration

```
public static Task<long> SumAsync<T>(this IDataQuery<T> source, Expression<Func<T, long>> selector)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataQuery <T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, System.Int64>>	selector	A projection function to apply to each element

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Int64>	Asynchronous call over Sum() LINQ function

Type Parameters

NAME	DESCRIPTION

NAME	DESCRIPTION
T	Object type

SumAsync<T>(IDataQuery<T>, Expression<Func<T, Nullable<Decimal>>>)

Asynchronous wrapper over Sum() LINQ function

Declaration

```
public static Task<decimal?> SumAsync<T>(this IDataQuery<T> source, Expression<Func<T, decimal?>> selector)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataQuery<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, System.Nullable< Decimal >>>	selector	A projection function to apply to each element

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Nullable< Decimal >>	Asynchronous call over Sum() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

SumAsync<T>(IDataQuery<T>, Expression<Func<T, Nullable<Double>>>)

Asynchronous wrapper over Sum() LINQ function

Declaration

```
public static Task<double?> SumAsync<T>(this IDataQuery<T> source, Expression<Func<T, double?>> selector)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataQuery<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, System.Nullable< Double >>>	selector	A projection function to apply to each element

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Nullable<Double>>	Asynchronous call over Sum() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

SumAsync<T>(IDataQuery<T>, Expression<Func<T, Nullable<Int32>>>)

Asynchronous wrapper over Sum() LINQ function

Declaration

```
public static Task<int?> SumAsync<T>(this IDataQuery<T> source, Expression<Func<T, int?>> selector)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataQuery<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, System.Nullable<System.Int32>>>	selector	A projection function to apply to each element

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Nullable<System.Int32>>	Asynchronous call over Sum() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

SumAsync<T>(IDataQuery<T>, Expression<Func<T, Nullable<Int64>>>)

Asynchronous wrapper over Sum() LINQ function

Declaration

```
public static Task<long?> SumAsync<T>(this IDataQuery<T> source, Expression<Func<T, long?>> selector)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataQuery<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, System.Nullable<System.Int64>>>	selector	A projection function to apply to each element

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Nullable<System.Int64>>	Asynchronous call over Sum() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

SumAsync<T>(IDataQuery<T>, Expression<Func<T, Nullable<Single>>>)

Asynchronous wrapper over Sum() LINQ function

Declaration

```
public static Task<float?> SumAsync<T>(this IDataQuery<T> source, Expression<Func<T, float?>> selector)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataQuery<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, System.Nullable< Single >>>	selector	A projection function to apply to each element

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Nullable< Single >>	Asynchronous call over Sum() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

SumAsync<T>(IDataQuery<T>, Expression<Func<T, Single>>)

Asynchronous wrapper over Sum() LINQ function

Declaration

```
public static Task<float> SumAsync<T>(this IDataQuery<T> source, Expression<Func<T, float>> selector)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataQuery<T>	source	Source object sequence
System.Linq.Expressions.Expression<System.Func<T, Single>>	selector	A projection function to apply to each element

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<Single>	Asynchronous call over Sum() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

ToArrayAsync<T>(IQueryable<T>)

Asynchronous wrapper over ToArray() LINQ function

Declaration

```
public static Task<T[]> ToArrayAsync<T>(this IQueryable<T> source)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<T[]>	Asynchronous call over ToArray() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

ToArrayAsync<T>(IEnumerable<T>, CancellationToken)

Asynchronous wrapper over ToArray() LINQ function

Declaration

```
public static Task<T[]> ToArrayAsync<T>(this IEnumerable<T> source, CancellationToken token)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence
System.Threading.CancellationToken	token	Cancellation token

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<T[]>	Asynchronous call over ToArray() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

ToArrayAsync<T>(IDataQuery<T>)

Asynchronous wrapper over ToArray() LINQ function

Declaration

```
public static Task<T[]> ToArrayAsync<T>(this IDataQuery<T> source)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataQuery<T>	source	Source object sequence

Returns

TYPE	DESCRIPTION

TYPE	DESCRIPTION
System.Threading.Tasks.Task<T[]>	Asynchronous call over ToArray() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

ToArrayAsync<T>(IDataQuery<T>, Func<JsonSettings<T>, JsonSettings<T>>)

Asynchronous wrapper over ToArray() LINQ function

Declaration

```
public static Task<T[]> ToArrayAsync<T>(this IDataQuery<T> source, Func<JsonSettings<T>, JsonSettings<T>>
settings)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
IDataQuery<T>	source	Source object sequence
System.Func<JsonSettings<T>, JsonSettings<T>>	settings	JSON serialization settings

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<T[]>	Asynchronous call over ToArray() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

ToDictionaryAsync<T, TKey, TValue>(IQueryable<T>, Func<T, TKey>, Func<T, TValue>)

Asynchronous wrapper over ToDictionary() LINQ function

Declaration

```
public static Task<Dictionary<TKey, TValue>> ToDictionaryAsync<T, TKey, TValue>(this IQueryable<T> source,
Func<T, TKey> keySelector, Func<T, TValue> elementSelector)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence
System.Func<T, TKey>	keySelector	Function to extract key from each element
System.Func<T, TValue>	elementSelector	Function to extract value from each element

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Collections.Generic.Dictionary<TKey, TValue>>	Asynchronous call over ToDictionary() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type
TKey	Dictionary key type
TValue	Dictionary value type

ToDictionaryAsync<T, TKey, TValue>(IQueryable<T>, Func<T, TKey>, Func<T, TValue>, CancellationToken)

Asynchronous wrapper over ToDictionary() LINQ function

Declaration

```
public static Task<Dictionary<TKey, TValue>> ToDictionaryAsync<T, TKey, TValue>(this IQueryable<T> source,
Func<T, TKey> keySelector, Func<T, TValue> elementSelector, CancellationToken token)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence
System.Func<T, TKey>	keySelector	Function to extract key from each element
System.Func<T, TValue>	elementSelector	Function to extract value from each element

TYPE	NAME	DESCRIPTION
System.Threading.CancellationToken	token	Cancellation token

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Collections.Generic.Dictionary<TKey, TValue> >	Asynchronous call over ToDictionary() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type
TKey	Dictionary key type
TValue	Dictionary value type

ToDictionaryAsync<T, TKey, TValue>(IDataQuery<T>, Func<T, TKey>, Func<T, TValue>)

Asynchronous wrapper over ToDictionary() LINQ function

Declaration

```
public static Task<Dictionary<TKey, TValue>> ToDictionaryAsync<T, TKey, TValue>(this IDataQuery<T> source,
Func<T, TKey> keySelector, Func<T, TValue> elementSelector)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataQuery<T>	source	Source object sequence
System.Func<T, TKey>	keySelector	Function to extract key from each element
System.Func<T, TValue>	elementSelector	Function to extract value from each element

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Collections.Generic.Dictionary<TKey, TValue> >	Asynchronous call over ToDictionary() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type
TKey	Dictionary key type
TValue	Dictionary value type

ToListAsync<T>(IEnumerable<T>)

Asynchronous wrapper over ToList() LINQ function

Declaration

```
public static Task<List<T>> ToListAsync<T>(this IEnumerable<T> source)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Collections.Generic.List<T> >	Asynchronous call over ToList() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

ToListAsync<T>(IEnumerable<T>, CancellationToken)

Asynchronous wrapper over ToList() LINQ function

Declaration

```
public static Task<List<T>> ToListAsync<T>(this IEnumerable<T> source, CancellationToken token)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.IQueryable<T>	source	Source object sequence

TYPE	NAME	DESCRIPTION
System.Threading.CancellationToken	token	Cancellation token

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Collections.Generic.List<T>>	Asynchronous call over ToList() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

ToListAsync<T>(IDataQuery<T>)

Asynchronous wrapper over ToList() LINQ function

Declaration

```
public static Task<List<T>> ToListAsync<T>(this IDataQuery<T> source)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataQuery<T>	source	Source object sequence

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Collections.Generic.List<T>>	Asynchronous call over ToList() LINQ function

Type Parameters

NAME	DESCRIPTION
T	Object type

Namespace XData.Database

Classes

[FieldInfo](#)

Field information

[TableInfo](#)

Table information

[XDataContext](#)

Data context Provides database adapter and SQL builder access

Interfaces

[ITransaction](#)

Transaction wrapper interface

Enums

[DataQueries](#)

Static adapter queries enumeration

Enum DataQueries

Static adapter queries enumeration

Namespace: [XData.Database](#)

Assembly: XData.docfx.dll

Syntax

```
public enum DataQueries
```

Fields

NAME	DESCRIPTION
SelectTableColumns	Table columns predefined query
SelectTableList	Tables predefined query

See Also

[SelectTableColumns](#)

[SelectTableList](#)

Extension Methods

- [SerializationExtensions.ToXml<DataQueries>\(\)](#)
- [CollectionsExtensions.SetValue<DataQueries, T>\(T\)](#)
- [CollectionsExtensions.AsEnum<DataQueries>\(\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(FilterOperation, String\)](#)
- [Processing.Do<DataQueries>\(Action<IProcess<DataQueries>>\[\]\)](#)
- [TypeExtensions.ChangeType<T>\(\)](#)
- [TypeExtensions.ChangeType\(Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.ConvertTo<T>\(\)](#)
- [TypeExtensions.EqualValue\(Enum\)](#)
- [SqlBlockExtensions.SetExpression<DataQueries, TResult>\(Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<DataQueries, TResult>\(Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Class FieldInfo

Field information

Inheritance

System.Object
FieldInfo

Inherited Members

System.Object.Equals(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.GetHashCode()
System.Object.GetType()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)

Namespace: [XData.Database](#)
Assembly: XData.docfx.dll

Syntax

```
[Serializable]  
public sealed class FieldInfo
```

Properties

IsAutoIncrement

Auto increment flag

Declaration

```
public bool IsAutoIncrement { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

IsPk

Primary key flag

Declaration

```
public bool IsPk { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

Name

Field name

Declaration

```
public string Name { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

NativeDataType

Field native data type

Declaration

```
public string NativeDataType { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Scale

Field scale

Declaration

```
public int Scale { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Int32	

Size

Field size

Declaration

```
public int Size { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Int32	

Type

Field type

Declaration

```
public DbType Type { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Data.DbType	

Methods

ToString()

Returns a string that represents the current object.

Declaration

```
public override string ToString()
```

Returns

TYPE	DESCRIPTION
System.String	A string that represents the current object.

Overrides

System.Object.ToString()

Extension Methods

- SerializationExtensions.ToXml<T>(T)
- CollectionsExtensions.SetValue<TKey, T>(TKey, T)
- CollectionsExtensions.AsEnum<T>(T)
- DataObjectQueryableExtensions.CompareVariable(Object, FilterOperation, String)
- Processing.Do<T>(T, Action<IProcess<T>>[])
- TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>)
- TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>, TResult)
- TypeExtensions.ChangeType<T>(Object)
- TypeExtensions.ChangeType(Object, Type)
- TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)
- TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)
- TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)
- TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)
- SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)
- QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Interface ITransaction

Transaction wrapper interface

Inherited Members

System.IDisposable.Dispose()

Namespace: [XData.Database](#)

Assembly: XData.docfx.dll

Syntax

```
public interface ITransaction : IDisposable
```

Methods

Commit()

Commit transaction

Declaration

```
bool Commit()
```

Returns

TYPE	DESCRIPTION
System.Boolean	Success flag

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Class TableInfo

Table information

Inheritance

System.Object
TableInfo

Implements

System.IEquatable<[TableInfo](#)>
System.ICloneable

Inherited Members

System.Object.Equals(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.GetHashCode()
System.Object.GetType()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)

Namespace: [XData.Database](#)
Assembly: XData.docfx.dll

Syntax

```
[Serializable]  
public class TableInfo : IEquatable<TableInfo>, ICloneable
```

Properties

Fields

Table fields

Declaration

```
public List<FieldInfo> Fields { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Collections.Generic.List< FieldInfo >	

Name

Table name

Declaration

```
public string Name { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Methods

Clone()

Creates a new object that is a copy of the current instance.

Declaration

```
public object Clone()
```

Returns

TYPE	DESCRIPTION
System.Object	A new object that is a copy of this instance.

Equals(TableInfo)

Indicates whether the current object is equal to another object of the same type.

Declaration

```
public virtual bool Equals(TableInfo other)
```

Parameters

TYPE	NAME	DESCRIPTION
TableInfo	other	An object to compare with this object.

Returns

TYPE	DESCRIPTION
System.Boolean	true if the current object is equal to the <code>other</code> parameter; otherwise, false.

RefreshFields(String, IConfiguration)

Refresh table info

Declaration

```
public TableInfo RefreshFields(string context, IConfiguration cfg = null)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	context	Context name
IConfiguration	cfg	Configuration

Returns

TYPE	DESCRIPTION

TYPE	DESCRIPTION
TableInfo	Refreshed table info

ToString()

Returns a string that represents the current object.

Declaration

```
public override string ToString()
```

Returns

TYPE	DESCRIPTION
System.String	A string that represents the current object.

Overrides

[System.Object.ToString\(\)](#)

Implements

[System.IEquatable<T>](#)

[System.ICloneable](#)

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Class XDataContext

Data context Provides database adapter and SQL builder access

Inheritance

System.Object
XDataContext

Implements

System.IDisposable

Inherited Members

System.Object.Equals(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.GetHashCode()
System.Object.GetType()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.ToString()

Namespace: [XData.Database](#)

Assembly: XData.docfx.dll

Syntax

```
public sealed class XDataContext : IDisposable
```

Properties

ConcurrencyErrorId

Concurrency error id

Declaration

```
public string ConcurrencyErrorId { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

ConnectionInfo

Context connection information

Declaration

```
public XConnectionInfo ConnectionInfo { get; }
```

Property Value

TYPE	DESCRIPTION
XConnectionInfo	

DatabaseAdapter

Database adapter

Declaration

```
public IDatabaseAdapter DatabaseAdapter { get; }
```

Property Value

TYPE	DESCRIPTION
IDatabaseAdapter	

DatabaseDialect

SQL builder

Declaration

```
public ISqlBuilder DatabaseDialect { get; }
```

Property Value

TYPE	DESCRIPTION
ISqlBuilder	

DialectName

SQL dialect name

Declaration

```
public string DialectName { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

DialectOptions

Dialect options supported by context

Declaration

```
public DialectOptions DialectOptions { get; }
```

Property Value

TYPE	DESCRIPTION
DialectOptions	

Name

Context name

Declaration

```
public string Name { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Proxy

Remote proxy

Declaration

```
public IProxy Proxy { get; }
```

Property Value

TYPE	DESCRIPTION
IProxy	

Methods

Dispose()

Declaration

```
public void Dispose()
```

Dispose(Boolean)

Declaration

```
public void Dispose(bool disposing)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Boolean	disposing	

Implements

System.IDisposable

Extension Methods

- SerializationExtensions.ToXml<T>(T)
- CollectionsExtensions.SetValue<TKey, T>(TKey, T)
- CollectionsExtensions.AsEnum<T>(T)
- DataObjectQueryableExtensions.CompareVariable(Object, FilterOperation, String)
- Processing.Do<T>(T, Action<IProcess<T>>[])
- TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>)
- TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>, TResult)
- TypeExtensions.ChangeType<T>(Object)
- TypeExtensions.ChangeType(Object, Type)
- TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)
- TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)
- TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)
- TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)
- SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)
- QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Namespace XData.Database.Adapter

Classes

[BaseDatabaseAdapter<TConnection, TParameter>](#)

Database adapter base class

[CallResult](#)

Stored procedure call result

[DatabaseAdapterAttribute](#)

Database adapter attribute

[Query](#)

SQL query

[QueryParam](#)

SQL query parameter

[QueryResult](#)

SQL query result

[QueryResultBase](#)

SQL query result

[StoredProcedure](#)

SQL stored procedure

[UdtSqlType](#)

User defined type (UDT) description

Interfaces

[IDatabaseAdapter](#)

Database adapter basic interface

[IDatabaseAdapterMetadata](#)

Database adapter description interface

Class BaseDatabaseAdapter<TConnection, TParameter>

Database adapter base class

Inheritance

System.Object
BaseDatabaseAdapter<TConnection, TParameter>

Implements

[IDatabaseAdapter](#)
System.IDisposable

Inherited Members

System.Object.Equals(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.GetHashCode()
System.Object.GetType()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.ToString()

Namespace: [XData.Database.Adapter](#)

Assembly: XData.docfx.dll

Syntax

```
[DatabaseAdapter("NONE")]
public abstract class BaseDatabaseAdapter<TConnection, TParameter> : IDatabaseAdapter, IDisposable where
TConnection : DbConnection, new()
    where TParameter : IDbDataParameter, new()
```

Type Parameters

NAME	DESCRIPTION
TConnection	
TParameter	

Properties

Config

Adapter configuration

Declaration

```
protected IConfiguration Config { get; }
```

Property Value

TYPE	DESCRIPTION
Microsoft.Extensions.Configuration.IConfiguration	

ConnectType

Adapter name

Declaration

```
public string ConnectType { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Context

Database context name

Declaration

```
protected string Context { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

DialectAdapter

SQL dialect adapter instance

Declaration

```
protected ISqlBuilder DialectAdapter { get; }
```

Property Value

TYPE	DESCRIPTION
ISqlBuilder	

Initialized

Declaration

```
protected bool Initialized { get; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

NotSupported

Check adapter supports user defined types (UDT)

Declaration

```
public virtual DialectOptions NotSupported { get; }
```

Property Value

TYPE	DESCRIPTION
DialectOptions	

Methods

AddSavePoint(Guid, IDbTransaction, String)

Add named save point

Declaration

```
protected abstract void AddSavePoint(Guid layer, IDbTransaction transaction, string savePoint)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Guid	layer	Layer
System.Data.IDbTransaction	transaction	Transaction
System.String	savePoint	Save point name

BeginTransaction(Guid, Boolean, IsolationLevel)

Begin transaction

Declaration

```
public string BeginTransaction(Guid layer, bool read = false, IsolationLevel isolation = IsolationLevel.ReadCommitted)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Guid	layer	Layer
System.Boolean	read	Read only transaction flag
System.Data.IsolationLevel	isolation	Transaction isolation level

Returns

TYPE	DESCRIPTION
System.String	Transaction key

Exceptions

TYPE	CONDITION
XDataSqlException	SQL exceptions while beginning transaction.

Call(Guid, StoredProcedure, CancellationToken)

Calls stored procedure

Declaration

```
public CallResult Call(Guid layer, StoredProcedure procedure, CancellationToken token)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Guid	layer	Layer
StoredProcedure	procedure	Stored procedure
System.Threading.CancellationToken	token	Cancellation token

Returns

TYPE	DESCRIPTION
CallResult	CallResult

Exceptions

TYPE	CONDITION
XDataSqlException	

Close(Guid)

Close database connection

Declaration

```
public void Close(Guid layer)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Guid	layer	

Exceptions

TYPE	CONDITION

TYPE	CONDITION
XDataSqlException	SQL exceptions while closing connection

Commit(Guid, String)

Commit transaction

Declaration

```
public bool Commit(Guid layer, string transactionId)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Guid	layer	Layer
System.String	transactionId	Transaction key

Returns

TYPE	DESCRIPTION
System.Boolean	Success flag

CreateCommand(TConnection)

Create specific SQL command

Declaration

```
protected virtual IDbCommand CreateCommand(TConnection connection)
```

Parameters

TYPE	NAME	DESCRIPTION
TConnection	connection	SQL connection

Returns

TYPE	DESCRIPTION
System.Data.IDbCommand	SQL command

Remarks

Override this when some special command parameters are required

CreateParameter(Guid, QueryParam)

Create common parameter

Create common parameter

Declaration

```
protected virtual TParameter CreateParameter(Guid layer, QueryParam parameter)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Guid	layer	Layer
QueryParam	parameter	Query parameter

Returns

TYPE	DESCRIPTION
TParameter	Parameter

Dispose()

IDisposable realization

Declaration

```
public void Dispose()
```

ExecuteNonQuery(Guid, Enum, Object[])

Executes SQL expression with no result set

Declaration

```
public QueryResult ExecuteNonQuery(Guid layer, Enum queryIndex, params object[] paramValues)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Guid	layer	Layer
System.Enum	queryIndex	Predefined query enumeration key
System.Object[]	paramValues	Query parameter values

Returns

TYPE	DESCRIPTION
QueryResult	Query result

Exceptions

TYPE	CONDITION
XDataSqlException	wrong SQL query parameters

ExecuteNonQuery(Guid, Query)

Executes SQL expression with no result set

Declaration

<pre>public QueryResult ExecuteNonQuery(Guid layer, Query query)</pre>

Parameters

TYPE	NAME	DESCRIPTION
System.Guid	layer	Layer
Query	query	Query

Returns

TYPE	DESCRIPTION
QueryResult	Query result

Exceptions

TYPE	CONDITION
XDataSqlException	wrong SQL query
XDataConcurrencyException	concurrency exception

ExecuteScalar(Guid, Enum, Object[])

Executes scalar returning query

Declaration

<pre>public object ExecuteScalar(Guid layer, Enum queryIndex, params object[] paramValues)</pre>

Parameters

TYPE	NAME	DESCRIPTION
System.Guid	layer	Layer

TYPE	NAME	DESCRIPTION
System.Enum	queryIndex	Predefined query enumeration key
System.Object[]	paramValues	Query parameter values

Returns

TYPE	DESCRIPTION
System.Object	Scalar value

Exceptions

TYPE	CONDITION
XDataSqlException	wrong SQL query parameters

ExecuteScalar(Guid, Query)

Executes scalar returning query

Declaration

public object ExecuteScalar(Guid layer, Query query)

Parameters

TYPE	NAME	DESCRIPTION
System.Guid	layer	Layer
Query	query	Query

Returns

TYPE	DESCRIPTION
System.Object	Reader

Exceptions

TYPE	CONDITION
XDataSqlException	wrong SQL query

TYPE	CONDITION
XDataConcurrencyException	concurrency exception

FillTable(Guid, Enum, CommandBehavior, Object[])

Executes query and fill data table

Declaration

```
public DataTable FillTable(Guid layer, Enum queryIndex, CommandBehavior behavior, params object[] paramValues)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Guid	layer	Layer
System.Enum	queryIndex	Predefined query enumeration key
System.Data.CommandBehavior	behavior	Command behavior
System.Object[]	paramValues	Query parameter values

Returns

TYPE	DESCRIPTION
System.Data.DataTable	Filled data table

Exceptions

TYPE	CONDITION
XDataSqlException	wrong SQL query parameters

FillTable(Guid, Enum, Object[])

Executes query and fill data table

Declaration

```
public DataTable FillTable(Guid layer, Enum queryIndex, params object[] paramValues)
```

Parameters

TYPE	NAME	DESCRIPTION

TYPE	NAME	DESCRIPTION
System.Guid	layer	Layer
System.Enum	queryIndex	Predefined query enumeration key
System.Object[]	paramValues	Query parameter values

Returns

TYPE	DESCRIPTION
System.Data.DataTable	Filled data table

Exceptions

TYPE	CONDITION
XDataSqlException	wrong SQL query parameters

FillTable(Guid, Query, CancellationToken, CommandBehavior)

Executes query and fill data table

Declaration

```
public DataTable FillTable(Guid layer, Query query, CancellationToken token, CommandBehavior behavior = CommandBehavior.Default)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Guid	layer	Layer
Query	query	Query
System.Threading.CancellationToken	token	Cancellation token
System.Data.CommandBehavior	behavior	Command behavior

Returns

TYPE	DESCRIPTION

TYPE	DESCRIPTION
System.Data.DataTable	Filled data table

Exceptions

TYPE	CONDITION
XDataSqlException	wrong SQL query

FixFieldType(DataRow)

Convert field data type

Declaration

```
protected virtual Type FixFieldType(DataRow columnInfo)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Data.DataRow	columnInfo	Field description from schema table

Returns

TYPE	DESCRIPTION
System.Type	Correct data type

GetConnection(Guid)

Database connection

Declaration

```
protected DbConnection GetConnection(Guid layer)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Guid	layer	

Returns

TYPE	DESCRIPTION
System.Data.Common.DbConnection	

GetNativeType(DbType)

Declaration

```
protected string GetNativeType(DbType type)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Data.DbType	type	

Returns

TYPE	DESCRIPTION
System.String	

Exceptions

TYPE	CONDITION
XDataAdapterException	

GetParameterValue(QueryParam)

Returns parameter value

Declaration

```
public object GetParameterValue(QueryParam parameter)
```

Parameters

TYPE	NAME	DESCRIPTION
QueryParam	parameter	Query parameter

Returns

TYPE	DESCRIPTION
System.Object	Parameter value

GetParameterValue<T>(QueryParam)

Declaration

```
public T GetParameterValue<T>(QueryParam parameter)
```

Parameters

TYPE	NAME	DESCRIPTION
QueryParam	parameter	

Returns

TYPE	DESCRIPTION
T	

NAME	DESCRIPTION
T	

Exceptions

TYPE	CONDITION
XDataAdapterException	

GetProcedureCall(ref IDbCommand, StoredProcedure, String, Boolean)

Declaration

```
protected virtual void GetProcedureCall(ref IDbCommand command, StoredProcedure procedure, string getResultName = null, bool closeCursor = false)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Data.IDbCommand	command	
StoredProcedure	procedure	
System.String	getResultName	
System.Boolean	closeCursor	

GetSqlType(String, ref Int32, ref Int32, Int32)

Declaration

```
protected DbType GetSqlType(string typeName, ref int size, ref int scale, int precision)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	typeName	
System.Int32	size	
System.Int32	scale	
System.Int32	precision	

Returns

TYPE	DESCRIPTION
System.Data.DbType	

Exceptions

TYPE	CONDITION
XDataAdapterException	Type not supported

GetTableFieldInfo(Guid, String)

Declaration

```
public abstract ReadOnlyCollection<FieldInfo> GetTableFieldInfo(Guid layer, string table)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Guid	layer	
System.String	table	

Returns

TYPE	DESCRIPTION
System.Collections.ObjectModel.ReadOnlyCollection< FieldInfo >	

GetTablesInfo(Guid)

Declaration

```
public abstract ReadOnlyCollection<TableInfo> GetTablesInfo(Guid layer)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Guid	layer	

Returns

TYPE	DESCRIPTION
System.Collections.ObjectModel.ReadOnlyCollection< TableInfo >	

GetUtdParameter(QueryParam)

Create and initialize user data type (UDT) parameter
Hint: Override this method to use UDT stored procedure parameters

Declaration

```
protected virtual IDataParameter GetUtdParameter(QueryParam parameter)
```

Parameters

TYPE	NAME	DESCRIPTION

TYPE	NAME	DESCRIPTION
QueryParam	parameter	Stored procedure parameter

Returns

TYPE	DESCRIPTION
System.Data.IDataParameter	System.Data.IDataParameter

Exceptions

TYPE	CONDITION
XDataAdapterException	User Defined Type (UTD)

Init()

Adapter initialization

Declaration

<code>protected abstract void Init()</code>

Init(ISqlBuilder, String, String, IConfiguration)

Adapter initialization logic

Declaration

<code>public void Init(ISqlBuilder dialectAdapter, string context, string connectionString, IConfiguration config)</code>

Parameters

TYPE	NAME	DESCRIPTION
ISqlBuilder	dialectAdapter	SQL dialect adapter
System.String	context	Context name
System.String	connectionString	Database connection string
Microsoft.Extensions.Configuration.IConfiguration	config	Adapter configuration

InitConnection(DbConnection)

Declaration

<code>protected virtual void InitConnection(DbConnection connection)</code>

Parameters

TYPE	NAME	DESCRIPTION
System.Data.Common.DbConnection	connection	

InTransaction(Guid)

Connection have not committed transaction flag

Declaration

```
public bool InTransaction(Guid layer)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Guid	layer	

Returns

TYPE	DESCRIPTION
System.Boolean	

IsConcurrencyError(Exception)

Check exception is a concurrency exception

Declaration

```
protected virtual bool IsConcurrencyError(Exception exception)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Exception	exception	Exception

Returns

TYPE	DESCRIPTION
System.Boolean	Exception is a concurrency exception

IsOpen(Guid)

Connection is open flag

Declaration

```
public bool IsOpen(Guid layer)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Guid	layer	

Returns

TYPE	DESCRIPTION
System.Boolean	

ReadNativeValue<T>(String, Object)

Read native SQL value

Declaration

```
protected virtual T ReadNativeValue<T>(string nativeSqlType, object value)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	nativeSqlType	Native SQL type name
System.Object	value	Value

Returns

TYPE	DESCRIPTION
T	Mapped value

Type Parameters

NAME	DESCRIPTION
T	Mapped type

Exceptions

TYPE	CONDITION
XDataAdapterException	

ReadUtd<T>(Object)

Read data from wrapped user defined type (UTD)

Declaration

```
protected virtual T ReadUtd<T>(object utdObject)
    where T : class
```

Parameters

TYPE	NAME	DESCRIPTION

TYPE	NAME	DESCRIPTION
System.Object	utdObject	Wrapped value

Returns

TYPE	DESCRIPTION
T	Unwrapped value

Type Parameters

NAME	DESCRIPTION
T	Type mapped to user defined type (UTD)

Exceptions

TYPE	CONDITION
XDataAdapterException	User Defined Type (UTD)

ReadUtdCollection<T>(Object)

Read data from wrapped user defined type (UTD)

Declaration

```
protected virtual ICollection<T> ReadUtdCollection<T>(object utdCollection)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Object	utdCollection	Wrapped collection

Returns

TYPE	DESCRIPTION
System.Collections.Generic.ICollection<T>	Unwrapped collection

Type Parameters

NAME	DESCRIPTION
T	Collection element type mapped to user defined type (UTD)

Exceptions

TYPE	CONDITION
XDataAdapterException	User Defined Type (UTD)

RegisterResultSet(String)

Register result set parameter of stored procedure

Declaration

```
protected virtual IDbDataParameter RegisterResultSet(string resultSetName)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	resultSetNames	Result set name

Returns

TYPE	DESCRIPTION
System.Data.IDbDataParameter	Result set parameter

Remarks

WARNING! Override this method to register result sets as out parameters

Exceptions

TYPE	CONDITION
XDataAdapterException	RegisterResultSet

See Also

- [ResultSets](#)
- [MultipleRecordsetProceduresSupported](#)
- [RegisterRecordsetRequired](#)

Rollback(Guid, String)

Rollback transaction

Declaration

```
public bool Rollback(Guid layer, string transactionId)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Guid	layer	Layer

TYPE	NAME	DESCRIPTION
System.String	transactionId	Transaction key

Returns

TYPE	DESCRIPTION
System.Boolean	Success flag

RollbackToSavePoint(Guid, IDbTransaction, String)

Rollback to named save point

Declaration

```
protected abstract void RollbackToSavePoint(Guid layer, IDbTransaction transaction, string savePoint)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Guid	layer	Layer
System.Data.IDbTransaction	transaction	Transaction
System.String	savePoint	Save point name

SafeConvert<T>(Object)

Safe conversion for Db specific types

Declaration

```
public virtual T SafeConvert<T>(object data)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Object	data	Source data

Returns

TYPE	DESCRIPTION
T	Source data converted to type

Type Parameters

NAME	DESCRIPTION
T	Result type

Remarks

Override this to convert some specific Db related types

See Also

[SafeTypeConvert](#)

SelectParameterValue<T>(QueryParam)

Declaration

```
public IEnumerable<T> SelectParameterValue<T>(QueryParam parameter)
```

Parameters

TYPE	NAME	DESCRIPTION
QueryParam	parameter	

Returns

TYPE	DESCRIPTION
System.Collections.Generic.IEnumerable<T>	

Type Parameters

NAME	DESCRIPTION
T	

Exceptions

TYPE	CONDITION
XDataAdapterException	

Test(Guid, Query)

Test query syntax WARNING! Throws a error if test failed!

Declaration

```
public void Test(Guid layer, Query query)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Guid	layer	Layer

TYPE	NAME	DESCRIPTION
Query	query	Query

Exceptions

TYPE	CONDITION
XDataSqlException	wrong SQL query

WrapUtd<T>(T, String)

Wrap data to user defined type (UTD)

Declaration

```
protected virtual object WrapUtd<T>(T value, string utdTypeName)
    where T : class
```

Parameters

TYPE	NAME	DESCRIPTION
T	value	Value to wrap
System.String	utdTypeName	User defined type name

Returns

TYPE	DESCRIPTION
System.Object	Wrapped value

Type Parameters

NAME	DESCRIPTION
T	Type mapped to user defined type (UTD)

Exceptions

TYPE	CONDITION
XDataAdapterException	User Defined Type (UTD)

WrapUtdCollection<T>(T[], Boolean, String, String)

Wrap data to collection (table or array) user defined type (UTD)

Declaration

protected virtual object WrapUtdCollection<T>(T[] value, bool isArray, string utdTypeName, string utdElementName = null)

Parameters

TYPE	NAME	DESCRIPTION
T[]	value	Collection to wrap
System.Boolean	isArray	Use VARARRAY
System.String	utdTypeName	User defined type name for collection
System.String	utdElementName	User defined type name for element (set to null for scalar type collection)

Returns

TYPE	DESCRIPTION
System.Object	Wrapped collection

Type Parameters

NAME	DESCRIPTION
T	Collection element type mapped to user defined type (UTD)

Exceptions

TYPE	CONDITION
XDataAdapterException	User Defined Type (UTD)

Implements

[IDatabaseAdapter](#)

System.IDisposable

Extension Methods

[SerializationExtensions.ToXml<T>\(T\)](#)

[CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)

[CollectionsExtensions.AsEnum<T>\(T\)](#)

[DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)

[Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)

[TypeExtensions.ChangeType<T>\(Object\)](#)

[TypeExtensions.ChangeType\(Object, Type\)](#)

TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)

TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)

SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)

QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Class CallResult

Stored procedure call result

Inheritance

System.Object
QueryResultBase
CallResult

Implements

System.Collections.Generic.IEnumerable<QueryParam>
System.Collections.IEnumerable

Inherited Members

QueryResultBase.AffectedRows
QueryResultBase.Item[Int32]
QueryResultBase.Item[String]
QueryResultBase.ParamsCount
QueryResultBase.Init(Int32, IDataParameterCollection)
QueryResultBase.Init(Int32, IEnumerable<IDataParameter>)
QueryResultBase.Init(Int32, IEnumerable<QueryParam>)
QueryResultBase.GetEnumerator()
QueryResultBase.IEnumerable.GetEnumerator()
System.Object.Equals(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.GetHashCode()
System.Object.GetType()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.ToString()

Namespace: XData.Database.Adapter

Assembly: XData.docfx.dll

Syntax

```
public sealed class CallResult : QueryResultBase, IEnumerable<QueryParam>, IEnumerable
```

Constructors

CallResult(StoredProcedure, IEnumerable<IDataParameter>, IList<DataTable>)

Stored procedure call result constructor

Declaration

```
public CallResult(StoredProcedure procedure, IEnumerable<IDataParameter> procedureParams, IList<DataTable> resultSets)
```

Parameters

TYPE	NAME	DESCRIPTION
StoredProcedure	procedure	Stored procedure
System.Collections.Generic.IEnumerable<System.Data.IDataParameter>	procedureParams	Procedure parameters

TYPE	NAME	DESCRIPTION
System.Collections.Generic.IList<System.Data.DataTable>	resultSets	Result sets list

CallResult(StoredProcedure, IEnumerable<QueryParam>, IList<DataTable>)

Stored procedure call result constructor

Declaration

```
public CallResult(StoredProcedure procedure, IEnumerable<QueryParam> procedureParams, IList<DataTable> resultSets)
```

Parameters

TYPE	NAME	DESCRIPTION
StoredProcedure	procedure	Stored procedure
System.Collections.Generic.IEnumerable< QueryParam >	procedureParams	Procedure parameters sequence
System.Collections.Generic.IList<System.Data.DataTable>	resultSets	Result sets list

CallResult(StoredProcedure, IDataParameterCollection, IList<DataTable>)

Stored procedure call result constructor

Declaration

```
public CallResult(StoredProcedure procedure, IDataParameterCollection procedureParams, IList<DataTable> resultSets)
```

Parameters

TYPE	NAME	DESCRIPTION
StoredProcedure	procedure	Stored procedure
System.Data.IDataParameterCollection	procedureParams	Procedure parameters
System.Collections.Generic.IList<System.Data.DataTable>	resultSets	Result sets list

Properties

ResultSets

Stored procedure result sets

Declaration

```
public ReadOnlyCollection<DataTable> ResultSets { get; }
```

Property Value

TYPE	DESCRIPTION
System.Collections.ObjectModel.ReadOnlyCollection<System.Data.DataTable>	

Remarks

Hint: Data table name contains result set name

Methods

ParamInit(IDataParameter)

Update out parameter value

Declaration

```
protected override QueryParam ParamInit(IDataParameter parameter)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Data.IDataParameter	parameter	Parameter

Returns

TYPE	DESCRIPTION
QueryParam	Query parameter

Overrides

[QueryResultBase.ParamInit\(IDataParameter\)](#)

Exceptions

TYPE	CONDITION
XDataRuntimeExcep	

Implements

System.Collections.Generic.IEnumerable<T>

System.Collections.IEnumerable

Extension Methods

[SerializationExtensions.ToXml<T>\(T\)](#)

[CollectionsExtensions.ToSortedSet<T>\(IEnumerable<T>, IComparer<T>\)](#)

[CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)

[CollectionsExtensions.AsEnum<T>\(T\)](#)

[DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)

[JoinedEnumerable.Inner<TElement>\(IEnumerable<TElement>\)](#)

[JoinedEnumerable.Outer<TElement>\(IEnumerable<TElement>\)](#)

[JoinedEnumerable.LeftOuterJoin<TOuter, TInner, TKey, TResult>\(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter,](#)

TKey>, Func<TInner, TKey>, Func<TOuter, TInner, TResult>, IEqualityComparer<TKey>)

JoinedEnumerable.RightOuterJoin<TOuter, TInner, TKey, TResult>(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter, TKey>, Func<TInner, TKey>, Func<TOuter, TInner, TResult>, IEqualityComparer<TKey>)

JoinedEnumerable.FullOuterJoin<TOuter, TInner, TKey, TResult>(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter, TKey>, Func<TInner, TKey>, Func<TOuter, TInner, TResult>, IEqualityComparer<TKey>)

Processing.Do<T>(T, Action<IProcess<T>>[])

TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>)

TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>, TResult)

TypeExtensions.ChangeType<T>(Object)

TypeExtensions.ChangeType(Object, Type)

TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)

TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)

SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)

QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

QueryDescriptionExtensions.ToDataSet<T>(IEnumerable<T>, String, ISqlBuilder)

Class DatabaseAdapterAttribute

Database adapter attribute

Inheritance

System.Object

System.Attribute

DatabaseAdapterAttribute

Implements

[IDatabaseAdapterMetadata](#)

Inherited Members

System.Attribute.Equals(System.Object)

System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Module)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.GetHashCode()

System.Attribute.IsDefaultAttribute()

System.Attribute.IsDefined(System.Reflection.Assembly, System.Type)

System.Attribute.IsDefined(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type)

System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.Module, System.Type)

System.Attribute.IsDefined(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type)

System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.Match(System.Object)

System.Attribute.TypeId

System.Object.Equals(System.Object, System.Object)

System.Object.GetType()

System.Object.MemberwiseClone()

System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.ToString()

Namespace: [XData.Database.Adapter](#)
Assembly: XData.docfx.dll

Syntax

```
[AttributeUsage(AttributeTargets.Class)]  
public class DatabaseAdapterAttribute : Attribute, IDatabaseAdapterMetadata
```

Constructors

DatabaseAdapterAttribute(String)

Database adapter attribute constructor

Declaration

```
public DatabaseAdapterAttribute(string adapterDatabase)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	adapterDatabase	Connection type

Properties

Database

Database connection type

Declaration

```
public string Database { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Implements

[IDatabaseAdapterMetadata](#)

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)

TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)

SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)

QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Interface IDatabaseAdapter

Database adapter basic interface

Inherited Members

System.IDisposable.Dispose()

Namespace: [XData.Database.Adapter](#)

Assembly: XData.docfx.dll

Syntax

```
public interface IDatabaseAdapter : IDisposable
```

Properties

ConnectType

Database connection type

Declaration

```
string ConnectType { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

NotSupported

Dialect options not supported by adapter

Declaration

```
DialectOptions NotSupported { get; }
```

Property Value

TYPE	DESCRIPTION
DialectOptions	

Methods

BeginTransaction(Guid, Boolean, IsolationLevel)

Begin transaction

Declaration

```
string BeginTransaction(Guid layer, bool read = false, IsolationLevel isolation = IsolationLevel.ReadCommitted)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Guid	layer	Layer

TYPE	NAME	DESCRIPTION
System.Boolean	read	Read only transaction flag
System.Data.IsolationLevel	isolation	Transaction isolation level

Returns

TYPE	DESCRIPTION
System.String	Success flag

Exceptions

TYPE	CONDITION
XDataSqlException	SQL exceptions while beginning transaction.

Call(Guid, StoredProcedure, CancellationToken)

Execute SQL stored procedure

Declaration

CallResult Call(Guid layer, StoredProcedure procedure, CancellationToken token)

Parameters

TYPE	NAME	DESCRIPTION
System.Guid	layer	Layer
StoredProcedure	procedure	Procedure
System.Threading.CancellationToken	token	Cancellation token

Returns

TYPE	DESCRIPTION
CallResult	Call result

Exceptions

TYPE	CONDITION

TYPE	CONDITION
XDataSqlException	wrong SQL stored procedure

Close(Guid)

Close database adapter

Declaration

```
void Close(Guid layer)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Guid	layer	Layer

Exceptions

TYPE	CONDITION
XDataSqlException	SQL exceptions while closing connection

Commit(Guid, String)

Commit transaction

Declaration

```
bool Commit(Guid layer, string transactionId)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Guid	layer	Layer
System.String	transactionId	Transaction id

Returns

TYPE	DESCRIPTION
System.Boolean	Success flag

ExecuteNonQuery(Guid, Enum, Object[])

Execute SQL statement

Declaration

```
QueryResult ExecuteNonQuery(Guid layer, Enum queryIndex, params object[] paramValues)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Guid	layer	Layer
System.Enum	queryIndex	Predefined query index
System.Object[]	paramValues	Query parameters

Returns

TYPE	DESCRIPTION
QueryResult	Query result

Exceptions

TYPE	CONDITION
XDataSqlException	wrong SQL query parameters

ExecuteNonQuery(Guid, Query)

Execute SQL statement

Declaration

```
QueryResult ExecuteNonQuery(Guid layer, Query query)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Guid	layer	Layer
Query	query	Query

Returns

TYPE	DESCRIPTION
QueryResult	Query result

Exceptions

TYPE	CONDITION
XDataSqlException	wrong SQL query

ExecuteScalar(Guid, Enum, Object[])

Retrieve scalar value

Declaration

```
object ExecuteScalar(Guid layer, Enum queryIndex, params object[] paramValues)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Guid	layer	Layer
System.Enum	queryIndex	Predefined query index
System.Object[]	paramValues	Query parameters

Returns

TYPE	DESCRIPTION
System.Object	Object

Exceptions

TYPE	CONDITION
XDataSqlException	wrong SQL query parameters

ExecuteScalar(Guid, Query)

Retrieve scalar value

Declaration

```
object ExecuteScalar(Guid layer, Query query)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Guid	layer	Layer

TYPE	NAME	DESCRIPTION
Query	query	Query

Returns

TYPE	DESCRIPTION
System.Object	Object

Exceptions

TYPE	CONDITION
XDataSqlException	wrong SQL query

FillTable(Guid, Enum, CommandBehavior, Object[])

Fill data table

Declaration

```
DataTable FillTable(Guid layer, Enum queryIndex, CommandBehavior behavior, params object[] paramValues)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Guid	layer	Layer
System.Enum	queryIndex	Predefined query index
System.Data.CommandBehavior	behavior	Command behavior
System.Object[]	paramValues	Query parameters

Returns

TYPE	DESCRIPTION
System.Data.DataTable	Data table

Exceptions

TYPE	CONDITION

TYPE	CONDITION
XDataSqlException	wrong SQL query parameters

FillTable(Guid, Enum, Object[])

Fill data table

Declaration

```
DataTable FillTable(Guid layer, Enum queryIndex, params object[] paramValues)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Guid	layer	Layer
System.Enum	queryIndex	Predefined query index
System.Object[]	paramValues	Query parameters

Returns

TYPE	DESCRIPTION
System.Data.DataTable	Data table

Exceptions

TYPE	CONDITION
XDataSqlException	wrong SQL query parameters

FillTable(Guid, Query, CancellationToken, CommandBehavior)

Fill data table

Declaration

```
DataTable FillTable(Guid layer, Query query, CancellationToken token, CommandBehavior behavior = CommandBehavior.Default)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Guid	layer	Layer

TYPE	NAME	DESCRIPTION
Query	query	Query
System.Threading.CancellationToken	token	Cancellation token
System.Data.CommandBehavior	behavior	Command behavior

Returns

TYPE	DESCRIPTION
System.Data.DataTable	Data table

Exceptions

TYPE	CONDITION
XDataSqlException	wrong SQL query

GetParameterValue(QueryParam)

Read parameter value

Declaration

```
object GetParameterValue(QueryParam parameter)
```

Parameters

TYPE	NAME	DESCRIPTION
QueryParam	parameter	Parameter

Returns

TYPE	DESCRIPTION
System.Object	Parameter value

GetParameterValue<T>(QueryParam)

Read parameter value

Declaration

```
T GetParameterValue<T>(QueryParam parameter)
```

Parameters

TYPE	NAME	DESCRIPTION
QueryParam	parameter	Parameter

Returns

TYPE	DESCRIPTION
T	Parameter value

Type Parameters

NAME	DESCRIPTION
T	

GetTableFieldInfo(Guid, String)

Returns table structure

Declaration

```
ReadOnlyCollection<FieldInfo> GetTableFieldInfo(Guid layer, string table)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Guid	layer	Layer
System.String	table	Table name

Returns

TYPE	DESCRIPTION
System.Collections.ObjectModel.ReadOnlyCollection< FieldInfo >	List of fields

GetTablesInfo(Guid)

Returns table list

Declaration

```
ReadOnlyCollection<TableInfo> GetTablesInfo(Guid layer)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Guid	layer	Layer

Returns

TYPE	DESCRIPTION
System.Collections.ObjectModel.ReadOnlyCollection< TableInfo >	List of table

Init(ISqlBuilder, String, String, IConfiguration)

Initialization routine

Declaration

```
void Init(ISqlBuilder dialectAdapter, string context, string connectionString, IConfiguration config)
```

Parameters

TYPE	NAME	DESCRIPTION
ISqlBuilder	dialectAdapter	SQL dialect adapter
System.String	context	Context name
System.String	connectionString	Connection string
Microsoft.Extensions.Configuration.IConfiguration	config	Adapter configuration

InTransaction(Guid)

Transaction context flag

Declaration

```
bool InTransaction(Guid layer)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Guid	layer	Layer

Returns

TYPE	DESCRIPTION
System.Boolean	

IsOpen(Guid)

Check adapter connection is open

Declaration

```
bool IsOpen(Guid layer)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Guid	layer	Layer

Returns

TYPE	DESCRIPTION
System.Boolean	

Rollback(Guid, String)

Rollback transaction

Declaration

```
bool Rollback(Guid layer, string transactionId)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Guid	layer	Layer
System.String	transactionId	Transaction id

Returns

TYPE	DESCRIPTION
System.Boolean	Success flag

SafeConvert<T>(Object)

Safe convert data

Declaration

```
T SafeConvert<T>(object data)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Object	data	Data to convert

Returns

TYPE	DESCRIPTION
T	Converted data

Type Parameters

NAME	DESCRIPTION
T	Result type

SelectParameterValue<T>(QueryParam)

Iterate user data type (UDT) valued table/array parameter value

Declaration

<pre>IEnumerable<T> SelectParameterValue<T>(QueryParam parameter)</pre>

Parameters

TYPE	NAME	DESCRIPTION
QueryParam	parameter	Parameter

Returns

TYPE	DESCRIPTION
System.Collections.Generic.IEnumerable<T>	Parameter value

Type Parameters

NAME	DESCRIPTION
T	

Test(Guid, Query)

Test query

Declaration

<pre>void Test(Guid layer, Query query)</pre>

Parameters

TYPE	NAME	DESCRIPTION
System.Guid	layer	Layer
Query	query	Query

Extension Methods

SerializationExtensions.ToXml<T>(T)

CollectionsExtensions.SetValue<TKey, T>(TKey, T)

CollectionsExtensions.AsEnum<T>(T)

DataObjectQueryableExtensions.CompareVariable(Object, FilterOperation, String)

Processing.Do<T>(T, Action<IProcess<T>>[])

TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>)

TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>, TResult)

TypeExtensions.ChangeType<T>(Object)

TypeExtensions.ChangeType(Object, Type)

TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)

TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)

SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)

QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Interface IDatabaseAdapterMetadata

Database adapter description interface

Namespace: [XData.Database.Adapter](#)

Assembly: XData.docfx.dll

Syntax

```
public interface IDatabaseAdapterMetadata
```

Properties

Database

Database connection type

Declaration

```
string Database { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Class Query

SQL query

Inheritance

System.Object

Query

Implements

System.Collections.Generic.IEnumerable<[QueryParam](#)>

System.Collections.IEnumerable

Inherited Members

System.Object.Equals(System.Object)

System.Object.Equals(System.Object, System.Object)

System.Object.GetHashCode()

System.Object.GetType()

System.Object.MemberwiseClone()

System.Object.ReferenceEquals(System.Object, System.Object)

Namespace: [XData.Database.Adapter](#)

Assembly: XData.docfx.dll

Syntax

```
public sealed class Query : IEnumerable<QueryParam>, IEnumerable
```

Constructors

Query(String, CommandType, QueryParam[])

Query constructor

Declaration

```
public Query(string sql, CommandType type = CommandType.Text, params QueryParam[] par)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	sql	Sql expression
System.Data.CommandType	type	Command type
QueryParam []	par	Parameters

Properties

Item[Int32]

Query parameters by index

Declaration

```
public QueryParam this[int i] { get; }
```

Parameters

TYPE	NAME	DESCRIPTION
System.Int32	i	Parameter index

Property Value

TYPE	DESCRIPTION
QueryParam	Query parameter

Item[String]

Query parameters by name

Declaration

```
public QueryParam this[string name] { get; set; }
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Parameter name

Property Value

TYPE	DESCRIPTION
QueryParam	Query parameter

Sql

Query SQL expression

Declaration

```
public string Sql { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Type

Query command type

Declaration

```
public CommandType Type { get; }
```

Property Value

TYPE	DESCRIPTION
System.Data.CommandType	

Methods

GetEnumerator()

Query parameters enumeration

Declaration

```
public IEnumerator<QueryParam> GetEnumerator()
```

Returns

TYPE	DESCRIPTION
System.Collections.Generic.IEnumerator<QueryParam>	Query parameters enumeration

ToString()

Returns a System.String that represents the current System.Object.

Declaration

```
public override string ToString()
```

Returns

TYPE	DESCRIPTION
System.String	A System.String that represents the current System.Object.

Overrides

System.Object.ToString()

Operators

Implicit(String to Query)

Query SQL expression

Declaration

```
public static implicit operator Query(string s)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	s	Sql expression

Returns

TYPE	DESCRIPTION
Query	Query

Explicit Interface Implementations

[IEnumerable.GetEnumerator\(\)](#)

[Query parameters enumeration](#)

Declaration

<code>IEnumerator IEnumerable.GetEnumerator()</code>

Returns

TYPE	DESCRIPTION
System.Collections.IEnumerator	Query parameters enumeration

Implements

[System.Collections.Generic.IEnumerable<T>](#)

[System.Collections.IEnumerable](#)

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.ToSortedSet<T>\(IEnumerable<T>, IComparer<T>\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [JoinedEnumerable.Inner<TElement>\(IEnumerable<TElement>\)](#)
- [JoinedEnumerable.Outer<TElement>\(IEnumerable<TElement>\)](#)
- [JoinedEnumerable.LeftOuterJoin<TOuter, TInner, TKey, TResult>\(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter, TKey>, Func<TInner, TKey>, Func<TOuter, TInner, TResult>, IEqualityComparer<TKey>\)](#)
- [JoinedEnumerable.RightOuterJoin<TOuter, TInner, TKey, TResult>\(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter, TKey>, Func<TInner, TKey>, Func<TOuter, TInner, TResult>, IEqualityComparer<TKey>\)](#)
- [JoinedEnumerable.FullOuterJoin<TOuter, TInner, TKey, TResult>\(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter, TKey>, Func<TInner, TKey>, Func<TOuter, TInner, TResult>, IEqualityComparer<TKey>\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.ToDataSet<T>\(IEnumerable<T>, String, ISqlBuilder\)](#)

Class QueryParam

SQL query parameter

Inheritance

System.Object
QueryParam

Inherited Members

System.Object.Equals(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.GetHashCode()
System.Object.GetType()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)

Namespace: [XData.Database.Adapter](#)

Assembly: XData.docfx.dll

Syntax

```
public class QueryParam
```

Constructors

QueryParam(String, Object, DbType, Int32)

Query parameter constructor

Declaration

```
public QueryParam(string name, object value = null, DbType type = DbType.String, int length = 0)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Parameter name
System.Object	value	Parameter value
System.Data.DbType	type	Parameter data type
System.Int32	length	Parameter value max length

QueryParam(QueryParam, Object)

Query parameter copy constructor

Declaration

```
public QueryParam(QueryParam param, object value = null)
```

Parameters

TYPE	NAME	DESCRIPTION
QueryParam	param	Parameter name
System.Object	value	Parameter value

Properties

Direction

Parameter direction

Declaration

```
public ParameterDirection Direction { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Data.ParameterDirection	

IsArray

Collection type array flag

Set to true to use with VARARRAY collections

WARNING! Using UDT can be not supported or limited by adapter!

Declaration

```
public bool IsArray { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

IsResultSet

Parameter is result set

Declaration

```
public bool IsResultSet { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

Length

Parameter value max length

Declaration

```
public int Length { get; }
```

Property Value

TYPE	DESCRIPTION
System.Int32	

Name

Parameter name

Declaration

```
public string Name { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

NativeSqlType

Parameter native SQL type

Declaration

```
public string NativeSqlType { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Type

Parameter data type

Declaration

```
public DbType Type { get; }
```

Property Value

TYPE	DESCRIPTION
System.Data.DbType	

UdtElementTypeName

Single object (or collection element) user defined type (UDT) SQL type name

WARNING! Specify full qualified type name!

WARNING! Using UDT can be not supported or limited by adapter!

Declaration


```
public string UdtElementTypeName { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

UdtMappedType

Class type mapped with user defined type (UDT) SQL type name

This type properties are mapped to SQL type by name (case insensitive)

WARNING! Beware to use property data types not supported by adapter or user defined type (UDT)!

WARNING! Using UDT can be not supported or limited by adapter!

Declaration

```
public Type UdtMappedType { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Type	

UdtTypeName

Collection user defined type (UDT) SQL type name

WARNING! Specify full qualified type name!

WARNING! Using UDT can be not supported or limited by adapter!

Declaration

```
public string UdtTypeName { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Value

Parameter value

Declaration

```
public object Value { get; }
```

Property Value

TYPE	DESCRIPTION
System.Object	

Methods

ToString()

Returns a System.String that represents the current System.Object.

Declaration

```
public override string ToString()
```

Returns

TYPE	DESCRIPTION
System.String	A System.String that represents the current System.Object.

Overrides

System.Object.ToString()

Extension Methods

- SerializationExtensions.ToXml<T>(T)
- CollectionsExtensions.SetValue<TKey, T>(TKey, T)
- CollectionsExtensions.AsEnum<T>(T)
- DataObjectQueryableExtensions.CompareVariable(Object, FilterOperation, String)
- Processing.Do<T>(T, Action<IProcess<T>>[])
- TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>)
- TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>, TResult)
- TypeExtensions.ChangeType<T>(Object)
- TypeExtensions.ChangeType(Object, Type)
- TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)
- TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)
- TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)
- TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)
- SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)
- QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Class QueryResult

SQL query result

Inheritance

System.Object

[QueryResultBase](#)

QueryResult

Implements

System.Collections.Generic.IEnumerable<[QueryParam](#)>

System.Collections.IEnumerable

Inherited Members

[QueryResultBase.AffectedRows](#)

[QueryResultBase.Item\[Int32\]](#)

[QueryResultBase.Item\[String\]](#)

[QueryResultBase.ParamsCount](#)

[QueryResultBase.Init\(Int32, IDataParameterCollection\)](#)

[QueryResultBase.Init\(Int32, IEnumerable<IDataParameter>\)](#)

[QueryResultBase.Init\(Int32, IEnumerable<QueryParam>\)](#)

[QueryResultBase.GetEnumerator\(\)](#)

[QueryResultBase.IEnumerable.GetEnumerator\(\)](#)

System.Object.Equals(System.Object)

System.Object.Equals(System.Object, System.Object)

System.Object.GetHashCode()

System.Object.GetType()

System.Object.MemberwiseClone()

System.Object.ReferenceEquals(System.Object, System.Object)

System.Object.ToString()

Namespace: [XData.Database.Adapter](#)

Assembly: XData.docfx.dll

Syntax

```
public sealed class QueryResult : QueryResultBase, IEnumerable<QueryParam>, IEnumerable
```

Constructors

QueryResult(Query, Int32, IEnumerable<QueryParam>)

SQL query result constructor

Declaration

```
public QueryResult(Query query, int affectedRows, IEnumerable<QueryParam> queryParams)
```

Parameters

TYPE	NAME	DESCRIPTION
Query	query	Query instance
System.Int32	affectedRows	Affected rows

TYPE	NAME	DESCRIPTION
System.Collections.Generic.IEnumerable< QueryParam >	queryParams	Query parameter sequence

QueryResult(Query, Int32, IDataParameterCollection)

SQL query result constructor

Declaration

```
public QueryResult(Query query, int affectedRows, IDataParameterCollection queryParams)
```

Parameters

TYPE	NAME	DESCRIPTION
Query	query	Query instance
System.Int32	affectedRows	Affected rows
System.Data.IDataParameterCollection	queryParams	Query parameters

Methods

ParamInit(IDataParameter)

Update out parameter value

Declaration

```
protected override QueryParam ParamInit(IDataParameter parameter)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Data.IDataParameter	parameter	Parameter

Returns

TYPE	DESCRIPTION
QueryParam	Query parameter

Overrides

[QueryResultBase.ParamInit\(IDataParameter\)](#)

Exceptions

TYPE	CONDITION
XDataRuntimeException	

Implements

[System.Collections.Generic.IEnumerable<T>](#)

[System.Collections.IEnumerable](#)

Extension Methods

[SerializationExtensions.ToXml<T>\(T\)](#)

[CollectionsExtensions.ToSortedSet<T>\(IEnumerable<T>, IComparer<T>\)](#)

[CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)

[CollectionsExtensions.AsEnum<T>\(T\)](#)

[DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)

[JoinedEnumerable.Inner<TElement>\(IEnumerable<TElement>\)](#)

[JoinedEnumerable.Outer<TElement>\(IEnumerable<TElement>\)](#)

[JoinedEnumerable.LeftOuterJoin<TOuter, TInner, TKey, TResult>\(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter, TKey>, Func<TInner, TKey>, Func<TOuter, TInner, TResult>, IEqualityComparer<TKey>\)](#)

[JoinedEnumerable.RightOuterJoin<TOuter, TInner, TKey, TResult>\(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter, TKey>, Func<TInner, TKey>, Func<TOuter, TInner, TResult>, IEqualityComparer<TKey>\)](#)

[JoinedEnumerable.FullOuterJoin<TOuter, TInner, TKey, TResult>\(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter, TKey>, Func<TInner, TKey>, Func<TOuter, TInner, TResult>, IEqualityComparer<TKey>\)](#)

[Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)

[TypeExtensions.ChangeType<T>\(Object\)](#)

[TypeExtensions.ChangeType\(Object, Type\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)

[SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)

[QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

[QueryDescriptionExtensions.ToDataSet<T>\(IEnumerable<T>, String, ISqlBuilder\)](#)

Class QueryResultBase

SQL query result

Inheritance

System.Object
QueryResultBase
[CallResult](#)
[QueryResult](#)

Implements

System.Collections.Generic.IEnumerable<[QueryParam](#)>
System.Collections.IEnumerable

Inherited Members

System.Object.Equals(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.GetHashCode()
System.Object.GetType()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.ToString()

Namespace: [XData.Database.Adapter](#)

Assembly: XData.docfx.dll

Syntax

```
public abstract class QueryResultBase : IEnumerable<QueryParam>, IEnumerable
```

Constructors

QueryResultBase()

Query result constructor

Declaration

```
protected QueryResultBase()
```

QueryResultBase(Int32, IEnumerable<QueryParam>)

Query result constructor

Declaration

```
protected QueryResultBase(int affectedRows, IEnumerable<QueryParam> queryParams)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Int32	affectedRows	Affected rows count
System.Collections.Generic.IEnumerable< QueryParam >	queryParams	Output parameters

Properties

AffectedRows

Count of rows affected by query

Declaration

```
public int AffectedRows { get; }
```

Property Value

TYPE	DESCRIPTION
System.Int32	

Item[Int32]

Query parameters by index

Declaration

```
public QueryParam this[int i] { get; }
```

Parameters

TYPE	NAME	DESCRIPTION
System.Int32	i	Parameter index

Property Value

TYPE	DESCRIPTION
QueryParam	Query parameter

Item[String]

Query parameters by name

Declaration

```
public QueryParam this[string name] { get; }
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Parameter name

Property Value

TYPE	DESCRIPTION
QueryParam	Query parameter

ParamsCount

Query parameters count

Declaration

```
public int ParamsCount { get; }
```

Property Value

TYPE	DESCRIPTION
System.Int32	

Methods

GetEnumerator()

Query parameters enumeration

Declaration

```
public IEnumerable<QueryParam> GetEnumerator()
```

Returns

TYPE	DESCRIPTION
System.Collections.Generic.IEnumerable< QueryParam >	Query parameters enumeration

Init(Int32, IEnumerable<IDataParameter>)

Query result initialization

Declaration

```
protected void Init(int affectedRows, IEnumerable<IDataParameter> queryParams)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Int32	affectedRows	Affected rows count
System.Collections.Generic.IEnumerable<System.Data.IDataParameter>	queryParams	Output parameters

Init(Int32, IEnumerable<QueryParam>)

Query result initialization

Declaration

```
protected void Init(int affectedRows, IEnumerable<QueryParam> queryParams)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Int32	affectedRows	Affected rows count
System.Collections.Generic.IEnumerable< QueryParam >	queryParams	Output parameters

Init(Int32, IDataParameterCollection)

Query result initialization

Declaration

```
protected void Init(int affectedRows, IDataParameterCollection queryParams)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Int32	affectedRows	Affected rows count
System.Data.IDataParameterCollection	queryParams	Output parameters

ParamInit(IDataParameter)

Output parameter initialization

Declaration

```
protected abstract QueryParam ParamInit(IDataParameter parameter)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Data.IDataParameter	parameter	Output parameter

Returns

TYPE	DESCRIPTION
QueryParam	

Explicit Interface Implementations

IEnumerable.GetEnumerator()

Query parameters enumeration

Declaration

```
IEnumerator IEnumerable.GetEnumerator()
```

Returns

TYPE	DESCRIPTION
System.Collections.IEnumerator	Query parameters enumeration

Implements

System.Collections.Generic.IEnumerable<T>

System.Collections.IEnumerable

Extension Methods

SerializationExtensions.ToXml<T>(T)

CollectionsExtensions.ToSortedSet<T>(IEnumerable<T>, IComparer<T>)

CollectionsExtensions.SetValue<TKey, T>(TKey, T)

CollectionsExtensions.AsEnum<T>(T)

DataObjectQueryableExtensions.CompareVariable(Object, FilterOperation, String)

JoinedEnumerable.Inner<TElement>(IEnumerable<TElement>)

JoinedEnumerable.Outer<TElement>(IEnumerable<TElement>)

JoinedEnumerable.LeftOuterJoin<TOuter, TInner, TKey, TResult>(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter, TKey>, Func<TInner, TKey>, Func<TOuter, TInner, TResult>, IEqualityComparer<TKey>)

JoinedEnumerable.RightOuterJoin<TOuter, TInner, TKey, TResult>(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter, TKey>, Func<TInner, TKey>, Func<TOuter, TInner, TResult>, IEqualityComparer<TKey>)

JoinedEnumerable.FullOuterJoin<TOuter, TInner, TKey, TResult>(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter, TKey>, Func<TInner, TKey>, Func<TOuter, TInner, TResult>, IEqualityComparer<TKey>)

Processing.Do<T>(T, Action<IProcess<T>>[])

TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>)

TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>, TResult)

TypeExtensions.ChangeType<T>(Object)

TypeExtensions.ChangeType(Object, Type)

TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)

TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)

SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)

QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

QueryDescriptionExtensions.ToDataSet<T>(IEnumerable<T>, String, ISqlBuilder)

Class StoredProcedure

SQL stored procedure

Inheritance

System.Object
StoredProcedure

Implements

System.Collections.Generic.IEnumerable<[QueryParam](#)>
System.Collections.IEnumerable

Inherited Members

System.Object.Equals(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.GetHashCode()
System.Object.GetType()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)

Namespace: [XData.Database.Adapter](#)

Assembly: XData.docfx.dll

Syntax

```
public sealed class StoredProcedure : IEnumerable<QueryParam>, IEnumerable
```

Constructors

StoredProcedure(String, QueryParam[])

Procedure constructor

Declaration

```
public StoredProcedure(string name, params QueryParam[] par)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Procedure name
QueryParam []	par	Parameters

Properties

Hint

Procedure hint

Declaration

```
public HintInfo Hint { get; set; }
```

Property Value

TYPE	DESCRIPTION
HintInfo	

Item[Int32]

Query parameters by index

Declaration

```
public QueryParam this[int i] { get; }
```

Parameters

TYPE	NAME	DESCRIPTION
System.Int32	i	Parameter index

Property Value

TYPE	DESCRIPTION
QueryParam	Query parameter

Item[String]

Procedure parameters by name

Declaration

```
public QueryParam this[string name] { get; set; }
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Parameter name

Property Value

TYPE	DESCRIPTION
QueryParam	Procedure parameter

Name

SQL stored procedure name

Declaration

```
public string Name { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

ResultSets

Stored procedure result sets names

Declaration

```
public IList<string> ResultSets { get; }
```

Property Value

TYPE	DESCRIPTION
System.Collections.Generic.IList<System.String>	

Methods

GetEnumerator()

Query parameters enumeration

Declaration

```
public IEnumerator<QueryParam> GetEnumerator()
```

Returns

TYPE	DESCRIPTION
System.Collections.Generic.IEnumerator<QueryParam>	Query parameters enumeration

ToString()

Returns a System.String that represents the current System.Object.

Declaration

```
public override string ToString()
```

Returns

TYPE	DESCRIPTION
System.String	A System.String that represents the current System.Object.

Overrides

System.Object.ToString()

Operators

Implicit(String to StoredProcedure)

Stored procedure name

Declaration

```
public static implicit operator StoredProcedure(string s)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	s	Procedure name

Returns

TYPE	DESCRIPTION
StoredProcedure	Stored procedure

Explicit Interface Implementations

[IEnumerable.GetEnumerator\(\)](#)

[Query parameters enumeration](#)

Declaration

```
IEnumerator IEnumerable.GetEnumerator()
```

Returns

TYPE	DESCRIPTION
System.Collections.IEnumerator	Query parameters enumeration

Implements

[System.Collections.Generic.IEnumerable<T>](#)

[System.Collections.IEnumerable](#)

Extension Methods

[SerializationExtensions.ToXml<T>\(T\)](#)

[CollectionsExtensions.ToSortedSet<T>\(IEnumerable<T>, IComparer<T>\)](#)

[CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)

[CollectionsExtensions.AsEnum<T>\(T\)](#)

[DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)

[JoinedEnumerable.Inner<TElement>\(IEnumerable<TElement>\)](#)

[JoinedEnumerable.Outer<TElement>\(IEnumerable<TElement>\)](#)

[JoinedEnumerable.LeftOuterJoin<TOuter, TInner, TKey, TResult>\(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter, TKey>, Func<TInner, TKey>, Func<TOuter, TInner, TResult>, IEqualityComparer<TKey>\)](#)

[JoinedEnumerable.RightOuterJoin<TOuter, TInner, TKey, TResult>\(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter, TKey>, Func<TInner, TKey>, Func<TOuter, TInner, TResult>, IEqualityComparer<TKey>\)](#)

[JoinedEnumerable.FullOuterJoin<TOuter, TInner, TKey, TResult>\(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter, TKey>, Func<TInner, TKey>, Func<TOuter, TInner, TResult>, IEqualityComparer<TKey>\)](#)

[Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)

[TypeExtensions.ChangeType<T>\(Object\)](#)

TypeExtensions.ChangeType(Object, Type)

TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)

TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)

SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)

QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

QueryDescriptionExtensions.ToDataSet<T>(IEnumerable<T>, String, ISqlBuilder)

Class UdtSqlType

User defined type (UDT) description

Inheritance

System.Object
UdtSqlType

Inherited Members

System.Object.Equals(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.GetHashCode()
System.Object.GetType()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.ToString()

Namespace: [XData.Database.Adapter](#)
Assembly: XData.docfx.dll

Syntax

```
public sealed class UdtSqlType
```

Properties

ElementSqlTypeName

User defined type (UDT) element type name for parameter

Declaration

```
public string ElementSqlTypeName { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

IsArray

User defined type (UDT) collection is VARARRAY for parameter

Declaration

```
public bool IsArray { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

MappedType

User defined type (UDT) mapped type

Declaration

```
public Type MappedType { get; set; }
```


Property Value

TYPE	DESCRIPTION
System.Type	

SqlTypeName

User defined type (UDT) name for parameter

Declaration

```
public string SqlTypeName { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Extension Methods

[SerializationExtensions.ToXml<T>\(T\)](#)

[CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)

[CollectionsExtensions.AsEnum<T>\(T\)](#)

[DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)

[Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)

[TypeExtensions.ChangeType<T>\(Object\)](#)

[TypeExtensions.ChangeType\(Object, Type\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)

[SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)

[QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Namespace XData.Database.Dialect

Classes

[BaseSqlBuilder](#)

SQL builder base class

[ExpressionContext](#)

SQL expression context

[MergeMap](#)

[XConnectionInfo](#)

Context connection info

Enums

[MergeFlag](#)

[SqlStringType](#)

SQL dialect specific format strings types

[SqlTypeSize](#)

Sql type size and scale applicable

Class BaseSqlBuilder

SQL builder base class

Inheritance

System.Object
BaseSqlBuilder

Implements

[ISqlBuilder](#)
System.Collections.Generic.IEnumerable<[Query](#)>
System.Collections.IEnumerable

Inherited Members

System.Object.Equals(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.GetHashCode()
System.Object.GetType()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.ToString()

Namespace: [XData.Database.Dialect](#)

Assembly: XData.docfx.dll

Syntax

```
[DatabaseAdapter("NONE")]  
public abstract class BaseSqlBuilder : ISqlBuilder, ISqlBuilderInternal, IEnumerable<Query>, IEnumerable
```

Fields

SqlStrings

SQL dialect specific format strings
WARNING! Mast be filled in constructor!

Declaration

```
protected SortedList<SqlStringType, string> SqlStrings
```

Field Value

TYPE	DESCRIPTION
System.Collections.Generic.SortedList< SqlStringType , System.String>	

Properties

Config

SQL dialect configuration

Declaration

```
protected IConfiguration Config { get; }
```

Property Value

TYPE	DESCRIPTION
Microsoft.Extensions.Configuration.IConfiguration	

Context

Context name

Declaration

```
protected string Context { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

DialectOptions

Dialect options

WARNING! Mast to be filled in constructor!

Declaration

```
public virtual DialectOptions DialectOptions { get; }
```

Property Value

TYPE	DESCRIPTION
DialectOptions	

Item[Enum]

Predefined queries enumerator

Declaration

```
public Query this[Enum i] { get; }
```

Parameters

TYPE	NAME	DESCRIPTION
System.Enum	i	Predefined query enumeration key

Property Value

TYPE	DESCRIPTION
Query	Query

Exceptions

TYPE	CONDITION
XDataAdapterException	

ParameterPrefix

Parameter prefix symbol

Declaration

```
public abstract string ParameterPrefix { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Queries

List of predefined queries

WARNING! Mast to be filled in constructor!

Declaration

```
protected List<Query> Queries { set; }
```

Property Value

TYPE	DESCRIPTION
System.Collections.Generic.List< Query >	

SqlBlockVariablePrefix

Declaration

```
protected virtual string SqlBlockVariablePrefix { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

SqlTypes

List of supported SQL types

WARNING! Mast to be filled in constructor!

Declaration

```
public DbType[] SqlTypes { get; protected set; }
```

Property Value

TYPE	DESCRIPTION
System.Data.DbType[]	

SqlTypeSizes

List of supported type size applicable

WARNING! Mast to be filled in constructor!

Declaration

```
public Dictionary<string, SqlTypeSize> SqlTypeSizes { get; protected set; }
```

Property Value

TYPE	DESCRIPTION
System.Collections.Generic.Dictionary<System.String, SqlTypeSize >	

TypeNames

List of supported column type names

WARNING! Mast to be filled in constructor!

Declaration

```
public string[] TypeNames { get; protected set; }
```

Property Value

TYPE	DESCRIPTION
System.String[]	

TypeSynonyms

List of supported type synonyms

WARNING! Mast to be filled in constructor!

Declaration

```
public string[] TypeSynonyms { get; protected set; }
```

Property Value

TYPE	DESCRIPTION
System.String[]	

Methods

AddOrderBy(IDataStructure, SortedList<String, Object>, ref StringBuilder, ref Query, ref List<Tuple<String, Type, String>>, ICollection<String>, ref SortedList<String, Tuple<String, String>>, ref Dictionary<String, Object>, Boolean)

Add order by statement to query

Declaration

```
protected virtual void AddOrderBy(IDataStructure structure, SortedList<string, object> filterValues, ref StringBuilder strCommand, ref Query query, ref List<Tuple<string, Type, string>> withSources, ICollection<string> skipped, ref SortedList<string, Tuple<string, string>> fields, ref Dictionary<string, object> paramValues, bool doNotSkip)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataStructure	structure	Structure metadata
System.Collections.Generic.SortedList<System.String, System.Object>	filterValues	Filter values
System.Text.StringBuilder	strCommand	Query string builder
Query	query	Query
System.Collections.Generic.List< Tuple <System.String, System.Type, System.String>>	withSources	Common-Table-Expression (WITH) data sources
System.Collections.Generic.ICollection<System.String>	skipped	Skipped sources collection
System.Collections.Generic.SortedList<System.String, Tuple <System.String, System.String>>	fields	Parsed fields
System.Collections.Generic.Dictionary<System.String, System.Object>	paramValues	Query parameter values
System.Boolean	doNotSkip	Don't skip empty sources

AddReturning2Command(ref StringBuilder, String, SortedList<Tuple<String, String>, Tuple<String, Boolean, Boolean>>, Boolean, Boolean)

Add returning auto increment identity column values statement in query

Declaration

```
protected abstract void AddReturning2Command(ref StringBuilder sb, string alias, SortedList<Tuple<string, string>, Tuple<string, bool, bool>> identity, bool defaultValues = false, bool close = false)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Text.StringBuilder	sb	Query string builder
System.String	alias	Table alias
System.Collections.Generic.SortedList< Tuple <System.String, System.String>, Tuple <System.String, System.Boolean, System.Boolean>>	identity	Identity column name and parameter name

TYPE	NAME	DESCRIPTION
System.Boolean	defaultValues	Insert default values flag
System.Boolean	close	Close values parenthesis flag

AddSkipLimitAndOrder(IDataStructure, SortedList<String, Object>, ref StringBuilder, Int32, String, ref Query, ref List<Tuple<String, Type, String>>, ICollection<String>, ref SortedList<String, Tuple<String, String>>, ref Dictionary<String, Object>, Boolean)

Add skip, limit, and order statements to query

Declaration

```
protected abstract void AddSkipLimitAndOrder(IDataStructure structure, SortedList<string, object>
filterValues, ref StringBuilder strCommand, int fromPosition, string selectAll, ref Query query, ref
List<Tuple<string, Type, string>> withSources, ICollection<string> skipped, ref SortedList<string,
Tuple<string, string>> fields, ref Dictionary<string, object> paramValues, bool doNotSkip = false)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataStructure	structure	Structure metadata
System.Collections.Generic.SortedList<System.String, System.Object>	filterValues	Filter values
System.Text.StringBuilder	strCommand	Query string builder
System.Int32	fromPosition	From statement start position
System.String	selectAll	All fields list string
Query	query	Query
System.Collections.Generic.List< Tuple <System.String, System.Type, System.String>>	withSources	Common-Table-Expression (WITH) data sources
System.Collections.Generic.ICollection<System.String>	skipped	Skipped sources collection
System.Collections.Generic.SortedList<System.String, Tuple <System.String, System.String>>	fields	Parsed fields

TYPE	NAME	DESCRIPTION
System.Collections.Generic.Dictionary<System.String, System.Object>	paramValues	Query parameter values
System.Boolean	doNotSkip	Don't skip empty sources

Beautify(String)

Returns formatted SQL string

Declaration

```
public static string Beautify(string sql)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	sql	SQL string to format

Returns

TYPE	DESCRIPTION
System.String	Formatted SQL string

BeautifyRtf(String)

Returns RTF formatted SQL string

Declaration

```
public static string BeautifyRtf(string sql)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	sql	SQL string to format

Returns

TYPE	DESCRIPTION
System.String	RTF formatted SQL string

BuildSqlBlock(Expression<SqlBlockDefinition>, Expression<Func<ISqlParamAdapter, ISqlParam>>[])

Declaration

```
public Query BuildSqlBlock(Expression<SqlBlockDefinition> sqlBlock, Expression<Func<ISqlParamAdapter, ISqlParam>>[] parameters)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<SqlBlockDefinition>	sqlBlock	
System.Linq.Expressions.Expression<System.Func<ISqlParamAdapter, ISqlParam>>[]	parameters	

Returns

TYPE	DESCRIPTION
Query	

BuildSqlBlock<T>(Expression<SqlBlockDefinition<T>>, Expression<Func<ISqlParamAdapter, ISqlParam>>[], out Func<DataRow, T>)

Declaration

```
public Query BuildSqlBlock<T>(Expression<SqlBlockDefinition<T>> sqlBlock, Expression<Func<ISqlParamAdapter, ISqlParam>>[] parameters, out Func<DataRow, T> mapper)
    where T : class, new()
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<SqlBlockDefinition<T>>	sqlBlock	
System.Linq.Expressions.Expression<System.Func<ISqlParamAdapter, ISqlParam>>[]	parameters	
System.Func<System.Data.DataRow, T>	mapper	

Returns

TYPE	DESCRIPTION
Query	

Type Parameters

NAME	DESCRIPTION
T	

CompleteInsert(String, SortedList<Tuple<String, String>, Tuple<String, Boolean, Boolean>>, Type)

Returns string contains SQL expression to complete insert statement

Declaration

```
protected virtual string CompleteInsert(string alias, SortedList<Tuple<string, string>, Tuple<string, bool, bool>> identity, Type objectType)
```

Parameters

TYPE	NAME	DESCRIPTION

TYPE	NAME	DESCRIPTION
System.String	alias	Base table alias
System.Collections.Generic.SortedList< Tuple <System.String, System.String>, Tuple <System.String, System.Boolean, System.Boolean> >	identity	Identity fields and query parameters
System.Type	objectType	Mapped object type

Returns

TYPE	DESCRIPTION
System.String	SQL expression to complete insert statement

ConvertTextExpr(IField, String)

Convert expression text for field native data type

Declaration

```
protected virtual string ConvertTextExpr(IField field, string exprText)
```

Parameters

TYPE	NAME	DESCRIPTION
IField	field	Field
System.String	exprText	Expression text

Returns

TYPE	DESCRIPTION
System.String	Expression text

ExtractXmlQuery(String, String, String[], IComparable[], String, KeyValuePair<String, String>[])

Returns query to extract xml data by XPath expression

Declaration

```
public Query ExtractXmlQuery(string table, string column, string[] tablePrimaryKey, IComparable[] primaryKeyValue, string path, params KeyValuePair<string, string>[] namespaces)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	table	Table name
System.String	column	Xml column name
System.String[]	tablePrimaryKey	Table primary key columns
IComparable[]	primaryKeyValue	Table primary key values
System.String	path	XPath expression
System.Collections.Generic.KeyValuePair<System.String, System.String> []	namespaces	XPath namespaces

Returns

TYPE	DESCRIPTION
Query	XPath result

Exceptions

TYPE	CONDITION
XDataQueryStructureException	Field not found

FormatConstValue(Object, out DbType)

Declaration

```
protected string FormatConstValue(object constVal, out DbType sqlType)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Object	constVal	
System.Data.DbType	sqlType	

Returns

TYPE	DESCRIPTION
System.String	

FormatFieldName(String, String)

Format field name

Declaration

```
public static string FormatFieldName(string alias, string fieldName)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	alias	Source alias
System.String	fieldName	Field name

Returns

TYPE	DESCRIPTION
System.String	Formatted field name

FormatQuery(Query, Object[])

Returns formatted query string with parameter values

Declaration

```
public static string FormatQuery(Query query, params object[] paramValues)
```

Parameters

TYPE	NAME	DESCRIPTION
Query	query	Query
System.Object[]	paramValues	Parameter values

Returns

TYPE	DESCRIPTION
System.String	Formatted SQL string

FormatSql(SqlStringType, Object[])

Format SQL dialect specific syntax strings

Hint: Override to realize ugly non standard syntax (if needed)

Declaration

```
protected virtual string FormatSql(SqlStringType sqlStringType, params object[] args)
```

Parameters

TYPE	NAME	DESCRIPTION
SqlStringType	sqlStringType	SQL syntax key
System.Object[]	args	Syntax arguments

Returns

TYPE	DESCRIPTION
System.String	

FormatStoredProcedure(StoredProcedure, Object[])

Returns formatted query string with parameter values

Declaration

```
public static string FormatStoredProcedure(StoredProcedure storedProcedure, params object[] paramValues)
```

Parameters

TYPE	NAME	DESCRIPTION
StoredProcedure	storedProcedure	Stored procedure
System.Object[]	paramValues	Parameter values

Returns

TYPE	DESCRIPTION
System.String	Formatted SQL string

GetAcquireBlobQuery(String, String, IEnumerable<String>, IComparable[], String)

Returns query to acquire blob data

Declaration

```
public Query GetAcquireBlobQuery(string table, string column, IEnumerable<string> tablePrimaryKey, IComparable[] primaryKeyValue, string nativeSqlType = null)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	table	Table name

TYPE	NAME	DESCRIPTION
System.String	column	Blob column name
System.Collections.Generic.IEnumerable<System.String>	tablePrimaryKey	Table primary key columns
IComparable[]	primaryKeyValue	Table primary key values
System.String	nativeSqlType	Native SQL type name

Returns

TYPE	DESCRIPTION
Query	Query

GetAcquireXmlQuery(String, String, String[], IComparable[])

Returns query to acquire xml data

Declaration

```
public Query GetAcquireXmlQuery(string table, string column, string[] tablePrimaryKey, IComparable[]
primaryKeyValue)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	table	Table name
System.String	column	Xml column name
System.String[]	tablePrimaryKey	Table primary key columns
IComparable[]	primaryKeyValue	Table primary key values

Returns

TYPE	DESCRIPTION
Query	Query

Exceptions

TYPE	CONDITION
XDataQueryStructureException	Field not found

GetAllQuery(IDataStructure)

Returns query to check compliance of all repository objects
Hint: Used in XData LINQ expressions

Declaration

```
public Query GetAllQuery(IDataStructure structure)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataStructure	structure	Structure metadata

Returns

TYPE	DESCRIPTION
Query	Query

GetAnyQuery(IDataStructure)

Returns query to check existence of repository objects
Hint: Used in XData LINQ expressions

Declaration

```
public Query GetAnyQuery(IDataStructure structure)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataStructure	structure	Structure metadata

Returns

TYPE	DESCRIPTION
Query	Query

GetBlobSizeQuery(String, String, String[], IComparable[])

Returns query to get Large Object size

Declaration


```
public Query GetBlobSizeQuery(string table, string column, string[] tablePrimaryKey, IComparable[]
primaryKeyValue)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	table	Table name contained LOB
System.String	column	LOB column name
System.String[]	tablePrimaryKey	Primary key column names
IComparable[]	primaryKeyValue	Primary key values

Returns

TYPE	DESCRIPTION
Query	Query to get Large Object size

GetCall(IDataStructure, String)

Declaration

```
public StoredProcedure GetCall(IDataStructure structure, string userName)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataStructure	structure	
System.String	userName	

Returns

TYPE	DESCRIPTION
StoredProcedure	

Exceptions

TYPE	CONDITION
XDataQueryStructureException	
XDataAdapterException	stored procedure

GetClearQuery(IDataStructure)

Returns query to clear all repository objects

Declaration

```
public Query GetClearQuery(IDataStructure structure)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataStructure	structure	Structure metadata

Returns

TYPE	DESCRIPTION
Query	Query

Exceptions

TYPE	CONDITION
XDataQueryStructureException	

GetConnectionInfo(String, Dictionary<CredentialsType, String>)

Declaration

```
public XConnectionInfo GetConnectionInfo(string connectionString, Dictionary<CredentialsType, string> credentials = null)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	connectionString	
System.Collections.Generic.Dictionary< CredentialsType , System.String>	credentials	

Returns

TYPE	DESCRIPTION
XConnectionInfo	

GetConnectionString(String, String, Boolean, String, String, String)

Compose specific connection string from parts

Declaration

```
protected abstract string GetConnectionString(string server, string schema, bool integratedSecurity, string user, string password, string partialConnectionString)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	server	Server name
System.String	schema	Schema name
System.Boolean	integratedSecurity	Integrated security flag
System.String	user	User login
System.String	password	User password
System.String	partialConnectionString	Additional connection string settings

Returns

TYPE	DESCRIPTION
System.String	Connection string

GetCountQuery(IDataStructure)

Returns query to select count of data objects in repository

Declaration

```
public Query GetCountQuery(IDataStructure structure)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataStructure	structure	Structure metadata

Returns

TYPE	DESCRIPTION
Query	Query

GetDeleteQuery(IDataStructure, IDataObject, String, Boolean)

Returns query to delete data object

Declaration

```
public Query GetDeleteQuery(IDataStructure structure, IDataObject data, string userName, bool ignoreConcurrency = true)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataStructure	structure	Structure metadata
IDataObject	data	Data object instance
System.String	userName	User name
System.Boolean	ignoreConcurrency	Ignore concurrency flag

Returns

TYPE	DESCRIPTION
Query	Query

Exceptions

TYPE	CONDITION
XDataQueryStructureException	

GetEnumerator()

Predefined queries enumerator

Declaration

```
public IEnumerable<Query> GetEnumerator()
```

Returns

TYPE	DESCRIPTION
System.Collections.Generic.IEnumerable< Query >	Predefined queries enumerator

GetExpressionText(IDataStructure, ExpressionContext, Type, Expression, out DbType)

Compose SQL expression from XData LINQ expression

Declaration

```
public string GetExpressionText(IDataStructure structure, ExpressionContext context, Type queryType, Expression expression, out DbType sqlType)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataStructure	structure	Structure metadata
ExpressionContext	context	Expression context
System.Type	queryType	Data query type
System.Linq.Expressions.Expression	expression	XData LINQ expression
System.Data.DbType	sqlType	Expression sql type

Returns

TYPE	DESCRIPTION
System.String	SQL expression

Exceptions

TYPE	CONDITION
XDataQueryStructureException	LINQ expression has unsupported type
XDataQueryStructureException	Property not found
XDataQueryStructureException	LINQ expression has unsupported method parameters
XDataAdapterException	

GetExprSize(IField)

Declaration

```
protected string GetExprSize(IField field)
```

Parameters

TYPE	NAME	DESCRIPTION
IField	field	

Returns

TYPE	DESCRIPTION
System.String	

GetInsertQuery(IDataStructure, IDataObject, String, Boolean)

Returns query to insert data object

Declaration

```
public Query GetInsertQuery(IDataStructure structure, IDataObject data, string userName, bool ignoreConcurrency = true)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataStructure	structure	Structure metadata
IDataObject	data	Data object instance
System.String	userName	User name
System.Boolean	ignoreConcurrency	Ignore concurrency flag

Returns

TYPE	DESCRIPTION
Query	Query

Exceptions

TYPE	CONDITION
XDataQueryStructureException	

GetLockQuery(IDataStructure)

Returns query to lock single data object

Declaration

```
public Query GetLockQuery(IDataStructure structure)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataStructure	structure	Structure metadata

Returns

TYPE	DESCRIPTION
Query	Query

GetOrderByString(IDataStructure, SortedList<String, Object>, ref Query, ref List<Tuple<String, Type, String>>, ICollection<String>, ref SortedList<String, Tuple<String, String>>, ref Dictionary<String, Object>, Boolean, Boolean)

Returns order by string

Declaration

```
protected string GetOrderByString(IDataStructure structure, SortedList<string, object> filterValues, ref Query query, ref List<Tuple<string, Type, string>> withSources, ICollection<string> skipped, ref SortedList<string, Tuple<string, string>> fields, ref Dictionary<string, object> paramValues, bool doNotSkip, bool useGrouping = false)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataStructure	structure	Structure metadata
System.Collections.Generic.SortedList<System.String, System.Object>	filterValues	Filter values
Query	query	Query
System.Collections.Generic.List<Tuple<System.String, System.Type, System.String> >	withSources	Common-Table-Expression (WITH) data sources
System.Collections.Generic.ICollection<System.String>	skipped	Skipped sources collection
System.Collections.Generic.SortedList<System.String, Tuple<System.String, System.String> >	fields	Parsed fields
System.Collections.Generic.Dictionary<System.String, System.Object>	paramValues	Query parameter values
System.Boolean	doNotSkip	Don't skip empty sources
System.Boolean	useGrouping	Used in GROUP BY generation

Returns

TYPE	DESCRIPTION
System.String	Order by string

Exceptions

TYPE	CONDITION
XDataQueryStructureException	Field not found

GetParameterName(String)

Formats field name as parameter name

Declaration

```
public virtual string GetParameterName(string fieldName)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	fieldName	Field name

Returns

TYPE	DESCRIPTION
System.String	Parameter name

GetRefreshQuery(IDataStructure)

Returns query to refresh single data object

Declaration

```
public Query GetRefreshQuery(IDataStructure structure)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataStructure	structure	Structure metadata

Returns

TYPE	DESCRIPTION
Query	Query

GetSafeConvertExpression(Type, Expression, Expression)

Safe convert expression

Declaration

```
public virtual Expression GetSafeConvertExpression(Type type, Expression dataRow, Expression fieldName)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Type	type	Type of expression
System.Linq.Expressions.Expression	dataRow	Data row expression
System.Linq.Expressions.Expression	fieldName	Field name expression

Returns

TYPE	DESCRIPTION
System.Linq.Expressions.Expression	Expression

GetSaveBlobQuery(String, String, IEnumerable<String>, IComparable[], ref Byte[], String)

Returns query to save blob data

Declaration

```
public Query GetSaveBlobQuery(string table, string column, IEnumerable<string> tablePrimaryKey, IComparable[] primaryKeyValue, ref byte[] body, string nativeSqlType = null)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	table	Table name
System.String	column	Blob column name
System.Collections.Generic.IEnumerable<System.String>	tablePrimaryKey	Table primary key columns
IComparable[]	primaryKeyValue	Table primary key values
Byte[]	body	Blob body

TYPE	NAME	DESCRIPTION
System.String	nativeSqlType	Native SQL type name

Returns

TYPE	DESCRIPTION
Query	Query

GetSaveXmlQuery(String, String, String[], IComparable[], ref String)

Returns query to save xml data

Declaration

```
public Query GetSaveXmlQuery(string table, string column, string[] tablePrimaryKey, IComparable[]
primaryKeyValue, ref string body)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	table	Table name
System.String	column	Xml column name
System.String[]	tablePrimaryKey	Table primary key columns
IComparable[]	primaryKeyValue	Table primary key values
System.String	body	Blob body

Returns

TYPE	DESCRIPTION
Query	Query

Exceptions

TYPE	CONDITION
XDataQueryStructureException	Field not found

GetSelectQuery(IDataStructure, Boolean, Boolean)

Returns query to select data objects

Declaration

```
public Query GetSelectQuery(IDataStructure structure, bool markProperties = false, bool doNotSkip = false)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataStructure	structure	Structure metadata
System.Boolean	markProperties	Mark properties with comments
System.Boolean	doNotSkip	Don't skip sources

Returns

TYPE	DESCRIPTION
Query	Query

GetSequenceName(String, String, String)

Returns sequence name

Declaration

```
public string GetSequenceName(string context, string table, string column)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	context	Context name
System.String	table	Table name
System.String	column	Column name

Returns

TYPE	DESCRIPTION
System.String	Sequence name

GetSqlTypeName(DbType)

Returns SQL type name for System.DbType

Declaration

```
protected virtual string GetSqlTypeName(DbType dbType)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Data.DbType	dbType	System.DbType enumeration key

Returns

TYPE	DESCRIPTION
System.String	SQL type name

Exceptions

TYPE	CONDITION
XDataAdapterException	Type not supported

GetUpdateQuery(IDataStructure, IDataObject, String, Boolean)

Returns query to update data object

Declaration

```
public Query GetUpdateQuery(IDataStructure structure, IDataObject data, string userName, bool ignoreConcurrency = true)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataStructure	structure	Structure metadata
IDataObject	data	Data object instance
System.String	userName	User name
System.Boolean	ignoreConcurrency	Ignore concurrency flag

Returns

TYPE	DESCRIPTION
Query	Query

Exceptions

TYPE	CONDITION
XDataQueryStructureException	

GetXmlNamespaces(KeyValuePair<String, String>[])

Returns Xml namespaces string

Declaration

<pre>protected abstract string GetXmlNamespaces(KeyValuePair<string, string>[] namespaces)</pre>

Parameters

TYPE	NAME	DESCRIPTION
System.Collections.Generic.KeyValuePair<System.String, System.String>[]	namespaces	Xml namespaces

Returns

TYPE	DESCRIPTION
System.String	Xml namespaces string

Init()

Dialect adapter initialization

Declaration

<pre>protected virtual void Init()</pre>

Init(String, IConfiguration, String)

Dialect adapter initialization

Declaration

<pre>public void Init(string context, IConfiguration config, string sequence)</pre>

Parameters

TYPE	NAME	DESCRIPTION
System.String	context	Context name
Microsoft.Extensions.Configuration.IConfiguration	config	Specific dialect adapter configuration Xml
System.String	sequence	Sequence generator assembly qualified class name

ParseConnectionString(String, ref String, ref String, ref Boolean, ref String, ref String, out String)

Parse specific connection string to parts

Declaration

```
protected abstract void ParseConnectionString(string connectionString, ref string server, ref string schema,
ref bool integratedSecurity, ref string user, ref string password, out string partialConnectionString)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	connectionString	Connection string
System.String	server	Server name
System.String	schema	Schema name
System.Boolean	integratedSecurity	Integrated security flag
System.String	user	User login
System.String	password	User password
System.String	partialConnectionString	Additional connection string settings

PrepareTempTables(IDataStructure, ref StringBuilder, Query, ICollection<String>, Boolean, Boolean)

Declaration

```
protected virtual bool PrepareTempTables(IDataStructure structure, ref StringBuilder strCommand, Query query,
ICollection<string> skippedSources, bool riseErrors, bool doNotSkip)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataStructure	structure	
System.Text.StringBuilder	strCommand	
Query	query	
System.Collections.Generic.ICollection<System.String>	skippedSources	
System.Boolean	riseErrors	
System.Boolean	doNotSkip	

Returns

TYPE	DESCRIPTION
System.Boolean	

RegisterResultSet(IResultSetDescription)

Declaration

<pre>protected virtual QueryParam RegisterResultSet(IResultSetDescription resultSet)</pre>

Parameters

TYPE	NAME	DESCRIPTION
IResultSetDescription	resultSet	

Returns

TYPE	DESCRIPTION
QueryParam	

SequenceGeneration(out Boolean)

Sequence generation supported

Declaration

<pre>public bool SequenceGeneration(out bool required)</pre>

Parameters

TYPE	NAME	DESCRIPTION
System.Boolean	required	

Returns

TYPE	DESCRIPTION
System.Boolean	Sequence generation supported flag

See Also

[GetSequenceName\(String, String, String\)](#)

[SequenceRequired](#)

SetQueryParameters(ref Query, IEnumerable<IParameterDescription>, SortedList<String, Object>, String, DialectOptions)

Declaration

<pre>public void SetQueryParameters(ref Query query, IEnumerable<IParameterDescription> parameters, SortedList<string, object> variableValues, string userName, DialectOptions dialectOptions)</pre>

Parameters

TYPE	NAME	DESCRIPTION
Query	query	
System.Collections.Generic.IEnumerable< IParacterDescription >	parameters	
System.Collections.Generic.SortedList<System.String, System.Object>	variableValues	
System.String	userName	
DialectOptions	dialectOptions	

Exceptions

TYPE	CONDITION
XDataAdapterException	

SetQueryParameterValues(ref Query, SortedList<String, Object>)

Set query parameter values

Declaration

```
public void SetQueryParameterValues(ref Query query, SortedList<string, object> paramValues)
```

Parameters

TYPE	NAME	DESCRIPTION
Query	query	Query
System.Collections.Generic.SortedList<System.String, System.Object>	paramValues	Parameter values

ToDataSet<TSource>(String, IEnumerable<TSource>)

Compose union structure over source object sequence

Declaration

```
public string ToDataSet<TSource>(string prefix, IEnumerable<TSource> source)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	prefix	Column prefix
System.Collections.Generic.IEnumerable<TSource>	source	Source sequence

Returns

TYPE	DESCRIPTION
System.String	Data source

Type Parameters

NAME	DESCRIPTION
TSource	Source type

Explicit Interface Implementations

IEnumerable.GetEnumerator()

Predefined queries enumerator

Declaration

IEnumerator IEnumerable.GetEnumerator()

Returns

TYPE	DESCRIPTION
System.Collections.IEnumerator	Predefined queries enumerator

Implements

[ISqlBuilder](#)

System.Collections.Generic.IEnumerable<T>

System.Collections.IEnumerable

Extension Methods

[SerializationExtensions.ToXml<T>\(T\)](#)

[CollectionsExtensions.ToSortedSet<T>\(IEnumerable<T>, IComparer<T>\)](#)

[CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)

[CollectionsExtensions.AsEnum<T>\(T\)](#)

[DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)

[JoinedEnumerable.Inner<TElement>\(IEnumerable<TElement>\)](#)

[JoinedEnumerable.Outer<TElement>\(IEnumerable<TElement>\)](#)

[JoinedEnumerable.LeftOuterJoin<TOuter, TInner, TKey, TResult>\(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter, TKey>, Func<TInner, TKey>, Func<TOuter, TInner, TResult>, IEqualityComparer<TKey>\)](#)

[JoinedEnumerable.RightOuterJoin<TOuter, TInner, TKey, TResult>\(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter, TKey>, Func<TInner, TKey>, Func<TOuter, TInner, TResult>, IEqualityComparer<TKey>\)](#)

[JoinedEnumerable.FullOuterJoin<TOuter, TInner, TKey, TResult>\(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter, TKey>, Func<TInner, TKey>, Func<TOuter, TInner, TResult>, IEqualityComparer<TKey>\)](#)

[Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)

[TypeExtensions.ChangeType<T>\(Object\)](#)

[TypeExtensions.ChangeType\(Object, Type\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)

TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)

SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)

QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

QueryDescriptionExtensions.ToDataSet<T>(IEnumerable<T>, String, ISqlBuilder)

Class ExpressionContext

SQL expression context

Inheritance

System.Object

ExpressionContext

Inherited Members

System.Object.Equals(System.Object)

System.Object.Equals(System.Object, System.Object)

System.Object.GetHashCode()

System.Object.GetType()

System.Object.MemberwiseClone()

System.Object.ReferenceEquals(System.Object, System.Object)

System.Object.ToString()

Namespace: [XData.Database.Dialect](#)

Assembly: XData.docfx.dll

Syntax

```
public class ExpressionContext
```

Properties

OuterQueries

Subquery outer queries

Declaration

```
public Dictionary<string, IDataQuery> OuterQueries { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Collections.Generic.Dictionary<System.String, IDataQuery >	

Query

Expression context query

Declaration

```
public IDataQuery Query { get; }
```

Property Value

TYPE	DESCRIPTION
IDataQuery	

Extension Methods

[SerializationExtensions.ToXml<T>\(T\)](#)

[CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)

[CollectionsExtensions.AsEnum<T>\(T\)](#)

[DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)

[Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)

TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>)

TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>, TResult)

TypeExtensions.ChangeType<T>(Object)

TypeExtensions.ChangeType(Object, Type)

TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)

TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)

SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)

QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Enum MergeFlag

Namespace: [XData.Database.Dialect](#)

Assembly: XData.docfx.dll

Syntax

```
[Flags]
public enum MergeFlag
```

Fields

NAME	DESCRIPTION
Insert	
Link	
None	
Update	

Extension Methods

- [SerializationExtensions.ToXml<MergeFlag>\(\)](#)
- [CollectionsExtensions.SetValue<MergeFlag, T>\(T\)](#)
- [CollectionsExtensions.AsEnum<MergeFlag>\(\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(FilterOperation, String\)](#)
- [Processing.Do<MergeFlag>\(Action<IProcess<MergeFlag>>\[\]\)](#)
- [TypeExtensions.ChangeType<T>\(\)](#)
- [TypeExtensions.ChangeType\(Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.ConvertTo<T>\(\)](#)
- [TypeExtensions.EqualValue\(Enum\)](#)
- [SqlBlockExtensions.SetExpression<MergeFlag, TResult>\(Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<MergeFlag, TResult>\(Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Class MergeMap

Inheritance

System.Object
MergeMap

Inherited Members

System.Object.Equals(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.GetHashCode()
System.Object.GetType()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.ToString()

Namespace: [XData.Database.Dialect](#)

Assembly: XData.docfx.dll

Syntax

```
public class MergeMap
```

Properties

InsertValue

Declaration

```
public string InsertValue { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

MergeFlag

Declaration

```
public MergeFlag MergeFlag { get; set; }
```

Property Value

TYPE	DESCRIPTION
MergeFlag	

SelectField

Declaration

```
public SelectedField SelectField { get; set; }
```

Property Value

TYPE	DESCRIPTION
SelectedField	

SourceField

Declaration

```
public string SourceField { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

SourceValue

Declaration

```
public string SourceValue { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

TargetField

Declaration

```
public string TargetField { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Extension Methods

- SerializationExtensions.ToXml<T>(T)
- CollectionsExtensions.SetValue<TKey, T>(TKey, T)
- CollectionsExtensions.AsEnum<T>(T)
- DataObjectQueryableExtensions.CompareVariable(Object, FilterOperation, String)
- Processing.Do<T>(T, Action<IProcess<T>>[])
- TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>)
- TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>, TResult)
- TypeExtensions.ChangeType<T>(Object)
- TypeExtensions.ChangeType(Object, Type)
- TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)
- TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)
- TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)
- TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)
- SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)
- QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Enum SqlStringType

SQL dialect specific format strings types

Namespace: [XData.Database.Dialect](#)

Assembly: XData.docfx.dll

Syntax

```
public enum SqlStringType
```

Fields

NAME	DESCRIPTION
AcquireBlob	
AcquireXml	
AfterDelete	
AfterInsert	
AfterUpdate	
All	
Any	
BeforeDelete	
BeforeInsert	
BeforeUpdate	
BooleanConstant	
Ceiling	
CheckConcurrency	
Clear	
ClearComplete	
ClearDeclare	
ClearField	
ClearSaveKeys	
Coalesce	
CompleteInsert	

NAME	DESCRIPTION
Concat	
Conditional	
Convert	
Date	
DateFormat	
DateTimeAdd	
DateTimeAddDays	
DateTimeAddHours	
DateTimeAddMilliseconds	
DateTimeAddMinutes	
DateTimeAddMonths	
DateTimeAddSeconds	
DateTimeAddYears	
DateTimeDaysInMonth	
DateTimeIsLeapYear	
Day	
DayOfWeek	
DayOfYear	
Days	
DaysTotal	
DictionaryTableName	
EmptyBlob	
EmptyGuid	
EmptyXml	
EndsWith	

NAME	DESCRIPTION
Except	
ExtractXml	
ExtractXmlWithNamespaces	
FieldFormat	
Floor	
GetBlobSize	
GetForeignKey	
GroupConvert	
GroupField	
GroupXmlField	
Hour	
Hours	
IndexOf	
InitInsert	
InsertValues	
Intersect	
IsNullOrEmpty	
IsNullOrWhiteSpace	
Length	
Lock	
LockPk	
LockPkJoin	
MaxSize	
Millisecond	
Minute	

NAME	DESCRIPTION
Minutes	
ModifiedHintAfter	
ModifiedHintBefore	
ModifiedTableName	
Month	
Months	
NewGuid	
NoneSelect	
Now	
PadLeft	
PadRight	
Power	
Random	
Remove	
Returning	
Round	
RowId	
RowIdArray	
RowIdFill	
RowIterator	
SaveBlob	
SaveXml	
Second	
Seconds	
SelectField	

NAME	DESCRIPTION
SelectHintBefore	
SelectXmlField	
SequenceNextVal	
SkipInsert	
SqlBlockDeclare	
SqlBlockDefault	
SqlBlockDelete	
SqlBlockExternalLoopVar	
SqlBlockForEach	
SqlBlockIf	
SqlBlockIfExists	
SqlBlockInsert	
SqlBlockMerge	
SqlBlockQuote	
SqlBlockReturn	
SqlBlockSelectInto	
SqlBlockUpdate	
SqlBlockVarAssign	
SqlParameter	
SqlScalarFunction	
SqlTableFunction	
Sqrt	
StartsWith	
Substring	
TableName	

NAME	DESCRIPTION
TempTableCreate	
TempTableDrop	
TimeOfDay	
TimeSpanFormat	
ToBoolean	
ToDateTime	
Today	
ToDecimal	
ToInt16	
ToInt32	
ToInt64	
ToString	
ToStringFormat	
ToXml	
TreeOrder	
Trim	
TrimEnd	
TrimStart	
UtcNow	
With	
WithRecursive	
XmlAsString	
XmlSourceName	
Year	
Years	

Extension Methods

SerializationExtensions.ToXml<SqlStringType>()
CollectionsExtensions.SetValue<SqlStringType, T>(T)
CollectionsExtensions.AsEnum<SqlStringType>()
DataObjectQueryableExtensions.CompareVariable(FilterOperation, String)
Processing.Do<SqlStringType>(Action<IProcess<SqlStringType>> [])
TypeExtensions.ChangeType<T>()
TypeExtensions.ChangeType(Type)
TypeExtensions.InvokeMethod<TRet>(Type[], Expression<Func<TRet>>, BindingFlags)
TypeExtensions.InvokeMethod(Type[], Expression<Action>, BindingFlags)
TypeExtensions.InvokeMethod<TRet>(Expression<Func<TRet>>, BindingFlags)
TypeExtensions.InvokeMethod(Expression<Action>, BindingFlags)
TypeExtensions.ConvertTo<T>()
TypeExtensions.EqualValue(Enum)
SqlBlockExtensions.SetExpression<SqlStringType, TResult>(Expression<Func<IBlockQueryAdapter, TResult>>)
QueryDescriptionExtensions.SetExpression<SqlStringType, TResult>(Expression<Func<IQueryStructureAdapter, TResult>>)

Enum SqlTypeSize

Sql type size and scale applicable

Namespace: [XData.Database.Dialect](#)

Assembly: XData.docfx.dll

Syntax

```
[Flags]
public enum SqlTypeSize
```

Fields

NAME	DESCRIPTION
None	None applicable
Precision	Precision applicable
Scale	Scale applicable
Size	Size applicable

Extension Methods

- [SerializationExtensions.ToXml<SqlTypeSize>\(\)](#)
- [CollectionsExtensions.SetValue<SqlTypeSize, T>\(T\)](#)
- [CollectionsExtensions.AsEnum<SqlTypeSize>\(\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(FilterOperation, String\)](#)
- [Processing.Do<SqlTypeSize>\(Action<IProcess<SqlTypeSize>>\[\]\)](#)
- [TypeExtensions.ChangeType<T>\(\)](#)
- [TypeExtensions.ChangeType\(Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.ConvertTo<T>\(\)](#)
- [TypeExtensions.EqualValue\(Enum\)](#)
- [SqlBlockExtensions.SetExpression<SqlTypeSize, TResult>\(Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<SqlTypeSize, TResult>\(Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Class XConnectionInfo

Context connection info

Inheritance

System.Object
XConnectionInfo

Inherited Members

System.Object.Equals(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.GetHashCode()
System.Object.GetType()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.ToString()

Namespace: [XData.Database.Dialect](#)
Assembly: XData.docfx.dll

Syntax

```
public class XConnectionInfo
```

Properties

DbLogin

Database user login

Declaration

```
public string DbLogin { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

IntegratedSecurity

Integrated security flag

Declaration

```
public bool IntegratedSecurity { get; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

Schema

Database name or schema name

Declaration

```
public string Schema { get; }
```


Property Value

TYPE	DESCRIPTION
System.String	

Server

Database server name

Declaration

```
public string Server { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Extension Methods

- SerializationExtensions.ToXml<T>(T)
- CollectionsExtensions.SetValue<TKey, T>(TKey, T)
- CollectionsExtensions.AsEnum<T>(T)
- DataObjectQueryableExtensions.CompareVariable(Object, FilterOperation, String)
- Processing.Do<T>(T, Action<IProcess<T>>[])
- TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>)
- TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>, TResult)
- TypeExtensions.ChangeType<T>(Object)
- TypeExtensions.ChangeType(Object, Type)
- TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)
- TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)
- TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)
- TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)
- SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)
- QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Namespace XData.Database.SqlBlock

Classes

[SqlBlockExtensions](#)

Sql block extensions

[SqlType](#)

SQL block variable/constant/parameter type description

Interfaces

[IBlockQueryAdapter](#)

SQL block query structure adapter

[IBlockQueryDescription](#)

Base SQL block query description interface

[IBlockQueryDescription<T>](#)

SQL block query description interface

[IBlockQueryStructure](#)

SQL block query structure interface

[IExpressionMergeAdapter](#)

Expression structure adapter for SQL block queries

[ILoopVariableAdapter<T>](#)

Loop variables mapping adapter

[IMergeAttributeAdapter](#)

Merge role structure adapter

[IMergeBlockAdapter](#)

Merge mapping adapter

[IPropertyMergeAdapter](#)

Dynamic structure adapter for SQL block queries

[ISetBlockAdapter](#)

Set/Insert block query adapter

[ISetSqlVarAdapter](#)

SQL variable assignment adapter

[ISqlBlock](#)

SQL block definition interface

[ISqlBlockAdapter](#)

SQL block adapter

[ISqlBlockAdapter<T>](#)

Adapter of SQL block returning data

[ISqlParam](#)

SQL block parameter interface

[ISqlParamAdapter](#)

Adapter to describe SQL block parameters

[ISqlTableAdapter](#)

[ISqlValue](#)

Base SQL block value description

[ISqlValue<T>](#)

Typed SQL value

[ISqlValueAdapter](#)

SQL value description adapter

[ISqlVarAdapter](#)

SQL variable value adapter

[ISqlVariable](#)

SQL variable description

Delegates

[SqlBlockDefinition](#)

SQL block definition expression type

[SqlBlockDefinition<T>](#)

SQL block definition expression type

Interface IBlockQueryAdapter

SQL block query structure adapter

Namespace: [XData.Database.SqlBlock](#)

Assembly: XData.docfx.dll

Syntax

```
public interface IBlockQueryAdapter
```

Methods

CastTo<T>()

Type cast modifier

Declaration

```
IBlockQueryAdapter CastTo<T>()
```

Returns

TYPE	DESCRIPTION
IBlockQueryAdapter	

Type Parameters

NAME	DESCRIPTION
T	Type to cast

Default<TValue>(String, DefaultType)

Default value description

Declaration

```
TValue Default<TValue>(string fieldName, DefaultType source)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	fieldName	Field name
DefaultType	source	Default value algorithm

Returns

TYPE	DESCRIPTION
TValue	Field value

Type Parameters

NAME	DESCRIPTION
TValue	Field type

Remarks

WARNING! Can be used in dynamic query mapping only!

Default<TValue>(String, DefaultType, TValue)

Default value description

Declaration

```
TValue Default<TValue>(string fieldName, DefaultType source, TValue value)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	fieldName	Field name
DefaultType	source	Default value algorithm
TValue	value	Default value

Returns

TYPE	DESCRIPTION
TValue	Field value

Type Parameters

NAME	DESCRIPTION
TValue	Field type

Remarks

WARNING! Can be used in dynamic query mapping only!

Default<TValue>(DefaultType)

Default value description

Declaration

```
TValue Default<TValue>(DefaultType source)
```

Parameters

TYPE	NAME	DESCRIPTION
DefaultType	source	Default value algorithm

Returns

TYPE	DESCRIPTION
TValue	Field value

Type Parameters

NAME	DESCRIPTION
TValue	Field type

Remarks

WARNING! Can be used in dynamic query mapping only!

Default<TValue>(DefaultType, TValue)

Default value description

Declaration

TValue Default<TValue>(DefaultType source, TValue value)

Parameters

TYPE	NAME	DESCRIPTION
DefaultType	source	Default value algorithm
TValue	value	Default value

Returns

TYPE	DESCRIPTION
TValue	Field value

Type Parameters

NAME	DESCRIPTION
TValue	Field type

Remarks

WARNING! Can be used in dynamic query mapping only!

Expr<TValue>(String, Expression<Func<IBlockQueryAdapter, TValue>>, DbType, Expression<Func<IExpressionMergeAdapter, IStructureFlag>>[])

SQL expression mapping

Declaration

```
TValue Expr<TValue>(string fieldName, Expression<Func<IBlockQueryAdapter, TValue>> expr, DbType dbType, params Expression<Func<IExpressionMergeAdapter, IStructureFlag>>[] attributes)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	fieldName	Expression field name
System.Linq.Expressions.Expression<System.Func< IBlockQueryAdapter , TValue> >	expr	Expression
System.Data.DbType	dbType	Expression database type
System.Linq.Expressions.Expression<System.Func< IExpressionMergeAdapter , IStructureFlag > > []	attributes	Property mapping attributes

Returns

TYPE	DESCRIPTION
TValue	Expression value

Type Parameters

NAME	DESCRIPTION
TValue	Expression result type

Remarks

WARNING! Can be used in dynamic query mapping only!

Expr<TValue>(String, DataExpressionType, String, DbType, Expression<Func<IExpressionMergeAdapter, IStructureFlag>>[])

SQL expression mapping

Declaration

```
TValue Expr<TValue>(string fieldName, DataExpressionType type, string exprText, DbType dbType, params Expression<Func<IExpressionMergeAdapter, IStructureFlag>>[] attributes)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	fieldName	Expression field name
DataExpressionType	type	Expression source type
System.String	exprText	Expression text
System.Data.DbType	dbType	Expression database type
System.Linq.Expressions.Expression<System.Func< IExpressionMergeAdapter , IStructureFlag >> []	attributes	Property mapping attributes

Returns

TYPE	DESCRIPTION
TValue	Expression value

Type Parameters

NAME	DESCRIPTION
TValue	Expression result type

Remarks

WARNING! Can be used in dynamic query mapping only!

Field<TValue>(Expression<Func<IPropertyMergeAdapter, IStructureFlag>>[])

Field mapping

Declaration

```
TValue Field<TValue>(params Expression<Func<IPropertyMergeAdapter, IStructureFlag>>[] attributes)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func< IPropertyMergeAdapter , IStructureFlag >> []	attributes	Property mapping attributes

Returns

TYPE	DESCRIPTION

TYPE	DESCRIPTION
TValue	Field value

Type Parameters

NAME	DESCRIPTION
TValue	Field type

Remarks

WARNING! Can be used in dynamic query mapping only!

Field<TValue>(String, String, Expression<Func<IPropertyMergeAdapter, IStructureFlag>>[])

Field mapping

Declaration

```
TValue Field<TValue>(string sourceAlias, string fieldName, params Expression<Func<IPropertyMergeAdapter, IStructureFlag>>[] attributes)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	sourceAlias	Source alias
System.String	fieldName	Field name
System.Linq.Expressions.Expression<System.Func< IPropertyMergeAdapter , IStructureFlag >>[]	attributes	Property mapping attributes

Returns

TYPE	DESCRIPTION
TValue	Field value

Type Parameters

NAME	DESCRIPTION
TValue	Field type

Remarks

WARNING! Can be used in dynamic query mapping only!

Ref<TValue>(String, String, Expression<Func<IPropertyMergeAdapter, IStructureFlag>>[])

Inner view property reference mapping

Declaration

```
TValue Ref<TValue>(string sourceAlias, string property, params Expression<Func<IPropertyMergeAdapter, IStructureFlag>>[] attributes)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	sourceAlias	Referenced property subquery alias
System.String	property	Referenced property name
System.Linq.Expressions.Expression<System.Func<IPropertyMergeAdapter, IStructureFlag>>[]	attributes	Property mapping attributes

Returns

TYPE	DESCRIPTION
TValue	Referenced property value

Type Parameters

NAME	DESCRIPTION
TValue	Referenced property type

Remarks

WARNING! Can be used in dynamic query mapping only!

```
Ref<TValue, TObj>(String, Expression<Func<TObj, Object>>, Expression<Func<IPropertyMergeAdapter, IStructureFlag>>[])
```

Inner view property reference mapping

Declaration

```
TValue Ref<TValue, TObj>(string sourceAlias, Expression<Func<TObj, object>> property, params Expression<Func<IPropertyMergeAdapter, IStructureFlag>>[] attributes)
    where TObj : class
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	sourceAlias	Referenced property subquery alias

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func<TObj, System.Object>>	property	Referenced property name
System.Linq.Expressions.Expression<System.Func<IPropertyMergeAdapter, IStructureFlag>>[]	attributes	Property mapping attributes

Returns

TYPE	DESCRIPTION
TValue	Referenced property value

Type Parameters

NAME	DESCRIPTION
TValue	Referenced property type
TObj	Referenced inner view type

Remarks

WARNING! Can be used in dynamic query mapping only!

Val<TValue>(Expression<Func<ISqlValueAdapter, ISqlValue<TValue>>>, Expression<Func<IMergeAttributeAdapter, IStructureFlag>>[])

SQL block variable/constant/parameter value mapping

Declaration

```
TValue Val<TValue>(Expression<Func<ISqlValueAdapter, ISqlValue<TValue>>> value, params
Expression<Func<IMergeAttributeAdapter, IStructureFlag>>[] attributes)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func<ISqlValueAdapter, ISqlValue<TValue>>>	value	SQL block value mapping expression
System.Linq.Expressions.Expression<System.Func<IMergeAttributeAdapter, IStructureFlag>>[]	attributes	Merge roles structure flags

Returns

TYPE	DESCRIPTION
TValue	Value

Type Parameters

NAME	DESCRIPTION
TValue	Value type

Extension Methods

SerializationExtensions.ToXml<T>(T)

CollectionsExtensions.SetValue<TKey, T>(TKey, T)

CollectionsExtensions.AsEnum<T>(T)

DataObjectQueryableExtensions.CompareVariable(Object, FilterOperation, String)

Processing.Do<T>(T, Action<IProcess<T>>[])

TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>)

TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>, TResult)

TypeExtensions.ChangeType<T>(Object)

TypeExtensions.ChangeType(Object, Type)

TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)

TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)

SqlBlockExtensions.Exists(IBlockQueryAdapter, String)

SqlBlockExtensions.Case<T, TResult>(IBlockQueryAdapter, Expression<Func<IBlockQueryAdapter, T>>,

Expression<Func<IBlockQueryAdapter, TResult>>, TaggedExpression<T, TResult>[])

SqlBlockExtensions.DateDiff<T>(IBlockQueryAdapter, Expression<Func<IBlockQueryAdapter, DateTime>>,

Expression<Func<IBlockQueryAdapter, DateTime>>, DateDiffKind)

SqlBlockExtensions.HasFlag<T>(IBlockQueryAdapter, Expression<Func<IBlockQueryAdapter, T>>,

Expression<Func<IBlockQueryAdapter, T>>, Expression<Func<IBlockQueryAdapter, T>>, Expression<Func<IBlockQueryAdapter, T>>)

SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)

SqlBlockExtensions.SqlFn<TResult>(IBlockQueryAdapter, String, Expression<Func<IBlockQueryAdapter, Object>>[])

SqlBlockExtensions.SqlParam<TResult>(IBlockQueryAdapter, String, DbType, Expression<Func<IProcedureParameterAdapter, IProcedureParameterFlag>>[])

SqlBlockExtensions.SqlExpr<TResult>(IBlockQueryAdapter, String, Expression<Func<IBlockQueryAdapter, TResult>>[])

QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Interface IBlockQueryDescription

Base SQL block query description interface

Namespace: [XData.Database.SqlBlock](#)

Assembly: [XData.docfx.dll](#)

Syntax

```
public interface IBlockQueryDescription
```

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Interface IBlockQueryDescription<T>

SQL block query description interface

Namespace: [XData.Database.SqlBlock](#)

Assembly: [XData.docfx.dll](#)

Syntax

```
public interface IBlockQueryDescription<T> : IBlockQueryDescription
```

Type Parameters

NAME	DESCRIPTION
T	SQL block query row mapped type

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Interface IBlockQueryStructure

SQL block query structure interface

Namespace: [XData.Database.SqlBlock](#)

Assembly: XData.docfx.dll

Syntax

```
public interface IBlockQueryStructure
```

Methods

GetStructure(String, Expression<Func<IQueryStructure, IQueryStructure>>)

Returns SQL block query structure

Declaration

```
IBlockQueryStructure GetStructure(string baseTableAlias, Expression<Func<IQueryStructure, IQueryStructure>> query)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	baseTableAlias	
System.Linq.Expressions.Expression<System.Func< IQueryStructure , IQueryStructure >>	query	

Returns

TYPE	DESCRIPTION
IBlockQueryStructure	SQL block query structure

See Also

[ExecuteSqlBlock](#)([IDataScope](#), System.Linq.Expressions.Expression<[SqlBlockDefinition](#)>, System.Linq.Expressions.Expression<System.Func<[ISqlParamAdapter](#), [ISqlParam](#)>> [])
[GetSqlBlockData](#)<T>([IDataScope](#), System.Linq.Expressions.Expression<[SqlBlockDefinition](#)<T>>, System.Linq.Expressions.Expression<System.Func<[ISqlParamAdapter](#), [ISqlParam](#)>> [])

GetStructure<T>(String, Expression<Func<IQueryStructure, IQueryStructure>>)

Returns SQL block query structure

Declaration

```
IBlockQueryStructure GetStructure<T>(string baseTableAlias, Expression<Func<IQueryStructure, IQueryStructure>> query)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	baseTableAlias	
System.Linq.Expressions.Expression<System.Func< IQueryStructure , IQueryStructure >>	query	

Returns

TYPE	DESCRIPTION
IBlockQueryStructure	SQL block query structure

Type Parameters

NAME	DESCRIPTION
T	Base data object type

See Also

[ExecuteSqlBlock](#)([IDataScope](#), System.Linq.Expressions.Expression<[SqlBlockDefinition](#)>, System.Linq.Expressions.Expression<System.Func<[ISqlParamAdapter](#), [ISqlParam](#)>> [])
[GetSqlBlockData](#)<T>([IDataScope](#), System.Linq.Expressions.Expression<[SqlBlockDefinition](#)<T>>, System.Linq.Expressions.Expression<System.Func<[ISqlParamAdapter](#), [ISqlParam](#)>> [])

Select<TResult>(Expression<Func<[IBlockQueryAdapter](#), TResult>>)

SQL block query result mapping

Declaration

<code>IBlockQueryDescription<TResult> Select<TResult>(Expression<Func<IBlockQueryAdapter, TResult>> mapper) where TResult : class</code>

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func< IBlockQueryAdapter , TResult>>	mapper	Mapping expression

Returns

TYPE	DESCRIPTION
IBlockQueryDescription <TResult>	Dynamic query description

Type Parameters

NAME	DESCRIPTION
TResult	Dynamic query result type

See Also

[GetStructure](#)(String, Expression<Func<[IQueryStructure](#), [IQueryStructure](#)>>)

Var(String)

SQL variable name including prefix

Declaration


```
string Var(string varName)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	varName	Variable name

Returns

TYPE	DESCRIPTION
System.String	SQL variable name including prefix

Extension Methods

[SerializationExtensions.ToXml<T>\(T\)](#)

[CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)

[CollectionsExtensions.AsEnum<T>\(T\)](#)

[DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)

[Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)

[TypeExtensions.ChangeType<T>\(Object\)](#)

[TypeExtensions.ChangeType\(Object, Type\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)

[SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)

[QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Interface IExpressionMergeAdapter

Expression structure adapter for SQL block queries

Inherited Members

[IExpressionPropertyAdapter.Size\(Int32\)](#)
[IExpressionPropertyAdapter.Size\(Int32, Int32\)](#)
[IExpressionPropertyAdapter.NativeSqlType\(String\)](#)
[IExpressionPropertyAdapter.Exists\(\)](#)
[IExpressionPropertyAdapter.Exists\(ExistsOperation\)](#)
[IPropertyMappingAdapter.Key\(\)](#)
[IPropertyMappingAdapter.Group\(DataGrouping\)](#)
[IPropertyMappingAdapter.Group\(Int32\)](#)
[IPropertyMappingAdapter.Hidden\(\)](#)
[IPropertyMappingAdapter.OrderBy\(Int32, Boolean\)](#)
[IPropertyMappingAdapter.OrderBy\(Int32\)](#)
[IMergeAttributeAdapter.Source\(String\)](#)
[IMergeAttributeAdapter.Link\(\)](#)
[IMergeAttributeAdapter.Update\(\)](#)
[IMergeAttributeAdapter.Insert\(\)](#)
[IMergeAttributeAdapter.Insert\(DefaultType\)](#)
[IMergeAttributeAdapter.Insert\(DefaultType, Object\)](#)

Namespace: [XData.Database.SqlBlock](#)

Assembly: [XData.docfx.dll](#)

Syntax

```
public interface IExpressionMergeAdapter : IExpressionPropertyAdapter, IPropertyMappingAdapter,
IMergeAttributeAdapter
```

Extension Methods

[SerializationExtensions.ToXml<T>\(T\)](#)
[CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
[CollectionsExtensions.AsEnum<T>\(T\)](#)
[DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
[Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
[TypeExtensions.ChangeType<T>\(Object\)](#)
[TypeExtensions.ChangeType\(Object, Type\)](#)
[TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
[TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
[TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
[TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
[SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
[QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Interface ILoopVariableAdapter<T>

Loop variables mapping adapter

Namespace: [XData.Database.SqlBlock](#)

Assembly: XData.docfx.dll

Syntax

```
public interface ILoopVariableAdapter<T>
```

Type Parameters

NAME	DESCRIPTION
T	Loop cursor type

Methods

Map<TVal>(String, Expression<Func<T, TVal>>)

Loop variable mapping expression

Declaration

```
void Map<TVal>(string variable, Expression<Func<T, TVal>> mapping)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	variable	Variable name
System.Linq.Expressions.Expression<System.Func<T, TVal>>	mapping	Mapped cursor property expression

Type Parameters

NAME	DESCRIPTION
TVal	

See Also

[ForEach<T>](#) (System.Linq.Expressions.Expression<System.Func<[IBlockQueryStructure](#), [IBlockQueryDescription](#)<T>>>, System.String, System.Linq.Expressions.Expression<System.Func<[ISqlBlockAdapter](#), [ISqlBlock](#)>>, System.Linq.Expressions.Expression<System.Action<[ILoopVariableAdapter](#)<T>>>[]])

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)

TypeExtensions.ChangeType(Object, Type)

TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)

TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)

SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)

QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Interface IMergeAttributeAdapter

Merge role structure adapter

Namespace: [XData.Database.SqlBlock](#)

Assembly: XData.docfx.dll

Syntax

```
public interface IMergeAttributeAdapter
```

Methods

Insert()

SQL value is used to insert into field on merge

Declaration

```
IStructureFlag Insert()
```

Returns

TYPE	DESCRIPTION
IStructureFlag	Structure flag

Insert(DefaultType)

SQL value is used to insert into field on merge

Declaration

```
IStructureFlag Insert(DefaultType defaultType)
```

Parameters

TYPE	NAME	DESCRIPTION
DefaultType	defaultType	Default value type

Returns

TYPE	DESCRIPTION
IStructureFlag	Structure flag

Insert(DefaultType, Object)

SQL value is used to insert into field on merge

Declaration

```
IStructureFlag Insert(DefaultType defaultType, object defaultValue)
```

Parameters

TYPE	NAME	DESCRIPTION
DefaultType	defaultType	Default value type
System.Object	defaultValue	Default value

Returns

TYPE	DESCRIPTION
IStructureFlag	Structure flag

Link()

SQL value linked to field on merge

Declaration

IStructureFlag Link()

Returns

TYPE	DESCRIPTION
IStructureFlag	Structure flag

Source(String)

SQL value linked to field on merge

Declaration

IStructureFlag Source(string fieldName)

Parameters

TYPE	NAME	DESCRIPTION
System.String	fieldName	Default value

Returns

TYPE	DESCRIPTION
IStructureFlag	Structure flag

Update()

SQL value is used to update field on merge

Declaration

IStructureFlag Update()

Returns

TYPE	DESCRIPTION
IStructureFlag	Structure flag

Extension Methods

[SerializationExtensions.ToXml<T>\(T\)](#)

[CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)

[CollectionsExtensions.AsEnum<T>\(T\)](#)

[DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)

[Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)

[TypeExtensions.ChangeType<T>\(Object\)](#)

[TypeExtensions.ChangeType\(Object, Type\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)

[SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)

[QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Interface IMergeBlockAdapter

Merge mapping adapter

Inherited Members

[ISetBlockAdapter.Value<TValue>\(Expression<Func<ISqlValueAdapter, ISqlValue<TValue>>>\)](#)
[ISetBlockAdapter.Value<TValue>\(String, Expression<Func<ISqlValueAdapter, ISqlValue<TValue>>>\)](#)

Namespace: [XData.Database.SqlBlock](#)

Assembly: [XData.docfx.dll](#)

Syntax

```
public interface IMergeBlockAdapter : ISetBlockAdapter
```

Methods

[Field<TValue>\(Expression<Func<ISqlValueAdapter, ISqlValue<TValue>>>, Expression<Func<IMergeAttributeAdapter, IStructureFlag>>\[\]\)](#)

Field mapping

Declaration

```
TValue Field<TValue>(Expression<Func<ISqlValueAdapter, ISqlValue<TValue>>> value, params Expression<Func<IMergeAttributeAdapter, IStructureFlag>>[] attributes)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func< ISqlValueAdapter , ISqlValue<TValue>>>	value	Property mapping value
System.Linq.Expressions.Expression<System.Func< IMergeAttributeAdapter , IStructureFlag >>[]	attributes	Merge roles structure flags

Returns

TYPE	DESCRIPTION
TValue	Field value

Type Parameters

NAME	DESCRIPTION
TValue	Field type

[Field<TValue>\(String, Expression<Func<ISqlValueAdapter, ISqlValue<TValue>>>, Expression<Func<IMergeAttributeAdapter, IStructureFlag>>\[\]\)](#)

Field mapping

Declaration

```
TValue Field<TValue>(string fieldName, Expression<Func<ISqlValueAdapter, ISqlValue<TValue>>> value, params Expression<Func<IMergeAttributeAdapter, IStructureFlag>>[] attributes)
```


Parameters

TYPE	NAME	DESCRIPTION
System.String	fieldName	Field name
System.Linq.Expressions.Expression<System.Func<ISqlValueAdapter, ISqlValue<TValue>>>	value	Property mapping value
System.Linq.Expressions.Expression<System.Func<IMergeAttributeAdapter, IStructureFlag>>[]	attributes	Merge roles structure flags

Returns

TYPE	DESCRIPTION
TValue	Field value

Type Parameters

NAME	DESCRIPTION
TValue	Field type

Extension Methods

- SerializationExtensions.ToXml<T>(T)
- CollectionsExtensions.SetValue<TKey, T>(TKey, T)
- CollectionsExtensions.AsEnum<T>(T)
- DataObjectQueryableExtensions.CompareVariable(Object, FilterOperation, String)
- Processing.Do<T>(T, Action<IProcess<T>>[])
- TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>)
- TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>, TResult)
- TypeExtensions.ChangeType<T>(Object)
- TypeExtensions.ChangeType(Object, Type)
- TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)
- TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)
- TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)
- TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)
- SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)
- QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Interface IPropertyMergeAdapter

Dynamic structure adapter for SQL block queries

Inherited Members

[IPropertyMappingAdapter.Key\(\)](#)
[IPropertyMappingAdapter.Group\(DataGrouping\)](#)
[IPropertyMappingAdapter.Group\(Int32\)](#)
[IPropertyMappingAdapter.Hidden\(\)](#)
[IPropertyMappingAdapter.OrderBy\(Int32, Boolean\)](#)
[IPropertyMappingAdapter.OrderBy\(Int32\)](#)
[IMergeAttributeAdapter.Source\(String\)](#)
[IMergeAttributeAdapter.Link\(\)](#)
[IMergeAttributeAdapter.Update\(\)](#)
[IMergeAttributeAdapter.Insert\(\)](#)
[IMergeAttributeAdapter.Insert\(DefaultType\)](#)
[IMergeAttributeAdapter.Insert\(DefaultType, Object\)](#)

Namespace: [XData.Database.SqlBlock](#)

Assembly: [XData.docfx.dll](#)

Syntax

```
public interface IPropertyMergeAdapter : IPropertyMappingAdapter, IMergeAttributeAdapter
```

Extension Methods

[SerializationExtensions.ToXml<T>\(T\)](#)
[CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
[CollectionsExtensions.AsEnum<T>\(T\)](#)
[DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
[Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
[TypeExtensions.ChangeType<T>\(Object\)](#)
[TypeExtensions.ChangeType\(Object, Type\)](#)
[TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
[TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
[TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
[TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
[SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
[QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Interface ISetBlockAdapter

Set/Insert block query adapter

Namespace: [XData.Database.SqlBlock](#)

Assembly: XData.docfx.dll

Syntax

```
public interface ISetBlockAdapter
```

Methods

Value<TValue>(Expression<Func<ISqlValueAdapter, ISqlValue<TValue>>>)

Field mapping

Declaration

```
TValue Value<TValue>(Expression<Func<ISqlValueAdapter, ISqlValue<TValue>>> value)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func< ISqlValueAdapter , ISqlValue <TValue>>>	value	Property mapping value

Returns

TYPE	DESCRIPTION
TValue	Field value

Type Parameters

NAME	DESCRIPTION
TValue	Field type

Value<TValue>(String, Expression<Func<ISqlValueAdapter, ISqlValue<TValue>>>)

Field mapping

Declaration

```
TValue Value<TValue>(string fieldName, Expression<Func<ISqlValueAdapter, ISqlValue<TValue>>> value)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	fieldName	Field name

System.Linq.Expressions.Expression<System.Func< ISqlValueAdapter , ISqlValue <TValue>>>	value	Property mapping value

Returns

TYPE	DESCRIPTION
TValue	Field value

Type Parameters

NAME	DESCRIPTION
TValue	Field type

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Interface ISetSqlVarAdapter

SQL variable assignment adapter

Namespace: [XData.Database.SqlBlock](#)

Assembly: XData.docfx.dll

Syntax

```
public interface ISetSqlVarAdapter
```

Methods

Var<T>(String, Expression<Func<ISqlValueAdapter, ISqlValue<T>>>)

SQL variable value assignment

Declaration

```
ISqlValue<T> Var<T>(string name, Expression<Func<ISqlValueAdapter, ISqlValue<T>>> value)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Variable name
System.Linq.Expressions.Expression<System.Func< ISqlValueAdapter , ISqlValue <T>>>	value	Value assignment

Returns

TYPE	DESCRIPTION
ISqlValue <T>	SQL value description

Type Parameters

NAME	DESCRIPTION
T	Value type

Extension Methods

- [SerializationExtensions.ToXml](#)<T>(T)
- [CollectionsExtensions.SetValue](#)<TKey, T>(TKey, T)
- [CollectionsExtensions.AsEnum](#)<T>(T)
- [DataObjectQueryableExtensions.CompareVariable](#)(Object, FilterOperation, String)
- [Processing.Do](#)<T>(T, Action<IProcess<T>>[])
- [TypeExtensions.TryConvert](#)<T, TResult>(T, Func<T, TResult>)
- [TypeExtensions.TryConvert](#)<T, TResult>(T, Func<T, TResult>, TResult)
- [TypeExtensions.ChangeType](#)<T>(Object)
- [TypeExtensions.ChangeType](#)(Object, Type)
- [TypeExtensions.InvokeMethod](#)<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)
- [TypeExtensions.InvokeMethod](#)(Object, Type[], Expression<Action>, BindingFlags)

TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)

SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)

QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Interface ISqlBlock

SQL block definition interface

Namespace: [XData.Database.SqlBlock](#)

Assembly: [XData.docfx.dll](#)

Syntax

```
public interface ISqlBlock
```

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Interface ISqlBlockAdapter

SQL block adapter

Namespace: [XData.Database.SqlBlock](#)

Assembly: XData.docfx.dll

Syntax

```
public interface ISqlBlockAdapter
```

Methods

Block(Expression<Func<ISqlBlockAdapter, ISqlBlock>>[])

Complex SQL block fences BEGIN...END

Declaration

```
ISqlBlock Block(params Expression<Func<ISqlBlockAdapter, ISqlBlock>>[] blocks)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func< ISqlBlockAdapter , ISqlBlock >>[]	blocks	Contained SQL blocks

Returns

TYPE	DESCRIPTION
ISqlBlock	SQL block description

See Also

[ExecuteSqlBlock\(IDataScope, System.Linq.Expressions.Expression<\[SqlBlockDefinition\]\(#\)>, System.Linq.Expressions.Expression<System.Func<\[ISqlParamAdapter\]\(#\), \[ISqlParam\]\(#\)>>\[\]\)](#)
[GetSqlBlockData<T>\(IDataScope, System.Linq.Expressions.Expression<\[SqlBlockDefinition\]\(#\)<T>>, System.Linq.Expressions.Expression<System.Func<\[ISqlParamAdapter\]\(#\), \[ISqlParam\]\(#\)>>\[\]\)](#)

Block(String)

Custom SQL block

Declaration

```
ISqlBlock Block(string customSqlBlock)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	customSqlBlock	Custom SQL block code

Returns

TYPE	DESCRIPTION
ISqlBlock	SQL block description

See Also

ExecuteSqlBlock(IDataScope, System.Linq.Expressions.Expression<SqlBlockDefinition>, System.Linq.Expressions.Expression<System.Func<ISqlParamAdapter, ISqlParam>>[])
GetSqlBlockData<T>(IDataScope, System.Linq.Expressions.Expression<SqlBlockDefinition<T>>, System.Linq.Expressions.Expression<System.Func<ISqlParamAdapter, ISqlParam>>[])

Block<T>(Expression<Func<ISqlBlockAdapter<T>, ISqlBlock>>[])

Complex SQL block fences BEGIN...END

Declaration

```
ISqlBlock Block<T>(params Expression<Func<ISqlBlockAdapter<T>, ISqlBlock>>[] blocks)
    where T : class, new()
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func<ISqlBlockAdapter<T>, ISqlBlock>>[]	blocks	Contained SQL blocks

Returns

TYPE	DESCRIPTION
ISqlBlock	SQL block description

Type Parameters

NAME	DESCRIPTION
T	Returned object type

See Also

GetSqlBlockData<T>(IDataScope, System.Linq.Expressions.Expression<SqlBlockDefinition<T>>, System.Linq.Expressions.Expression<System.Func<ISqlParamAdapter, ISqlParam>>[])

Declare(Expression<Func<ISqlVariable, ISqlValue>>[])

Declare variables SQL block

Declaration

```
ISqlBlock Declare(params Expression<Func<ISqlVariable, ISqlValue>>[] variables)
```

Parameters

TYPE	NAME	DESCRIPTION

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func< ISqlVariable , ISqlValue >> []	variables	Variable declaration descriptions

Returns

TYPE	DESCRIPTION
ISqlBlock	SQL block description

See Also

[ExecuteSqlBlock](#)([IDataScope](#), System.Linq.Expressions.Expression<[SqlBlockDefinition](#)>, System.Linq.Expressions.Expression<System.Func<[ISqlParamAdapter](#), [ISqlParam](#)>> [])
[GetSqlBlockData](#)<T>([IDataScope](#), System.Linq.Expressions.Expression<[SqlBlockDefinition](#)<T>>, System.Linq.Expressions.Expression<System.Func<[ISqlParamAdapter](#), [ISqlParam](#)>> [])

Delete(Expression<Func<IBlockQueryStructure, IBlockQueryStructure>>)

DELETE SQL block

Declaration

```
ISqlBlock Delete(Expression<Func<IBlockQueryStructure, IBlockQueryStructure>> query)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func< IBlockQueryStructure , IBlockQueryStructure >> >	query	Delete query structure

Returns

TYPE	DESCRIPTION
ISqlBlock	SQL block description

See Also

[ExecuteSqlBlock](#)([IDataScope](#), System.Linq.Expressions.Expression<[SqlBlockDefinition](#)>, System.Linq.Expressions.Expression<System.Func<[ISqlParamAdapter](#), [ISqlParam](#)>> [])
[GetSqlBlockData](#)<T>([IDataScope](#), System.Linq.Expressions.Expression<[SqlBlockDefinition](#)<T>>, System.Linq.Expressions.Expression<System.Func<[ISqlParamAdapter](#), [ISqlParam](#)>> [])

ForEach<T>(Expression<Func<IBlockQueryStructure, IBlockQueryDescription<T>>>, String, Expression<Func<[ISqlBlockAdapter](#), [ISqlBlock](#)>>, Expression<Action<[ILoopVariableAdapter](#)<T>>>[])

Cursor iteration SQL block

Declaration

```
ISqlBlock ForEach<T>(Expression<Func<IBlockQueryStructure, IBlockQueryDescription<T>>> selectQuery, string cursorName, Expression<Func<ISqlBlockAdapter, ISqlBlock>> body, params Expression<Action<ILoopVariableAdapter<T>>>[] mapping)
    where T : class
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func< IBlockQueryStructure , IBlockQueryDescription <T>>>>	selectQuery	Cursor query description
System.String	cursorName	Cursor name
System.Linq.Expressions.Expression<System.Func< ISqlBlockAdapter , ISqlBlock >>>	body	Loop body SQL block
System.Linq.Expressions.Expression<System.Action< ILoopVariableAdapter <T>>>>[]	mapping	Loop variables mapping

Returns

TYPE	DESCRIPTION
ISqlBlock	SQL block description

Type Parameters

NAME	DESCRIPTION
T	Cursor mapped type

See Also

[ExecuteSqlBlock](#)([IDataScope](#), System.Linq.Expressions.Expression<[SqlBlockDefinition](#)>, System.Linq.Expressions.Expression<System.Func<[ISqlParamAdapter](#), [ISqlParam](#)>>>[])
[GetSqlBlockData](#)<T>([IDataScope](#), System.Linq.Expressions.Expression<[SqlBlockDefinition](#)<T>>, System.Linq.Expressions.Expression<System.Func<[ISqlParamAdapter](#), [ISqlParam](#)>>>[])

If(Expression<Func<[IBlockQueryStructure](#), [IBlockQueryStructure](#)>>, FilterOperation, Expression<Func<[ISqlBlockAdapter](#), [ISqlBlock](#)>>[])

IF [NOT] EXISTS(...) SQL block

Declaration

ISqlBlock If(Expression<Func< IBlockQueryStructure , IBlockQueryStructure >> existsStructure, FilterOperation operation, params Expression<Func< ISqlBlockAdapter , ISqlBlock >>[] parts)

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func< IBlockQueryStructure , IBlockQueryStructure >>>	existsStructure	Exists select structure

TYPE	NAME	DESCRIPTION
FilterOperation	operation	Comparison operation (Exists or NotExists only)
System.Linq.Expressions.Expression<System.Func< ISqlBlockAdapter , ISqlBlock >> []	parts	True-part SQL block and (optionally) false-part SQL block

Returns

TYPE	DESCRIPTION
ISqlBlock	SQL block description

See Also

```
ExecuteSqlBlock(IDataScope, System.Linq.Expressions.Expression<SqlBlockDefinition>,
System.Linq.Expressions.Expression<System.Func<ISqlParamAdapter, ISqlParam>> [])
GetSqlBlockData<T>(IDataScope, System.Linq.Expressions.Expression<SqlBlockDefinition<T>>,
System.Linq.Expressions.Expression<System.Func<ISqlParamAdapter, ISqlParam>> [])
```

If<T>(Expression<Func<[ISqlValueAdapter](#), [ISqlValue](#)<T>>>, Expression<Func<[ISqlValueAdapter](#), [ISqlValue](#)<T>>>,
FilterOperation, Expression<Func<[ISqlBlockAdapter](#), [ISqlBlock](#)>> [])

IF SQL block

Declaration

```
ISqlBlock If<T>(Expression<Func<ISqlValueAdapter, ISqlValue<T>>> check, Expression<Func<ISqlValueAdapter,
ISqlValue<T>>> value, FilterOperation oper, params Expression<Func<ISqlBlockAdapter, ISqlBlock>>[] parts)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func< ISqlValueAdapter , ISqlValue <T>>>	check	SQL value to check
System.Linq.Expressions.Expression<System.Func< ISqlValueAdapter , ISqlValue <T>>>	value	SQL value to compare with <code>check</code> Can be null to describe IS NULL or NOT IS NULL comparisons
FilterOperation	oper	Comparison operation
System.Linq.Expressions.Expression<System.Func< ISqlBlockAdapter , ISqlBlock >> []	parts	True-part SQL block and (optionally) false-part SQL block

Returns

TYPE	DESCRIPTION
ISqlBlock	SQL block description

Type Parameters

NAME	DESCRIPTION
T	

See Also

[ExecuteSqlBlock\(IDataScope, System.Linq.Expressions.Expression<\[SqlBlockDefinition\]\(#\)>, System.Linq.Expressions.Expression<System.Func<\[ISqlParamAdapter\]\(#\), \[ISqlParam\]\(#\)>>\[\]\)](#)
[GetSqlBlockData<T>\(IDataScope, System.Linq.Expressions.Expression<\[SqlBlockDefinition\]\(#\)<T>>, System.Linq.Expressions.Expression<System.Func<\[ISqlParamAdapter\]\(#\), \[ISqlParam\]\(#\)>>\[\]\)](#)

[Insert<TResult>\(String, DataTable, Expression<Func<\[ISetBlockAdapter\]\(#\), TResult>>\)](#)

INSERT INTO...VALUES(...),VALUES(...)... SQL block

Declaration

```
ISqlBlock Insert<TResult>(string tableName, DataTable data, Expression<Func<ISetBlockAdapter, TResult>>
mapper)
    where TResult : class
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	tableName	Table to insert
System.Data.DataTable	data	Data table filled by data Access data by Data<T>(String) method inside <code>expressions</code>
System.Linq.Expressions.Expression<System.Func< ISetBlockAdapter , TResult>>	mapper	Fields to insert values mapper

Returns

TYPE	DESCRIPTION
ISqlBlock	SQL block description

Type Parameters

NAME	DESCRIPTION
TResult	

See Also

[Data<T>\(String\)](#)

```
ExecuteSqlBlock(IDataScope, System.Linq.Expressions.Expression<SqlBlockDefinition>,
System.Linq.Expressions.Expression<System.Func<ISqlParamAdapter, ISqlParam>> []))
GetSqlBlockData<T>(IDataScope, System.Linq.Expressions.Expression<SqlBlockDefinition<T>>,
System.Linq.Expressions.Expression<System.Func<ISqlParamAdapter, ISqlParam>> []))
```

```
Insert<T>(String, Expression<Func<IBlockQueryStructure, IBlockQueryDescription<T>>>)
```

INSERT INTO... SELECT SQL block

Declaration

```
ISqlBlock Insert<T>(string tableName, Expression<Func<IBlockQueryStructure, IBlockQueryDescription<T>>>
selectQuery)
    where T : class
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	tableName	Table to insert
System.Linq.Expressions.Expression<System.Func<IBlockQueryStructure, IBlockQueryDescription<T>>>	selectQuery	Select query description Use same fields order as in <code>fields</code> parameter

Returns

TYPE	DESCRIPTION
ISqlBlock	SQL block description

Type Parameters

NAME	DESCRIPTION
T	

See Also

```
ExecuteSqlBlock(IDataScope, System.Linq.Expressions.Expression<SqlBlockDefinition>,
System.Linq.Expressions.Expression<System.Func<ISqlParamAdapter, ISqlParam>> []))
GetSqlBlockData<T>(IDataScope, System.Linq.Expressions.Expression<SqlBlockDefinition<T>>,
System.Linq.Expressions.Expression<System.Func<ISqlParamAdapter, ISqlParam>> []))
```

```
Insert<TResult>(String, Expression<Func<ISetBlockAdapter, TResult>>)
```

Single INSERT INTO...VALUES SQL block

Declaration

```
ISqlBlock Insert<TResult>(string tableName, Expression<Func<ISetBlockAdapter, TResult>> mapper)
    where TResult : class
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	tableName	Table to insert
System.Linq.Expressions.Expression<System.Func< ISetBlockAdapter , TResult>>	mapper	Fields to insert values mapper

Returns

TYPE	DESCRIPTION
ISqlBlock	SQL block description

Type Parameters

NAME	DESCRIPTION
TResult	

See Also

[ExecuteSqlBlock](#)([IDataScope](#), System.Linq.Expressions.Expression<[SqlBlockDefinition](#)>, System.Linq.Expressions.Expression<System.Func<[ISqlParamAdapter](#), [ISqlParam](#)>> [])
[GetSqlBlockData](#)<T>([IDataScope](#), System.Linq.Expressions.Expression<[SqlBlockDefinition](#)<T>>, System.Linq.Expressions.Expression<System.Func<[ISqlParamAdapter](#), [ISqlParam](#)>> [])

Merge<T>(String, String, String, Expression<Func<[IMergeBlockAdapter](#), T>>)

MERGE SQL block

Declaration

```
ISqlBlock Merge<T>(string tableName, string targetAlias, string sourceAlias,
Expression<Func<IMergeBlockAdapter, T>> mapper)
    where T : class
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	tableName	Table name to merge with
System.String	targetAlias	Target table alias
System.String	sourceAlias	Data source alias
System.Linq.Expressions.Expression<System.Func< IMergeBlockAdapter , T>>	mapper	Data source mapper Use Link() , Update() and Insert() structure flags to set merge role of data source columns

Returns

TYPE	DESCRIPTION
ISqlBlock	SQL block description

Type Parameters

NAME	DESCRIPTION
T	

See Also

[ExecuteSqlBlock\(IDataScope, System.Linq.Expressions.Expression<\[SqlBlockDefinition\]\(#\)>, System.Linq.Expressions.Expression<System.Func<\[ISqlParamAdapter\]\(#\), \[ISqlParam\]\(#\)>>\[\]\)](#)
[GetSqlBlockData<T>\(IDataScope, System.Linq.Expressions.Expression<\[SqlBlockDefinition\]\(#\)<T>>, System.Linq.Expressions.Expression<System.Func<\[ISqlParamAdapter\]\(#\), \[ISqlParam\]\(#\)>>\[\]\)](#)

Merge<T, TSource>(String, String, Expression<Func<IBlockQueryStructure, IBlockQueryDescription<TSource>>>, String, Expression<Func<IMergeBlockAdapter, T>>)

MERGE SQL block

Declaration

```
ISqlBlock Merge<T, TSource>(string tableName, string targetAlias, Expression<Func<IBlockQueryStructure, IBlockQueryDescription<TSource>>> sourceQuery, string sourceAlias, Expression<Func<IMergeBlockAdapter, T>> mapper)
    where T : class where TSource : class
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	tableName	Table name to merge with
System.String	targetAlias	Target table alias
System.Linq.Expressions.Expression<System.Func< IBlockQueryStructure , IBlockQueryDescription <TSource>>>	sourceQuery	Source subquery description
System.String	sourceAlias	Data source alias
System.Linq.Expressions.Expression<System.Func< IMergeBlockAdapter , T>>	mapper	Data source mapper Use Link() , Update() and Insert() structure flags to set merge role of data source columns

Returns

TYPE	DESCRIPTION
ISqlBlock	SQL block description

Type Parameters

NAME	DESCRIPTION
T	
TSource	

See Also

[ExecuteSqlBlock\(IDataScope, System.Linq.Expressions.Expression<\[SqlBlockDefinition\]\(#\)>, System.Linq.Expressions.Expression<System.Func<\[ISqlParamAdapter\]\(#\), \[ISqlParam\]\(#\)>>\[\]\)](#)
[GetSqlBlockData<T>\(IDataScope, System.Linq.Expressions.Expression<\[SqlBlockDefinition\]\(#\)<T>>, System.Linq.Expressions.Expression<System.Func<\[ISqlParamAdapter\]\(#\), \[ISqlParam\]\(#\)>>\[\]\)](#)

SetVar(Expression<Func<ISetSqlVarAdapter, ISqlValue>>[])

SQL variables initialization SQL block

Declaration

```
ISqlBlock SetVar(params Expression<Func<ISetSqlVarAdapter, ISqlValue>>[] variables)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func< ISetSqlVarAdapter , ISqlValue >>[]	variables	Variable value descriptions

Returns

TYPE	DESCRIPTION
ISqlBlock	SQL block description

See Also

[Declare\(Expression<Func<ISqlVariable, ISqlValue>>\[\]\)](#)
[ExecuteSqlBlock\(IDataScope, System.Linq.Expressions.Expression<\[SqlBlockDefinition\]\(#\)>, System.Linq.Expressions.Expression<System.Func<\[ISqlParamAdapter\]\(#\), \[ISqlParam\]\(#\)>>\[\]\)](#)
[GetSqlBlockData<T>\(IDataScope, System.Linq.Expressions.Expression<\[SqlBlockDefinition\]\(#\)<T>>, System.Linq.Expressions.Expression<System.Func<\[ISqlParamAdapter\]\(#\), \[ISqlParam\]\(#\)>>\[\]\)](#)

SetVar<TValue>(Expression<Func<IBlockQueryStructure, IBlockQueryDescription<TValue>>>, Expression<Func<ISqlVarAdapter, ISqlValue>>[])

SQL variables initialization using SELECT... INTO statement SQL block

Declaration

```
ISqlBlock SetVar<TValue>(Expression<Func<IBlockQueryStructure, IBlockQueryDescription<TValue>>> selectQuery, params Expression<Func<ISqlVarAdapter, ISqlValue>>[] variables)
```

Parameters

TYPE	NAME	DESCRIPTION

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func< IBlockQueryStructure , IBlockQueryDescription <TValue>>>	selectQuery	Query description used as data source
System.Linq.Expressions.Expression<System.Func< ISqlVarAdapter , ISqlValue >>[]	variables	Variable value descriptions

Returns

TYPE	DESCRIPTION
ISqlBlock	SQL block description

Type Parameters

NAME	DESCRIPTION
TValue	

See Also

[Declare\(Expression<Func<ISqlVariable, ISqlValue>>\[\]\)](#)

[ExecuteSqlBlock\(IDataScope, System.Linq.Expressions.Expression<\[SqlBlockDefinition\]\(#\)>,](#)

[System.Linq.Expressions.Expression<System.Func<\[ISqlParamAdapter\]\(#\), \[ISqlParam\]\(#\)>>\[\]\)](#)

[GetSqlBlockData<T>\(IDataScope, System.Linq.Expressions.Expression<\[SqlBlockDefinition\]\(#\)<T>>,](#)

[System.Linq.Expressions.Expression<System.Func<\[ISqlParamAdapter\]\(#\), \[ISqlParam\]\(#\)>>\[\]\)](#)

[Update<T>\(Expression<Func<IBlockQueryStructure, IBlockQueryStructure>>, Expression<Func<ISetBlockAdapter, T>>\)](#)

UPDATE SQL block

Declaration

ISqlBlock Update<T>(Expression<Func<IBlockQueryStructure, IBlockQueryStructure>> query, Expression<Func<ISetBlockAdapter, T>> setExpr) where T : class

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func< IBlockQueryStructure , IBlockQueryStructure >>	query	SQL block query description
System.Linq.Expressions.Expression<System.Func< ISetBlockAdapter , T>>	setExpr	Set expression description

Returns

TYPE	DESCRIPTION
ISqlBlock	SQL block description

Type Parameters

NAME	DESCRIPTION
T	

See Also

[ExecuteSqlBlock\(IDataScope, System.Linq.Expressions.Expression<SqlBlockDefinition>, System.Linq.Expressions.Expression<System.Func<ISqlParamAdapter, ISqlParam>>\[\]\)](#)
[GetSqlBlockData<T>\(IDataScope, System.Linq.Expressions.Expression<SqlBlockDefinition<T>>, System.Linq.Expressions.Expression<System.Func<ISqlParamAdapter, ISqlParam>>\[\]\)](#)

Extension Methods

[SerializationExtensions.ToXml<T>\(T\)](#)
[CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
[CollectionsExtensions.AsEnum<T>\(T\)](#)
[DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
[Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
[TypeExtensions.ChangeType<T>\(Object\)](#)
[TypeExtensions.ChangeType\(Object, Type\)](#)
[TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
[TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
[TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
[TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
[SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
[QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Interface ISqlBlockAdapter<T>

Adapter of SQL block returning data

Inherited Members

ISqlBlockAdapter.Block(Expression<Func<ISqlBlockAdapter, ISqlBlock>>[])
ISqlBlockAdapter.Block<T>(Expression<Func<ISqlBlockAdapter<T>, ISqlBlock>>[])
ISqlBlockAdapter.Block(String)
ISqlBlockAdapter.Declare(Expression<Func<ISqlVariable, ISqlValue>>[])
ISqlBlockAdapter.SetVar(Expression<Func<ISetSqlVarAdapter, ISqlValue>>[])
ISqlBlockAdapter.SetVar<TValue>(Expression<Func<IBlockQueryStructure, IBlockQueryDescription<TValue>>>, Expression<Func<ISqlVarAdapter, ISqlValue>>[])
ISqlBlockAdapter.Update<T>(Expression<Func<IBlockQueryStructure, IBlockQueryStructure>>, Expression<Func<ISetBlockAdapter, T>>)
ISqlBlockAdapter.Insert<TResult>(String, Expression<Func<ISetBlockAdapter, TResult>>)
ISqlBlockAdapter.Insert<TResult>(String, DataTable, Expression<Func<ISetBlockAdapter, TResult>>)
ISqlBlockAdapter.Insert<T>(String, Expression<Func<IBlockQueryStructure, IBlockQueryDescription<T>>>)
ISqlBlockAdapter.Delete(Expression<Func<IBlockQueryStructure, IBlockQueryStructure>>)
ISqlBlockAdapter.Merge<T>(String, String, String, Expression<Func<IMergeBlockAdapter, T>>)
ISqlBlockAdapter.Merge<T, TSource>(String, String, Expression<Func<IBlockQueryStructure, IBlockQueryDescription<TSource>>>, String, Expression<Func<IMergeBlockAdapter, T>>)
ISqlBlockAdapter.If<T>(Expression<Func<ISqlValueAdapter, ISqlValue<T>>>, Expression<Func<ISqlValueAdapter, ISqlValue<T>>>, FilterOperation, Expression<Func<ISqlBlockAdapter, ISqlBlock>>[])
ISqlBlockAdapter.If(Expression<Func<IBlockQueryStructure, IBlockQueryStructure>>, FilterOperation, Expression<Func<ISqlBlockAdapter, ISqlBlock>>[])
ISqlBlockAdapter.ForEach<T>(Expression<Func<IBlockQueryStructure, IBlockQueryDescription<T>>>, String, Expression<Func<ISqlBlockAdapter, ISqlBlock>>, Expression<Action<ILoopVariableAdapter<T>>>[])

Namespace: XData.Database.SqlBlock

Assembly: XData.docfx.dll

Syntax

```
public interface ISqlBlockAdapter<T> : ISqlBlockAdapter where T : class, new()
```

Type Parameters

NAME	DESCRIPTION
T	Returned data type

Methods

Return(Expression<Func<IBlockQueryStructure, IBlockQueryDescription<T>>>)

Returning data out of sql block

Declaration

```
ISqlBlock Return(Expression<Func<IBlockQueryStructure, IBlockQueryDescription<T>>> selectQuery)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func< IBlockQueryStructure , IBlockQueryDescription <T>>>	selectQuery	Select query description

Returns

TYPE	DESCRIPTION
ISqlBlock	SQL block description

See Also

[GetSqlBlockData<T>](#) ([IDataScope](#), System.Linq.Expressions.Expression<[SqlBlockDefinition](#)<T>>, System.Linq.Expressions.Expression<System.Func<[ISqlParamAdapter](#), [ISqlParam](#)>> [])

Extension Methods

- [SerializationExtensions.ToXml<T>](#) (T)
- [CollectionsExtensions.SetValue<TKey, T>](#) (TKey, T)
- [CollectionsExtensions.AsEnum<T>](#) (T)
- [DataObjectQueryableExtensions.CompareVariable](#) (Object, FilterOperation, String)
- [Processing.Do<T>](#) (T, Action<IProcess<T>> [])
- [TypeExtensions.TryConvert<T, TResult>](#) (T, Func<T, TResult>)
- [TypeExtensions.TryConvert<T, TResult>](#) (T, Func<T, TResult>, TResult)
- [TypeExtensions.ChangeType<T>](#) (Object)
- [TypeExtensions.ChangeType](#) (Object, Type)
- [TypeExtensions.InvokeMethod<TRet>](#) (Object, Type[], Expression<Func<TRet>>, BindingFlags)
- [TypeExtensions.InvokeMethod](#) (Object, Type[], Expression<Action>, BindingFlags)
- [TypeExtensions.InvokeMethod<TRet>](#) (Object, Expression<Func<TRet>>, BindingFlags)
- [TypeExtensions.InvokeMethod](#) (Object, Expression<Action>, BindingFlags)
- [SqlBlockExtensions.SetExpression<TTag, TResult>](#) (TTag, Expression<Func<[IBlockQueryAdapter](#), TResult>>)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>](#) (TTag, Expression<Func<[IQueryStructureAdapter](#), TResult>>)

Interface ISqlParam

SQL block parameter interface

Namespace: [XData.Database.SqlBlock](#)

Assembly: XData.docfx.dll

Syntax

```
public interface ISqlParam
```

Methods

Set<T>(T)

Set SQL block parameter value

Declaration

```
ISqlParam Set<T>(T value)
```

Parameters

TYPE	NAME	DESCRIPTION
T	value	Parameter value

Returns

TYPE	DESCRIPTION
ISqlParam	SQL block parameter

Type Parameters

NAME	DESCRIPTION
T	Type of parameter value

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)

```
QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)
```

Interface ISqlParamAdapter

Adapter to describe SQL block parameters

Namespace: [XData.Database.SqlBlock](#)

Assembly: XData.docfx.dll

Syntax

```
public interface ISqlParamAdapter
```

Methods

Param<T>(String, DbType)

SQL block parameter

Declaration

```
ISqlParam Param<T>(string name, DbType dbType)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Parameter name
System.Data.DbType	dbType	Parameter database type

Returns

TYPE	DESCRIPTION
ISqlParam	SQL block parameter description

Type Parameters

NAME	DESCRIPTION
T	Parameter mapped type

Param<T>(String, DbType, Int32)

SQL block parameter

Declaration

```
ISqlParam Param<T>(string name, DbType dbType, int length)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Parameter name

TYPE	NAME	DESCRIPTION
System.Data.DbType	dbType	Parameter database type
System.Int32	length	Parameter length

Returns

TYPE	DESCRIPTION
ISqlParam	SQL block parameter description

Type Parameters

NAME	DESCRIPTION
T	Parameter mapped typ

Param<T>(String, DbType, String)

SQL block parameter

Declaration

```
ISqlParam Param<T>(string name, DbType dbType, string nativeSqlType)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Parameter name
System.Data.DbType	dbType	Parameter database type
System.String	nativeSqlType	Parameter native SQL type

Returns

TYPE	DESCRIPTION
ISqlParam	SQL block parameter description

Type Parameters

NAME	DESCRIPTION

NAME	DESCRIPTION
T	Parameter mapped typ

Param<T>(String, DbType, String, Int32)

SQL block parameter

Declaration

```
ISqlParam Param<T>(string name, DbType dbType, string nativeSqlType, int length)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Parameter name
System.Data.DbType	dbType	Parameter database type
System.String	nativeSqlType	Parameter native SQL type
System.Int32	length	Parameter length

Returns

TYPE	DESCRIPTION
ISqlParam	SQL block parameter description

Type Parameters

NAME	DESCRIPTION
T	Parameter mapped typ

Param<T>(String, SqlType)

SQL block parameter

Declaration

```
ISqlParam Param<T>(string name, SqlType type)
```

Parameters

TYPE	NAME	DESCRIPTION

TYPE	NAME	DESCRIPTION
System.String	name	Parameter name
SqlType	type	Parameter type

Returns

TYPE	DESCRIPTION
ISqlParam	SQL block parameter description

Type Parameters

NAME	DESCRIPTION
T	Parameter mapped type

Extension Methods

- SerializationExtensions.ToXml<T>(T)
- CollectionsExtensions.SetValue<TKey, T>(TKey, T)
- CollectionsExtensions.AsEnum<T>(T)
- DataObjectQueryableExtensions.CompareVariable(Object, FilterOperation, String)
- Processing.Do<T>(T, Action<IProcess<T>>[])
- TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>)
- TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>, TResult)
- TypeExtensions.ChangeType<T>(Object)
- TypeExtensions.ChangeType(Object, Type)
- TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)
- TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)
- TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)
- TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)
- SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)
- QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Interface ISqlTableAdapter

Namespace: [XData.Database.SqlBlock](#)

Assembly: XData.docfx.dll

Syntax

```
public interface ISqlTableAdapter
```

Methods

Field(String, DbType)

Declaration

```
void Field(string fieldName, DbType dbType)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	fieldName	
System.Data.DbType	dbType	

Field(String, DbType, Int32)

Declaration

```
void Field(string fieldName, DbType dbType, int length)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	fieldName	
System.Data.DbType	dbType	
System.Int32	length	

Field(String, String)

Declaration

```
void Field(string fieldName, string nativeSqlType)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	fieldName	
System.String	nativeSqlType	

Field(String, String, Int32)

Declaration

```
void Field(string fieldName, string nativeSqlType, int length)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	fieldName	
System.String	nativeSqlType	
System.Int32	length	

Field(String, SqlType)

Declaration

<pre>void Field(string fieldName, SqlType sqlType)</pre>

Parameters

TYPE	NAME	DESCRIPTION
System.String	fieldName	
SqlType	sqlType	

Extension Methods

[SerializationExtensions.ToXml<T>\(T\)](#)

[CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)

[CollectionsExtensions.AsEnum<T>\(T\)](#)

[DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)

[Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)

[TypeExtensions.ChangeType<T>\(Object\)](#)

[TypeExtensions.ChangeType\(Object, Type\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)

[SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)

[QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Interface ISqlValue

Base SQL block value description

Namespace: [XData.Database.SqlBlock](#)

Assembly: [XData.docfx.dll](#)

Syntax

```
public interface ISqlValue
```

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Interface ISqlValue<T>

Typed SQL value

Namespace: [XData.Database.SqlBlock](#)

Assembly: [XData.docfx.dll](#)

Syntax

```
public interface ISqlValue<T> : ISqlValue
```

Type Parameters

NAME	DESCRIPTION
T	

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Interface ISqlValueAdapter

SQL value description adapter

Inherited Members

[ISqlVarAdapter.Var<T>\(String\)](#)

Namespace: [XData.Database.SqlBlock](#)

Assembly: XData.docfx.dll

Syntax

```
public interface ISqlValueAdapter : ISqlVarAdapter
```

Methods

Const<T>(T, DbType)

SQL constant value

Declaration

```
ISqlValue<T> Const<T>(T value, DbType dbType)
```

Parameters

TYPE	NAME	DESCRIPTION
T	value	Constant value
System.Data.DbType	dbType	Constant DbType

Returns

TYPE	DESCRIPTION
ISqlValue<T>	SQL value description

Type Parameters

NAME	DESCRIPTION
T	Constant mapped type

Data<T>(String)

Data value

Declaration

```
ISqlValue<T> Data<T>(string name)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Data column or parameter name

Returns

TYPE	DESCRIPTION
ISqlValue<T>	SQL value description

Type Parameters

NAME	DESCRIPTION
T	Data type

Default<T>(DefaultType)

Default value mapping

Declaration

ISqlValue<T> Default<T>(DefaultType defaultType)

Parameters

TYPE	NAME	DESCRIPTION
DefaultType	defaultType	Default value type

Returns

TYPE	DESCRIPTION
ISqlValue<T>	SQL value description

Type Parameters

NAME	DESCRIPTION
T	Value mapped type

Default<T>(DefaultType, Object)

Default value mapping

Declaration

ISqlValue<T> Default<T>(DefaultType defaultType, object defaultValue)

Parameters

TYPE	NAME	DESCRIPTION
DefaultType	defaultType	Default value type
System.Object	defaultValue	Default value

Returns

TYPE	DESCRIPTION
ISqlValue<T>	SQL value description

Type Parameters

NAME	DESCRIPTION
T	Value mapped type

Expr<T>(Expression<Func<IBlockQueryAdapter, T>>, DbType)

SQL expression mapping

Declaration

ISqlValue<T> Expr<T>(Expression<Func<IBlockQueryAdapter, T>> expr, DbType dbType)

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func<IBlockQueryAdapter, T>>	expr	Expression
System.Data.DbType	dbType	Expression database type

Returns

TYPE	DESCRIPTION
ISqlValue<T>	SQL value description

Type Parameters

NAME	DESCRIPTION
T	Value mapped type

Expr<T>(String, DbType)

SQL expression mapping

Declaration

```
ISqlValue<T> Expr<T>(string exprText, DbType dbType)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	exprText	Expression text
System.Data.DbType	dbType	Expression database type

Returns

TYPE	DESCRIPTION
ISqlValue<T>	SQL value description

Type Parameters

NAME	DESCRIPTION
T	Value mapped type

Field<T>(String, String)

SQL field mapping

Declaration

```
ISqlValue<T> Field<T>(string alias, string fieldName)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	alias	Data source alias
System.String	fieldName	Field name

Returns

TYPE	DESCRIPTION
ISqlValue<T>	SQL value description

Type Parameters

NAME	DESCRIPTION
T	Value mapped type

Subquery<T>(Expression<Func<IBlockQueryStructure, IBlockQueryStructure>>, Expression<Func<IBlockQueryAdapter, T>>)

Single column subquery as value

Declaration

```
ISqlValue<T> Subquery<T>(Expression<Func<IBlockQueryStructure, IBlockQueryStructure>> query, Expression<Func<IBlockQueryAdapter, T>> field)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func<IBlockQueryStructure, IBlockQueryStructure>>	query	Query structure
System.Linq.Expressions.Expression<System.Func<IBlockQueryAdapter, T>>	field	Result field

Returns

TYPE	DESCRIPTION
ISqlValue<T>	SQL value description

Type Parameters

NAME	DESCRIPTION
T	Value mapped type

Subquery<T>(Expression<Func<IBlockQueryStructure, IBlockQueryStructure>>, Expression<Func<IBlockQueryAdapter, T>>, DataStructureFlag)

Single column subquery as value

Declaration

```
ISqlValue<T> Subquery<T>(Expression<Func<IBlockQueryStructure, IBlockQueryStructure>> query, Expression<Func<IBlockQueryAdapter, T>> field, DataStructureFlag flag)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func<IBlockQueryStructure, IBlockQueryStructure>>	query	Query structure

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func< IBlockQueryAdapter , T>>	field	Result field
DataStructureFlag	flag	Query structure flag

Returns

TYPE	DESCRIPTION
ISqlValue <T>	SQL value description

Type Parameters

NAME	DESCRIPTION
T	Value mapped type

Extension Methods

- [SerializationExtensions.ToXml](#)<T>(T)
- [CollectionsExtensions.SetValue](#)<TKey, T>(TKey, T)
- [CollectionsExtensions.AsEnum](#)<T>(T)
- [DataObjectQueryableExtensions.CompareVariable](#)(Object, FilterOperation, String)
- [Processing.Do](#)<T>(T, Action<IProcess<T>>[])
- [TypeExtensions.TryConvert](#)<T, TResult>(T, Func<T, TResult>)
- [TypeExtensions.TryConvert](#)<T, TResult>(T, Func<T, TResult>, TResult)
- [TypeExtensions.ChangeType](#)<T>(Object)
- [TypeExtensions.ChangeType](#)(Object, Type)
- [TypeExtensions.InvokeMethod](#)<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)
- [TypeExtensions.InvokeMethod](#)(Object, Type[], Expression<Action>, BindingFlags)
- [TypeExtensions.InvokeMethod](#)<TRet>(Object, Expression<Func<TRet>>, BindingFlags)
- [TypeExtensions.InvokeMethod](#)(Object, Expression<Action>, BindingFlags)
- [SqlBlockExtensions.SetExpression](#)<TTag, TResult>(TTag, Expression<Func<[IBlockQueryAdapter](#), TResult>>)
- [QueryDescriptionExtensions.SetExpression](#)<TTag, TResult>(TTag, Expression<Func<[IQueryStructureAdapter](#), TResult>>)

Interface ISqlVarAdapter

SQL variable value adapter

Namespace: [XData.Database.SqlBlock](#)

Assembly: XData.docfx.dll

Syntax

```
public interface ISqlVarAdapter
```

Methods

Var<T>(String)

SQL variable value

Declaration

```
ISqlValue<T> Var<T>(string name)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Variable name

Returns

TYPE	DESCRIPTION
ISqlValue<T>	SQL value description

Type Parameters

NAME	DESCRIPTION
T	Variable mapped type

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)

```
QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)
```

Interface ISqlVariable

SQL variable description

Namespace: [XData.Database.SqlBlock](#)

Assembly: XData.docfx.dll

Syntax

```
public interface ISqlVariable
```

Methods

Var<T>(String, DbType)

SQL variable value

Declaration

```
ISqlValue Var<T>(string name, DbType dbType)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Variable name
System.Data.DbType	dbType	Database type

Returns

TYPE	DESCRIPTION
ISqlValue	SQL value description

Type Parameters

NAME	DESCRIPTION
T	Variable mapped type

Var<T>(String, DbType, Int32)

SQL variable value

Declaration

```
ISqlValue Var<T>(string name, DbType dbType, int length)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Variable name

TYPE	NAME	DESCRIPTION
System.Data.DbType	dbType	Database type
System.Int32	length	Database type length

Returns

TYPE	DESCRIPTION
ISqlValue	SQL value description

Type Parameters

NAME	DESCRIPTION
T	Variable mapped type

Var<T>(String, DbType, Int32, Int32)

SQL variable value

Declaration

ISqlValue Var<T>(string name, DbType dbType, int length, int scale)

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Variable name
System.Data.DbType	dbType	Database type
System.Int32	length	Database type length
System.Int32	scale	Database type scale

Returns

TYPE	DESCRIPTION
ISqlValue	SQL value description

Type Parameters

NAME	DESCRIPTION
T	Variable mapped type

Var<T>(String, Expression<Action<ISqlTableAdapter>>[])

SQL variable value

Declaration

```
ISqlValue Var<T>(string name, params Expression<Action<ISqlTableAdapter>>[] fields)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Variable name
System.Linq.Expressions.Expression<System.Action< ISqlTableAdapter >> > []	fields	Table variable fields description

Returns

TYPE	DESCRIPTION
ISqlValue	SQL value description

Type Parameters

NAME	DESCRIPTION
T	Variable mapped type

Var<T>(String, String)

Sql variable value

Declaration

```
ISqlValue Var<T>(string name, string nativeSqlType)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Variable name
System.String	nativeSqlType	Native database type name

Returns

TYPE	DESCRIPTION
ISqlValue	Sql value description

Type Parameters

NAME	DESCRIPTION
T	Variable mapped type

Var<T>(String, String, Int32)

SQL variable value

Declaration

```
ISqlValue Var<T>(string name, string nativeSqlType, int length)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Variable name
System.String	nativeSqlType	Native database type name
System.Int32	length	Database type length

Returns

TYPE	DESCRIPTION
ISqlValue	SQL value description

Type Parameters

NAME	DESCRIPTION
T	Variable mapped type

Var<T>(String, String, Int32, Int32)

SQL variable value

Declaration

```
ISqlValue Var<T>(string name, string nativeSqlType, int length, int scale)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Variable name
System.String	nativeSqlType	Native database type name
System.Int32	length	Database type length
System.Int32	scale	Database type scale

Returns

TYPE	DESCRIPTION
ISqlValue	SQL value description

Type Parameters

NAME	DESCRIPTION
T	Variable mapped type

Extension Methods

[SerializationExtensions.ToXml<T>\(T\)](#)

[CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)

[CollectionsExtensions.AsEnum<T>\(T\)](#)

[DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)

[Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)

[TypeExtensions.ChangeType<T>\(Object\)](#)

[TypeExtensions.ChangeType\(Object, Type\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)

[SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)

[QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Delegate SqlBlockDefinition

SQL block definition expression type

Namespace: [XData.Database.SqlBlock](#)

Assembly: XData.docfx.dll

Syntax

```
public delegate ISqlBlock SqlBlockDefinition(ISqlBlockAdapter adapter);
```

Parameters

TYPE	NAME	DESCRIPTION
ISqlBlockAdapter	adapter	SQL block adapter interface

Returns

TYPE	DESCRIPTION
ISqlBlock	SQL block definition

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Delegate SqlBlockDefinition<T>

SQL block definition expression type

Namespace: [XData.Database.SqlBlock](#)

Assembly: XData.docfx.dll

Syntax

```
public delegate ISqlBlock SqlBlockDefinition<T>(ISqlBlockAdapter<T> adapter)
    where T : class, new();
```

Parameters

TYPE	NAME	DESCRIPTION
ISqlBlockAdapter<T>	adapter	SQL block adapter typed interface

Returns

TYPE	DESCRIPTION
ISqlBlock	SQL block definition

Type Parameters

NAME	DESCRIPTION
T	SQL block result mapped type

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Class SqlBlockExtensions

Sql block extensions

Inheritance

System.Object

SqlBlockExtensions

Inherited Members

System.Object.Equals(System.Object)

System.Object.Equals(System.Object, System.Object)

System.Object.GetHashCode()

System.Object.GetType()

System.Object.MemberwiseClone()

System.Object.ReferenceEquals(System.Object, System.Object)

System.Object.ToString()

Namespace: [XData.Database.SqlBlock](#)

Assembly: XData.docfx.dll

Syntax

```
public static class SqlBlockExtensions
```

Methods

Case<T, TResult>(IBlockQueryAdapter, Expression<Func<IBlockQueryAdapter, T>>, Expression<Func<IBlockQueryAdapter, TResult>>, TaggedExpression<T, TResult>[])

Case

Declaration

```
public static TResult Case<T, TResult>(this IBlockQueryAdapter adapter, Expression<Func<IBlockQueryAdapter, T>> expression, Expression<Func<IBlockQueryAdapter, TResult>> elseValue, params TaggedExpression<T, TResult>[] switchValues)
```

Parameters

TYPE	NAME	DESCRIPTION
IBlockQueryAdapter	adapter	Source query structure adapter
System.Linq.Expressions.Expression<System.Func< IBlockQueryAdapter , T>>	expression	Case expression
System.Linq.Expressions.Expression<System.Func< IBlockQueryAdapter , TResult>>	elseValue	Else value
TaggedExpression <T, TResult>[]	switchValues	Case switch values

Returns

TYPE	DESCRIPTION

TYPE	DESCRIPTION
TResult	Case result

Type Parameters

NAME	DESCRIPTION
T	SQL case source data mapped type
TResult	SQL case result mapped type

Exceptions

TYPE	CONDITION
XDataRuntimeException	Function accessible in XData LINQ expressions only

DateDiff<T>(IBlockQueryAdapter, Expression<Func<IBlockQueryAdapter, DateTime>>, Expression<Func<IBlockQueryAdapter, DateTime>>, DateDiffKind)

Dates difference

Declaration

```
public static T DateDiff<T>(this IBlockQueryAdapter adapter, Expression<Func<IBlockQueryAdapter, DateTime>>
finalDate, Expression<Func<IBlockQueryAdapter, DateTime>> startDate, DateDiffKind kind)
```

Parameters

TYPE	NAME	DESCRIPTION
IBlockQueryAdapter	adapter	Source query structure adapter
System.Linq.Expressions.Expression<System.Func< IBlockQueryAdapter , System.DateTime>>	finalDate	Final date
System.Linq.Expressions.Expression<System.Func< IBlockQueryAdapter , System.DateTime>>	startDate	Start date
DateDiffKind	kind	Dates differences kind

Returns

TYPE	DESCRIPTION
T	Dates difference

Type Parameters

NAME	DESCRIPTION
T	SQL function DateDiff result data mapped type

Exceptions

TYPE	CONDITION
XDataRuntimeException	Function accessible in XData LINQ expressions only

ExecuteSqlBlock(IDataScope, Expression<SqlBlockDefinition>, Expression<Func<ISqlParamAdapter, ISqlParam>>[])

Execute SQL block

Declaration

```
public static QueryResult ExecuteSqlBlock(this IDataScope dataScope, Expression<SqlBlockDefinition> block,
params Expression<Func<ISqlParamAdapter, ISqlParam>>[] parameters)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataScope	dataScope	Data scope
System.Linq.Expressions.Expression< SqlBlockDefinition >	block	Sql block structure
System.Linq.Expressions.Expression<System.Func< ISqlParamAdapter , ISqlParam >>[]	parameters	Sql block parameters

Returns

TYPE	DESCRIPTION
QueryResult	SQL block executed over database

Exists(IBlockQueryAdapter, String)

Exists

Declaration

```
public static bool Exists(this IBlockQueryAdapter adapter, string subQueryAlias)
```

Parameters

TYPE	NAME	DESCRIPTION
IBlockQueryAdapter	adapter	Source query structure adapter

TYPE	NAME	DESCRIPTION
System.String	subQueryAlias	Subquery alias

Returns

TYPE	DESCRIPTION
System.Boolean	

Exceptions

TYPE	CONDITION
XDataRuntimeException	Function accessible in XData LINQ expressions only

GetSqlBlockData<T>(IDataScope, Expression<SqlBlockDefinition<T>>, Expression<Func<ISqlParamAdapter, ISqlParam>>[])

Select SQL block data

Declaration

```
public static IEnumerable<T> GetSqlBlockData<T>(this IDataScope dataScope, Expression<SqlBlockDefinition<T>>
block, params Expression<Func<ISqlParamAdapter, ISqlParam>>[] parameters)
    where T : class, new()
```

Parameters

TYPE	NAME	DESCRIPTION
IDataScope	dataScope	Data scope layer
System.Linq.Expressions.Expression< SqlBlockDefinition <T>>	block	Sql block structure
System.Linq.Expressions.Expression<System.Func< ISqlParamAdapter , ISqlParam >>[]	parameters	Sql block parameters

Returns

TYPE	DESCRIPTION
System.Collections.Generic.IEnumerable<T>	SQL block data selected and mapped

Type Parameters

NAME	DESCRIPTION
T	SQL block result mapped type

HasFlag<T>(IBlockQueryAdapter, Expression<Func<IBlockQueryAdapter, T>>, Expression<Func<IBlockQueryAdapter, T>>, Expression<Func<IBlockQueryAdapter, T>>, Expression<Func<IBlockQueryAdapter, T>>, Expression<Func<IBlockQueryAdapter, T>>)

Bit mask has flag

Declaration

```
public static T HasFlag<T>(this IBlockQueryAdapter adapter, Expression<Func<IBlockQueryAdapter, T>> expression, Expression<Func<IBlockQueryAdapter, T>> flagValue, Expression<Func<IBlockQueryAdapter, T>> trueValue, Expression<Func<IBlockQueryAdapter, T>> falseValue)
```

Parameters

TYPE	NAME	DESCRIPTION
IBlockQueryAdapter	adapter	Source query structure adapter
System.Linq.Expressions.Expression<System.Func< IBlockQueryAdapter , T>>	expression	Expression to test
System.Linq.Expressions.Expression<System.Func< IBlockQueryAdapter , T>>	flagValue	Flag value
System.Linq.Expressions.Expression<System.Func< IBlockQueryAdapter , T>>	trueValue	Result if flag exists
System.Linq.Expressions.Expression<System.Func< IBlockQueryAdapter , T>>	falseValue	Result if flag not exists

Returns

TYPE	DESCRIPTION
T	Bit mask has flag than trueValue else falseValue

Type Parameters

NAME	DESCRIPTION
T	SQL bitmask flag test result data mapped type

Exceptions

TYPE	CONDITION
XDataRuntimeException	Function accessible in XData LINQ expressions only

SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)

Set expression to tag

Declaration

```
public static TaggedExpression<TTag, TResult> SetExpression<TTag, TResult>(this TTag key,
Expression<Func<IBlockQueryAdapter, TResult>> expression)
```

Parameters

TYPE	NAME	DESCRIPTION
TTag	key	Tag value
System.Linq.Expressions.Expression<System.Func< IBlockQueryAdapter , TResult>>	expression	Expression

Returns

TYPE	DESCRIPTION
TaggedExpression <TTag, TResult>	Tagged expression

Type Parameters

NAME	DESCRIPTION
TTag	Tag type
TResult	Expression result

SqlExpr<TResult>(IBlockQueryAdapter, String, Expression<Func<IBlockQueryAdapter, TResult>>[])

Plain SQL expression with XData LINQ expression placeholders

Declaration

```
public static TResult SqlExpr<TResult>(this IBlockQueryAdapter adapter, string format, params
Expression<Func<IBlockQueryAdapter, TResult>>[] expressions)
```

Parameters

TYPE	NAME	DESCRIPTION
IBlockQueryAdapter	adapter	Source query structure adapter
System.String	format	SQL expression body in System.String.Format(String, Object[]) style
System.Linq.Expressions.Expression<System.Func< IBlockQueryAdapter , TResult>>[]	expressions	XData LINQ parameter expressions

Returns

TYPE	DESCRIPTION
TResult	SQL expression result

Type Parameters

NAME	DESCRIPTION
TResult	SQL expression result mapped type

Remarks

WARNING! Plain SQL expressions can use specific SQL dialect features!

Data mapping may be not compatible with other database engines!

Exceptions

TYPE	CONDITION
XDataRuntimeException	Function accessible in XData LINQ expressions only

SqlFn<TResult>(IBlockQueryAdapter, String, Expression<Func<IBlockQueryAdapter, Object>>[])

SQL function call

Declaration

```
public static TResult SqlFn<TResult>(this IBlockQueryAdapter adapter, string name, params
Expression<Func<IBlockQueryAdapter, object>>[] parameters)
```

Parameters

TYPE	NAME	DESCRIPTION
IBlockQueryAdapter	adapter	Source query structure adapter
System.String	name	SQL function full qualified name
System.Linq.Expressions.Expression<System.Func< IBlockQueryAdapter , System.Object>>[]	parameters	Function parameters descriptions

Returns

TYPE	DESCRIPTION
TResult	Function call result

Type Parameters

NAME	DESCRIPTION
TResult	SQL function result mapped type

Exceptions

TYPE	CONDITION
XDataRuntimeException	Function accessible in XData LINQ expressions only

SqlParam<TResult>(IBlockQueryAdapter, String, DbType, Expression<Func<IProcedureParameterAdapter, IProcedureParameterFlag>>[])

SQL parameter definition

Declaration

```
public static TResult SqlParam<TResult>(this IBlockQueryAdapter adapter, string binding, DbType type, params Expression<Func<IProcedureParameterAdapter, IProcedureParameterFlag>>[] attributes)
```

Parameters

TYPE	NAME	DESCRIPTION
IBlockQueryAdapter	adapter	Query structure adapter
System.String	binding	Parameter data binding name
System.Data.DbType	type	Parameter database type
System.Linq.Expressions.Expression<System.Func< IProcedureParameterAdapter , IProcedureParameterFlag >> []	attributes	Parameter mapping attributes

Returns

TYPE	DESCRIPTION
TResult	Parameter

Type Parameters

NAME	DESCRIPTION
TResult	Parameter mapped type

Exceptions

TYPE	CONDITION
XDataRuntimeException	Function accessible in XData LINQ expressions only

Class SqlType

SQL block variable/constant/parameter type description

Inheritance

System.Object
SqlType

Inherited Members

System.Object.Equals(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.GetHashCode()
System.Object.GetType()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.ToString()

Namespace: [XData.Database.SqlBlock](#)

Assembly: XData.docfx.dll

Syntax

```
public class SqlType
```

Properties

DbType

Database type

Declaration

```
public DbType DbType { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Data.DbType	

Length

Value type length

Declaration

```
public int Length { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Int32	

NativeSqlType

Value type native name

Declaration

```
public string NativeSqlType { get; set; }
```


Property Value

TYPE	DESCRIPTION
System.String	

Scale

Value type scale

Declaration

```
public int Scale { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Int32	

Extension Methods

- SerializationExtensions.ToXml<T>(T)
- CollectionsExtensions.SetValue<TKey, T>(TKey, T)
- CollectionsExtensions.AsEnum<T>(T)
- DataObjectQueryableExtensions.CompareVariable(Object, FilterOperation, String)
- Processing.Do<T>(T, Action<IProcess<T>>[])
- TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>)
- TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>, TResult)
- TypeExtensions.ChangeType<T>(Object)
- TypeExtensions.ChangeType(Object, Type)
- TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)
- TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)
- TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)
- TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)
- SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)
- QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Namespace XData.DataSource.Structure

Classes

[DataQueryState](#)

Data query state

[DataStructureState](#)

Data structure state

[Field](#)

Field object

[FilterCombinationState](#)

Filter combination state

[FilterName](#)

FilterName helper class to initialize named [RuntimeFilter](#)

[FilterState](#)

Filter state info

[Grouping](#)

Grouping element

[Hierarchy](#)

Hierarchy

[Order](#)

Order by element

[RepositoryState](#)

Repository state

[RuntimeFilter](#)

Runtime value filter

Structs

[SelectedField](#)

Selected field description

[TaggedValue](#)

Named value

Interfaces

[IConstantFilter](#)

Constant value filter interface

[IDataSource](#)

Data object source

[IDataStructure](#)

Data object structure

[IDictionaryFilter](#)

Dictionary filter interface

[IExpressionFilter](#)

Expression filter interface

[IExternalLink](#)

External link interface

[IField](#)

Field interface

[IFilter](#)

Filter interface

[IFilterCombination](#)

Filter combination interface

[ILink](#)

Link interface

[IInnerViewLink](#)

Inner view link interface

[INamedFilter](#)

Named filter interface

[IOptionalFilter](#)

Optional filter interface

Hint: Optional filters can rise flag to skip their sources if no value assigned in runtime or their subquery is skipped by other optional filters

[IProcedure](#)

Procedure based structure

[IRangeFilter](#)

Range filter interface

[IRuntimeFilter](#)

Runtime filter interface

[ISqlPart](#)

Subquery base interface

[ISubqueryFilter](#)

Subquery filter interface

[ISubqueryLink](#)

Subquery link interface

[I Table](#)

Table

[I TableLike](#)

Temporary table

[I TempTable](#)

[IXmlSource](#)

XML data source

Class DataQueryState

Data query state

Inheritance

System.Object

DataQueryState

[RepositoryState](#)

Inherited Members

System.Object.Equals(System.Object)

System.Object.Equals(System.Object, System.Object)

System.Object.GetHashCode()

System.Object.GetType()

System.Object.MemberwiseClone()

System.Object.ReferenceEquals(System.Object, System.Object)

Namespace: [XData.DataSource.Structure](#)

Assembly: XData.docfx.dll

Syntax

```
[Serializable]
[DataContract]
public class DataQueryState
```

Properties

AssemblyName

Element type assembly name

Declaration

```
[DataMember]
public string AssemblyName { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Context

Context

Declaration

```
[DataMember]
public string Context { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

ElementType

Element type name

Declaration

```
[DataMember]
public string ElementType { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Structure

DataSetStructure

Declaration

```
[DataMember]
public DataSetStructureState Structure { get; set; }
```

Property Value

TYPE	DESCRIPTION
DataSetStructureState	

Methods

ToString()

Returns a System.String that represents the current System.Object.

Declaration

```
public override string ToString()
```

Returns

TYPE	DESCRIPTION
System.String	A System.String that represents the current System.Object.

Overrides

System.Object.ToString()

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)

TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)

SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)

QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Class DataStructureState

Data structure state

Inheritance

System.Object

DataStructureState

Implements

System.IEquatable<[DataStructureState](#)>

Inherited Members

System.Object.Equals(System.Object)

System.Object.Equals(System.Object, System.Object)

System.Object.GetHashCode()

System.Object.GetType()

System.Object.MemberwiseClone()

System.Object.ReferenceEquals(System.Object, System.Object)

System.Object.ToString()

Namespace: [XData.DataSource.Structure](#)

Assembly: XData.docfx.dll

Syntax

```
[Serializable]
[DataContract]
public sealed class DataStructureState : IEquatable<DataStructureState>
```

Properties

Alias

Subquery alias

Declaration

```
[DataMember]
public string Alias { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Context

Structure context

Declaration

```
[DataMember]
public string Context { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Distinct

Distinct flag

Declaration

```
[DataMember]
public bool Distinct { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

Filters

Filter combinations

Declaration

```
[DataMember]
public FilterCombinationState Filters { get; set; }
```

Property Value

TYPE	DESCRIPTION
FilterCombinationState	

FilterValues

Filter values

Declaration

```
[DataMember]
public TaggedValue[] FilterValues { get; set; }
```

Property Value

TYPE	DESCRIPTION
TaggedValue[]	

Grouping

Grouping

Declaration

```
[DataMember]
public Grouping[] Grouping { get; set; }
```

Property Value

TYPE	DESCRIPTION
Grouping[]	

IsReadOnly

Read only flag

Declaration

```
[DataMember]
public bool IsReadOnly { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

Limit

Limit count

Declaration

```
[DataMember]
public int Limit { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Int32	

Name

Table name

Declaration

```
[DataMember]
public string Name { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

OrderBy

Order

Declaration

```
[DataMember]
public Order[] OrderBy { get; set; }
```

Property Value

TYPE	DESCRIPTION
Order []	

Reverse

Reverse order flag

Declaration

```
[DataMember]
public bool Reverse { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

Skip

Skip count

Declaration

```
[DataMember]
public int Skip { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Int32	

Sources

Dynamic sources states

Declaration

```
[DataMember]
public DataStructureState[] Sources { get; set; }
```

Property Value

TYPE	DESCRIPTION
DataStructureState[]	

SourceTypeType

Source type

Declaration

```
[DataMember]
public DataSourceType SourceTypeType { get; set; }
```

Property Value

TYPE	DESCRIPTION
DataSourceType	

StructureType

Structure type name

Declaration

```
[DataMember]
public string StructureType { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Union

Union combination

Declaration

```
[DataMember]
public DataCombination Union { get; set; }
```

Property Value

TYPE	DESCRIPTION
DataCombination	

Variables

Variables

Declaration

```
[DataMember]
public TaggedValue[] Variables { get; set; }
```

Property Value

TYPE	DESCRIPTION
TaggedValue[]	

Methods

Equals(DataStructureState)

DataStructureState equality check

Declaration

```
public bool Equals(DataStructureState other)
```

Parameters

TYPE	NAME	DESCRIPTION
DataStructureState	other	DataStructureState instance to compare

Returns

TYPE	DESCRIPTION
System.Boolean	Instances are contains the same data

Implements

System.IEquatable<T>

Extension Methods

- SerializationExtensions.ToXml<T>(T)
- CollectionsExtensions.SetValue<TKey, T>(TKey, T)
- CollectionsExtensions.AsEnum<T>(T)
- DataObjectQueryableExtensions.CompareVariable(Object, FilterOperation, String)
- Processing.Do<T>(T, Action<IProcess<T>>[])
- TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>)
- TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>, TResult)
- TypeExtensions.ChangeType<T>(Object)
- TypeExtensions.ChangeType(Object, Type)
- TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)
- TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)
- TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)
- TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)
- SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)
- QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Class Field

Field object

Inheritance

System.Object
Field

Implements

[IField](#)
System.ICloneable

Inherited Members

System.Object.Equals(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.GetHashCode()
System.Object.GetType()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.ToString()

Namespace: [XData.DataSource.Structure](#)

Assembly: XData.docfx.dll

Syntax

```
[Serializable]  
[DataContract]  
public class Field : IField, ICloneable
```

Properties

DefaultFeature

Always use the default value (also when updating)

Declaration

```
[DataMember]  
public DefaultFeature DefaultFeature { get; set; }
```

Property Value

TYPE	DESCRIPTION
DefaultFeature	

DefaultType

Default source

Declaration

```
[DataMember]  
public DefaultType DefaultType { get; set; }
```

Property Value

TYPE	DESCRIPTION
DefaultType	

DefaultValue

Default value

Declaration

```
[DataMember]
public object DefaultValue { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Object	

EnumValues

Enumeration values

Declaration

```
[DataMember]
public KeyValuePair<long, string>[] EnumValues { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Collections.Generic.KeyValuePair<System.Int64, System.String>[]	

Exists

Exists operation

Declaration

```
[DataMember]
public ExistsOperation Exists { get; set; }
```

Property Value

TYPE	DESCRIPTION
ExistsOperation	

ExprScale

Expression scale

Declaration

```
[DataMember]
public int ExprScale { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Int32	

ExprSize

Expression size

Declaration

```
[DataMember]
public int ExprSize { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Int32	

ExprText

Expression text

Declaration

```
[DataMember]
public string ExprText { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

ExprType

Expression type

Declaration

```
[DataMember]
public DataExpressionType ExprType { get; set; }
```

Property Value

TYPE	DESCRIPTION
DataExpressionType	

FieldName

Field name

Declaration

```
[DataMember]
public string FieldName { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Flags

Flags

Declaration

```
[DataMember]
public DataPropertyFlag Flags { get; set; }
```

Property Value

TYPE	DESCRIPTION
DataPropertyFlag	

IsArray

User defined type (UDT) collection is VARARRAY for field

Declaration

```
public bool IsArray { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

IsAutoIncrement

Autoincrement field flag

Declaration

```
[DataMember]
public bool IsAutoIncrement { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

IsSelectable

Selectable flag

Declaration

```
[DataMember]
public bool IsSelectable { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

IsTablePk

Table primary key flag

Declaration

```
[DataMember]
public bool IsTablePk { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

Name

Property name

Declaration

```
[DataMember]
public string Name { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

NativeSqlType

Native SQL type name

Declaration

```
[DataMember]
public string NativeSqlType { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Remarks

Used to specify native SQL column type when conflicted with default type mapping

WARNING! May be not supported by database adapter!

SqlType

Sql type

Declaration

```
[DataMember]
public DbType SqlType { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Data.DbType	

UdtElementTypeName

User defined type (UDT) element type name for field

Declaration

```
[DataMember]
public string UdtElementTypeName { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

UdtMappedType

User defined type (UDT) mapped type for field

Declaration

```
[DataMember]
public Type UdtMappedType { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Type	

UdtTypeName

User defined type (UDT) name for field

Declaration

```
[DataMember]
public string UdtTypeName { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

UpdateWithExprText

Update source expression text

Declaration

```
[DataMember]
public string UpdateWithExprText { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

UpdateWithExprType

Update source expression type

Declaration

```
[DataMember]
public DataExpressionType UpdateWithExprType { get; set; }
```

Property Value

TYPE	DESCRIPTION
DataExpressionType	

XPath

XML field value XPath

Declaration

```
[DataMember]
public string XPath { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Remarks

Used to specify native SQL column type when conflicted with default type mapping

WARNING! May be not supported by database adapter!

Methods

Clone()

Creates a new object that is a copy of the current instance.

Declaration

```
public object Clone()
```

Returns

TYPE	DESCRIPTION
System.Object	A new object that is a copy of this instance.

Implements

[IField](#)

System.ICloneable

Extension Methods

[SerializationExtensions.ToXml<T>\(T\)](#)

[CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)

[CollectionsExtensions.AsEnum<T>\(T\)](#)

[DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)

[Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)

TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>, TResult)

TypeExtensions.ChangeType<T>(Object)

TypeExtensions.ChangeType(Object, Type)

TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)

TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)

SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)

QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Class FilterCombinationState

Filter combination state

Inheritance

System.Object
FilterCombinationState

Implements

System.IEquatable<[FilterCombinationState](#)>

Inherited Members

System.Object.Equals(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.GetHashCode()
System.Object.GetType()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.ToString()

Namespace: [XData.DataSource.Structure](#)
Assembly: XData.docfx.dll

Syntax

```
[Serializable]  
[DataContract]  
public sealed class FilterCombinationState : IEquatable<FilterCombinationState>
```

Properties

Combination

Combination

Declaration

```
[DataMember]  
public Combination Combination { get; set; }
```

Property Value

TYPE	DESCRIPTION
Combination	

FilterCombinations

Nested filter combinations

Declaration

```
[DataMember]  
public FilterCombinationState[] FilterCombinations { get; set; }
```

Property Value

TYPE	DESCRIPTION
FilterCombinationState []	

Filters

Filter states

Declaration

```
[DataMember]
public FilterState[] Filters { get; set; }
```

Property Value

TYPE	DESCRIPTION
FilterState[]	

Methods

Equals(FilterCombinationState)

FilterCombinationState equality check

Declaration

```
public bool Equals(FilterCombinationState other)
```

Parameters

TYPE	NAME	DESCRIPTION
FilterCombinationState	other	FilterCombinationState instance to compare

Returns

TYPE	DESCRIPTION
System.Boolean	Instances are contains the same data

Implements

System.IEquatable<T>

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)

```
QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)
```


Class FilterName

FilterName helper class to initialize named [RuntimeFilter](#)

Inheritance

System.Object
FilterName

Inherited Members

System.Object.Equals(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.GetHashCode()
System.Object.GetType()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.ToString()

Namespace: [XData.DataSource.Structure](#)

Assembly: XData.docfx.dll

Syntax

```
public class FilterName
```

Constructors

FilterName(String)

FilterName constructor

Declaration

```
public FilterName(string name)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	

Properties

Name

Filter name

Declaration

```
public string Name { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Extension Methods

[SerializationExtensions.ToXml<T>\(T\)](#)
[CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
[CollectionsExtensions.AsEnum<T>\(T\)](#)

DataObjectQueryableExtensions.CompareVariable(Object, FilterOperation, String)
Processing.Do<T>(T, Action<IProcess<T>>[])
TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>)
TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>, TResult)
TypeExtensions.ChangeType<T>(Object)
TypeExtensions.ChangeType(Object, Type)
TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)
TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)
TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)
TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)
SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)
QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Class FilterState

Filter state info

Inheritance

System.Object
FilterState

Implements

[ISubqueryLink](#)
[IInnerViewLink](#)
[IInnerLink](#)
[IDictionaryFilter](#)
[ISubqueryFilter](#)
[IExpressionFilter](#)
[IRangeFilter](#)
[IConstantFilter](#)
[IRuntimeFilter](#)
[INamedFilter](#)
[IOptionalFilter](#)
[IFilter](#)
System.ICloneable
System.IEquatable<[FilterState](#)>

Inherited Members

System.Object.Equals(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.GetHashCode()
System.Object.GetType()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.ToString()

Namespace: [XData.DataSource.Structure](#)
Assembly: XData.docfx.dll

Syntax

```
[Serializable]
[DataContract]
public sealed class FilterState : ISubqueryLink, IInnerViewLink, IInnerLink, IDictionaryFilter,
ISubqueryFilter, IExpressionFilter, IRangeFilter, IConstantFilter, IRuntimeFilter, INamedFilter,
IOptionalFilter, IFilter, ICloneable, IEquatable<FilterState>
```

Properties

Combination

Parent filter combination name

Declaration

```
[DataMember]
public string Combination { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

ConstantType

Constant type

Declaration

```
[DataMember]  
public FilterConstantType ConstantType { get; set; }
```

Property Value

TYPE	DESCRIPTION
FilterConstantType	

ConstantValue

Filter constant value

Declaration

```
[DataMember]  
public object ConstantValue { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Object	

DictionaryCode

Dictionary filter code column name

Declaration

```
[DataMember]  
public string DictionaryCode { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

DictionaryId

Dictionary filter id column name

Declaration

```
[DataMember]  
public string DictionaryId { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

DictionaryTableName

Dictionary table name

Declaration

```
[DataMember]  
public string DictionaryTableName { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

DictionaryValue

Dictionary value

Declaration

```
[DataMember]  
public string DictionaryValue { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Dynamic

Filter dynamic flag

Declaration

```
[DataMember]  
public bool Dynamic { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

ExprText

Expression filter text

Declaration

```
[DataMember]  
public string ExprText { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Field

Filter field name

Declaration

```
[DataMember]  
public string Field { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

FilterType

Filter type

Declaration

```
[DataMember]  
public FilterType FilterType { get; set; }
```

Property Value

TYPE	DESCRIPTION
FilterType	

IsVariableValue

Dictionary filter value is variable flag

Declaration

```
[DataMember]  
public bool IsVariableValue { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

IteratorFilter

Iterator filter flag

Declaration

```
[DataMember]  
public bool IteratorFilter { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

LinkedAlias

Linked subquery source alias

Declaration

```
[DataMember]
public string LinkedAlias { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

LinkedField

Linked field name

Declaration

```
[DataMember]
public string LinkedField { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

LinkedPropertyName

Outer link linked property name

Declaration

```
[DataMember]
public string LinkedPropertyName { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

LinkedSourceAlias

Linked source alias

Declaration

```
[DataMember]
public string LinkedSourceAlias { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

LinkedType

Linked subquery type name

Declaration

```
[DataMember]  
public string LinkedType { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Name

Filter name

Declaration

```
[DataMember]  
public string Name { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Nullable

Filter null value handling type

Declaration

```
[DataMember]  
public FilterNullable Nullable { get; set; }
```

Property Value

TYPE	DESCRIPTION
FilterNullable	

Operation

Filter operation

Declaration

```
[DataMember]  
public FilterOperation Operation { get; set; }
```

Property Value

TYPE	DESCRIPTION
FilterOperation	

Primary

Primary filter flag

Declaration

```
[DataMember]  
public bool Primary { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

PropertyName

Outer link property name

Declaration

```
[DataMember]  
public string PropertyName { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

RangeValues

Range filter values

Declaration

```
[DataMember]  
public object[] RangeValues { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Object[]	

SourceAlias

Filter field source alias

Declaration

```
[DataMember]  
public string SourceAlias { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

SubqueryAlias

Filter subquery alias

Declaration

```
[DataMember]  
public string SubqueryAlias { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Methods

Clone()

Clone filter state

Declaration

```
public object Clone()
```

Returns

TYPE	DESCRIPTION
System.Object	Empty filter state

Equals(FilterState)

FilterState equality check

Declaration

```
public bool Equals(FilterState other)
```

Parameters

TYPE	NAME	DESCRIPTION
FilterState	other	FilterState instance to compare

Returns

TYPE	DESCRIPTION
System.Boolean	Instances are contains the same data

GetAliases()

Link aliases

Declaration

```
public string[] GetAliases()
```

Returns

TYPE	DESCRIPTION
System.String[]	

GetCaption()

Filter caption

Declaration

```
public string GetCaption()
```

Returns

TYPE	DESCRIPTION
System.String	

GetIsLink()

Filter link flag

Declaration

```
public bool GetIsLink()
```

Returns

TYPE	DESCRIPTION
System.Boolean	

Implements

[ISubqueryLink](#)

[IInnerViewLink](#)

[IInnerLink](#)

[IDictionaryFilter](#)

[ISubqueryFilter](#)

[IExpressionFilter](#)

[IRangeFilter](#)

[IConstantFilter](#)

[IRuntimeFilter](#)

[INamedFilter](#)

[IOptionalFilter](#)

[IFilter](#)

[System.ICloneable](#)

[System.IEquatable<T>](#)

Extension Methods

[SerializationExtensions.ToXml<T>\(T\)](#)

CollectionsExtensions.SetValue<TKey, T>(TKey, T)
CollectionsExtensions.AsEnum<T>(T)
DataObjectQueryableExtensions.CompareVariable(Object, FilterOperation, String)
Processing.Do<T>(T, Action<IProcess<T>>[])
TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>)
TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>, TResult)
TypeExtensions.ChangeType<T>(Object)
TypeExtensions.ChangeType(Object, Type)
TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)
TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)
TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)
TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)
SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)
QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Class Grouping

Grouping element

Inheritance

System.Object

Grouping

Implements

System.IEquatable<[Grouping](#)>

Inherited Members

System.Object.Equals(System.Object)

System.Object.Equals(System.Object, System.Object)

System.Object.GetHashCode()

System.Object.GetType()

System.Object.MemberwiseClone()

System.Object.ReferenceEquals(System.Object, System.Object)

System.Object.ToString()

Namespace: [XData.DataSource.Structure](#)

Assembly: XData.docfx.dll

Syntax

```
[Serializable]
[DataContract]
public sealed class Grouping : IEquatable<Grouping>
```

Properties

FieldName

Field name

Declaration

```
[DataMember]
public string FieldName { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

GroupFunction

Group function

Declaration

```
[DataMember]
public DataGrouping GroupFunction { get; set; }
```

Property Value

TYPE	DESCRIPTION
DataGrouping	

GroupOrder

Group function

Declaration

```
[DataMember]
public int GroupOrder { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Int32	

PropertyName

Property name

Declaration

```
[DataMember]
public string PropertyName { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

PropertyOwnerType

Property owner type name

Declaration

```
[DataMember]
public string PropertyOwnerType { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

SourceAlias

Source alias

Declaration

```
[DataMember]
public string SourceAlias { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Methods

Equals(Grouping)

Grouping equality check

Declaration

```
public bool Equals(Grouping other)
```

Parameters

TYPE	NAME	DESCRIPTION
Grouping	other	Grouping instance to compare

Returns

TYPE	DESCRIPTION
System.Boolean	Instances are contains the same data

Implements

System.IEquatable<T>

Extension Methods

- SerializationExtensions.ToXml<T>(T)
- CollectionsExtensions.SetValue<TKey, T>(TKey, T)
- CollectionsExtensions.AsEnum<T>(T)
- DataObjectQueryableExtensions.CompareVariable(Object, FilterOperation, String)
- Processing.Do<T>(T, Action<IProcess<T>>[])
- TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>)
- TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>, TResult)
- TypeExtensions.ChangeType<T>(Object)
- TypeExtensions.ChangeType(Object, Type)
- TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)
- TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)
- TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)
- TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)
- SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)
- QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Class Hierarchy

Hierarchy

Inheritance

System.Object

Hierarchy

Inherited Members

- System.Object.Equals(System.Object)
- System.Object.Equals(System.Object, System.Object)
- System.Object.GetHashCode()
- System.Object.GetType()
- System.Object.MemberwiseClone()
- System.Object.ReferenceEquals(System.Object, System.Object)
- System.Object.ToString()

Namespace: [XData.DataSource.Structure](#)

Assembly: XData.docfx.dll

Syntax

```
public sealed class Hierarchy
```

Constructors

Hierarchy(String, String, String, String)

Hierarchy constructor

Declaration

```
public Hierarchy(string name, string parent, string linkParent, string linkChild)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Hierarchy table name
System.String	parent	Parent link field
System.String	linkParent	Hierarchy table parent link
System.String	linkChild	Hierarchy table child link

Properties

LinkChild

Hierarchy table child link

Declaration

```
public string LinkChild { get; }
```


Property Value

TYPE	DESCRIPTION
System.String	

LinkParent

Hierarchy table parent link

Declaration

```
public string LinkParent { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Name

Hierarchy table name

Declaration

```
public string Name { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Parent

Parent link field

Declaration

```
public string Parent { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Extension Methods

[SerializationExtensions.ToXml<T>\(T\)](#)

[CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)

[CollectionsExtensions.AsEnum<T>\(T\)](#)

[DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)

[Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)

[TypeExtensions.ChangeType<T>\(Object\)](#)

[TypeExtensions.ChangeType\(Object, Type\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)

TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)

TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)

SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)

QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Interface IConstantFilter

Constant value filter interface

Inherited Members

[INamedFilter.Name](#)

[IFilter.FilterType](#)

[IFilter.SourceAlias](#)

[IFilter.Field](#)

[IFilter.Operation](#)

[System.ICloneable.Clone\(\)](#)

Namespace: [XData.DataSource.Structure](#)

Assembly: XData.docfx.dll

Syntax

```
public interface IConstantFilter : INamedFilter, IFilter, ICloneable
```

Properties

ConstantType

Constant value type

Declaration

```
FilterConstantType ConstantType { get; }
```

Property Value

TYPE	DESCRIPTION
FilterConstantType	

ConstantValue

Constant value

Declaration

```
object ConstantValue { get; }
```

Property Value

TYPE	DESCRIPTION
System.Object	

Nullable

Constant value null value handling mode

Declaration

```
FilterNullable Nullable { get; }
```

Property Value

TYPE	DESCRIPTION
FilterNullable	

Extension Methods

[SerializationExtensions.ToXml<T>\(T\)](#)
[CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
[CollectionsExtensions.AsEnum<T>\(T\)](#)
[DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
[Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
[TypeExtensions.ChangeType<T>\(Object\)](#)
[TypeExtensions.ChangeType\(Object, Type\)](#)
[TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
[TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
[TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
[TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
[SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
[QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Interface IDataSource

Data object source

Inherited Members

System.ICloneable.Clone()

Namespace: [XData.DataSource.Structure](#)

Assembly: XData.docfx.dll

Syntax

```
public interface IDataSource : ICloneable
```

Properties

Alias

Source alias

Declaration

```
string Alias { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

IsBase

Base table flag

Declaration

```
bool IsBase { get; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

SourceType

Source type

Declaration

```
DataSourceType SourceType { get; }
```

Property Value

TYPE	DESCRIPTION
DataSourceType	

Methods

GetField(String, String, out String, out String)

Returns property field

Declaration

```
IField GetField(string sourceAlias, string fieldName, out string alias, out string fullName)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	sourceAlias	Source alias
System.String	fieldName	Field name
System.String	alias	Relative source alias
System.String	fullName	Relative field full name

Returns

TYPE	DESCRIPTION
IField	Field object

GetField(String, out String, out String)

Returns property field

Declaration

```
IField GetField(string propertyName, out string alias, out string fieldName)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	propertyName	Property name
System.String	alias	Relative source alias
System.String	fieldName	Relative field full name

Returns

TYPE	DESCRIPTION
IField	Field object

GetFields()

Returns selected fields

Declaration

```
IEnumerable<SelectedField> GetFields()
```

Returns

TYPE	DESCRIPTION
System.Collections.Generic.IEnumerable< SelectedField >	Selected fields list

GetFields(Type)

Returns selected fields of subquery with type

Declaration

```
IEnumerable<SelectedField> GetFields(Type type)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Type	type	

Returns

TYPE	DESCRIPTION
System.Collections.Generic.IEnumerable< SelectedField >	Selected fields list

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Interface IDataStructure

Data object structure

Namespace: [XData.DataSource.Structure](#)

Assembly: XData.docfx.dll

Syntax

```
public interface IDataStructure
```

Properties

BaseSource

Base source

Declaration

```
ITableLike BaseSource { get; }
```

Property Value

TYPE	DESCRIPTION
ITableLike	

Context

Data structure context

Declaration

```
string Context { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Distinct

Structure distinct values flag

Declaration

```
bool Distinct { get; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

FilterValues

Runtime filter values

Declaration

```
SortedList<string, object> FilterValues { get; }
```


Property Value

TYPE	DESCRIPTION
System.Collections.Generic.SortedList<System.String, System.Object>	

Flags

Data object flags

Declaration

DataStructureFlag Flags { get; }

Property Value

TYPE	DESCRIPTION
DataStructureFlag	

GroupBy

Group by list

Declaration

List<Grouping> GroupBy { get; }

Property Value

TYPE	DESCRIPTION
System.Collections.Generic.List< Grouping >	

Hint

Query hints

Declaration

HintInfo[] Hint { get; set; }

Property Value

TYPE	DESCRIPTION
HintInfo []	

Inverse

Structure filters inverse flag

Declaration

bool Inverse { get; }

Property Value

TYPE	DESCRIPTION
System.Boolean	

IsReadOnly

Read only structure flag

Declaration

```
bool IsReadOnly { get; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

IsSubquery

Subquery flag

Declaration

```
bool IsSubquery { get; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

IsUnion

Combined structure flag

Declaration

```
bool IsUnion { get; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

Limit

Limit objects count

Declaration

```
int? Limit { get; }
```

Property Value

TYPE	DESCRIPTION
System.Nullable<System.Int32>	

ObjectType

Structure data object type

Declaration

```
Type ObjectType { get; }
```

Property Value

TYPE	DESCRIPTION
System.Type	

OrderBy

Order by list

Declaration

```
List<Order> OrderBy { get; }
```

Property Value

TYPE	DESCRIPTION
System.Collections.Generic.List< Order >	

Parameters

Stored procedure parameters definitions

Declaration

```
List<IParameterDescription> Parameters { get; }
```

Property Value

TYPE	DESCRIPTION
System.Collections.Generic.List< IParameterDescription >	

ResultSets

Stored procedure result sets

Declaration

```
IResultSetDescription[] ResultSets { get; }
```

Property Value

TYPE	DESCRIPTION
IResultSetDescription []	

ReverseOrder

Structure reversion order flag

Declaration

```
bool ReverseOrder { get; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

RootFilterCombination

Structure root filter combination

Declaration

```
IFilterCombination RootFilterCombination { get; }
```

Property Value

TYPE	DESCRIPTION
IFilterCombination	

Skip

Skip objects count

Declaration

```
int? Skip { get; }
```

Property Value

TYPE	DESCRIPTION
System.Nullable<System.Int32>	

Sources

Sources

Declaration

```
SortedSet<IDataSource> Sources { get; }
```

Property Value

TYPE	DESCRIPTION
System.Collections.Generic.SortedSet< IDataSource >	

StructureType

Structure type full name

Declaration

```
string StructureType { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

UnionCombination

Data object structure combination

Declaration

```
DataCombination UnionCombination { get; }
```

Property Value

TYPE	DESCRIPTION
DataCombination	

Methods

FillSkippedSources(SortedList<String, Object>, out ICollection<String>)

Fill skipped sources list

Declaration

```
bool FillSkippedSources(SortedList<string, object> filterValues, out ICollection<string> skipped)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Collections.Generic.SortedList<System.String, System.Object>	filterValues	Filter values
System.Collections.Generic.ICollection<System.String>	skipped	Returned skipped sources list

Returns

TYPE	DESCRIPTION
System.Boolean	Structure skipped flag

GetField(String, String, out String, out String)

Returns field

Declaration

```
IField GetField(string sourceAlias, string fieldName, out string alias, out string fullName)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	sourceAlias	Source alias
System.String	fieldName	Field name

TYPE	NAME	DESCRIPTION
System.String	alias	Relative alias
System.String	fullName	Relative full name of field

Returns

TYPE	DESCRIPTION
IField	Field object

GetFields()

Returns selectable fields

Declaration

```
IEnumerable<SelectedField> GetFields()
```

Returns

TYPE	DESCRIPTION
System.Collections.Generic.IEnumerable< SelectedField >	Selectable fields sequence

GetParent(String)

Returns parent structure by object type

Declaration

```
IDataStructure GetParent(string parentType)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	parentType	Parent object type

Returns

TYPE	DESCRIPTION
IDataStructure	Structure

GetPrimaryKey()

Returns data object unique key fields

Declaration

```
IEnumerable<SelectedField> GetPrimaryKey()
```

Returns

TYPE	DESCRIPTION
System.Collections.Generic.IEnumerable< SelectedField >	Unique key fields sequence

GetSubqueryVariables(String)

Subquery variables

Declaration

```
SortedList<string, object> GetSubqueryVariables(string subqueryAlias)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	subqueryAlias	Subquery alias

Returns

TYPE	DESCRIPTION
System.Collections.Generic.SortedList<System.String, System.Object>	Subquery variables

GetVariableValue(String)

Returns variable value

Declaration

```
object GetVariableValue(string variableName)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	variableName	Variable name

Returns

TYPE	DESCRIPTION
System.Object	Variable value

IsOuterJoined(String, String)

Check source is outer joined to parent

Declaration

```
bool IsOuterJoined(string alias, string parentAlias)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	alias	Checked source alias
System.String	parentAlias	Parent source alias

Returns

TYPE	DESCRIPTION
System.Boolean	Outer joined flag

Extension Methods

[SerializationExtensions.ToXml<T>\(T\)](#)

[CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)

[CollectionsExtensions.AsEnum<T>\(T\)](#)

[DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)

[Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)

[TypeExtensions.ChangeType<T>\(Object\)](#)

[TypeExtensions.ChangeType\(Object, Type\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)

[SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)

[QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Interface IDictionaryFilter

Dictionary filter interface

Inherited Members

[IFilter.FilterType](#)

[IFilter.SourceAlias](#)

[IFilter.Field](#)

[IFilter.Operation](#)

[System.ICloneable.Clone\(\)](#)

Namespace: [XData.DataSource.Structure](#)

Assembly: XData.docfx.dll

Syntax

```
public interface IDictionaryFilter : IFilter, ICloneable
```

Properties

DictionaryCode

Dictionary code field

Declaration

```
string DictionaryCode { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

DictionaryId

Dictionary id field

Declaration

```
string DictionaryId { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

DictionaryTableName

Dictionary table name

Declaration

```
string DictionaryTableName { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

DictionaryValue

Dictionary search value

Declaration

```
string DictionaryValue { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

IsVariableValue

Dictionary search value is variable name flag

Declaration

```
bool IsVariableValue { get; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

Extension Methods

[SerializationExtensions.ToXml<T>\(T\)](#)

[CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)

[CollectionsExtensions.AsEnum<T>\(T\)](#)

[DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)

[Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)

[TypeExtensions.ChangeType<T>\(Object\)](#)

[TypeExtensions.ChangeType\(Object, Type\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)

[SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)

[QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Interface IExpressionFilter

Expression filter interface

Inherited Members

[IFilter.FilterType](#)

[IFilter.SourceAlias](#)

[IFilter.Field](#)

[IFilter.Operation](#)

[System.ICloneable.Clone\(\)](#)

Namespace: [XData.DataSource.Structure](#)

Assembly: XData.docfx.dll

Syntax

```
public interface IExpressionFilter : IFilter, ICloneable
```

Properties

ExprText

Expression text

Declaration

```
string ExprText { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

IteratorFilter

Iterator filter flag

Declaration

```
bool IteratorFilter { get; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

Extension Methods

[SerializationExtensions.ToXml<T>\(T\)](#)

[CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)

[CollectionsExtensions.AsEnum<T>\(T\)](#)

[DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)

[Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)

[TypeExtensions.ChangeType<T>\(Object\)](#)

[TypeExtensions.ChangeType\(Object, Type\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)

TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)

TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)

SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)

QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Interface IExternalLink

External link interface

Inherited Members

System.ICloneable.Clone()

Namespace: [XData.DataSource.Structure](#)

Assembly: XData.docfx.dll

Syntax

```
public interface IExternalLink : ICloneable
```

Properties

ChildLinkType

Child link type

Declaration

```
Type ChildLinkType { get; }
```

Property Value

TYPE	DESCRIPTION
System.Type	

ChildProperty

Child property

Declaration

```
string ChildProperty { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

ChildType

Child type

Declaration

```
Type ChildType { get; }
```

Property Value

TYPE	DESCRIPTION
System.Type	

Combination

Filter combination name

Declaration

```
string Combination { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

DirectLink

Direct link flag

Declaration

```
bool DirectLink { get; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

FilterName

Filter name

Declaration

```
string FilterName { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

MasterRefresh

Master refresh flag

Declaration

```
bool MasterRefresh { get; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

Nullable

Null value handling

Declaration

```
FilterNullable Nullable { get; }
```

Property Value

TYPE	DESCRIPTION
FilterNullable	

Operation

Filter operation

Declaration

```
FilterOperation Operation { get; }
```

Property Value

TYPE	DESCRIPTION
FilterOperation	

PrimaryFilter

Primary filter flag

Declaration

```
bool PrimaryFilter { get; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

PropertyName

Property name

Declaration

```
string PropertyName { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

TreeFilter

Tree filter flag

Declaration

```
bool TreeFilter { get; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

Extension Methods

SerializationExtensions.ToXml<T>(T)

CollectionsExtensions.SetValue<TKey, T>(TKey, T)

CollectionsExtensions.AsEnum<T>(T)

DataObjectQueryableExtensions.CompareVariable(Object, FilterOperation, String)

Processing.Do<T>(T, Action<IProcess<T>>[])

TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>)

TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>, TResult)

TypeExtensions.ChangeType<T>(Object)

TypeExtensions.ChangeType(Object, Type)

TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)

TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)

SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)

QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Interface IField

Field interface

Inherited Members

System.ICloneable.Clone()

Namespace: [XData.DataSource.Structure](#)

Assembly: XData.docfx.dll

Syntax

```
public interface IField : ICloneable
```

Properties

DefaultFeature

Default extended features

Declaration

```
DefaultFeature DefaultFeature { get; }
```

Property Value

TYPE	DESCRIPTION
DefaultFeature	

DefaultType

Default source

Declaration

```
DefaultType DefaultType { get; }
```

Property Value

TYPE	DESCRIPTION
DefaultType	

DefaultValue

Default value

Declaration

```
object DefaultValue { get; }
```

Property Value

TYPE	DESCRIPTION
System.Object	

EnumValues

Enumeration values

Declaration

```
KeyValuePair<long, string>[] EnumValues { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Collections.Generic.KeyValuePair<System.Int64, System.String>[]	

Exists

Exists operation

Declaration

```
ExistsOperation Exists { get; }
```

Property Value

TYPE	DESCRIPTION
ExistsOperation	

ExprScale

Expression scale

Declaration

```
int ExprScale { get; }
```

Property Value

TYPE	DESCRIPTION
System.Int32	

ExprSize

Expression size

Declaration

```
int ExprSize { get; }
```

Property Value

TYPE	DESCRIPTION
System.Int32	

ExprText

Expression text

Declaration

```
string ExprText { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

ExprType

Expression type

Declaration

```
DataExpressionType ExprType { get; }
```

Property Value

TYPE	DESCRIPTION
DataExpressionType	

FieldName

Field name

Declaration

```
string FieldName { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Flags

Flags

Declaration

```
DataPropertyFlag Flags { get; set; }
```

Property Value

TYPE	DESCRIPTION
DataPropertyFlag	

IsAutoIncrement

Auto increment field flag

Declaration

```
bool IsAutoIncrement { get; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

IsSelectable

Selectable flag

Declaration

```
bool IsSelectable { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

IsTablePk

Table primary key flag

Declaration

```
bool IsTablePk { get; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

Name

Property name

Declaration

```
string Name { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

NativeSqlType

Native SQL type name

Declaration

```
string NativeSqlType { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Remarks

Used to specify native SQL column type when conflicted with default type mapping

WARNING! May be not supported by database adapter!

SqlType

SQL type

Declaration

```
DbType SqlType { get; }
```

Property Value

TYPE	DESCRIPTION
System.Data.DbType	

UpdateWithExprText

Update source expression text

Declaration

```
string UpdateWithExprText { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

UpdateWithExprType

Update source expression type

Declaration

```
DataExpressionType UpdateWithExprType { get; }
```

Property Value

TYPE	DESCRIPTION
DataExpressionType	

XPath

XML property value XPath

Declaration

```
string XPath { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Extension Methods

[SerializationExtensions.ToXml<T>\(T\)](#)

[CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)

[CollectionsExtensions.AsEnum<T>\(T\)](#)

DataObjectQueryableExtensions.CompareVariable(Object, FilterOperation, String)
Processing.Do<T>(T, Action<IProcess<T>>[])
TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>)
TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>, TResult)
TypeExtensions.ChangeType<T>(Object)
TypeExtensions.ChangeType(Object, Type)
TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)
TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)
TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)
TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)
SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)
QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Interface IFilter

Filter interface

Inherited Members

System.ICloneable.Clone()

Namespace: [XData.DataSource.Structure](#)

Assembly: XData.docfx.dll

Syntax

```
public interface IFilter : ICloneable
```

Properties

Field

Field name

Declaration

```
string Field { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

FilterType

Filter type

Declaration

```
FilterType FilterType { get; }
```

Property Value

TYPE	DESCRIPTION
FilterType	

Operation

Filter operation

Declaration

```
FilterOperation Operation { get; }
```

Property Value

TYPE	DESCRIPTION
FilterOperation	

SourceAlias

Source alias

Declaration

```
string SourceAlias { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Extension Methods

- SerializationExtensions.ToXml<T>(T)
- CollectionsExtensions.SetValue<TKey, T>(TKey, T)
- CollectionsExtensions.AsEnum<T>(T)
- DataObjectQueryableExtensions.CompareVariable(Object, FilterOperation, String)
- Processing.Do<T>(T, Action<IProcess<T>>[])
- TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>)
- TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>, TResult)
- TypeExtensions.ChangeType<T>(Object)
- TypeExtensions.ChangeType(Object, Type)
- TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)
- TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)
- TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)
- TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)
- SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)
- QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Interface IFilterCombination

Filter combination interface

Inherited Members

System.ICloneable.Clone()

Namespace: [XData.DataSource.Structure](#)

Assembly: XData.docfx.dll

Syntax

```
public interface IFilterCombination : ICloneable
```

Properties

Combination

Logical combination

Declaration

```
Combination Combination { get; }
```

Property Value

TYPE	DESCRIPTION
Combination	

FilterCombinations

Child combinations

Declaration

```
List<IFilterCombination> FilterCombinations { get; }
```

Property Value

TYPE	DESCRIPTION
System.Collections.Generic.List< IFilterCombination >	

Filters

Filters

Declaration

```
List<IFilter> Filters { get; }
```

Property Value

TYPE	DESCRIPTION
System.Collections.Generic.List< IFilter >	

Name

Filter combination name

Declaration

```
string Name { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Methods

GetLinks(String)

Returns links

Declaration

```
IEnumerable<IInnerLink> GetLinks(string alias)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	alias	Source alias

Returns

TYPE	DESCRIPTION
System.Collections.Generic.IEnumerable< IInnerLink >	Links sequence

GetLinks(String[], Boolean)

Returns links

Declaration

```
IEnumerable<IInnerLink> GetLinks(string[] aliases, bool exact)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String[]	aliases	Array of source aliases
System.Boolean	exact	Exact comparison flag

Returns

TYPE	DESCRIPTION
System.Collections.Generic.IEnumerable< IInnerLink >	Links sequence

GetLinks(IDataStructure, String)

Returns links

Declaration

```
IEnumerable<IInnerLink> GetLinks(IDataStructure structure, string alias)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataStructure	structure	Structure
System.String	alias	Source alias

Returns

TYPE	DESCRIPTION
System.Collections.Generic.IEnumerable< IInnerLink >	Links sequence

GetPrimaryFilters()

Returns primary filters

Declaration

```
List<IOptionalFilter> GetPrimaryFilters()
```

Returns

TYPE	DESCRIPTION
System.Collections.Generic.List< IOptionalFilter >	Primary filters list

GetValueFilters()

Return value filters list

Declaration

```
IEnumerable<IFilter> GetValueFilters()
```

Returns

TYPE	DESCRIPTION
System.Collections.Generic.IEnumerable< IFilter >	Value filters list

Extension Methods

[SerializationExtensions.ToXml<T>\(T\)](#)

[CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)

CollectionsExtensions.AsEnum<T>(T)
DataObjectQueryableExtensions.CompareVariable(Object, FilterOperation, String)
Processing.Do<T>(T, Action<IProcess<T>>[])
TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>)
TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>, TResult)
TypeExtensions.ChangeType<T>(Object)
TypeExtensions.ChangeType(Object, Type)
TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)
TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)
TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)
TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)
SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)
QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Interface IInnerLink

Link interface

Inherited Members

[IOptionalFilter.Primary](#)

[IFilter.FilterType](#)

[IFilter.SourceAlias](#)

[IFilter.Field](#)

[IFilter.Operation](#)

[System.ICloneable.Clone\(\)](#)

Namespace: [XData.DataSource.Structure](#)

Assembly: XData.docfx.dll

Syntax

```
public interface IInnerLink : IOptionalFilter, IFilter, ICloneable
```

Properties

LinkedField

Linked field

Declaration

```
string LinkedField { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

LinkedSourceAlias

Linked source alias

Declaration

```
string LinkedSourceAlias { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Methods

GetAliases()

Link aliases

Declaration

```
string[] GetAliases()
```

Returns

TYPE	DESCRIPTION
System.String[]	

GetIsLink()

Link flag

Declaration

bool GetIsLink()

Returns

TYPE	DESCRIPTION
System.Boolean	

Extension Methods

- SerializationExtensions.ToXml<T>(T)
- CollectionsExtensions.SetValue<TKey, T>(TKey, T)
- CollectionsExtensions.AsEnum<T>(T)
- DataObjectQueryableExtensions.CompareVariable(Object, FilterOperation, String)
- Processing.Do<T>(T, Action<IProcess<T>>[])
- TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>)
- TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>, TResult)
- TypeExtensions.ChangeType<T>(Object)
- TypeExtensions.ChangeType(Object, Type)
- TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)
- TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)
- TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)
- TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)
- SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)
- QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Interface IInnerViewLink

Inner view link interface

Inherited Members

- [IInnerLink.LinkedSourceAlias](#)
- [IInnerLink.LinkedField](#)
- [IInnerLink.GetIsLink\(\)](#)
- [IInnerLink.GetAliases\(\)](#)
- [IOptionalFilter.Primary](#)
- [IFilter.FilterType](#)
- [IFilter.SourceAlias](#)
- [IFilter.Field](#)
- [IFilter.Operation](#)
- [System.ICloneable.Clone\(\)](#)

Namespace: [XData.DataSource.Structure](#)
Assembly: [XData.docfx.dll](#)

Syntax

```
public interface IInnerViewLink : IInnerLink, IOptionalFilter, IFilter, ICloneable
```

Properties

LinkedAlias

Linked source alias

Declaration

```
string LinkedAlias { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Interface INamedFilter

Named filter interface

Inherited Members

[IFilter.FilterType](#)

[IFilter.SourceAlias](#)

[IFilter.Field](#)

[IFilter.Operation](#)

[System.ICloneable.Clone\(\)](#)

Namespace: [XData.DataSource.Structure](#)

Assembly: XData.docfx.dll

Syntax

```
public interface INamedFilter : IFilter, ICloneable
```

Properties

Name

Filter name

Declaration

```
string Name { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Extension Methods

[SerializationExtensions.ToXml<T>\(T\)](#)

[CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)

[CollectionsExtensions.AsEnum<T>\(T\)](#)

[DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)

[Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)

[TypeExtensions.ChangeType<T>\(Object\)](#)

[TypeExtensions.ChangeType\(Object, Type\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)

[SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)

[QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Interface IOptionalFilter

Optional filter interface

Hint: Optional filters can rise flag to skip their sources if no value assigned in runtime or their subquery is skipped by other optional filters

Inherited Members

- [IFilter.FilterType](#)
- [IFilter.SourceAlias](#)
- [IFilter.Field](#)
- [IFilter.Operation](#)
- [System.ICloneable.Clone\(\)](#)

Namespace: [XData.DataSource.Structure](#)
Assembly: XData.docfx.dll

Syntax

```
public interface IOptionalFilter : IFilter, ICloneable
```

Properties

Primary

Primary filter flag

Declaration

```
bool Primary { get; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Interface IProcedure

Procedure based structure

Inherited Members

- [IDataSource.SourceType](#)
- [IDataSource.Alias](#)
- [IDataSource.IsBase](#)
- [IDataSource.GetFields\(\)](#)
- [IDataSource.GetFields\(Type\)](#)
- [IDataSource.GetField\(String, String, String\)](#)
- [IDataSource.GetField\(String, String, String, String\)](#)
- [System.ICloneable.Clone\(\)](#)

Namespace: [XData.DataSource.Structure](#)

Assembly: XData.docfx.dll

Syntax

```
public interface IProcedure : IDataSource, ICloneable
```

Properties

CallBased

SQL procedure/function is call based

Declaration

```
bool CallBased { get; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

DefaultResultSet

SQL stored procedure/function default result set name

Declaration

```
string DefaultResultSet { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Hint

SQL stored procedure/function hint

Declaration

```
HintInfo Hint { get; }
```

Property Value

TYPE	DESCRIPTION
HintInfo	

Name

SQL stored procedure/function full qualified name

Declaration

```
string Name { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Parameters

SQL stored procedure/function parameters descriptions

Declaration

```
IEnumerable<IParameterDescription> Parameters { get; }
```

Property Value

TYPE	DESCRIPTION
System.Collections.Generic.IEnumerable< IParameterDescription >	

ProcedureType

SQL stored procedure/function type

Declaration

```
ProcedureType ProcedureType { get; }
```

Property Value

TYPE	DESCRIPTION
ProcedureType	

See Also

[ProcedureType](#)

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)

TypeExtensions.ChangeType(Object, Type)

TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)

TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)

SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)

QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Interface IRangeFilter

Range filter interface

Inherited Members

[IFilter.FilterType](#)

[IFilter.SourceAlias](#)

[IFilter.Field](#)

[IFilter.Operation](#)

[System.ICloneable.Clone\(\)](#)

Namespace: [XData.DataSource.Structure](#)

Assembly: XData.docfx.dll

Syntax

```
public interface IRangeFilter : IFilter, ICloneable
```

Properties

RangeValues

Range values

Declaration

```
object[] RangeValues { get; }
```

Property Value

TYPE	DESCRIPTION
System.Object[]	

Extension Methods

[SerializationExtensions.ToXml<T>\(T\)](#)

[CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)

[CollectionsExtensions.AsEnum<T>\(T\)](#)

[DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)

[Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)

[TypeExtensions.ChangeType<T>\(Object\)](#)

[TypeExtensions.ChangeType\(Object, Type\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)

[SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)

[QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Interface IRuntimeFilter

Runtime filter interface

Inherited Members

- [INamedFilter.Name](#)
- [IOptionalFilter.Primary](#)
- [IFilter.FilterType](#)
- [IFilter.SourceAlias](#)
- [IFilter.Field](#)
- [IFilter.Operation](#)
- [System.ICloneable.Clone\(\)](#)

Namespace: [XData.DataSource.Structure](#)

Assembly: XData.docfx.dll

Syntax

```
public interface IRuntimeFilter : INamedFilter, IOptionalFilter, IFilter, ICloneable
```

Properties

Nullable

Null value handling

Declaration

```
FilterNullable Nullable { get; }
```

Property Value

TYPE	DESCRIPTION
FilterNullable	

Methods

GetCaption()

Filter caption

Declaration

```
string GetCaption()
```

Returns

TYPE	DESCRIPTION
System.String	

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)

TypeExtensions.ChangeType<T>(Object)

TypeExtensions.ChangeType(Object, Type)

TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)

TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)

SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)

QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Interface ISqlPart

Subquery base interface

Inherited Members

- [IDataSource.SourceType](#)
- [IDataSource.Alias](#)
- [IDataSource.IsBase](#)
- [IDataSource.GetFields\(\)](#)
- [IDataSource.GetFields\(Type\)](#)
- [IDataSource.GetField\(String, String, String\)](#)
- [IDataSource.GetField\(String, String, String, String\)](#)
- [System.ICloneable.Clone\(\)](#)

Namespace: [XData.DataSource.Structure](#)
Assembly: XData.docfx.dll

Syntax

```
public interface ISqlPart : IDataSource, ICloneable
```

Properties

Dynamic

Subquery dynamic flag

Declaration

```
bool Dynamic { get; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

ObjectType

Subquery object type name

Declaration

```
string ObjectType { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Structure

Subquery structure

Declaration

```
IDataStructure Structure { get; }
```

Property Value

TYPE	DESCRIPTION
IDataStructure	

Methods

FindTempTables()

Find temporary tables recursive

Declaration

<code>IEnumerable<ITempTable> FindTempTables()</code>

Returns

TYPE	DESCRIPTION
<code>System.Collections.Generic.IEnumerable<ITempTable></code>	Found temporary tables sequence

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Interface ISubqueryFilter

Subquery filter interface

Inherited Members

- [IOptionalFilter.Primary](#)
- [IFilter.FilterType](#)
- [IFilter.SourceAlias](#)
- [IFilter.Field](#)
- [IFilter.Operation](#)
- [System.ICloneable.Clone\(\)](#)

Namespace: [XData.DataSource.Structure](#)

Assembly: [XData.docfx.dll](#)

Syntax

```
public interface ISubqueryFilter : IOptionalFilter, IFilter, ICloneable
```

Properties

SubqueryAlias

Subquery alias

Declaration

```
string SubqueryAlias { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Interface ISubqueryLink

Subquery link interface

Inherited Members

[IInnerLink.LinkedSourceAlias](#)

[IInnerLink.LinkedField](#)

[IInnerLink.GetIsLink\(\)](#)

[IInnerLink.GetAliases\(\)](#)

[IOptionalFilter.Primary](#)

[IFilter.FilterType](#)

[IFilter.SourceAlias](#)

[IFilter.Field](#)

[IFilter.Operation](#)

[System.ICloneable.Clone\(\)](#)

Namespace: [XData.DataSource.Structure](#)

Assembly: XData.docfx.dll

Syntax

```
public interface ISubqueryLink : IInnerLink, IOptionalFilter, IFilter, ICloneable
```

Properties

LinkedPropertyName

Declaration

```
string LinkedPropertyName { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

LinkedType

Linked type full name

Declaration

```
string LinkedType { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

PropertyName

Declaration

```
string PropertyName { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Extension Methods

- SerializationExtensions.ToXml<T>(T)
- CollectionsExtensions.SetValue<TKey, T>(TKey, T)
- CollectionsExtensions.AsEnum<T>(T)
- DataObjectQueryableExtensions.CompareVariable(Object, FilterOperation, String)
- Processing.Do<T>(T, Action<IProcess<T>>[])
- TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>)
- TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>, TResult)
- TypeExtensions.ChangeType<T>(Object)
- TypeExtensions.ChangeType(Object, Type)
- TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)
- TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)
- TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)
- TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)
- SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)
- QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Interface ITable

Table

Inherited Members

- [ITableLike.Name](#)
- [ITableLike.Fields](#)
- [IDataSource.SourceType](#)
- [IDataSource.Alias](#)
- [IDataSource.IsBase](#)
- [IDataSource.GetFields\(\)](#)
- [IDataSource.GetFields\(Type\)](#)
- [IDataSource.GetField\(String, String, String\)](#)
- [IDataSource.GetField\(String, String, String, String\)](#)
- [System.ICloneable.Clone\(\)](#)

Namespace: [XData.DataSource.Structure](#)
Assembly: XData.docfx.dll

Syntax

```
public interface ITable : ITableLike, IDataSource, ICloneable
```

Properties

Hierarchies

Hierarchies

Declaration

```
List<Hierarchy> Hierarchies { get; }
```

Property Value

TYPE	DESCRIPTION
System.Collections.Generic.List< Hierarchy >	

Hint

Table hint

Declaration

```
HintInfo[] Hint { get; }
```

Property Value

TYPE	DESCRIPTION
HintInfo []	

ParentAlias

Parent table alias

Declaration

```
string ParentAlias { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Extension Methods

SerializationExtensions.ToXml<T>(T)

CollectionsExtensions.SetValue<TKey, T>(TKey, T)

CollectionsExtensions.AsEnum<T>(T)

DataObjectQueryableExtensions.CompareVariable(Object, FilterOperation, String)

Processing.Do<T>(T, Action<IProcess<T>>[])

TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>)

TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>, TResult)

TypeExtensions.ChangeType<T>(Object)

TypeExtensions.ChangeType(Object, Type)

TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)

TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)

SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)

QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Interface ITableLike

Temporary table

Inherited Members

- [IDataSource.SourceType](#)
- [IDataSource.Alias](#)
- [IDataSource.IsBase](#)
- [IDataSource.GetFields\(\)](#)
- [IDataSource.GetFields\(Type\)](#)
- [IDataSource.GetField\(String, String, String\)](#)
- [IDataSource.GetField\(String, String, String, String\)](#)
- [System.ICloneable.Clone\(\)](#)

Namespace: [XData.DataSource.Structure](#)

Assembly: XData.docfx.dll

Syntax

```
public interface ITableLike : IDataSource, ICloneable
```

Properties

Fields

Fields

Declaration

```
SortedSet<IField> Fields { get; }
```

Property Value

TYPE	DESCRIPTION
System.Collections.Generic.SortedSet< IField >	

Name

Table name

Declaration

```
string Name { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)

TypeExtensions.ChangeType<T>(Object)

TypeExtensions.ChangeType(Object, Type)

TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)

TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)

SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)

QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Interface ITempTable

Inherited Members

- [ITableLike.Name](#)
- [ITableLike.Fields](#)
- [IDataSource.SourceType](#)
- [IDataSource.Alias](#)
- [IDataSource.IsBase](#)
- [IDataSource.GetFields\(\)](#)
- [IDataSource.GetFields\(Type\)](#)
- [IDataSource.GetField\(String, String, String\)](#)
- [IDataSource.GetField\(String, String, String, String\)](#)
- [System.ICloneable.Clone\(\)](#)

Namespace: [XData.DataSource.Structure](#)

Assembly: [XData.docfx.dll](#)

Syntax

```
public interface ITempTable : ITableLike, IDataSource, ICloneable
```

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Interface IXmlSource

XML data source

Inherited Members

- [ITableLike.Name](#)
- [ITableLike.Fields](#)
- [IDataSource.SourceType](#)
- [IDataSource.Alias](#)
- [IDataSource.IsBase](#)
- [IDataSource.GetFields\(\)](#)
- [IDataSource.GetFields\(Type\)](#)
- [IDataSource.GetField\(String, String, String\)](#)
- [IDataSource.GetField\(String, String, String, String\)](#)
- [System.ICloneable.Clone\(\)](#)

Namespace: [XData.DataSource.Structure](#)
Assembly: [XData.docfx.dll](#)

Syntax

```
public interface IXmlSource : ITableLike, IDataSource, ICloneable
```

Properties

RootXPath

Table name

Declaration

```
string RootXPath { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Class Order

Order by element

Inheritance

System.Object
Order

Implements

System.IEquatable<[Order](#)>

Inherited Members

System.Object.Equals(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.GetHashCode()
System.Object.GetType()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.ToString()

Namespace: [XData.DataSource.Structure](#)
Assembly: XData.docfx.dll

Syntax

```
[Serializable]
[DataContract]
public sealed class Order : IEquatable<Order>
```

Properties

Descending

Descending flag

Declaration

```
[DataMember]
public bool Descending { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

FieldName

Field name

Declaration

```
[DataMember]
public string FieldName { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

PropertyName

Property name

Declaration

```
[DataMember]
public string PropertyName { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

PropertyOwnerType

Property owner type name

Declaration

```
[DataMember]
public string PropertyOwnerType { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

SourceAlias

Source alias

Declaration

```
[DataMember]
public string SourceAlias { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Methods

Equals(Order)

Order equality check

Declaration

```
public bool Equals(Order other)
```

Parameters

TYPE	NAME	DESCRIPTION
Order	other	Order instance to compare

Returns

TYPE	DESCRIPTION
System.Boolean	Instances are contains the same data

Implements

System.IEquatable<T>

Extension Methods

[SerializationExtensions.ToXml<T>\(T\)](#)

[CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)

[CollectionsExtensions.AsEnum<T>\(T\)](#)

[DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)

[Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)

[TypeExtensions.ChangeType<T>\(Object\)](#)

[TypeExtensions.ChangeType\(Object, Type\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)

[SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)

[QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Class RepositoryState

Repository state

Inheritance

System.Object

[DataQueryState](#)

RepositoryState

Inherited Members

[DataQueryState.AssemblyName](#)

[DataQueryState.ElementType](#)

[DataQueryState.Context](#)

[DataQueryState.Structure](#)

[DataQueryState.ToString\(\)](#)

System.Object.Equals(System.Object)

System.Object.Equals(System.Object, System.Object)

System.Object.GetHashCode()

System.Object.GetType()

System.Object.MemberwiseClone()

System.Object.ReferenceEquals(System.Object, System.Object)

Namespace: [XData.DataSource.Structure](#)

Assembly: XData.docfx.dll

Syntax

```
[Serializable]
[DataContract]
public sealed class RepositoryState : DataQueryState
```

Properties

Alias

Alias

Declaration

```
[DataMember]
public string Alias { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Fixed

Fixed query flag

Declaration

```
[DataMember]
public bool Fixed { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

IsSupervisor

Security session supervisor flag

Declaration

```
[DataMember]
public bool IsSupervisor { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

SecurityRoles

Security session roles

Declaration

```
[DataMember]
public string[] SecurityRoles { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String[]	

SecuritySessionClass

Security session class

Declaration

```
[DataMember]
public string SecuritySessionClass { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

ShowSubtree

Show subtree flag

Declaration

```
[DataMember]
public bool ShowSubtree { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

UserName

Security user name

Declaration

```
[DataMember]  
public string UserName { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Extension Methods

- SerializationExtensions.ToXml<T>(T)
- CollectionsExtensions.SetValue<TKey, T>(TKey, T)
- CollectionsExtensions.AsEnum<T>(T)
- DataObjectQueryableExtensions.CompareVariable(Object, FilterOperation, String)
- Processing.Do<T>(T, Action<IProcess<T>>[])
- TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>)
- TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>, TResult)
- TypeExtensions.ChangeType<T>(Object)
- TypeExtensions.ChangeType(Object, Type)
- TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)
- TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)
- TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)
- TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)
- SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)
- QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Class RuntimeFilter

Runtime value filter

Inheritance

System.Object
RuntimeFilter

Inherited Members

System.Object.Equals(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.GetHashCode()
System.Object.GetType()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.ToString()

Namespace: [XData.DataSource.Structure](#)
Assembly: XData.docfx.dll

Syntax

```
public sealed class RuntimeFilter
```

Constructors

RuntimeFilter(String, String)

Runtime filter constructor

Declaration

```
public RuntimeFilter(string propertyName, string filterName = null)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	propertyName	Filter property name
System.String	filterName	Filter name

Properties

Combination

Filter combination name

Declaration

```
public string Combination { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Name

Filter name

Declaration

```
public string Name { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Nullable

Filter null value handling

Declaration

```
public FilterNullable Nullable { get; set; }
```

Property Value

TYPE	DESCRIPTION
FilterNullable	

Operation

Filter operation

Declaration

```
public FilterOperation Operation { get; set; }
```

Property Value

TYPE	DESCRIPTION
FilterOperation	

PrimaryFilter

Primary filter flag

Declaration

```
public bool PrimaryFilter { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

PropertyName

Property name

Declaration

```
public string PropertyName { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Operators

Implicit(String to RuntimeFilter)

Cast operator for runtime filter based on property

Declaration

```
public static implicit operator RuntimeFilter(string propertyName)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	propertyName	Property name

Returns

TYPE	DESCRIPTION
RuntimeFilter	

Implicit(FilterName to RuntimeFilter)

Cast operator for named runtime filter

Declaration

```
public static implicit operator RuntimeFilter(FilterName filterName)
```

Parameters

TYPE	NAME	DESCRIPTION
FilterName	filterName	Filter name

Returns

TYPE	DESCRIPTION
RuntimeFilter	

Extension Methods

[SerializationExtensions.ToXml<T>\(T\)](#)

[CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)

[CollectionsExtensions.AsEnum<T>\(T\)](#)

[DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)

[Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)

TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>)

TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>, TResult)

TypeExtensions.ChangeType<T>(Object)

TypeExtensions.ChangeType(Object, Type)

TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)

TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)

SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)

QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Struct SelectedField

Selected field description

Inherited Members

- System.ValueType.Equals(System.Object)
- System.ValueType.GetHashCode()
- System.ValueType.ToString()
- System.Object.Equals(System.Object, System.Object)
- System.Object.GetType()
- System.Object.ReferenceEquals(System.Object, System.Object)

Namespace: [XData.DataSource.Structure](#)

Assembly: XData.docfx.dll

Syntax

```
public struct SelectedField
```

Constructors

SelectedField(IField, String, String, String)

Selected field constructor

Declaration

```
public SelectedField(IField field, string alias, string fieldName, string property = null)
```

Parameters

TYPE	NAME	DESCRIPTION
IField	field	Field object
System.String	alias	Source alias
System.String	fieldName	Field name
System.String	property	Property name

Fields

Alias

Alias

Declaration

```
public readonly string Alias
```

Field Value

TYPE	DESCRIPTION
System.String	

Field

Field description

Declaration

```
public readonly IField Field
```

Field Value

TYPE	DESCRIPTION
IField	

FieldName

Full field name

Declaration

```
public readonly string FieldName
```

Field Value

TYPE	DESCRIPTION
System.String	

Property

Property name

Declaration

```
public readonly string Property
```

Field Value

TYPE	DESCRIPTION
System.String	

Extension Methods

- SerializationExtensions.ToXml<T>(T)
- CollectionsExtensions.SetValue<TKey, T>(TKey, T)
- CollectionsExtensions.AsEnum<T>(T)
- DataObjectQueryableExtensions.CompareVariable(Object, FilterOperation, String)
- Processing.Do<T>(T, Action<IProcess<T>>[])
- TypeExtensions.ChangeType<T>(Object)
- TypeExtensions.ChangeType(Object, Type)
- TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)
- TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)
- TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)
- TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)

SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)

QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Struct TaggedValue

Named value

Inherited Members

- System.ValueType.Equals(System.Object)
- System.ValueType.GetHashCode()
- System.ValueType.ToString()
- System.Object.Equals(System.Object, System.Object)
- System.Object.GetType()
- System.Object.ReferenceEquals(System.Object, System.Object)

Namespace: [XData.DataSource.Structure](#)

Assembly: [XData.docfx.dll](#)

Syntax

```
[Serializable]
[DataContract]
public struct TaggedValue
```

Properties

Key

Name

Declaration

```
[DataMember]
public string Key { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Value

Value

Declaration

```
[DataMember]
public object Value { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Object	

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)

TypeExtensions.ChangeType(Object, Type)

TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)

TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)

SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)

QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Namespace XData.Exceptions

Classes

[AdditionalInfo](#)

Exception and log additional info

[XDataAdapterException](#)

Adapter exception

[XDataConcurrencyException](#)

Concurrency exception

[XDataConfigurationException](#)

XData configuration exception

[XDataException](#)

Base XData exception class

[XDataExceptionAttribute](#)

XData exception attribute

[XDataQueryStructureException](#)

Query structure build failed

[XDataRuntimeException](#)

XData runtime exception

[XDataSecurityException](#)

XData security related exception

[XDataSqlException](#)

SQL exception

Enums

[XDataAdapterExceptionType](#)

XData adapter exception types

[XDataConfigurationExceptionType](#)

XData configuration exception types

[XDataQueryStructureExceptionType](#)

Query structure exception types

[XDataRuntimeExceptionType](#)

XData runtime exception type

Class AdditionalInfo

Exception and log additional info

Inheritance

System.Object
AdditionalInfo

Inherited Members

System.Object.Equals(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.GetHashCode()
System.Object.GetType()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.ToString()

Namespace: [XData.Exceptions](#)

Assembly: XData.docfx.dll

Syntax

```
[Serializable]  
public sealed class AdditionalInfo
```

Properties

Name

Additional info name

Declaration

```
public string Name { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Value

Additional info value

Declaration

```
public object Value { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Object	

Operators

Implicit(KeyValuePair<String, Object> to AdditionalInfo)

Exception additional info

Declaration

```
public static implicit operator AdditionalInfo(KeyValuePair<string, object> source)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Collections.Generic.KeyValuePair<System.String, System.Object>	source	Source key value pair

Returns

TYPE	DESCRIPTION
AdditionalInfo	Exception additional info

Implicit(KeyValuePair<String, String> to AdditionalInfo)

Exception additional info

Declaration

```
public static implicit operator AdditionalInfo(KeyValuePair<string, string> source)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Collections.Generic.KeyValuePair<System.String, System.String>	source	Source key value pair

Returns

TYPE	DESCRIPTION
AdditionalInfo	Exception additional info

Extension Methods

[SerializationExtensions.ToXml<T>\(T\)](#)

[CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)

[CollectionsExtensions.AsEnum<T>\(T\)](#)

[DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)

[Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)

[TypeExtensions.ChangeType<T>\(Object\)](#)

[TypeExtensions.ChangeType\(Object, Type\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)

[SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)

[QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Class XDataAdapterException

Adapter exception

Inheritance

System.Object
System.Exception
System.ApplicationException
[XDataException](#)
XDataAdapterException

Implements

System.Runtime.Serialization.ISerializable

Inherited Members

System.Exception.GetBaseException()
System.Exception.GetObjectData(System.Runtime.Serialization.SerializationInfo, System.Runtime.Serialization.StreamingContext)
System.Exception.GetType()
System.Exception.ToString()
System.Exception.Data
System.Exception.HelpLink
System.Exception.HResult
System.Exception.InnerException
System.Exception.Message
System.Exception.Source
System.Exception.StackTrace
System.Exception.TargetSite
System.Exception.SerializeObjectState
System.Object.Equals(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.GetHashCode()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)

Namespace: [XData.Exceptions](#)

Assembly: XData.docfx.dll

Syntax

```
[Serializable]  
public class XDataAdapterException : XDataException, ISerializable
```

Constructors

XDataAdapterException(XDataAdapterExceptionType, String)

Adapter exception

Declaration

```
public XDataAdapterException(XDataAdapterExceptionType type, string path)
```

Parameters

TYPE	NAME	DESCRIPTION
XDataAdapterExceptionType	type	Exception type

TYPE	NAME	DESCRIPTION
System.String	path	Adapter file path

XDataAdapterException(XDataAdapterExceptionType, String, Type)

Adapter exception

Declaration

```
public XDataAdapterException(XDataAdapterExceptionType type, string details, Type adapterType)
```

Parameters

TYPE	NAME	DESCRIPTION
XDataAdapterExceptionType	type	Exception type
System.String	details	Exception details
System.Type	adapterType	Adapter type

XDataAdapterException(XDataAdapterExceptionType, Type)

Adapter exception

Declaration

```
public XDataAdapterException(XDataAdapterExceptionType type, Type adapterType)
```

Parameters

TYPE	NAME	DESCRIPTION
XDataAdapterExceptionType	type	Exception type
System.Type	adapterType	Adapter type

Properties

AdapterType

Adapter type

Declaration

```
public Type AdapterType { get; }
```

Property Value

TYPE	DESCRIPTION
System.Type	

Details

Exception details

Declaration

```
public string Details { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

FilePath

Adapter file path

Declaration

```
public string FilePath { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Type

Adapter exception type

Declaration

```
public XDataAdapterExceptionType Type { get; }
```

Property Value

TYPE	DESCRIPTION
XDataAdapterExceptionType	

Implements

System.Runtime.Serialization.ISerializable

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)

TypeExtensions.ChangeType(Object, Type)

TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)

TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)

SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)

QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Enum XDataAdapterExceptionType

XData adapter exception types

Namespace: [XData.Exceptions](#)

Assembly: [XData.docfx.dll](#)

Syntax

```
public enum XDataAdapterExceptionType
```

Fields

NAME	DESCRIPTION
AdapterClassNotFoundError	Adapter class not found
AdapterConstructorFailedError	Adapter constructor failed
ComplexPrimaryKeyNotSupportedError	Complex primary key is not supported in this case
FunctionIsNotSupportedError	Function is not supported
GetSqlSyntaxTemplateError	SQL syntax not provided by adapter error
SqlBlockProceduralSyntaxNotSupportedError	SQL block procedural syntax is not supported
SqlBuilderClassNotFoundError	Sql builder class not found
SqlBuilderConstructorFailedError	Sql builder constructor failed
SyntaxNotSupportedError	SQL syntax construction is not supported
TypeNotSupportedError	Data type is not supported

Extension Methods

- [SerializationExtensions.ToXml<XDataAdapterExceptionType>\(\)](#)
- [CollectionsExtensions.SetValue<XDataAdapterExceptionType, T>\(T\)](#)
- [CollectionsExtensions.AsEnum<XDataAdapterExceptionType>\(\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(FilterOperation, String\)](#)
- [Processing.Do<XDataAdapterExceptionType>\(Action<IProcess<XDataAdapterExceptionType>> > \[\]\)](#)
- [TypeExtensions.ChangeType<T>\(\)](#)

TypeExtensions.ChangeType(Type)

TypeExtensions.InvokeMethod<TRet>(Type[], Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Type[], Expression<Action>, BindingFlags)

TypeExtensions.InvokeMethod<TRet>(Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Expression<Action>, BindingFlags)

TypeExtensions.ConvertTo<T>()

TypeExtensions.EqualValue(Enum)

SqlBlockExtensions.SetExpression<XDataAdapterExceptionType, TResult>(Expression<Func<IBlockQueryAdapter, TResult>>)

QueryDescriptionExtensions.SetExpression<XDataAdapterExceptionType, TResult>(Expression<Func<IQueryStructureAdapter, TResult>>)

Class XDataConcurrencyException

Concurrency exception

Inheritance

System.Object
System.Exception
System.ApplicationException
[XDataException](#)
XDataConcurrencyException

Implements

System.Runtime.Serialization.ISerializable

Inherited Members

System.Exception.GetBaseException()
System.Exception.GetObjectData(System.Runtime.Serialization.SerializationInfo, System.Runtime.Serialization.StreamingContext)
System.Exception.GetType()
System.Exception.ToString()
System.Exception.Data
System.Exception.HelpLink
System.Exception.HResult
System.Exception.InnerException
System.Exception.Message
System.Exception.Source
System.Exception.StackTrace
System.Exception.TargetSite
System.Exception.SerializeObjectState
System.Object.Equals(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.GetHashCode()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)

Namespace: [XData.Exceptions](#)

Assembly: XData.docfx.dll

Syntax

```
[Serializable]  
public sealed class XDataConcurrencyException : XDataException, ISerializable
```

Constructors

XDataConcurrencyException(Exception, Query)

Concurrency exception

Declaration

```
public XDataConcurrencyException(Exception exception, Query query)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Exception	exception	Inner exception

TYPE	NAME	DESCRIPTION
Query	query	Query

XDataConcurrencyException(Exception, AdditionalInfo[])

Concurrency exception

Declaration

```
public XDataConcurrencyException(Exception exception, params AdditionalInfo[] addInfo)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Exception	exception	Inner exception
AdditionalInfo[]	addInfo	Additional information

XDataConcurrencyException(String)

Concurrency exception

Declaration

```
public XDataConcurrencyException(string exception)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	exception	Exception text

Properties

Query

Query

Declaration

```
public Query Query { get; }
```

Property Value

TYPE	DESCRIPTION
Query	

Implements

System.Runtime.Serialization.ISerializable

Extension Methods

SerializationExtensions.ToXml<T>(T)
CollectionsExtensions.SetValue<TKey, T>(TKey, T)
CollectionsExtensions.AsEnum<T>(T)
DataObjectQueryableExtensions.CompareVariable(Object, FilterOperation, String)
Processing.Do<T>(T, Action<IProcess<T>>[])
TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>)
TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>, TResult)
TypeExtensions.ChangeType<T>(Object)
TypeExtensions.ChangeType(Object, Type)
TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)
TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)
TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)
TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)
SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)
QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Class XDataConfigurationException

XData configuration exception

Inheritance

- System.Object
- System.Exception
- System.ApplicationException
- XDataException
- XDataConfigurationException

Implements

- System.Runtime.Serialization.ISerializable

Inherited Members

- System.Exception.GetBaseException()
- System.Exception.GetObjectData(System.Runtime.Serialization.SerializationInfo, System.Runtime.Serialization.StreamingContext)
- System.Exception.GetType()
- System.Exception.ToString()
- System.Exception.Data
- System.Exception.HelpLink
- System.Exception.HResult
- System.Exception.InnerException
- System.Exception.Message
- System.Exception.Source
- System.Exception.StackTrace
- System.Exception.TargetSite
- System.Exception.SerializeObjectState
- System.Object.Equals(System.Object)
- System.Object.Equals(System.Object, System.Object)
- System.Object.GetHashCode()
- System.Object.MemberwiseClone()
- System.Object.ReferenceEquals(System.Object, System.Object)

Namespace: [XData.Exceptions](#)

Assembly: XData.docfx.dll

Syntax

```
[Serializable]
public class XDataConfigurationException : XDataException, ISerializable
```

Constructors

XDataConfigurationException(XDataConfigurationExceptionType)

Query structure build failed

Declaration

```
public XDataConfigurationException(XDataConfigurationExceptionType type)
```

Parameters

TYPE	NAME	DESCRIPTION
XDataConfigurationExceptionType	type	Query structure exception type

XDataConfigurationException(XDataConfigurationExceptionType, String)

Query structure build failed

Declaration

```
public XDataConfigurationException(XDataConfigurationExceptionType type, string details)
```

Parameters

TYPE	NAME	DESCRIPTION
XDataConfigurationExceptionType	type	Query structure exception type
System.String	details	Error details

XDataConfigurationException(XDataConfigurationExceptionType, String, Exception)

Query structure build failed

Declaration

```
public XDataConfigurationException(XDataConfigurationExceptionType type, string details, Exception inner)
```

Parameters

TYPE	NAME	DESCRIPTION
XDataConfigurationExceptionType	type	Query structure exception type
System.String	details	Error details
System.Exception	inner	Inner exception

Properties

DetailsInfo

Exception details info

Declaration

```
public string DetailsInfo { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Type

Type of exception

Declaration

```
public XDataConfigurationExceptionType Type { get; }
```

Property Value

TYPE	DESCRIPTION
XDataConfigurationExceptionType	

Implements

[System.Runtime.Serialization.ISerializable](#)

Extension Methods

[SerializationExtensions.ToXml<T>\(T\)](#)

[CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)

[CollectionsExtensions.AsEnum<T>\(T\)](#)

[DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)

[Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)

[TypeExtensions.ChangeType<T>\(Object\)](#)

[TypeExtensions.ChangeType\(Object, Type\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)

[SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)

[QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Enum XDataConfigurationExceptionType

XData configuration exception types

Namespace: [XData.Exceptions](#)

Assembly: [XData.docfx.dll](#)

Syntax

```
public enum XDataConfigurationExceptionType
```

Fields

NAME	DESCRIPTION
AdapterAssemblyNotFoundError	Adapter assembly not found
AdapterNotConfiguredError	Data adapter not configured
ConfigurationFileLoadError	Configuration file load error
ConfigurationFileSaveError	Configuration file save error
ConfigurationXmlReadError	Configuration xml read error
ConnectionStringHasWrongFormatError	Connection string has wrong format
ContextNotConfiguredError	Data context not configured
DefaultContextNotFoundError	Default data context is not found
ProxyNotConfiguredError	Proxy not configured

Extension Methods

- [SerializationExtensions.ToXml<XDataConfigurationExceptionType>\(\)](#)
- [CollectionsExtensions.SetValue<XDataConfigurationExceptionType, T>\(T\)](#)
- [CollectionsExtensions.AsEnum<XDataConfigurationExceptionType>\(\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(FilterOperation, String\)](#)
- [Processing.Do<XDataConfigurationExceptionType>\(Action<IProcess<XDataConfigurationExceptionType>> > \[\]\)](#)
- [TypeExtensions.ChangeType<T>\(\)](#)
- [TypeExtensions.ChangeType\(Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Expression<Func<TRet>>, BindingFlags\)](#)

TypeExtensions.InvokeMethod(Expression<Action>, BindingFlags)

TypeExtensions.ConvertTo<T>()

TypeExtensions.EqualValue(Enum)

SqlBlockExtensions.SetExpression<XDataConfigurationExceptionType, TResult>(Expression<Func<IBlockQueryAdapter, TResult>>)

QueryDescriptionExtensions.SetExpression<XDataConfigurationExceptionType, TResult>

(Expression<Func<IQueryStructureAdapter, TResult>>)

Class XDataException

Base XData exception class

Inheritance

System.Object

System.Exception

System.ApplicationException

XDataException

[XDataAdapterException](#)

[XDataConcurrencyException](#)

[XDataConfigurationException](#)

[XDataQueryStructureException](#)

[XDataRuntimeErrorException](#)

[XDataSecurityException](#)

[XDataSqlException](#)

Implements

System.Runtime.Serialization.ISerializable

Inherited Members

System.Exception.GetBaseException()

System.Exception.GetObjectData(System.Runtime.Serialization.SerializationInfo, System.Runtime.Serialization.StreamingContext)

System.Exception.GetType()

System.Exception.ToString()

System.Exception.Data

System.Exception.HelpLink

System.Exception.HResult

System.Exception.InnerException

System.Exception.Message

System.Exception.Source

System.Exception.StackTrace

System.Exception.TargetSite

System.Exception.SerializeObjectState

System.Object.Equals(System.Object)

System.Object.Equals(System.Object, System.Object)

System.Object.GetHashCode()

System.Object.MemberwiseClone()

System.Object.ReferenceEquals(System.Object, System.Object)

Namespace: [XData.Exceptions](#)

Assembly: XData.docfx.dll

Syntax

```
public abstract class XDataException : ApplicationException, ISerializable
```

Constructors

XDataException(String)

Base XData exception class constructor

Declaration

```
protected XDataException(string message)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	message	

XDataException(String, Exception)

Base XData exception class constructor

Declaration

```
protected XDataException(string message, Exception innerException)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	message	
System.Exception	innerException	

Implements

System.Runtime.Serialization.ISerializable

Extension Methods

- SerializationExtensions.ToXml<T>(T)
- CollectionsExtensions.SetValue<TKey, T>(TKey, T)
- CollectionsExtensions.AsEnum<T>(T)
- DataObjectQueryableExtensions.CompareVariable(Object, FilterOperation, String)
- Processing.Do<T>(T, Action<IProcess<T>>[])
- TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>)
- TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>, TResult)
- TypeExtensions.ChangeType<T>(Object)
- TypeExtensions.ChangeType(Object, Type)
- TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)
- TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)
- TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)
- TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)
- SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)
- QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Class XDataExceptionAttribute

XData exception attribute

Inheritance

System.Object

System.Attribute

XDataExceptionAttribute

Inherited Members

System.Attribute.Equals(System.Object)

System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Module)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.GetHashCode()

System.Attribute.IsDefaultAttribute()

System.Attribute.IsDefined(System.Reflection.Assembly, System.Type)

System.Attribute.IsDefined(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type)

System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.Module, System.Type)

System.Attribute.IsDefined(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type)

System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.Match(System.Object)

System.Attribute.TypeId

System.Object.Equals(System.Object, System.Object)

System.Object.GetType()

System.Object.MemberwiseClone()

System.Object.ReferenceEquals(System.Object, System.Object)

System.Object.ToString()

Namespace: [XData.Exceptions](#)

Syntax

```
public class XDataExceptionAttribute : Attribute
```

Extension Methods

[SerializationExtensions.ToXml<T>\(T\)](#)

[CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)

[CollectionsExtensions.AsEnum<T>\(T\)](#)

[DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)

[Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)

[TypeExtensions.ChangeType<T>\(Object\)](#)

[TypeExtensions.ChangeType\(Object, Type\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)

[SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)

[QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Class XDataQueryStructureException

Query structure build failed

Inheritance

System.Object
System.Exception
System.ApplicationException
[XDataException](#)
XDataQueryStructureException

Implements

System.Runtime.Serialization.ISerializable

Inherited Members

System.Exception.GetBaseException()
System.Exception.GetObjectData(System.Runtime.Serialization.SerializationInfo, System.Runtime.Serialization.StreamingContext)
System.Exception.GetType()
System.Exception.ToString()
System.Exception.Data
System.Exception.HelpLink
System.Exception.HResult
System.Exception.InnerException
System.Exception.Message
System.Exception.Source
System.Exception.StackTrace
System.Exception.TargetSite
System.Exception.SerializeObjectState
System.Object.Equals(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.GetHashCode()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)

Namespace: [XData.Exceptions](#)

Assembly: XData.docfx.dll

Syntax

```
[Serializable]  
public sealed class XDataQueryStructureException : XDataException, ISerializable
```

Constructors

XDataQueryStructureException(XDataQueryStructureExceptionType, String)

Query structure build failed

Declaration

```
public XDataQueryStructureException(XDataQueryStructureExceptionType type, string details)
```

Parameters

TYPE	NAME	DESCRIPTION
XDataQueryStructureExceptionType	type	Query structure exception type

TYPE	NAME	DESCRIPTION
System.String	details	Error details

XDataQueryStructureException(XDataQueryStructureExceptionType, String, Type)

Query structure build failed

Declaration

```
public XDataQueryStructureException(XDataQueryStructureExceptionType type, string details, Type dataObjectType)
```

Parameters

TYPE	NAME	DESCRIPTION
XDataQueryStructureExceptionType	type	Query structure exception type
System.String	details	Error details
System.Type	dataObjectType	Data object type

XDataQueryStructureException(XDataQueryStructureExceptionType, String, Type, AdditionalInfo[])

Query structure build failed

Declaration

```
public XDataQueryStructureException(XDataQueryStructureExceptionType type, string details, Type dataObjectType, params AdditionalInfo[] addInfo)
```

Parameters

TYPE	NAME	DESCRIPTION
XDataQueryStructureExceptionType	type	Query structure exception type
System.String	details	Error details
System.Type	dataObjectType	Data object type
AdditionalInfo[]	addInfo	Additional information

Exceptions

TYPE	CONDITION
System.NotImplementedException	

XDataQueryStructureException(XDataQueryStructureExceptionType, Type)

Query structure build failed

Declaration

```
public XDataQueryStructureException(XDataQueryStructureExceptionType type, Type dataObjectType)
```

Parameters

TYPE	NAME	DESCRIPTION
XDataQueryStructureExceptionType	type	Query structure exception type
System.Type	dataObjectType	Data object type

Properties

DataObjectType

Data object type

Declaration

```
public Type DataObjectType { get; }
```

Property Value

TYPE	DESCRIPTION
System.Type	

DetailsInfo

Exception details info

Declaration

```
public string DetailsInfo { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Type

Type of exception

Declaration

```
public XDataQueryStructureExceptionType Type { get; }
```

Property Value

TYPE	DESCRIPTION
XDataQueryStructureExceptionType	

Implements

System.Runtime.Serialization.ISerializable

Extension Methods

- SerializationExtensions.ToXml<T>(T)
- CollectionsExtensions.SetValue<TKey, T>(TKey, T)
- CollectionsExtensions.AsEnum<T>(T)
- DataObjectQueryableExtensions.CompareVariable(Object, FilterOperation, String)
- Processing.Do<T>(T, Action<IProcess<T>>[])
- TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>)
- TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>, TResult)
- TypeExtensions.ChangeType<T>(Object)
- TypeExtensions.ChangeType(Object, Type)
- TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)
- TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)
- TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)
- TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)
- SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)
- QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Enum XDataQueryStructureExceptionType

Query structure exception types

Namespace: [XData.Exceptions](#)

Assembly: XData.docfx.dll

Syntax

```
public enum XDataQueryStructureExceptionType
```

Fields

NAME	DESCRIPTION
AddTreeFilterError	Tree filter (START WITH) generation failed
AddTreeLinkError	Tree link (CONNECT BY PRIOR) filter generation failed
CallNonProcedureBasedStructureError	Call of non procedure based structure
DictionaryTableHasComplexPrimaryKeyError	Dictionary table has complex primary key
DictionaryTableHasNoPrimaryKeyError	Dictionary table has no primary key
EmptyQueryError	Empty query text generated error
FieldNotFoundError	Field not found
FilterCombinationNotFoundError	Filter combination not found
FindPropertyError	Property not found error
GetParentStructureError	Parent structure type not found error
HierarchyOnComplexPrimaryKeyTableError	Table with hierarchy has complex primary key
HierarchyOnNoPrimaryKeyTableError	Table with hierarchy has no primary key
IdPropertyNotFoundError	Id property not found

NAME	DESCRIPTION
InnerViewNotFoundError	No inner view found for alias
LinkedPropertyNotFoundError	Linked property not found
LinkToProcedureFoundError	SQL stored procedure can not be linked
LinqExpressionCannotBeParsedError	LINQ expression can not be parsed
LinqExpressionHasUnknownMethodError	LINQ expression contains unknown method error
LinqExpressionHasUnsupportedMethodError	LINQ expression contains unsupported method
LinqExpressionHasUnsupportedMethodParametersError	LINQ expression contains unsupported method parameter count error
LinqExpressionHasUnsupportedTypeError	LINQ expression contains unsupported in this case expression type
NoDataSourceMappingFoundError	Data source mapping not found
NoElementsAtPositionError	Sequence contains no elements at position error
NoElementsError	Sequence contains no elements error
NoMappingFoundError	No mapping found for data object
NoneBasedObjectHasTablesError	Can't build data object based on NONE table contains other tables error
NoPropertyMappingFoundError	Property mapping not found
NoSelectedFieldsFoundError	Selected fields list is empty error
PropertyEnumValueNotFoundError	Property enum values not found

NAME	DESCRIPTION
PropertyNotFoundError	Property not found (or not mapped)
ResultSetMappingExpressionError	SQL stored procedure / function result set expression mapping used
ResultSetNotRegisteredError	Result set not registered
RuntimeFilterUsedInsideDynamicQuery	Runtime filter used inside dynamic query
SubqueryAliasDuplicatedError	Subquery alias duplicated
SubqueryNotFoundError	No subquery found for alias
TableNotFoundError	Table source not found
WrongDataSourceTypeError	Wrong data source type used
WrongLoadChildPropertyTypeError	Serialize child repository wrong target property type

Extension Methods

[SerializationExtensions.ToXml<XDataQueryStructureExceptionType>\(\)](#)

[CollectionsExtensions.SetValue<XDataQueryStructureExceptionType, T>\(T\)](#)

[CollectionsExtensions.AsEnum<XDataQueryStructureExceptionType>\(\)](#)

[DataObjectQueryableExtensions.CompareVariable\(FilterOperation, String\)](#)

[Processing.Do<XDataQueryStructureExceptionType>\(Action<IProcess<XDataQueryStructureExceptionType>> >\(\)\)](#)

[TypeExtensions.ChangeType<T>\(\)](#)

[TypeExtensions.ChangeType\(Type\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Type\[\], Expression<Action>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Expression<Action>, BindingFlags\)](#)

[TypeExtensions.ConvertTo<T>\(\)](#)

[TypeExtensions.EqualValue\(Enum\)](#)

[SqlBlockExtensions.SetExpression<XDataQueryStructureExceptionType, TResult>\(Expression<Func<IBlockQueryAdapter, TResult>> >\)](#)

[QueryDescriptionExtensions.SetExpression<XDataQueryStructureExceptionType, TResult>\(Expression<Func<IQueryStructureAdapter, TResult>> >\)](#)

Class XDataRuntimeException

XData runtime exception

Inheritance

- System.Object
- System.Exception
- System.ApplicationException
- XDataException
- XDataRuntimeException

Implements

- System.Runtime.Serialization.ISerializable

Inherited Members

- System.Exception.GetBaseException()
- System.Exception.GetObjectData(System.Runtime.Serialization.SerializationInfo, System.Runtime.Serialization.StreamingContext)
- System.Exception.GetType()
- System.Exception.ToString()
- System.Exception.Data
- System.Exception.HelpLink
- System.Exception.HResult
- System.Exception.InnerException
- System.Exception.Message
- System.Exception.Source
- System.Exception.StackTrace
- System.Exception.TargetSite
- System.Exception.SerializeObjectState
- System.Object.Equals(System.Object)
- System.Object.Equals(System.Object, System.Object)
- System.Object.GetHashCode()
- System.Object.MemberwiseClone()
- System.Object.ReferenceEquals(System.Object, System.Object)

Namespace: [XData.Exceptions](#)

Assembly: XData.docfx.dll

Syntax

```
[Serializable]
public sealed class XDataRuntimeException : XDataException, ISerializable
```

Constructors

XDataRuntimeException(XDataRuntimeExceptionType)

XData runtime exception

Declaration

```
public XDataRuntimeException(XDataRuntimeExceptionType type)
```

Parameters

TYPE	NAME	DESCRIPTION
XDataRuntimeExceptionType	type	Exception type

XDataRuntimeException(XDataRuntimeExceptionType, Exception)

XData runtime exception

Declaration

```
public XDataRuntimeException(XDataRuntimeExceptionType type, Exception exception)
```

Parameters

TYPE	NAME	DESCRIPTION
XDataRuntimeExceptionType	type	Exception type
System.Exception	exception	Inner exception

XDataRuntimeException(XDataRuntimeExceptionType, Exception, Type)

XData runtime exception

Declaration

```
public XDataRuntimeException(XDataRuntimeExceptionType type, Exception exception, Type dataObjectType)
```

Parameters

TYPE	NAME	DESCRIPTION
XDataRuntimeExceptionType	type	Exception type
System.Exception	exception	Inner exception
System.Type	dataObjectType	Data object type

XDataRuntimeException(XDataRuntimeExceptionType, Exception, Type, AdditionalInfo[])

XData runtime exception

Declaration

```
public XDataRuntimeException(XDataRuntimeExceptionType type, Exception exception, Type dataObjectType, params AdditionalInfo[] addInfo)
```

Parameters

TYPE	NAME	DESCRIPTION
XDataRuntimeExceptionType	type	Exception type
System.Exception	exception	Inner exception

TYPE	NAME	DESCRIPTION
System.Type	dataObjectType	Data object type
AdditionalInfo[]	addInfo	Additional information

XDataRuntimeException(XDataRuntimeExceptionType, String)

XData runtime exception

Declaration

```
public XDataRuntimeException(XDataRuntimeExceptionType type, string details)
```

Parameters

TYPE	NAME	DESCRIPTION
XDataRuntimeExceptionType	type	Exception type
System.String	details	Exception details info

XDataRuntimeException(XDataRuntimeExceptionType, String, Type)

XData runtime exception

Declaration

```
public XDataRuntimeException(XDataRuntimeExceptionType type, string details, Type dataObjectType)
```

Parameters

TYPE	NAME	DESCRIPTION
XDataRuntimeExceptionType	type	Exception type
System.String	details	Exception details info
System.Type	dataObjectType	Data object type

XDataRuntimeException(XDataRuntimeExceptionType, String, Type, AdditionalInfo[])

XData runtime exception

Declaration

```
public XDataRuntimeException(XDataRuntimeExceptionType type, string details, Type dataObjectType, params AdditionalInfo[] addInfo)
```

Parameters

TYPE	NAME	DESCRIPTION
XDataRuntimeExceptionType	type	Exception type
System.String	details	Exception details info
System.Type	dataObjectType	Data object type
AdditionalInfo[]	addInfo	Additional info

XDataRuntimeException(XDataRuntimeExceptionType, Type)

XData runtime exception

Declaration

```
public XDataRuntimeException(XDataRuntimeExceptionType type, Type dataObjectType)
```

Parameters

TYPE	NAME	DESCRIPTION
XDataRuntimeExceptionType	type	Exception type
System.Type	dataObjectType	Data object type

Properties

DataObjectType

Data object type

Declaration

```
public Type DataObjectType { get; }
```

Property Value

TYPE	DESCRIPTION
System.Type	

DetailsInfo

Exception details info

Declaration

```
public string DetailsInfo { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Type

Type of exception

Declaration

```
public XDataRuntimeExceptionType Type { get; }
```

Property Value

TYPE	DESCRIPTION
XDataRuntimeExceptionType	

Implements

System.Runtime.Serialization.ISerializable

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Enum XDataRuntimeExceptionType

XData runtime exception type

Namespace: [XData.Exceptions](#)

Assembly: XData.docfx.dll

Syntax

```
public enum XDataRuntimeExceptionType
```

Fields

NAME	DESCRIPTION
ArgumentError	Argument error
AssemblyNotFoundError	Assembly not found by path
AttachedHandlersNotFoundError	Attached handlers collection not found
CustomActionAttributeNotFoundError	CustomActionAttribute not found
CustomLogicError	Custom logic error
DataConversionError	Data conversion error
DataMappingAttributeNotFoundError	Data mapping attribute not found
DataMappingError	Data mapping error
FunctionLockedError	Function is locked
GetIndexedAttributeError	IndexedAttribute not found
GetOriginalDataError	Original data not found
GetPropertyValueError	Property not found

NAME	DESCRIPTION
GetRepositoryError	Object not attached to repository
GetSerializedDataFieldError	SerializedData field not found
GetStateError	Object state not found
PropertyIsReadOnlyError	Property is read only
RepositoryIsReadOnlyError	Repository marked as read only
ServerCallError	XData Application Server call error
SessionCreationFailedError	Application server session creation failed
SessionNotFoundError	Application server session not found
TypeLoadingError	Type loading error
TypeNotFoundError	Type not found in assembly
WorkSetError	Work set data access error

Extension Methods

[SerializationExtensions.ToXml<XDataRuntimeExceptionType>\(\)](#)

[CollectionsExtensions.SetValue<XDataRuntimeExceptionType, T>\(T\)](#)

[CollectionsExtensions.AsEnum<XDataRuntimeExceptionType>\(\)](#)

[DataObjectQueryableExtensions.CompareVariable\(FilterOperation, String\)](#)

[Processing.Do<XDataRuntimeExceptionType>\(Action<IProcess<XDataRuntimeExceptionType>> >\(\)\)](#)

[TypeExtensions.ChangeType<T>\(\)](#)

[TypeExtensions.ChangeType\(Type\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Type\[\], Expression<Action>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Expression<Action>, BindingFlags\)](#)

[TypeExtensions.ConvertTo<T>\(\)](#)

[TypeExtensions.EqualValue\(Enum\)](#)

[SqlBlockExtensions.SetExpression<XDataRuntimeExceptionType, TResult>\(Expression<Func<IBlockQueryAdapter, TResult>> >\)](#)

```
QueryDescriptionExtensions.SetExpression<XDataRuntimeExceptionType, TResult>(Expression<Func<IQueryStructureAdapter, TResult>>)
```

Class XDataSecurityException

XData security related exception

Inheritance

- System.Object
- System.Exception
- System.ApplicationException
- XDataException
- XDataSecurityException

Implements

- System.Runtime.Serialization.ISerializable

Inherited Members

- System.Exception.GetBaseException()
- System.Exception.GetObjectData(System.Runtime.Serialization.SerializationInfo, System.Runtime.Serialization.StreamingContext)
- System.Exception.GetType()
- System.Exception.ToString()
- System.Exception.Data
- System.Exception.HelpLink
- System.Exception.HResult
- System.Exception.InnerException
- System.Exception.Message
- System.Exception.Source
- System.Exception.StackTrace
- System.Exception.TargetSite
- System.Exception.SerializeObjectState
- System.Object.Equals(System.Object)
- System.Object.Equals(System.Object, System.Object)
- System.Object.GetHashCode()
- System.Object.MemberwiseClone()
- System.Object.ReferenceEquals(System.Object, System.Object)

Namespace: [XData.Exceptions](#)

Assembly: XData.docfx.dll

Syntax

```
[Serializable]
public sealed class XDataSecurityException : XDataException, ISerializable
```

Constructors

XDataSecurityException(String)

XData security exception

Declaration

```
public XDataSecurityException(string details)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	details	Exception details info

XDataSecurityException(String, Exception)

XData security exception

Declaration

```
public XDataSecurityException(string details, Exception exception)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	details	Exception details info
System.Exception	exception	Inner exception

XDataSecurityException(String, Exception, Type)

XData security exception

Declaration

```
public XDataSecurityException(string details, Exception exception, Type dataObjectType)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	details	Exception details info
System.Exception	exception	Inner exception
System.Type	dataObjectType	Data object type

XDataSecurityException(String, Exception, Type, AdditionalInfo[])

XData security exception

Declaration

```
public XDataSecurityException(string details, Exception exception, Type dataObjectType, params AdditionalInfo[] addInfo)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	details	Exception details info
System.Exception	exception	Inner exception

TYPE	NAME	DESCRIPTION
System.Type	dataObjectType	Data object type
AdditionalInfo[]	addInfo	Additional information

XDataSecurityException(String, Type)

XData security exception

Declaration

```
public XDataSecurityException(string details, Type dataObjectType)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	details	Exception details info
System.Type	dataObjectType	Data object type

XDataSecurityException(String, Type, AdditionalInfo[])

XData security exception

Declaration

```
public XDataSecurityException(string details, Type dataObjectType, params AdditionalInfo[] addInfo)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	details	Exception details info
System.Type	dataObjectType	Data object type
AdditionalInfo[]	addInfo	Additional info

Properties

DataObjectType

Data object type

Declaration

```
public Type DataObjectType { get; }
```


Property Value

TYPE	DESCRIPTION
System.Type	

DetailsInfo

Exception details info

Declaration

```
public string DetailsInfo { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Implements

System.Runtime.Serialization.ISerializable

Extension Methods

- SerializationExtensions.ToXml<T>(T)
- CollectionsExtensions.SetValue<TKey, T>(TKey, T)
- CollectionsExtensions.AsEnum<T>(T)
- DataObjectQueryableExtensions.CompareVariable(Object, FilterOperation, String)
- Processing.Do<T>(T, Action<IProcess<T>>[])
- TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>)
- TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>, TResult)
- TypeExtensions.ChangeType<T>(Object)
- TypeExtensions.ChangeType(Object, Type)
- TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)
- TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)
- TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)
- TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)
- SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)
- QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Class XDataSqlException

SQL exception

Inheritance

System.Object
System.Exception
System.ApplicationException
[XDataException](#)
XDataSqlException

Implements

System.Runtime.Serialization.ISerializable

Inherited Members

System.Exception.GetBaseException()
System.Exception.GetObjectData(System.Runtime.Serialization.SerializationInfo, System.Runtime.Serialization.StreamingContext)
System.Exception.GetType()
System.Exception.ToString()
System.Exception.Data
System.Exception.HelpLink
System.Exception.HResult
System.Exception.InnerException
System.Exception.Message
System.Exception.Source
System.Exception.StackTrace
System.Exception.TargetSite
System.Exception.SerializeObjectState
System.Object.Equals(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.GetHashCode()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)

Namespace: [XData.Exceptions](#)

Assembly: XData.docfx.dll

Syntax

```
[Serializable]  
public sealed class XDataSqlException : XDataException, ISerializable
```

Constructors

XDataSqlException(Exception, Query)

SQL exception

Declaration

```
public XDataSqlException(Exception exception, Query query)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Exception	exception	Inner exception

TYPE	NAME	DESCRIPTION
Query	query	Query

XDataSqlException(Exception, Query, AdditionalInfo[])

SQL exception

Declaration

```
public XDataSqlException(Exception exception, Query query, params AdditionalInfo[] addInfo)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Exception	exception	Inner exception
Query	query	Query
AdditionalInfo[]	addInfo	Additional information

XDataSqlException(Exception, AdditionalInfo[])

SQL exception

Declaration

```
public XDataSqlException(Exception exception, params AdditionalInfo[] addInfo)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Exception	exception	Inner exception
AdditionalInfo[]	addInfo	Additional information

Properties

Query

Query

Declaration

```
public Query Query { get; }
```

Property Value

TYPE	DESCRIPTION
Query	

Implements

System.Runtime.Serialization.ISerializable

Extension Methods

- SerializationExtensions.ToXml<T>(T)
- CollectionsExtensions.SetValue<TKey, T>(TKey, T)
- CollectionsExtensions.AsEnum<T>(T)
- DataObjectQueryableExtensions.CompareVariable(Object, FilterOperation, String)
- Processing.Do<T>(T, Action<IProcess<T>>[])
- TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>)
- TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>, TResult)
- TypeExtensions.ChangeType<T>(Object)
- TypeExtensions.ChangeType(Object, Type)
- TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)
- TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)
- TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)
- TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)
- SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)
- QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Namespace XData.Extensions

Classes

[CollectionsExtensions](#)

Collections extension functions

[DataObjectQueryableExtensions](#)

XData LINQ query extensions

[ExpressionExtensions](#)

Expression extensions

[GeneralPropertyComparer<T>](#)

[IndexAttribute](#)

Attribute to compare objects by property value

[Indexer<T>](#)

Indexed objects comparer

Hint: Objects can be indexed by annotation attribute [IndexAttribute](#)

[JoinedEnumerable](#)

Joined enumerable extensions class

[JoinedEnumerable<T>](#)

Helper class to use outer joins

[Processing](#)

Object processing

[TypeComparer](#)

Type equality comparer

[TypeExtensions](#)

Type extensions

[TypeExtensions.DummyType](#)

Interfaces

[IProcess<T>](#)

Object processing interface

Class CollectionsExtensions

Collections extension functions

Inheritance

System.Object

CollectionsExtensions

Inherited Members

System.Object.Equals(System.Object)

System.Object.Equals(System.Object, System.Object)

System.Object.GetHashCode()

System.Object.GetType()

System.Object.MemberwiseClone()

System.Object.ReferenceEquals(System.Object, System.Object)

System.Object.ToString()

Namespace: [XData.Extensions](#)

Assembly: XData.docfx.dll

Syntax

```
public static class CollectionsExtensions
```

Methods

AsEnum<T>(T)

Returns single item enumerable

Declaration

```
public static IEnumerable<T> AsEnum<T>(this T item)
```

Parameters

TYPE	NAME	DESCRIPTION
T	item	Item

Returns

TYPE	DESCRIPTION
System.Collections.Generic.IEnumerable<T>	Single item enumerable

Type Parameters

NAME	DESCRIPTION
T	Item type

Clear<T>(List<T>, Action<T>)

Clear list content with action call on every element

Declaration

```
public static void Clear<T>(this List<T> list, Action<T> action)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Collections.Generic.List<T>	list	List
System.Action<T>	action	Action

Type Parameters

NAME	DESCRIPTION
T	List element type

GetOrAdd<T>(IList<T>, Func<T, Boolean>, Func<T>)

Returns first found element by predicate or add selector result and return it

Declaration

```
public static T GetOrAdd<T>(this IList<T> list, Func<T, bool> predicate, Func<T> selector)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Collections.Generic.IList<T>	list	List
System.Func<T, System.Boolean>	predicate	Find element predicate
System.Func<T>	selector	Create element logic

Returns

TYPE	DESCRIPTION
T	Found or crested element

Type Parameters

NAME	DESCRIPTION
T	Element type

Exceptions

TYPE	CONDITION
XDataRuntimeException	Argument is null

HasSameValuesAs(ICollection<TaggedValue>, ICollection<TaggedValue>)

Dictionary comparison test

Declaration

```
public static bool HasSameValuesAs(this ICollection<TaggedValue> source, ICollection<TaggedValue> other)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Collections.Generic.ICollection< TaggedValue >	source	Source dictionary
System.Collections.Generic.ICollection< TaggedValue >	other	Matched dictionary

Returns

TYPE	DESCRIPTION
System.Boolean	

HasSameValuesAs<TKey, TValue>(ICollection<KeyValuePair<TKey, TValue>>, ICollection<KeyValuePair<TKey, TValue>>)

Dictionary comparison test

Declaration

```
public static bool HasSameValuesAs<TKey, TValue>(this ICollection<KeyValuePair<TKey, TValue>> source,
ICollection<KeyValuePair<TKey, TValue>> other)
    where TValue : IComparable
```

Parameters

TYPE	NAME	DESCRIPTION
System.Collections.Generic.ICollection<System.Collections.Generic.KeyValuePair<TKey, TValue>>	source	Source dictionary
System.Collections.Generic.ICollection<System.Collections.Generic.KeyValuePair<TKey, TValue>>	other	Matched dictionary

Returns

TYPE	DESCRIPTION
System.Boolean	

Type Parameters

NAME	DESCRIPTION
TKey	Type of dictionary key
TValue	Type of dictionary value

HasSameValuesAs<TKey, TValue>(IDictionary<TKey, TValue>, IDictionary<TKey, TValue>)

Dictionary comparison test

Declaration

```
public static bool HasSameValuesAs<TKey, TValue>(this IDictionary<TKey, TValue> source, IDictionary<TKey, TValue> other)
    where TValue : IEquatable<TValue>
```

Parameters

TYPE	NAME	DESCRIPTION
System.Collections.Generic.IDictionary<TKey, TValue>	source	Source dictionary
System.Collections.Generic.IDictionary<TKey, TValue>	other	Matched dictionary

Returns

TYPE	DESCRIPTION
System.Boolean	

Type Parameters

NAME	DESCRIPTION
TKey	Type of dictionary key
TValue	Type of dictionary value

IsEmpty(ICollection<IComparable>)

IComparable array IsEmpty helper

Declaration

```
public static bool IsEmpty(this ICollection<IComparable> fields)
```

Parameters

TYPE	NAME	DESCRIPTION

TYPE	NAME	DESCRIPTION
System.Collections.Generic.ICollection<IComparable>	fields	Array of values

Returns

TYPE	DESCRIPTION
System.Boolean	Any of fields is null or DBNull

RemoveAll<TKey, TValue>(ConcurrentDictionary<TKey, TValue>, Func<TKey, Boolean>)

Remove all elements matched predicate

Declaration

```
public static void RemoveAll<TKey, TValue>(this ConcurrentDictionary<TKey, TValue> collection, Func<TKey, bool> predicate)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Collections.Concurrent.ConcurrentDictionary<TKey, TValue>	collection	Concurrent dictionary
System.Func<TKey, System.Boolean>	predicate	Filtering predicate

Type Parameters

NAME	DESCRIPTION
TKey	Key type
TValue	Value type

SetValue<TKey, T>(TKey, T)

Named value pair builder

Declaration

```
public static KeyValuePair<TKey, T> SetValue<TKey, T>(this TKey name, T value)
```

Parameters

TYPE	NAME	DESCRIPTION
TKey	name	Name

TYPE	NAME	DESCRIPTION
T	value	Value

Returns

TYPE	DESCRIPTION
System.Collections.Generic.KeyValuePair<TKey, T>	Named value pair

Type Parameters

NAME	DESCRIPTION
TKey	Key type
T	Value type

ToDictionary<TKey, TValue>(IEnumerable<KeyValuePair<TKey, TValue>>)

ToDictionary helper

Declaration

```
public static Dictionary<TKey, TValue> ToDictionary<TKey, TValue>(this IEnumerable<KeyValuePair<TKey, TValue>> pairs)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Collections.Generic.IEnumerable<System.Collections.Generic.KeyValuePair<TKey, TValue>>	pairs	KeyValuePair collection

Returns

TYPE	DESCRIPTION
System.Collections.Generic.Dictionary<TKey, TValue>	Typed Dictionary

Type Parameters

NAME	DESCRIPTION
TKey	Dictionary key type
TValue	Dictionary value type

ToDictionary<TKey, TValue>(IEnumerable<IGrouping<TKey, TValue>>)

ToDictionary groupings helper

Declaration

```
public static Dictionary<TKey, List<TValue>> ToDictionary<TKey, TValue>(this IEnumerable<IGrouping<TKey, TValue>> groupings)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Collections.Generic.IEnumerable<IGrouping<TKey, TValue>>	groupings	Groupings

Returns

TYPE	DESCRIPTION
System.Collections.Generic.Dictionary<TKey, System.Collections.Generic.List<TValue>>	Typed Dictionary

Type Parameters

NAME	DESCRIPTION
TKey	Dictionary key type
TValue	Dictionary value type

ToSortedList<TKey, TValue>(IEnumerable<KeyValuePair<TKey, TValue>>)

ToSortedList helper

Declaration

```
public static SortedList<TKey, TValue> ToSortedList<TKey, TValue>(this IEnumerable<KeyValuePair<TKey, TValue>> pairs)
    where TKey : IComparable
```

Parameters

TYPE	NAME	DESCRIPTION
System.Collections.Generic.IEnumerable<System.Collections.Generic.KeyValuePair<TKey, TValue>>	pairs	KeyValuePair collection

Returns

TYPE	DESCRIPTION
System.Collections.Generic.SortedList<TKey, TValue>	Typed SortedList

Type Parameters

NAME	DESCRIPTION
TKey	Dictionary key type
TValue	Dictionary value type

ToSortedList<TKey, TValue>(IEnumerable<IGrouping<TKey, TValue>>)

ToSortedList groupings helper

Declaration

```
public static SortedList<TKey, List<TValue>> ToSortedList<TKey, TValue>(this IEnumerable<IGrouping<TKey,
TValue>> groupings)
    where TKey : IComparable
```

Parameters

TYPE	NAME	DESCRIPTION
System.Collections.Generic.IEnumerable<IGrouping<TKey, TValue> >	groupings	Groupings

Returns

TYPE	DESCRIPTION
System.Collections.Generic.SortedList<TKey, System.Collections.Generic.List<TValue> >	Typed SortedList

Type Parameters

NAME	DESCRIPTION
TKey	SortedList key type
TValue	SortedList value type

ToSortedSet<T>(IEnumerable<T>, IComparer<T>)

ToSortedSet helper

Declaration

```
public static SortedSet<T> ToSortedSet<T>(this IEnumerable<T> source, IComparer<T> comparer = null)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Collections.Generic.IEnumerable<T>	source	Source sequence

TYPE	NAME	DESCRIPTION
System.Collections.Generic.IComparer<T>	comparer	Source type comparer

Returns

TYPE	DESCRIPTION
System.Collections.Generic.SortedSet<T>	Typed SortedSet

Type Parameters

NAME	DESCRIPTION
T	SortedSet type

Remarks

If comparer is not set [GetInstance\(\)](#) used

WARNING! To use [Indexer<T>](#) one of System.IComparable<T> properties of source type must be marked by [IndexAttribute](#)

TryGetValue<TKey, TValue>(IEnumerable<KeyValuePair<TKey, TValue>>, TKey)

Try get value helper

Declaration

```
public static object TryGetValue<TKey, TValue>(this IEnumerable<KeyValuePair<TKey, TValue>> source, TKey key)
    where TKey : IEquatable<TKey>
```

Parameters

TYPE	NAME	DESCRIPTION
System.Collections.Generic.IEnumerable<System.Collections.Generic.KeyValuePair<TKey, TValue> >	source	Sequence of key value pairs
TKey	key	Key value

Returns

TYPE	DESCRIPTION
System.Object	

Type Parameters

NAME	DESCRIPTION
TKey	Key type

NAME	DESCRIPTION
TValue	Value type

Class DataObjectQueryableExtensions

XData LINQ query extensions

Inheritance

System.Object
DataObjectQueryableExtensions

Inherited Members

System.Object.Equals(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.GetHashCode()
System.Object.GetType()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.ToString()

Namespace: [XData.Extensions](#)

Assembly: XData.docfx.dll

Syntax

```
public static class DataObjectQueryableExtensions
```

Methods

CompareToCurrentDate(DateTime, FilterOperation)

Compare value to current date

Declaration

```
public static bool CompareToCurrentDate(this DateTime source, FilterOperation operation)
```

Parameters

TYPE	NAME	DESCRIPTION
System.DateTime	source	Source property
FilterOperation	operation	Comparison operation

Returns

TYPE	DESCRIPTION
System.Boolean	Comparison result

Exceptions

TYPE	CONDITION
XDataQueryStructureException	LINQ expression has unsupported method parameters

See Also

[CompareToCurrentDateTime\(DateTime, FilterOperation\)](#)
[CompareToCurrentDateTimeUtc\(DateTime, FilterOperation\)](#)

CompareToCurrentDate(Nullable<DateTime>, FilterOperation)

Compare value to current date

Declaration

```
public static bool CompareToCurrentDate(this DateTime? source, FilterOperation operation)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Nullable<System.DateTime>	source	Source property
FilterOperation	operation	Comparison operation

Returns

TYPE	DESCRIPTION
System.Boolean	Comparison result

Exceptions

TYPE	CONDITION
XDataQueryStructureException	LINQ expression has unsupported method parameters

See Also

[CompareToCurrentDate\(DateTime, FilterOperation\)](#)
[CompareToCurrentDateTime\(DateTime, FilterOperation\)](#)
[CompareToCurrentDateTimeUtc\(DateTime, FilterOperation\)](#)

CompareToCurrentDateTime(DateTime, FilterOperation)

Compare value to current date and time

Declaration

```
public static bool CompareToCurrentDateTime(this DateTime source, FilterOperation operation)
```

Parameters

TYPE	NAME	DESCRIPTION
System.DateTime	source	Source property
FilterOperation	operation	Comparison operation

Returns

TYPE	DESCRIPTION
System.Boolean	Comparison result

Exceptions

TYPE	CONDITION
XDataQueryStructureException	LINQ expression has unsupported method parameters

See Also

[CompareToCurrentDate\(DateTime, FilterOperation\)](#)

[CompareToCurrentDateTimeUtc\(DateTime, FilterOperation\)](#)

[CompareToCurrentDateTime\(Nullable<DateTime>, FilterOperation\)](#)

Compare value to current date and time

Declaration

```
public static bool CompareToCurrentDateTime(this DateTime? source, FilterOperation operation)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Nullable<System.DateTime>	source	Source property
FilterOperation	operation	Comparison operation

Returns

TYPE	DESCRIPTION
System.Boolean	Comparison result

Exceptions

TYPE	CONDITION
XDataQueryStructureException	LINQ expression has unsupported method parameters

See Also

[CompareToCurrentDate\(DateTime, FilterOperation\)](#)

[CompareToCurrentDateTime\(DateTime, FilterOperation\)](#)

[CompareToCurrentDateTimeUtc\(DateTime, FilterOperation\)](#)

[CompareToCurrentDateTimeUtc\(DateTime, FilterOperation\)](#)

Compare value to current date and time UTC

Declaration

```
public static bool CompareToCurrentDateTimeUtc(this DateTime source, FilterOperation operation)
```

Parameters

TYPE	NAME	DESCRIPTION
System.DateTime	source	Source property
FilterOperation	operation	Comparison operation

Returns

TYPE	DESCRIPTION
System.Boolean	Comparison result

Exceptions

TYPE	CONDITION
XDataQueryStructureException	LINQ expression has unsupported method parameters

See Also

[CompareToCurrentDate\(DateTime, FilterOperation\)](#)

[CompareToCurrentDateTime\(DateTime, FilterOperation\)](#)

CompareToCurrentDateTimeUtc(Nullable<DateTime>, FilterOperation)

Compare value to current date and time UTC

Declaration

```
public static bool CompareToCurrentDateTimeUtc(this DateTime? source, FilterOperation operation)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Nullable<System.DateTime>	source	Source property
FilterOperation	operation	Comparison operation

Returns

TYPE	DESCRIPTION
System.Boolean	Comparison result

Exceptions

TYPE	CONDITION
XDataQueryStructureException	LINQ expression has unsupported method parameters

See Also

[CompareToCurrentDate\(DateTime, FilterOperation\)](#)

[CompareToCurrentDateTime\(DateTime, FilterOperation\)](#)

[CompareToCurrentDateTimeUtc\(DateTime, FilterOperation\)](#)

CompareVariable(Object, FilterOperation, String)

Compare value to variable value WARNING! Method accessible only in XData LINQ expressions!

Declaration

```
public static bool CompareVariable(this object source, FilterOperation operation, string variableName)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Object	source	Source property
FilterOperation	operation	Comparison operation
System.String	variableName	Variable name

Returns

TYPE	DESCRIPTION
System.Boolean	Comparison result

Exceptions

TYPE	CONDITION
XDataRuntimeException	Function is accessible in XData LINQ expressions only

Like(String, String)

Like

Declaration

```
public static bool Like(this string source, string pattern)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	source	Source property
System.String	pattern	Like pattern

Returns

TYPE	DESCRIPTION
System.Boolean	Like result

Exceptions

TYPE	CONDITION
XDataRuntimeException	Like pattern can not be null or empty

See Also

[Like\(String, String, Char\)](#)

Like(String, String, Char)

Like

Declaration

```
public static bool Like(this string source, string pattern, char escapeChar)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	source	Source property
System.String	pattern	Like pattern
System.Char	escapeChar	Escape character

Returns

TYPE	DESCRIPTION
System.Boolean	Like result

Exceptions

TYPE	CONDITION
XDataRuntimeException	Like pattern can not be null or empty

See Also

[Like\(String, String\)](#)

Class ExpressionExtensions

Expression extensions

Inheritance

System.Object

ExpressionExtensions

Inherited Members

System.Object.Equals(System.Object)

System.Object.Equals(System.Object, System.Object)

System.Object.GetHashCode()

System.Object.GetType()

System.Object.MemberwiseClone()

System.Object.ReferenceEquals(System.Object, System.Object)

System.Object.ToString()

Namespace: [XData.Extensions](#)

Assembly: XData.docfx.dll

Syntax

```
public static class ExpressionExtensions
```

Class GeneralPropertyComparer<T>

Inheritance

System.Object
GeneralPropertyComparer<T>

Implements

System.Collections.Generic.IEqualityComparer<T>

Inherited Members

System.Object.Equals(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.GetHashCode()
System.Object.GetType()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.ToString()

Namespace: [XData.Extensions](#)

Assembly: XData.docfx.dll

Syntax

```
public class GeneralPropertyComparer<T> : IEqualityComparer<T>
```

Type Parameters

NAME	DESCRIPTION
T	

Constructors

GeneralPropertyComparer(Func<T, Object>[])

Declaration

```
public GeneralPropertyComparer(Func<T, object>[] expr)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Func<T, System.Object>[]	expr	

Methods

Equals(T, T)

Declaration

```
public bool Equals(T left, T right)
```

Parameters

TYPE	NAME	DESCRIPTION
T	left	
T	right	

Returns

TYPE	DESCRIPTION
System.Boolean	

GetHashCode(T)

Declaration

```
public int GetHashCode(T obj)
```

Parameters

TYPE	NAME	DESCRIPTION
T	obj	

Returns

TYPE	DESCRIPTION
System.Int32	

Implements

System.Collections.Generic.IEqualityComparer<T>

Extension Methods

- SerializationExtensions.ToXml<T>(T)
- CollectionsExtensions.SetValue<TKey, T>(TKey, T)
- CollectionsExtensions.AsEnum<T>(T)
- DataObjectQueryableExtensions.CompareVariable(Object, FilterOperation, String)
- Processing.Do<T>(T, Action<IProcess<T>>[])
- TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>)
- TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>, TResult)
- TypeExtensions.ChangeType<T>(Object)
- TypeExtensions.ChangeType(Object, Type)
- TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)
- TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)
- TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)
- TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)
- SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)
- QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Class IndexAttribute

Attribute to compare objects by property value

Inheritance

System.Object
System.Attribute
IndexAttribute

Inherited Members

System.Attribute.Equals(System.Object)
System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type)
System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type, System.Boolean)
System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type)
System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type, System.Boolean)
System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type)
System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type, System.Boolean)
System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type)
System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.Assembly)
System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type)
System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo)
System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type)
System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.Module)
System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type)
System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo)
System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type)
System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type, System.Boolean)
System.Attribute.GetHashCode()
System.Attribute.IsDefaultAttribute()
System.Attribute.IsDefined(System.Reflection.Assembly, System.Type)
System.Attribute.IsDefined(System.Reflection.Assembly, System.Type, System.Boolean)
System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type)
System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type, System.Boolean)
System.Attribute.IsDefined(System.Reflection.Module, System.Type)
System.Attribute.IsDefined(System.Reflection.Module, System.Type, System.Boolean)
System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type)
System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type, System.Boolean)
System.Attribute.Match(System.Object)
System.Attribute.TypeId
System.Object.Equals(System.Object, System.Object)
System.Object.GetType()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.ToString()

Namespace: [XData.Extensions](#)

Syntax

```
[AttributeUsage(AttributeTargets.Property)]  
public class IndexAttribute : Attribute
```

Extension Methods

[SerializationExtensions.ToXml<T>\(T\)](#)

[CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)

[CollectionsExtensions.AsEnum<T>\(T\)](#)

[DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)

[Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)

[TypeExtensions.ChangeType<T>\(Object\)](#)

[TypeExtensions.ChangeType\(Object, Type\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)

[SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)

[QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Class Indexer<T>

Indexed objects comparer
Hint: Objects can be indexed by annotation attribute [IndexAttribute](#)

Inheritance

System.Object
Indexer<T>

Implements

System.Collections.Generic.IComparer<T>
System.Collections.Generic.IEqualityComparer<T>

Inherited Members

System.Object.Equals(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.GetHashCode()
System.Object.GetType()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.ToString()

Namespace: [XData.Extensions](#)

Assembly: XData.docfx.dll

Syntax

```
public class Indexer<T> : IIndexer, IComparer<T>, IEqualityComparer<T>
```

Type Parameters

NAME	DESCRIPTION
T	Indexed object type

Constructors

Indexer(String)

Indexer parametrized constructor

Declaration

```
public Indexer(string propertyName)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	propertyName	Comparable property name

Methods

Compare(T, T)

Compares two objects and returns a value indicating whether one is less than, equal to, or greater than the other.

Declaration

```
public int Compare(T x, T y)
```

Parameters

TYPE	NAME	DESCRIPTION
T	x	The first object to compare.
T	y	The second object to compare.

Returns

TYPE	DESCRIPTION
System.Int32	A signed integer that indicates the relative values of <code>x</code> and <code>y</code> , as shown in the following table. Value Meaning Less than zero <code>x</code> is less than <code>y</code> . Zero <code>x</code> equals <code>y</code> . Greater than zero <code>x</code> is greater than <code>y</code> .

Exceptions

TYPE	CONDITION
XDataRuntimeException	IndexedAttribute not found

Equals(T, T)

Determines whether the specified objects are equal.

Declaration

```
public bool Equals(T x, T y)
```

Parameters

TYPE	NAME	DESCRIPTION
T	x	The first object to compare.
T	y	The second object to compare.

Returns

TYPE	DESCRIPTION
System.Boolean	true if the specified objects are equal; otherwise, false.

Exceptions

TYPE	CONDITION
XDataRuntimeException	IndexedAttribute not found

GetHashCode(T)

Returns a hash code for the specified object.

Declaration

```
public int GetHashCode(T obj)
```

Parameters

TYPE	NAME	DESCRIPTION
T	obj	The System.Object for which a hash code is to be returned.

Returns

TYPE	DESCRIPTION
System.Int32	A hash code for the specified object.

Exceptions

TYPE	CONDITION
System.ArgumentNullException	The type of <code>obj</code> is a reference type and <code>obj</code> is null.
XDataRuntimeException	IndexedAttribute not found

GetInstance()

Singleton comparer instance

Declaration

```
public static Indexer<T> GetInstance()
```

Returns

TYPE	DESCRIPTION
Indexer<T>	Comparer instance

Implements

- System.Collections.Generic.IComparer<T>
- System.Collections.Generic.IEqualityComparer<T>

Extension Methods

SerializationExtensions.ToXml<T>(T)

CollectionsExtensions.SetValue<TKey, T>(TKey, T)

CollectionsExtensions.AsEnum<T>(T)

DataObjectQueryableExtensions.CompareVariable(Object, FilterOperation, String)

Processing.Do<T>(T, Action<IProcess<T>>[])

TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>)

TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>, TResult)

TypeExtensions.ChangeType<T>(Object)

TypeExtensions.ChangeType(Object, Type)

TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)

TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)

SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)

QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Interface IProcess<T>

Object processing interface

Namespace: [XData.Extensions](#)

Assembly: XData.docfx.dll

Syntax

```
public interface IProcess<T>
```

Type Parameters

NAME	DESCRIPTION
T	Object type

Properties

Instance

Processing source object instance

Declaration

```
T Instance { get; set; }
```

Property Value

TYPE	DESCRIPTION
T	

State

Intermediate object processing status

Declaration

```
dynamic State { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Object	

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)

TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)

TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)

SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)

QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Class JoinedEnumerable

Joined enumerable extensions class

Inheritance

System.Object
JoinedEnumerable

Inherited Members

System.Object.Equals(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.GetHashCode()
System.Object.GetType()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.ToString()

Namespace: [XData.Extensions](#)
Assembly: XData.docfx.dll

Syntax

```
public static class JoinedEnumerable
```

Methods

FullOuterJoin<TOuter, TInner, TKey, TResult>(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter, TKey>, Func<TInner, TKey>, Func<TOuter, TInner, TResult>, IEqualityComparer<TKey>)

Full outer join extension

Declaration

```
public static IEnumerable<TResult> FullOuterJoin<TOuter, TInner, TKey, TResult>(this IEnumerable<TOuter> outer, IEnumerable<TInner> inner, Func<TOuter, TKey> outerKeySelector, Func<TInner, TKey> innerKeySelector, Func<TOuter, TInner, TResult> resultSelector, IEqualityComparer<TKey> comparer = null)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Collections.Generic.IEnumerable<TOuter>	outer	Outer enumerable
System.Collections.Generic.IEnumerable<TInner>	inner	Inner enumerable
System.Func<TOuter, TKey>	outerKeySelector	Outer key selector
System.Func<TInner, TKey>	innerKeySelector	Inner key selector
System.Func<TOuter, TInner, TResult>	resultSelector	Result selector

TYPE	NAME	DESCRIPTION
System.Collections.Generic.IEqualityComparer<TKey>	comparer	Key type comparer

Returns

TYPE	DESCRIPTION
System.Collections.Generic.IEnumerable<TResult>	Joined sequence

Type Parameters

NAME	DESCRIPTION
TOuter	Outer enumerable type
TInner	Inner enumerable type
TKey	Join key type
TResult	Result sequence type

See Also

[Join<TOuter, TInner, TKey, TResult>\(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter, TKey>, Func<TInner, TKey>, Func<TOuter, TInner, TResult>, IEqualityComparer<TKey>\)](#)

Inner<TElement>(IEnumerable<TElement>)

Mark enumerable as inner

Declaration

```
public static JoinedEnumerable<TElement> Inner<TElement>(this IEnumerable<TElement> source)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Collections.Generic.IEnumerable<TElement>	source	Source sequence

Returns

TYPE	DESCRIPTION
JoinedEnumerable<TElement>	Joined inner enumerable

Type Parameters

NAME	DESCRIPTION
TElement	Element type

See Also

Outer<TElement>(IEnumerable<TElement>)
Join<TOuter, TInner, TKey, TResult>(JoinedEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter, TKey>, Func<TInner, TKey>, Func<TOuter, TInner, TResult>, IEqualityComparer<TKey>)

Join<TOuter, TInner, TKey, TResult>(JoinedEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter, TKey>, Func<TInner, TKey>, Func<TOuter, TInner, TResult>, IEqualityComparer<TKey>)

Join extension for Joined enumerable

Declaration

```
public static IEnumerable<TResult> Join<TOuter, TInner, TKey, TResult>(this JoinedEnumerable<TOuter> outer,
IEnumerable<TInner> inner, Func<TOuter, TKey> outerKeySelector, Func<TInner, TKey> innerKeySelector,
Func<TOuter, TInner, TResult> resultSelector, IEqualityComparer<TKey> comparer = null)
```

Parameters

TYPE	NAME	DESCRIPTION
JoinedEnumerable<TOuter>	outer	Outer enumerable
System.Collections.Generic.IEnumerable<TInner>	inner	Inner enumerable
System.Func<TOuter, TKey>	outerKeySelector	Outer key selector
System.Func<TInner, TKey>	innerKeySelector	Inner key selector
System.Func<TOuter, TInner, TResult>	resultSelector	Result selector
System.Collections.Generic.IEqualityComparer<TKey>	comparer	Key type comparer

Returns

TYPE	DESCRIPTION
System.Collections.Generic.IEnumerable<TResult>	Joined sequence

Type Parameters

NAME	DESCRIPTION

NAME	DESCRIPTION
TOuter	Outer enumerable type
TInner	Inner enumerable type
TKey	Join key type
TResult	Result sequence type

Exceptions

TYPE	CONDITION
XDataRuntimeException	Argument null

See Also

[Inner<TElement>\(IEnumerable<TElement>\)](#)
[Outer<TElement>\(IEnumerable<TElement>\)](#)
[LeftOuterJoin<TOuter, TInner, TKey, TResult>\(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter, TKey>, Func<TInner, TKey>, Func<TOuter, TInner, TResult>, IEqualityComparer<TKey>\)](#)
[RightOuterJoin<TOuter, TInner, TKey, TResult>\(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter, TKey>, Func<TInner, TKey>, Func<TOuter, TInner, TResult>, IEqualityComparer<TKey>\)](#)
[FullOuterJoin<TOuter, TInner, TKey, TResult>\(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter, TKey>, Func<TInner, TKey>, Func<TOuter, TInner, TResult>, IEqualityComparer<TKey>\)](#)

[LeftOuterJoin<TOuter, TInner, TKey, TResult>\(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter, TKey>, Func<TInner, TKey>, Func<TOuter, TInner, TResult>, IEqualityComparer<TKey>\)](#)

Left outer join extension

Declaration

```
public static IEnumerable<TResult> LeftOuterJoin<TOuter, TInner, TKey, TResult>(this IEnumerable<TOuter>
outer, IEnumerable<TInner> inner, Func<TOuter, TKey> outerKeySelector, Func<TInner, TKey> innerKeySelector,
Func<TOuter, TInner, TResult> resultSelector, IEqualityComparer<TKey> comparer = null)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Collections.Generic.IEnumerable<TOuter>	outer	Outer enumerable
System.Collections.Generic.IEnumerable<TInner>	inner	Inner enumerable
System.Func<TOuter, TKey>	outerKeySelector	Outer key selector

TYPE	NAME	DESCRIPTION
System.Func<TInner, TKey>	innerKeySelector	Inner key selector
System.Func<TOuter, TInner, TResult>	resultSelector	Result selector
System.Collections.Generic.IEqualityComparer<TKey>	comparer	Key type comparer

Returns

TYPE	DESCRIPTION
System.Collections.Generic.IEnumerable<TResult>	Joined sequence

Type Parameters

NAME	DESCRIPTION
TOuter	Outer enumerable type
TInner	Inner enumerable type
TKey	Join key type
TResult	Result sequence type

See Also

[Join<TOuter, TInner, TKey, TResult>\(JoinedEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter, TKey>, Func<TInner, TKey>, Func<TOuter, TInner, TResult>, IEqualityComparer<TKey>\)](#)

Outer<TElement>(IEnumerable<TElement>)

Mark enumerable as outer

Declaration

```
public static JoinedEnumerable<TElement> Outer<TElement>(this IEnumerable<TElement> source)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Collections.Generic.IEnumerable<TElement>	source	Source sequence

Returns

TYPE	DESCRIPTION
JoinedEnumerable<TElement>	Joined inner enumerable

Type Parameters

NAME	DESCRIPTION
TElement	Element type

See Also

[Inner<TElement>\(IEnumerable<TElement>\)](#)
[Join<TOuter, TInner, TKey, TResult>\(JoinedEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter, TKey>, Func<TInner, TKey>, Func<TOuter, TInner, TResult>, IEqualityComparer<TKey>\)](#)

[RightOuterJoin<TOuter, TInner, TKey, TResult>\(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter, TKey>, Func<TInner, TKey>, Func<TOuter, TInner, TResult>, IEqualityComparer<TKey>\)](#)

Right outer join extension

Declaration

```
public static IEnumerable<TResult> RightOuterJoin<TOuter, TInner, TKey, TResult>(this IEnumerable<TOuter>
outer, IEnumerable<TInner> inner, Func<TOuter, TKey> outerKeySelector, Func<TInner, TKey> innerKeySelector,
Func<TOuter, TInner, TResult> resultSelector, IEqualityComparer<TKey> comparer = null)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Collections.Generic.IEnumerable<TOuter>	outer	Outer enumerable
System.Collections.Generic.IEnumerable<TInner>	inner	Inner enumerable
System.Func<TOuter, TKey>	outerKeySelector	Outer key selector
System.Func<TInner, TKey>	innerKeySelector	Inner key selector
System.Func<TOuter, TInner, TResult>	resultSelector	Result selector
System.Collections.Generic.IEqualityComparer<TKey>	comparer	Key type comparer

Returns

TYPE	DESCRIPTION

TYPE	DESCRIPTION
System.Collections.Generic.IEnumerable<TResult>	Joined sequence

Type Parameters

NAME	DESCRIPTION
TOuter	Outer enumerable type
TInner	Inner enumerable type
TKey	Join key type
TResult	Result sequence type

See Also

[Join<TOuter, TInner, TKey, TResult>](#) ([JoinedEnumerable<TOuter>](#), [IEnumerable<TInner>](#), [Func<TOuter, TKey>](#), [Func<TInner, TKey>](#), [Func<TOuter, TInner, TResult>](#), [IEqualityComparer<TKey>](#))

See Also

[JoinedEnumerable<T>](#)

Class JoinedEnumerable<T>

Helper class to use outer joins

Inheritance

System.Object
JoinedEnumerable<T>

Implements

System.Collections.Generic.IEnumerable<T>
System.Collections.IEnumerable

Inherited Members

System.Object.Equals(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.GetHashCode()
System.Object.GetType()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.ToString()

Namespace: [XData.Extensions](#)

Assembly: XData.docfx.dll

Syntax

```
public class JoinedEnumerable<T> : IEnumerable<T>, IEnumerable
```

Type Parameters

NAME	DESCRIPTION
T	Element type

Constructors

JoinedEnumerable(IEnumerable<T>)

Joined enumerable constructor

Declaration

```
public JoinedEnumerable(IEnumerable<T> source)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Collections.Generic.IEnumerable<T>	source	

Fields

IsOuter

Outer joined enumerable

Declaration

```
public bool IsOuter
```

Field Value

TYPE	DESCRIPTION
System.Boolean	

Explicit Interface Implementations

IEnumerable<T>.GetEnumerator()

Declaration

IEnumerator<T> IEnumerable<T>.GetEnumerator()

Returns

TYPE	DESCRIPTION
System.Collections.Generic.IEnumerator<T>	

IEnumerable.GetEnumerator()

Declaration

IEnumerator IEnumerable.GetEnumerator()

Returns

TYPE	DESCRIPTION
System.Collections.IEnumerator	

Implements

System.Collections.Generic.IEnumerable<T>
System.Collections.IEnumerable

Extension Methods

- SerializationExtensions.ToXml<T>(T)
- CollectionsExtensions.ToSortedSet<T>(IEnumerable<T>, IComparer<T>)
- CollectionsExtensions.SetValue<TKey, T>(TKey, T)
- CollectionsExtensions.AsEnum<T>(T)
- DataObjectQueryableExtensions.CompareVariable(Object, FilterOperation, String)
- JoinedEnumerable.Inner<TElement>(IEnumerable<TElement>)
- JoinedEnumerable.Outer<TElement>(IEnumerable<TElement>)
- JoinedEnumerable.Join<TOuter, TInner, TKey, TResult>(JoinedEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter, TKey>, Func<TInner, TKey>, Func<TOuter, TInner, TResult>, IEqualityComparer<TKey>)
- JoinedEnumerable.LeftOuterJoin<TOuter, TInner, TKey, TResult>(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter, TKey>, Func<TInner, TKey>, Func<TOuter, TInner, TResult>, IEqualityComparer<TKey>)
- JoinedEnumerable.RightOuterJoin<TOuter, TInner, TKey, TResult>(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter, TKey>, Func<TInner, TKey>, Func<TOuter, TInner, TResult>, IEqualityComparer<TKey>)
- JoinedEnumerable.FullOuterJoin<TOuter, TInner, TKey, TResult>(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter, TKey>, Func<TInner, TKey>, Func<TOuter, TInner, TResult>, IEqualityComparer<TKey>)
- Processing.Do<T>(T, Action<IProcess<T>>[])
- TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>)
- TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>, TResult)
- TypeExtensions.ChangeType<T>(Object)
- TypeExtensions.ChangeType(Object, Type)

TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)

TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)

SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)

QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

QueryDescriptionExtensions.ToDataSet<T>(IEnumerable<T>, String, ISqlBuilder)

Class Processing

Object processing

Inheritance

System.Object

Processing

Inherited Members

- System.Object.Equals(System.Object)
- System.Object.Equals(System.Object, System.Object)
- System.Object.GetHashCode()
- System.Object.GetType()
- System.Object.MemberwiseClone()
- System.Object.ReferenceEquals(System.Object, System.Object)
- System.Object.ToString()

Namespace: [XData.Extensions](#)

Assembly: XData.docfx.dll

Syntax

```
public static class Processing
```

Methods

Do<T>(T, Action<IProcess<T>>[])

Object processing method

Declaration

```
public static T Do<T>(this T source, params Action<IProcess<T>>[] actions)
```

Parameters

TYPE	NAME	DESCRIPTION
T	source	Source object
System.Action< IProcess <T>>[]	actions	Actions of processing

Returns

TYPE	DESCRIPTION
T	Source object after processing

Type Parameters

NAME	DESCRIPTION
T	Object type

Class TypeComparer

Type equality comparer

Inheritance

System.Object
TypeComparer

Implements

System.Collections.Generic.IEqualityComparer<System.Type>

Inherited Members

System.Object.Equals(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.GetHashCode()
System.Object.GetType()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.ToString()

Namespace: [XData.Extensions](#)

Assembly: XData.docfx.dll

Syntax

```
public class TypeComparer : IEqualityComparer<Type>
```

Properties

Instance

Type equality comparer instance

Declaration

```
public static TypeComparer Instance { get; }
```

Property Value

TYPE	DESCRIPTION
TypeComparer	

Methods

Equals(Type, Type)

Determines whether the specified types are equal.

Declaration

```
public bool Equals(Type x, Type y)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Type	x	The first type to compare.

TYPE	NAME	DESCRIPTION
System.Type	y	The second type to compare.

Returns

TYPE	DESCRIPTION
System.Boolean	true if the specified types are equal; otherwise, false.

GetHashCode(Type)

Returns a hash code for the specified type.

Declaration

```
public int GetHashCode(Type obj)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Type	obj	The type for which a hash code is to be returned.

Returns

TYPE	DESCRIPTION
System.Int32	A hash code for the specified type.

Implements

System.Collections.Generic.IEqualityComparer<T>

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Class TypeExtensions

Type extensions

Inheritance

System.Object

TypeExtensions

Inherited Members

- System.Object.Equals(System.Object)
- System.Object.Equals(System.Object, System.Object)
- System.Object.GetHashCode()
- System.Object.GetType()
- System.Object.MemberwiseClone()
- System.Object.ReferenceEquals(System.Object, System.Object)
- System.Object.ToString()

Namespace: [XData.Extensions](#)

Assembly: XData.docfx.dll

Syntax

```
public static class TypeExtensions
```

Methods

ArrayCast(Type, ICollection)

Cast object array to specific type array

Declaration

```
public static object ArrayCast(this Type elementType, ICollection collection)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Type	elementType	Type of result array elements
System.Collections.ICollection	collection	Source collection

Returns

TYPE	DESCRIPTION
System.Object	Typed array

ChangeType(Object, Type)

Type convert

Declaration

```
public static object ChangeType(this object value, Type conversion)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Object	value	Value to convert
System.Type	conversion	Target type

Returns

TYPE	DESCRIPTION
System.Object	Target converted value

ChangeType<T>(Object)

Type convert

Declaration

```
public static T ChangeType<T>(this object value)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Object	value	Value to convert

Returns

TYPE	DESCRIPTION
T	Target converted value

Type Parameters

NAME	DESCRIPTION
T	Target type

ConvertTo<T>(Enum)

Convert enumeration value to another enumeration by hash code

Declaration

```
public static T ConvertTo<T>(this Enum source)
```

Parameters

TYPE	NAME	DESCRIPTION

TYPE	NAME	DESCRIPTION
System.Enum	source	Source enumeration value

Returns

TYPE	DESCRIPTION
T	Target enumeration value

Type Parameters

NAME	DESCRIPTION
T	Target enumeration type

EqualValue(Enum, Enum)

Compare Enumeration values by hash codes

Declaration

```
public static bool EqualValue(this Enum source, Enum other)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Enum	source	Source enumeration value
System.Enum	other	Other enumeration value

Returns

TYPE	DESCRIPTION
System.Boolean	Values are equal

GetDefault(Type)

Get type default value

Declaration

```
public static object GetDefault(this Type type)
```

Parameters

TYPE	NAME	DESCRIPTION

TYPE	NAME	DESCRIPTION
System.Type	type	Type

Returns

TYPE	DESCRIPTION
System.Object	Default value

GetDefault<TType>()

Get type default value

Declaration

```
public static object GetDefault<TType>()
```

Returns

TYPE	DESCRIPTION
System.Object	Default value

Type Parameters

NAME	DESCRIPTION
TType	Type

GetEnumKeyValuePairs(Type)

Declaration

```
public static IEnumerable<KeyValuePair<long, string>> GetEnumKeyValuePairs(this Type enumType)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Type	enumType	

Returns

TYPE	DESCRIPTION
System.Collections.Generic.IEnumerable<System.Collections.Generic.KeyValuePair<System.Int64, System.String>>	

GetEnumValue(Type, Int64)

Returns enum id and value pair for specified id

Declaration

```
public static KeyValuePair<long, string>? GetEnumValue(this Type enumType, long id)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Type	enumType	Enum type
System.Int64	id	Enum key

Returns

TYPE	DESCRIPTION
System.Nullable<System.Collections.Generic.KeyValuePair<System.Int64, System.String>>	Enum id and value pair

GetGenericMethod(Type, String, IEnumerable<Type>)

Returns generic method with typed arguments

Declaration

```
public static MethodInfo GetGenericMethod(this Type type, string name, IEnumerable<Type> parameterTypes)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Type	type	Method type
System.String	name	Method name
System.Collections.Generic.IEnumerable<System.Type>	parameterTypes	Parameter types

Returns

TYPE	DESCRIPTION
System.Reflection.MethodInfo	Reflection method info

Exceptions

TYPE	CONDITION
XDataRuntimeException	Method name can not be null or empty
XDataRuntimeException	Method parameter types can not be null

GetLocalizedEnumKeyValuePairs(Type)

Returns id and name pairs for all enum values

Declaration

```
public static IEnumerable<KeyValuePair<long, string>> GetLocalizedEnumKeyValuePairs(this Type enumType)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Type	enumType	Enum type

Returns

TYPE	DESCRIPTION
System.Collections.Generic.IEnumerable<System.Collections.Generic.KeyValuePair<System.Int64, System.String>>	Sequence of id and name pairs for all enum values

GetLocalizedEnumValue(Type, Int64)

Returns enum id and localized value pair for specified id

Declaration

```
public static KeyValuePair<long, string>? GetLocalizedEnumValue(this Type enumType, long id)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Type	enumType	Enum type
System.Int64	id	Enum key

Returns

TYPE	DESCRIPTION
System.Nullable<System.Collections.Generic.KeyValuePair<System.Int64, System.String>>	Enum id and localized value pair

GetRealType(Type)

Get Generic base type

Declaration

```
public static Type GetRealType(this Type type)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Type	type	Type

Returns

TYPE	DESCRIPTION
System.Type	Base type

Implements<T>(Type, T)

Test type implements an interface

Declaration

```
public static bool Implements<T>(this Type type, T interfaceType)
    where T : Type
```

Parameters

TYPE	NAME	DESCRIPTION
System.Type	type	Type
T	interfaceType	Interface

Returns

TYPE	DESCRIPTION
System.Boolean	Type is implements an interface

Type Parameters

NAME	DESCRIPTION
T	Interface type

InvokeMethod(Object, Expression<Action>, BindingFlags)

Invoke method with replacing base object type wrapper

Declaration

```
public static void InvokeMethod(this object obj, Expression<Action> expr, BindingFlags bindingFlags =
    BindingFlags.Default)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Object	obj	Object to invoke method with
System.Linq.Expressions.Expression< Action >	expr	Call expression
System.Reflection.BindingFlags	bindingFlags	Binding flags

Exceptions

TYPE	CONDITION
XDataRuntimeException	

InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)

Invoke generic method wrapper

Declaration

```
public static void InvokeMethod(this object obj, Type[] types, Expression<Action> expr, BindingFlags bindingFlags = BindingFlags.Default)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Object	obj	Object to invoke method (set null to call static method)
System.Type[]	types	Generic arguments types array
System.Linq.Expressions.Expression< Action >	expr	Call expression (set generic arguments to any valid types)
System.Reflection.BindingFlags	bindingFlags	Binding flags

Exceptions

TYPE	CONDITION
XDataRuntimeException	

InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)

Invoke method with replacing base object type wrapper

Declaration

```
public static TRet InvokeMethod<TRet>(this object obj, Expression<Func<TRet>> expr, BindingFlags bindingFlags = BindingFlags.Default)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Object	obj	Object to invoke method with
System.Linq.Expressions.Expression<System.Func<TRet>>	expr	Call expression
System.Reflection.BindingFlags	bindingFlags	Binding flags

Returns

TYPE	DESCRIPTION
TRet	Call return value

Type Parameters

NAME	DESCRIPTION
TRet	Return value type

Exceptions

TYPE	CONDITION
XDataRuntimeException	

InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)

Invoke generic method wrapper

Declaration

```
public static TRet InvokeMethod<TRet>(this object obj, Type[] types, Expression<Func<TRet>> expr, BindingFlags bindingFlags = BindingFlags.Default)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Object	obj	Object to invoke method (set null to call static method)
System.Type[]	types	Generic arguments types array
System.Linq.Expressions.Expression<System.Func<TRet>>	expr	Call expression (set generic arguments to any valid types)

TYPE	NAME	DESCRIPTION
System.Reflection.BindingFlags	bindingFlags	Binding flags

Returns

TYPE	DESCRIPTION
TRet	Call return value

Type Parameters

NAME	DESCRIPTION
TRet	Return value type

Exceptions

TYPE	CONDITION
XDataRuntimeException	

IsBasedOn<TType>(Type)

Check generic base type

Declaration

```
public static bool IsBasedOn<TType>(this Type type)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Type	type	Generic type

Returns

TYPE	DESCRIPTION
System.Boolean	Generic type is based on TType

Type Parameters

NAME	DESCRIPTION
TType	Base type to check

IsBinary(Type)

Check Lob type

Declaration

```
public static bool IsBinary(this Type type)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Type	type	Type to check

Returns

TYPE	DESCRIPTION
System.Boolean	Type is binary

IsEnum(Type)

Check Enum type

Declaration

```
public static bool IsEnum(this Type type)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Type	type	Type to check

Returns

TYPE	DESCRIPTION
System.Boolean	Type is binary

IsNullable(Type)

Check nullable type

Declaration

```
public static bool IsNullable(this Type type)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Type	type	Type to check

Returns

TYPE	DESCRIPTION
System.Boolean	Type is nullable

IsXml(Type)

Check Xml type

Declaration

```
public static bool IsXml(this Type type)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Type	type	Type to check

Returns

TYPE	DESCRIPTION
System.Boolean	Type is binary

NullWrap<T>(T)

Null value wrapper

Declaration

```
public static T NullWrap<T>(T source)
    where T : class, new()
```

Parameters

TYPE	NAME	DESCRIPTION
T	source	Source

Returns

TYPE	DESCRIPTION
T	Source or empty object of source type

Type Parameters

NAME	DESCRIPTION
T	Source type

NullWrap<T>(T[])

Null value wrapper

Declaration

```
public static T[] NullWrap<T>(T[] source)
```

Parameters

TYPE	NAME	DESCRIPTION
T[]	source	Source array

Returns

TYPE	DESCRIPTION
T[]	Source or empty array

Type Parameters

NAME	DESCRIPTION
T	Array element type

ParseEnum(Type, String)

Returns id for specified enum type and string value

Declaration

```
public static long? ParseEnum(this Type enumType, string value)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Type	enumType	Enum type
System.String	value	String value of enum

Returns

TYPE	DESCRIPTION
System.Nullable<System.Int64>	Enum id

ParseEnumRange(Type, ICollection<Object>)

Parse enum values range

Declaration

```
public static object[] ParseEnumRange(this Type enumType, ICollection<object> values)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Type	enumType	Enum type
System.Collections.Generic.ICollection<System.Object>	values	Enum values range

Returns

TYPE	DESCRIPTION
System.Object[]	Enum id array

See Also

[ParseEnum\(Type, String\)](#)

ParseType(String)

Parse type full name

Declaration

```
public static Type ParseType(this string typeFullName)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	typeFullName	

Returns

TYPE	DESCRIPTION
System.Type	

TryConvert<T, TResult>(T, Func<T, TResult>)

Try to convert value

Declaration

```
public static TResult TryConvert<T, TResult>(this T source, Func<T, TResult> convert)
    where T : class
```

Parameters

TYPE	NAME	DESCRIPTION
T	source	Source value

TYPE	NAME	DESCRIPTION
System.Func<T, TResult>	convert	Conversion logic

Returns

TYPE	DESCRIPTION
TResult	Converted value or default of TResult

Type Parameters

NAME	DESCRIPTION
T	Source type
TResult	Result type

TryConvert<T, TResult>(T, Func<T, TResult>, TResult)

Try to convert value

Declaration

```
public static TResult TryConvert<T, TResult>(this T source, Func<T, TResult> convert, TResult defaultValue)
    where T : class
```

Parameters

TYPE	NAME	DESCRIPTION
T	source	Source value
System.Func<T, TResult>	convert	Conversion logic
TResult	defaultValue	Default result value

Returns

TYPE	DESCRIPTION
TResult	Converted value or default of TResult

Type Parameters

NAME	DESCRIPTION

NAME	DESCRIPTION
T	Source type
TResult	Result type

Class TypeExtensions.DummyType

Inheritance

System.Object

TypeExtensions.DummyType

Implements

[IDataObject](#)

[ISqlObject](#)

Inherited Members

System.Object.Equals(System.Object)

System.Object.Equals(System.Object, System.Object)

System.Object.GetHashCode()

System.Object.GetType()

System.Object.MemberwiseClone()

System.Object.ReferenceEquals(System.Object, System.Object)

System.Object.ToString()

Namespace: [XData.Extensions](#)

Assembly: XData.docfx.dll

Syntax

```
public sealed class DummyType : IDataObject, ISqlObject
```

Implements

[IDataObject](#)

[ISqlObject](#)

Extension Methods

[DataObjectExtensions.Modify<T>\(T, Action<T>\[\]\)](#)

[DataObjectExtensions.GetProperties<T>\(T, IEnumerable<String>, DataVersion\)](#)

[DataObjectExtensions.GetProperties<T>\(T, String\[\]\)](#)

[DataObjectExtensions.Refresh<T>\(T\)](#)

[DataObjectExtensions.Execute<T>\(T, String, IDictionary<String, Action<Byte\[\]>>, IDictionary<String, Func<Byte\[\], Byte\[\]>>\)](#)

[DataObjectExtensions.Execute<T>\(T, Expression<Func<CustomLogic<T>>>, IDictionary<String, Action<Byte\[\]>>, IDictionary<String, Func<Byte\[\], Byte\[\]>>\)](#)

[DataObjectExtensions.PostData<T>\(T, String, Func<Byte\[\]>\)](#)

[DataObjectExtensions.Callback<T>\(T, String, ref Byte\[\]\)](#)

[DataObjectExtensions.GetProperties<T>\(T, DataVersion, String\[\]\)](#)

[DataObjectExtensions.IsCleared<T>\(T, String\)](#)

[DataObjectExtensions.IsChanged<T>\(T, String\)](#)

[DataObjectExtensions.IsChanged<T>\(T, Expression<Func<T, Object>>\[\]\)](#)

[DataObjectExtensions.IsEmpty<T>\(T, String\)](#)

[DataObjectExtensions.IsEmpty<T>\(T, Expression<Func<T, Object>>\)](#)

[DataObjectExtensions.Submit<T>\(T, out T, DataSubmitFlag\)](#)

[DataObjectExtensions.Submit<T>\(T, Func<JsonSettings<T>>, JsonSettings<T>>, out T, DataSubmitFlag\)](#)

[DataObjectExtensions.Submit<T>\(T, DataSubmitFlag\)](#)

[DataObjectExtensions.Lock<T>\(T\)](#)

[DataObjectExtensions.SetCurrent<T>\(T\)](#)

[DataObjectExtensions.GetProperty<TRet>\(IDataObject, String\)](#)

[DataObjectExtensions.GetContext\(IDataObject\)](#)

[DataObjectExtensions.GetLayer\(IDataObject\)](#)

[DataObjectExtensions.GetRepositoryAlias\(IDataObject\)](#)

DataObjectExtensions.GetProperty<T, TRet>(T, Expression<Func<T, TRet>>)

DataObjectExtensions.GetProperty<T, TRet>(T, Expression<Func<T, TRet>>, DataVersion)

DataObjectExtensions.AssignProperty<T, TValue>(T, String, TValue)

DataObjectExtensions.Copy<T>(T, T, String[])

DataObjectExtensions.Copy<T>(T, T, ApplyFlag, String[])

DataObjectExtensions.LoadLob<T>(T, Expression<Func<T, Object>>[])

DataObjectExtensions.LoadLob<T>(T)

DataObjectExtensions.LoadLob<T>(T, String[])

DataObjectExtensions.WithJsonSettings<T>(T, Func<JsonSettings<T>, JsonSettings<T>>)

DataObjectExtensions.GetParameter<T, TRet>(T, String)

DataObjectExtensions.GetParameter<T, TRet>(T, Expression<Func<T, TRet>>)

DataObjectExtensions.GetResultSet<T, TRes>(T, Expression<Func<T, IEnumerable<TRes>>>)

DataObjectExtensions.GetRepository<T>(T)

DataObjectExtensions.SetAttachedHandlers<T>(T, AttachedHandler<T>[])

DataObjectExtensions.IsAttached(IDataObject, Type)

DataObjectExtensions.GetState<T>(T)

DataObjectExtensions.CheckState<T>(T, DataObjectState)

DataObjectExtensions.SetDeleted<T>(T, Boolean)

SerializationExtensions.ToXml<T>(T)

SerializationExtensions.Serialize<T>(T)

SerializationExtensions.Restore<T>(T, SerializedData)

AsyncExtensions.ExecuteAsync<T>(T, Expression<Func<CustomLogic<T>>>, IDictionary<String, Action<Byte[]>>>, IDictionary<String, Func<Byte[], Byte[]>>>)

AsyncExtensions.ExecuteAsync<T>(T, Expression<Func<CustomLogic<T>>>, IDictionary<String, Action<Byte[]>>>)

AsyncExtensions.ExecuteAsync<T>(T, Expression<Func<CustomLogic<T>>>, IDictionary<String, Func<Byte[], Byte[]>>>)

AsyncExtensions.ExecuteAsync<T>(T, String, IDictionary<String, Action<Byte[]>>>, IDictionary<String, Func<Byte[], Byte[]>>>)

AsyncExtensions.ExecuteAsync<T>(T, String, IDictionary<String, Action<Byte[]>>>)

AsyncExtensions.ExecuteAsync<T>(T, String, IDictionary<String, Func<Byte[], Byte[]>>>)

AsyncExtensions.GetParameterAsync<T, TRet>(T, Expression<Func<T, TRet>>)

AsyncExtensions.GetParameterAsync<T, TRet>(T, String)

AsyncExtensions.GetResultSetAsync<T, TRes>(T, Expression<Func<T, IEnumerable<TRes>>>)

AsyncExtensions.SubmitAsync<T>(T, DataSubmitFlag)

AsyncExtensions.SubmitAndReturnAsync<T>(T, DataSubmitFlag)

AsyncExtensions.SubmitAndReturnAsync<T>(T, Func<JsonSettings<T>, JsonSettings<T>>, DataSubmitFlag)

AsyncExtensions.LoadLobAsync<T>(T)

AsyncExtensions.LoadLobAsync<T>(T, String[])

AsyncExtensions.LoadLobAsync<T>(T, Expression<Func<T, Object>>[])

CollectionsExtensions.SetValue<TKey, T>(TKey, T)

CollectionsExtensions.AsEnum<T>(T)

DataObjectQueryableExtensions.CompareVariable(Object, FilterOperation, String)

Processing.Do<T>(T, Action<IProcess<T>>[])

TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>)

TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>, TResult)

TypeExtensions.ChangeType<T>(Object)

TypeExtensions.ChangeType(Object, Type)

TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)

TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)

SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)

QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

QueryDescriptionExtensions.GetProperty(ISqlObject, String)

Namespace XData.Interfaces

Classes

[AdapterConfig](#)

Adapter configuration section

[ContextConfig](#)

Context config section

[DialectConfig](#)

SQL dialect configuration section

[ProxyConfig](#)

Proxy configuration section

Interfaces

[IConfiguration](#)

XData configuration

[ILogWriter](#)

XData log writer interface

[IProxy](#)

Data object proxy interface

[IProxyManager](#)

Proxy manager

[ISecuritySession](#)

Security session interface

[ISequenceNameRule](#)

Sequence name rule interface

[ISqlBuilder](#)

SQL builder interface

Class AdapterConfig

Adapter configuration section

Inheritance

System.Object
AdapterConfig

Inherited Members

System.Object.Equals(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.GetHashCode()
System.Object.GetType()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.ToString()

Namespace: [XData.Interfaces](#)
Assembly: XData.docfx.dll

Syntax

```
public sealed class AdapterConfig
```

Remarks

WARNING! All attributes are required!

Examples

```
<adapter name="MSSQL" file="C:\XData\XMsSqlAdapter.dll"/>
```

Properties

AssemblyName

Adapter assembly name

Declaration

```
public string AssemblyName { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Config

Adapter configuration

Declaration

```
public IConfiguration Config { get; set; }
```

Property Value

TYPE	DESCRIPTION
Microsoft.Extensions.Configuration.IConfiguration	

Name

Adapter name

Declaration

```
public string Name { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Remarks

Must be the same as [AdapterName](#)

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

See Also

- [Name](#)
- [AssemblyName](#)

Class ContextConfig

Context config section

Inheritance

System.Object
ContextConfig

Inherited Members

System.Object.Equals(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.GetHashCode()
System.Object.GetType()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)

Namespace: [XData.Interfaces](#)

Assembly: XData.docfx.dll

Syntax

```
public sealed class ContextConfig
```

Remarks

WARNING! All attributes are required!

Examples

```
<context name="TEST" adapter="MSSQL"/>
```

Properties

AdapterName

Context adapter name

Declaration

```
public string AdapterName { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Remarks

Mast be the same as [Name](#)

ConcurrencyErrorId

Concurrency error id

Declaration

```
public string ConcurrencyErrorId { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

ConnectionString

Connection string

Declaration

```
public string ConnectionString { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Remarks

Received from ConnectionStrings configuration section by name same as [Name](#)

DialectName

Dialect adapter name

Declaration

```
public string DialectName { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Remarks

Mast be the same as [Name](#)

IsDefault

Default context flag

Declaration

```
public bool IsDefault { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

Name

Context name

Declaration

```
public string Name { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

SequenceNameRule

Sequence name rule class name

Declaration

```
public string SequenceNameRule { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Methods

ToString()

Returns a string that represents the current object.

Declaration

```
public override string ToString()
```

Returns

TYPE	DESCRIPTION
System.String	A string that represents the current object.

Overrides

System.Object.ToString()

Extension Methods

[SerializationExtensions.ToXml<T>\(T\)](#)

[CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)

[CollectionsExtensions.AsEnum<T>\(T\)](#)

[DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)

[Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)

[TypeExtensions.ChangeType<T>\(Object\)](#)

[TypeExtensions.ChangeType\(Object, Type\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)

[SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)

[QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

See Also

Name
AdapterName
ConnectionString
IsDefault
SequenceNameRule

Class DialectConfig

SQL dialect configuration section

Inheritance

System.Object
DialectConfig

Inherited Members

System.Object.Equals(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.GetHashCode()
System.Object.GetType()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.ToString()

Namespace: [XData.Interfaces](#)
Assembly: XData.docfx.dll

Syntax

```
public sealed class DialectConfig
```

Remarks

WARNING! All attributes are required!

Examples

```
<dialect name="MSSQL" file="C:\XData\XMsSqlDialect.dll"/>
```

Properties

AssemblyName

Adapter assembly name

Declaration

```
public string AssemblyName { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Config

Adapter configuration

Declaration

```
public IConfiguration Config { get; set; }
```

Property Value

TYPE	DESCRIPTION
Microsoft.Extensions.Configuration.IConfiguration	

Name

Adapter name

Declaration

```
public string Name { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Remarks

Mast be the same as [DialectName](#)

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

See Also

- [Name](#)
- [AssemblyName](#)

Interface IConfiguration

XData configuration

Namespace: [XData.Interfaces](#)

Assembly: XData.docfx.dll

Syntax

```
public interface IConfiguration
```

Examples

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <!-- *** WARNING! configSections section with declaration of "xdata" is required and mast be first
child section of configuration! -->
  <configSections>
    <section name="xdata" type="System.Configuration.IgnoreSectionHandler" />
  </configSections>
  <!-- *** You can configure Your application to direct connect database OR use application server -->
  <!-- *** Endpoint required using application server only -->
  <!--<system.serviceModel>
  <client>
    <endpoint address="net.tcp://localhost:8733/AppServer/"
      binding="netTcpBinding"
      contract="XDataProxy.IAppServer"
      name="XAppServer" />
  </client>
  </system.serviceModel>-->
  <!-- *** Connection strings required using direct connect only -->
  <connectionStrings>
    <add name="TEST" connectionString="Data Source=(local);Initial Catalog=XDataTest;User
Id=test;Password=test" />
  </connectionStrings>
  <!-- *** xdata section (see XDataConfiguration.DefaultContext, XDataConfiguration.LogWriter
documentation) -->
  <xdata default="TEST" log="XData.Logging.Trace.TraceWriter, XData.Logging.Trace">
    <!-- *** Specific SQL data access adapter required using direct database access (see
XDataConfiguration.Adapters documentation) -->
    <adapter name="MSSQL" file="C:\Projects\Net40\XData\_Debug\XMsSqlAdapter.dll"/>
    <!-- *** Specific SQL dialect adapter (see XDataConfiguration.Dialects documentation) -->
    <dialect name="MSSQL" file="C:\Projects\Net40\XData\_Debug\XMsSqlDialect.dll"/>
    <!-- *** Proxy adapter required using application server only -->
    <!--<adapter name="PROXY" file="C:\XData\XProxyAdapter.dll"/>-->
    <!-- *** Context can be configured by direct or proxy adapter according Your needs (see
XDataConfiguration.Contexts, ContextConfig.SequenceNameRule documentation) -->
    <!--<context name="TEST" adapter="PROXY"/>-->
    <context name="TEST" adapter="MSSQL" sequence="XMyCustomLib.MyDbSequence, XMyCustomLib"/>
    <!-- *** Context proxy configuration required using application server only (see
XDataConfiguration.ProxyList documentation) -->
    <!--<proxy name="TEST" adapter="MSSQL" endpoint="XAppServer" />-->
  </xdata>
</configuration>
```

Properties

Adapters

Registered adapters collection

Declaration

```
List<AdapterConfig> Adapters { get; }
```

Property Value

TYPE	DESCRIPTION
System.Collections.Generic.List< AdapterConfig >	

See Also

[AdapterConfig](#)

Contexts

Registered contexts collection

Declaration

List<ContextConfig> Contexts { get; }

Property Value

TYPE	DESCRIPTION
System.Collections.Generic.List< ContextConfig >	

See Also

[ContextConfig](#)

DefaultContext

Default context name

Declaration

string DefaultContext { get; }

Property Value

TYPE	DESCRIPTION
System.String	

Remarks

WARNING! Required configuration attribute!

See Also

[Context](#)

[Context](#)

Dialects

Registered SQL dialects collection

Declaration

List<DialectConfig> Dialects { get; }

Property Value

TYPE	DESCRIPTION
System.Collections.Generic.List< DialectConfig >	

See Also

[DialectConfig](#)

LifeTimeLimit

XData objects lifetime limit (minutes)

Declaration

```
int LifeTimeLimit { get; }
```

Property Value

TYPE	DESCRIPTION
System.Int32	

LogWriter

Log writer class name

Declaration

```
string LogWriter { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

MonitoringContext

Health monitoring context name

Declaration

```
string MonitoringContext { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

See Also

[Context](#)

[Context](#)

ProxyList

Registered proxy collection

Declaration

```
List<ProxyConfig> ProxyList { get; }
```

Property Value

TYPE	DESCRIPTION
System.Collections.Generic.List< ProxyConfig >	

See Also

[ProxyConfig](#)

SecurityContext

Security context name

Declaration

```
string SecurityContext { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

See Also

[Context](#)

[Context](#)

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

See Also

- [Adapters](#)
- [Dialects](#)
- [Contexts](#)
- [ProxyList](#)
- [DefaultContext](#)
- [LogWriter](#)

Interface ILogWriter

XData log writer interface

Namespace: [XData.Interfaces](#)

Assembly: XData.docfx.dll

Syntax

```
public interface ILogWriter
```

Methods

Write(MessageType, String)

Write message to log

Declaration

```
void Write(MessageType msgType, string msgText)
```

Parameters

TYPE	NAME	DESCRIPTION
MessageType	msgType	Message type
System.String	msgText	Message text

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Interface IProxy

Data object proxy interface

Inherited Members

System.IDisposable.Dispose()

Namespace: [XData.Interfaces](#)

Assembly: XData.docfx.dll

Syntax

```
public interface IProxy : IDisposable
```

Methods

Close(Guid)

Close proxy

Declaration

```
void Close(Guid layer)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Guid	layer	Layer to close (Specify Guid.Empty to close all layers)

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

See Also

[Proxy](#)

Interface IProxyManager

Proxy manager

Namespace: [XData.Interfaces](#)

Assembly: XData.docfx.dll

Syntax

```
public interface IProxyManager
```

Methods

GetProxy(String)

Returns XAppServer connection proxy object

Declaration

```
IProxy GetProxy(string endpointName)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	endpointName	

Returns

TYPE	DESCRIPTION
IProxy	

GetProxyRepository(Guid, RepositoryState)

Returns proxy repository using state

Declaration

```
IRepository GetProxyRepository(Guid layer, RepositoryState state)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Guid	layer	Repository layer
RepositoryState	state	Repository state

Returns

TYPE	DESCRIPTION
IRepository	Repository

GetProxyRepository<T>(Guid, String, String, ISecuritySession, Variable[])

Returns proxy repository

Declaration

```
IRepository<T> GetProxyRepository<T>(Guid layer = default(Guid), string alias = null, string context = null,
ISecuritySession security = null, params Variable[] variables)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
System.Guid	layer	Repository layer
System.String	alias	Repository alias
System.String	context	Repository context
ISecuritySession	security	Repository security context
Variable[]	variables	Repository variables

Returns

TYPE	DESCRIPTION
IRepository<T>	Repository

Type Parameters

NAME	DESCRIPTION
T	Repository element type

Extension Methods

- SerializationExtensions.ToXml<T>(T)
- CollectionsExtensions.SetValue<TKey, T>(TKey, T)
- CollectionsExtensions.AsEnum<T>(T)
- DataObjectQueryableExtensions.CompareVariable(Object, FilterOperation, String)
- Processing.Do<T>(T, Action<IProcess<T>> [])
- TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>)
- TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>, TResult)
- TypeExtensions.ChangeType<T>(Object)
- TypeExtensions.ChangeType(Object, Type)
- TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)
- TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)
- TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)

SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)

QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Interface ISecuritySession

Security session interface

Namespace: [XData.Interfaces](#)

Assembly: XData.docfx.dll

Syntax

```
public interface ISecuritySession
```

Properties

IsSupervisor

Current user is supervisor flag

Declaration

```
bool IsSupervisor { get; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

Roles

Current user roles list

Declaration

```
string[] Roles { get; }
```

Property Value

TYPE	DESCRIPTION
System.String[]	

UserName

Current user name

Declaration

```
string UserName { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Methods

FilterGrantedObjects<T>(String, Expression)

Converts initial IQueryable expression filtering granted objects only

Declaration

```
Expression FilterGrantedObjects<T>(string context, Expression allDataExpr)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	context	Context name
System.Linq.Expressions.Expression	allDataExpr	Initial IQueryable expression

Returns

TYPE	DESCRIPTION
System.Linq.Expressions.Expression	Modified IQueryable expression

Type Parameters

NAME	DESCRIPTION
T	Data object type

GetGrantedActions<T>(String)

Returns list of granted actions for current user

Declaration

```
string[] GetGrantedActions<T>(string context = null)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	context	Context name

Returns

TYPE	DESCRIPTION
System.String[]	Actions list

Type Parameters

NAME	DESCRIPTION
T	Data object type

GetRestrictedProperties<T>(String)

Returns list of restricted properties for current user

Declaration

```
string[] GetRestrictedProperties<T>(string context = null)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	context	Context name

Returns

TYPE	DESCRIPTION
System.String[]	Properties name list

Type Parameters

NAME	DESCRIPTION
T	Data object type

Extension Methods

[SerializationExtensions.ToXml<T>\(T\)](#)

[CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)

[CollectionsExtensions.AsEnum<T>\(T\)](#)

[DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)

[Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)

[TypeExtensions.ChangeType<T>\(Object\)](#)

[TypeExtensions.ChangeType\(Object, Type\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)

[SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)

[QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Interface ISequenceNameRule

Sequence name rule interface

Namespace: [XData.Interfaces](#)

Assembly: XData.docfx.dll

Syntax

```
public interface ISequenceNameRule
```

Methods

GetSequenceName(String, String, String)

Returns sequence name

Declaration

```
string GetSequenceName(string context, string tableName, string columnName)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	context	Context name
System.String	tableName	Table name
System.String	columnName	Column name

Returns

TYPE	DESCRIPTION
System.String	Sequence name

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)

```
QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)
```

Interface ISqlBuilder

SQL builder interface

Namespace: [XData.Interfaces](#)

Assembly: XData.docfx.dll

Syntax

```
public interface ISqlBuilder
```

Properties

DialectOptions

Dialect options

WARNING! Mast to be filled in constructor!

Declaration

```
DialectOptions DialectOptions { get; }
```

Property Value

TYPE	DESCRIPTION
DialectOptions	

Item[Enum]

Predefined queries

Declaration

```
Query this[Enum i] { get; }
```

Parameters

TYPE	NAME	DESCRIPTION
System.Enum	i	Query enumerated index

Property Value

TYPE	DESCRIPTION
Query	Query

ParameterPrefix

Parameter prefix in SQL dialect

Declaration

```
string ParameterPrefix { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

SqlTypes

List of supported SQL types
WARNING! Mast to be filled in constructor!

Declaration

```
DbType[] SqlTypes { get; }
```

Property Value

TYPE	DESCRIPTION
System.Data.DbType[]	

SqlTypeSizes

List of supported type size applicable
WARNING! Mast to be filled in constructor!

Declaration

```
Dictionary<string, SqlTypeSize> SqlTypeSizes { get; }
```

Property Value

TYPE	DESCRIPTION
System.Collections.Generic.Dictionary<System.String, SqlTypeSize >	

TypeNames

List of supported column type names
WARNING! Mast to be filled in constructor!

Declaration

```
string[] TypeNames { get; }
```

Property Value

TYPE	DESCRIPTION
System.String[]	

TypeSynonyms

List of supported type synonyms
WARNING! Mast to be filled in constructor!

Declaration

```
string[] TypeSynonyms { get; }
```

Property Value

TYPE	DESCRIPTION
System.String[]	

Methods

ExtractXmlQuery(String, String, String[], IComparable[], String, KeyValuePair<String, String>[])

Returns query to extract XML data by XPath expression

Declaration

```
Query ExtractXmlQuery(string table, string column, string[] tablePrimaryKey, IComparable[] primaryKeyValue, string path, params KeyValuePair<string, string>[] namespaces)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	table	Table name
System.String	column	XML column name
System.String[]	tablePrimaryKey	Table primary key columns
IComparable[]	primaryKeyValue	Table primary key values
System.String	path	XPath expression
System.Collections.Generic.KeyValuePair<System.String, System.String> []	namespaces	XPath namespaces

Returns

TYPE	DESCRIPTION
Query	XPath result

GetAcquireBlobQuery(String, String, IEnumerable<String>, IComparable[], String)

Returns query to acquire blob data

Declaration

```
Query GetAcquireBlobQuery(string table, string column, IEnumerable<string> tablePrimaryKey, IComparable[] primaryKeyValue, string nativeSqlType = null)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	table	Table name
System.String	column	Blob column name
System.Collections.Generic.IEnumerable<System.String>	tablePrimaryKey	Table primary key columns
IComparable[]	primaryKeyValue	Table primary key values
System.String	nativeSqlType	Native SQL type name

Returns

TYPE	DESCRIPTION
Query	Query

GetAcquireXmlQuery(String, String, String[], IComparable[])

Returns query to acquire XML data

Declaration

```
Query GetAcquireXmlQuery(string table, string column, string[] tablePrimaryKey, IComparable[] primaryKeyValue)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	table	Table name
System.String	column	XML column name
System.String[]	tablePrimaryKey	Table primary key columns
IComparable[]	primaryKeyValue	Table primary key values

Returns

TYPE	DESCRIPTION
Query	Query

GetAllQuery(IDataStructure)

Returns query to check compliance of all repository objects Hint: Used in XData LINQ expressions

Declaration

```
Query GetAllQuery(IDataStructure structure)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataStructure	structure	Structure metadata

Returns

TYPE	DESCRIPTION
Query	Query

GetAnyQuery(IDataStructure)

Returns query to check existence of repository objects Hint: Used in XData LINQ expressions

Declaration

```
Query GetAnyQuery(IDataStructure structure)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataStructure	structure	Structure metadata

Returns

TYPE	DESCRIPTION
Query	Query

GetBlobSizeQuery(String, String, String[], IComparable[])

Returns query to get Large Object size

Declaration

```
Query GetBlobSizeQuery(string table, string column, string[] tablePrimaryKey, IComparable[] primaryKeyValue)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	table	Table name contained LOB

TYPE	NAME	DESCRIPTION
System.String	column	LOB column name
System.String[]	tablePrimaryKey	Primary key column names
IComparable[]	primaryKeyValue	Primary key values

Returns

TYPE	DESCRIPTION
Query	Query to get Large Object size

GetCall(IDataStructure, String)

Returns stored procedure call definition

Declaration

StoredProcedure GetCall(IDataStructure dataStructure, string userName)

Parameters

TYPE	NAME	DESCRIPTION
IDataStructure	dataStructure	Structure
System.String	userName	User name

Returns

TYPE	DESCRIPTION
StoredProcedure	Stored procedure definition

GetClearQuery(IDataStructure)

Returns query to clear all repository objects

Declaration

Query GetClearQuery(IDataStructure structure)

Parameters

TYPE	NAME	DESCRIPTION

TYPE	NAME	DESCRIPTION
IDataStructure	structure	Structure metadata

Returns

TYPE	DESCRIPTION
Query	Query

GetCountQuery(IDataStructure)

Returns query to select count of data objects in repository

Declaration

```
Query GetCountQuery(IDataStructure structure)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataStructure	structure	Structure metadata

Returns

TYPE	DESCRIPTION
Query	Query

GetDeleteQuery(IDataStructure, IDataObject, String, Boolean)

Returns query to delete data object

Declaration

```
Query GetDeleteQuery(IDataStructure structure, IDataObject data, string userName, bool ignoreConcurrency = true)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataStructure	structure	Structure metadata
IDataObject	data	Data object instance
System.String	userName	User name

TYPE	NAME	DESCRIPTION
System.Boolean	ignoreConcurrency	Ignore concurrency flag

Returns

TYPE	DESCRIPTION
Query	Query

GetInsertQuery(IDataStructure, IDataObject, String, Boolean)

Returns query to insert data object

Declaration

```
Query GetInsertQuery(IDataStructure structure, IDataObject data, string userName, bool ignoreConcurrency = true)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataStructure	structure	Structure metadata
IDataObject	data	Data object instance
System.String	userName	User name
System.Boolean	ignoreConcurrency	Ignore concurrency flag

Returns

TYPE	DESCRIPTION
Query	Query

GetLockQuery(IDataStructure)

Returns query to lock single data object

Declaration

```
Query GetLockQuery(IDataStructure structure)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataStructure	structure	Structure metadata

Returns

TYPE	DESCRIPTION
Query	Query

GetParameterName(String)

Formats field name as parameter name

Declaration

```
string GetParameterName(string fieldName)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	fieldName	Field name

Returns

TYPE	DESCRIPTION
System.String	Parameter name

GetRefreshQuery(IDataStructure)

Returns query to refresh single data object

Declaration

```
Query GetRefreshQuery(IDataStructure structure)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataStructure	structure	Structure metadata

Returns

TYPE	DESCRIPTION
Query	Query

GetSafeConvertExpression(Type, Expression, Expression)

Safe convert expression

Declaration

```
Expression GetSafeConvertExpression(Type type, Expression dataRow, Expression fieldName)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Type	type	Type of expression
System.Linq.Expressions.Expression	dataRow	Data row expression
System.Linq.Expressions.Expression	fieldName	Field name expression

Returns

TYPE	DESCRIPTION
System.Linq.Expressions.Expression	Expression

GetSaveBlobQuery(String, String, IEnumerable<String>, IComparable[], ref Byte[], String)

Returns query to save blob data

Declaration

```
Query GetSaveBlobQuery(string table, string column, IEnumerable<string> tablePrimaryKey, IComparable[] primaryKeyValue, ref byte[] body, string nativeSqlType = null)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	table	Table name
System.String	column	Blob column name
System.Collections.Generic.IEnumerable<System.String>	tablePrimaryKey	Table primary key columns
IComparable[]	primaryKeyValue	Table primary key values
Byte[]	body	Blob body

TYPE	NAME	DESCRIPTION
System.String	nativeSqlType	Native SQL type name

Returns

TYPE	DESCRIPTION
Query	Query

GetSaveXmlQuery(String, String, String[], IComparable[], ref String)

Returns query to save XML data

Declaration

```
Query GetSaveXmlQuery(string table, string column, string[] tablePrimaryKey, IComparable[] primaryKeyValue,
ref string body)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	table	Table name
System.String	column	XML column name
System.String[]	tablePrimaryKey	Table primary key columns
IComparable[]	primaryKeyValue	Table primary key values
System.String	body	Blob body

Returns

TYPE	DESCRIPTION
Query	Query

GetSelectQuery(IDataStructure, Boolean, Boolean)

Returns query to select data objects

Declaration

```
Query GetSelectQuery(IDataStructure structure, bool markProperties = false, bool doNotSkip = false)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataStructure	structure	Structure metadata
System.Boolean	markProperties	Mark properties with comments
System.Boolean	doNotSkip	Don't skip sources

Returns

TYPE	DESCRIPTION
Query	Query

GetUpdateQuery(IDataStructure, IDataObject, String, Boolean)

Returns query to update data object

Declaration

```
Query GetUpdateQuery(IDataStructure structure, IDataObject data, string userName, bool ignoreConcurrency = true)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataStructure	structure	Structure metadata
IDataObject	data	Data object instance
System.String	userName	User name
System.Boolean	ignoreConcurrency	Ignore concurrency flag

Returns

TYPE	DESCRIPTION
Query	Query

Init(String, IConfiguration, String)

SQL builder initialization

Declaration

```
void Init(string context, IConfiguration config, string sequence)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	context	
Microsoft.Extensions.Configuration.IConfiguration	config	
System.String	sequence	

SetQueryParameterValues(ref Query, SortedList<String, Object>)

Set query parameter values

Declaration

```
void SetQueryParameterValues(ref Query query, SortedList<string, object> paramValues)
```

Parameters

TYPE	NAME	DESCRIPTION
Query	query	Query
System.Collections.Generic.SortedList<System.String, System.Object>	paramValues	Parameter values

Extension Methods

[SerializationExtensions.ToXml<T>\(T\)](#)

[CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)

[CollectionsExtensions.AsEnum<T>\(T\)](#)

[DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)

[Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)

[TypeExtensions.ChangeType<T>\(Object\)](#)

[TypeExtensions.ChangeType\(Object, Type\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)

[SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)

[QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Class ProxyConfig

Proxy configuration section

Inheritance

System.Object
ProxyConfig

Inherited Members

System.Object.Equals(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.GetHashCode()
System.Object.GetType()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.ToString()

Namespace: [XData.Interfaces](#)

Assembly: XData.docfx.dll

Syntax

```
public sealed class ProxyConfig
```

Remarks

WARNING! All attributes are required!

Examples

```
<proxy name="TEST" dialect="MSSQL" endpoint="XAppServer"/>
```

Properties

EndpointName

XData application server WCF endpoint name

Declaration

```
public string EndpointName { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Name

Proxy name

Declaration

```
public string Name { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Remarks

Must be the same as [Name](#)

Extension Methods

[SerializationExtensions.ToXml<T>\(T\)](#)

[CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)

[CollectionsExtensions.AsEnum<T>\(T\)](#)

[DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)

[Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)

[TypeExtensions.ChangeType<T>\(Object\)](#)

[TypeExtensions.ChangeType\(Object, Type\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)

[SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)

[QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

See Also

[Name](#)

[EndpointName](#)

Namespace XData.Json

Classes

[JsonDataObjectConverter<T>](#)

[JsonSettings<T>](#)

JSON serialization settings

Enums

[HideFlags](#)

Serialization hide flag for properties

[LinkMode](#)

Link serialization mode

Enum HideFlags

Serialization hide flag for properties

Namespace: [XData.Json](#)

Assembly: XData.docfx.dll

Syntax

```
[Flags]
public enum HideFlags
```

Fields

NAME	DESCRIPTION
Columns	Hide all columns
ReadOnly	Hide all Read only properties

Extension Methods

- [SerializationExtensions.ToXml<HideFlags>\(\)](#)
- [CollectionsExtensions.SetValue<HideFlags, T>\(T\)](#)
- [CollectionsExtensions.AsEnum<HideFlags>\(\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(FilterOperation, String\)](#)
- [Processing.Do<HideFlags>\(Action<IProcess<HideFlags>>\[\]\)](#)
- [TypeExtensions.ChangeType<T>\(\)](#)
- [TypeExtensions.ChangeType\(Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.ConvertTo<T>\(\)](#)
- [TypeExtensions.EqualValue\(Enum\)](#)
- [SqlBlockExtensions.SetExpression<HideFlags, TResult>\(Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<HideFlags, TResult>\(Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Class JsonDataObjectConverter<T>

Inheritance

System.Object
Newtonsoft.Json.JsonConverter
JsonDataObjectConverter<T>

Inherited Members

Newtonsoft.Json.JsonConverter.GetSchema()
Newtonsoft.Json.JsonConverter.CanRead
Newtonsoft.Json.JsonConverter.CanWrite
System.Object.Equals(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.GetHashCode()
System.Object.GetType()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.ToString()

Namespace: [XData.Json](#)
Assembly: XData.docfx.dll

Syntax

```
public class JsonDataObjectConverter<T> : JsonConverter where T : class, IDataObject
```

Type Parameters

NAME	DESCRIPTION
T	

Methods

CanConvert(Type)

Declaration

```
public override bool CanConvert(Type objectType)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Type	objectType	

Returns

TYPE	DESCRIPTION
System.Boolean	

Overrides

Newtonsoft.Json.JsonConverter.CanConvert(System.Type)

ReadJson(JsonReader, Type, Object, JsonSerializer)

Declaration

```
public override object ReadJson(JsonReader reader, Type objectType, object existingValue, JsonSerializer serializer)
```

Parameters

TYPE	NAME	DESCRIPTION
Newtonsoft.Json.JsonReader	reader	
System.Type	objectType	
System.Object	existingValue	
Newtonsoft.Json.JsonSerializer	serializer	

Returns

TYPE	DESCRIPTION
System.Object	

Overrides

Newtonsoft.Json.JsonConverter.ReadJson(Newtonsoft.Json.JsonReader, System.Type, System.Object, Newtonsoft.Json.JsonSerializer)

WriteJson(JsonWriter, Object, JsonSerializer)

Declaration

```
public override void WriteJson(JsonWriter writer, object value, JsonSerializer serializer)
```

Parameters

TYPE	NAME	DESCRIPTION
Newtonsoft.Json.JsonWriter	writer	
System.Object	value	
Newtonsoft.Json.JsonSerializer	serializer	

Overrides

Newtonsoft.Json.JsonConverter.WriteJson(Newtonsoft.Json.JsonWriter, System.Object, Newtonsoft.Json.JsonSerializer)

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)

TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)

SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)

QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Class JsonSettings<T>

JSON serialization settings

Inheritance

System.Object
JsonSettings<T>

Inherited Members

System.Object.Equals(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.GetHashCode()
System.Object.GetType()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.ToString()

Namespace: [XData.Json](#)
Assembly: XData.docfx.dll

Syntax

```
public class JsonSettings<T>
    where T : class, IDataObject, new()
```

Type Parameters

NAME	DESCRIPTION
T	Object type

Constructors

JsonSettings()

Declaration

```
public JsonSettings()
```

Methods

DeleteFlag<TChild>(Expression<Func<T, TChild[]>>, Expression<Func<TChild, Boolean>>)

Delete child flag

Declaration

```
public JsonSettings<T> DeleteFlag<TChild>(Expression<Func<T, TChild[]>> child, Expression<Func<TChild, bool>>
property)
    where TChild : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func<T, TChild[]>>	child	Child instance

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func<TChild, System.Boolean>>	property	Delete child flag property

Returns

TYPE	DESCRIPTION
JsonSettings<T>	JSON settings

Type Parameters

NAME	DESCRIPTION
TChild	

DeleteFlag<TChild>(Expression<Func<T, TChild>>, Expression<Func<TChild, Boolean>>)

Delete child flag

Declaration

```
public JsonSettings<T> DeleteFlag<TChild>(Expression<Func<T, TChild>> child, Expression<Func<TChild, bool>>
property)
    where TChild : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func<T, TChild>>	child	Child instance
System.Linq.Expressions.Expression<System.Func<TChild, System.Boolean>>	property	Delete child flag property

Returns

TYPE	DESCRIPTION
JsonSettings<T>	JSON settings

Type Parameters

NAME	DESCRIPTION
TChild	

Hide()

Hide properties from serialization

Declaration

```
public JsonSettings<T> Hide()
```

Returns

TYPE	DESCRIPTION
JsonSettings<T>	JSON settings

Hide(Expression<Func<T, Object>>[])

Allow properties serialization

Declaration

```
public JsonSettings<T> Hide(params Expression<Func<T, object>>[] properties)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func<T, System.Object>>[]	properties	Properties extensions

Returns

TYPE	DESCRIPTION
JsonSettings<T>	JSON settings

Hide(String[])

Hide properties from serialization

Declaration

```
public JsonSettings<T> Hide(params string[] properties)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String[]	properties	Properties names

Returns

TYPE	DESCRIPTION
JsonSettings<T>	JSON settings

Hide(HideFlags)

Allow properties serialization

Declaration

```
public JsonSettings<T> Hide(HideFlags flags)
```

Parameters

TYPE	NAME	DESCRIPTION
HideFlags	flags	Hide all properties corresponding flags

Returns

TYPE	DESCRIPTION
JsonSettings<T>	JSON settings

LinkMode(LinkMode)

Link serialization mode

Declaration

```
public JsonSettings<T> LinkMode(LinkMode mode)
```

Parameters

TYPE	NAME	DESCRIPTION
LinkMode	mode	Link serialization mode

Returns

TYPE	DESCRIPTION
JsonSettings<T>	JSON settings

LinkMode(LinkMode, Expression<Func<T, Object>>[])

Link serialization mode

Declaration

```
public JsonSettings<T> LinkMode(LinkMode mode, params Expression<Func<T, object>>[] properties)
```

Parameters

TYPE	NAME	DESCRIPTION
LinkMode	mode	Link serialization mode
System.Linq.Expressions.Expression<System.Func<T, System.Object>>[]	properties	Properties extensions

Returns

TYPE	DESCRIPTION
JsonSettings<T>	JSON settings

LinkMode(LinkMode, String[])

Link serialization mode

Declaration

```
public JsonSettings<T> LinkMode(LinkMode mode, params string[] properties)
```

Parameters

TYPE	NAME	DESCRIPTION
LinkMode	mode	Link serialization mode
System.String[]	properties	Properties names

Returns

TYPE	DESCRIPTION
JsonSettings<T>	JSON settings

LoadLob()

Load large objects

Declaration

```
public JsonSettings<T> LoadLob()
```

Returns

TYPE	DESCRIPTION
JsonSettings<T>	JSON settings

LoadLob(Expression<Func<T, Object>>[])

Load large objects

Declaration

```
public JsonSettings<T> LoadLob(params Expression<Func<T, object>>[] properties)
```

Parameters

TYPE	NAME	DESCRIPTION

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func<T, System.Object>> []	properties	Properties extensions

Returns

TYPE	DESCRIPTION
JsonSettings<T>	JSON settings

LoadLob(String[])

Load large objects

Declaration

```
public JsonSettings<T> LoadLob(params string[] properties)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String[]	properties	Properties names

Returns

TYPE	DESCRIPTION
JsonSettings<T>	JSON settings

Show()

Allow properties serialization

Declaration

```
public JsonSettings<T> Show()
```

Returns

TYPE	DESCRIPTION
JsonSettings<T>	JSON settings

Show(Expression<Func<T, Object>>[])

Allow properties serialization

Declaration

```
public JsonSettings<T> Show(params Expression<Func<T, object>>[] properties)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func<T, System.Object> > []	properties	Properties extensions

Returns

TYPE	DESCRIPTION
JsonSettings <T>	JSON settings

Show(String[])

Allow properties serialization

Declaration

public JsonSettings<T> Show(params string[] properties)

Parameters

TYPE	NAME	DESCRIPTION
System.String[]	properties	Properties names

Returns

TYPE	DESCRIPTION
JsonSettings <T>	JSON settings

Extension Methods

[SerializationExtensions.ToXml](#)<T>(T)

[CollectionsExtensions.SetValue](#)<TKey, T>(TKey, T)

[CollectionsExtensions.AsEnum](#)<T>(T)

[DataObjectQueryableExtensions.CompareVariable](#)(Object, FilterOperation, String)

[Processing.Do](#)<T>(T, Action<IProcess<T> > [])

[TypeExtensions.TryConvert](#)<T, TResult>(T, Func<T, TResult>)

[TypeExtensions.TryConvert](#)<T, TResult>(T, Func<T, TResult>, TResult)

[TypeExtensions.ChangeType](#)<T>(Object)

[TypeExtensions.ChangeType](#)(Object, Type)

[TypeExtensions.InvokeMethod](#)<TRet>(Object, Type[], Expression<Func<TRet> >, BindingFlags)

[TypeExtensions.InvokeMethod](#)(Object, Type[], Expression<Action>, BindingFlags)

[TypeExtensions.InvokeMethod](#)<TRet>(Object, Expression<Func<TRet> >, BindingFlags)

[TypeExtensions.InvokeMethod](#)(Object, Expression<Action>, BindingFlags)

[SqlBlockExtensions.SetExpression](#)<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult> >)

[QueryDescriptionExtensions.SetExpression](#)<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult> >)

Enum LinkMode

Link serialization mode

Namespace: [XData.Json](#)
Assembly: XData.docfx.dll

Syntax

```
public enum LinkMode
```

Fields

NAME	DESCRIPTION
Default	Default link mode
Source	Link is serialized as limited Source object only
Value	Link is serialized as Value only

Extension Methods

- [SerializationExtensions.ToXml<LinkMode>\(\)](#)
- [CollectionsExtensions.SetValue<LinkMode, T>\(T\)](#)
- [CollectionsExtensions.AsEnum<LinkMode>\(\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(FilterOperation, String\)](#)
- [Processing.Do<LinkMode>\(Action<IProcess<LinkMode>>\[\]\)](#)
- [TypeExtensions.ChangeType<T>\(\)](#)
- [TypeExtensions.ChangeType\(Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.ConvertTo<T>\(\)](#)
- [TypeExtensions.EqualValue\(Enum\)](#)
- [SqlBlockExtensions.SetExpression<LinkMode, TResult>\(Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<LinkMode, TResult>\(Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Namespace XData.Mapping

Classes

[ActionAttribute](#)

Data action attribute

[ColumnAttribute](#)

Column mapping attribute

[ColumnDefaultAttribute](#)

Column default value attribute

[ColumnExpressionAttribute](#)

Column expression attribute

[ColumnUpdateWithAttribute](#)

Column update source expression attribute

[ConstantFilterAttribute](#)

Data object constant value filter attribute

[DataObjectAttribute](#)

Data object metadata attribute

[DataSourceAttribute](#)

Base data object source attribute

[DataTableAttribute](#)

Data object table attribute

[DictionaryFilterAttribute](#)

Data object dictionary filter attribute

[ExpressionFilterAttribute](#)

Data object plain SQL expression filter attribute

[ExternalLinkAttribute](#)

Data object external link to other (child) data objects attribute

[FilterAttribute](#)

Base inner filter attribute

[FilterBaseAttribute](#)

Base filter description attribute

[FilterCombinationAttribute](#)

Data object filter combination (filter group) attribute

[HierarchyAttribute](#)

Hierarchy attribute

[HintAttribute](#)

[HintInfo](#)

Hint description

[InnerViewAttribute](#)

Data object inner view attribute

[Link<TVal, TSrc>](#)

Dictionary link wrapper

[LinkAttribute](#)

Base link attribute

[LinkPropertyAttribute](#)

Data object link property attribute

[Lob](#)

Blob field wrapper

[OptionalFilterAttribute](#)

Base optional filter attribute

[ParameterAttribute](#)

SQL stored procedure/function parameter

[ProcedureAttribute](#)

Data object SQL procedure / function attribute

[PropertyAttribute](#)

Data object property attribute

[PropertyBaseAttribute](#)

Data property base attribute

[PropertyDefaultAttribute](#)

Data object property default value attribute

[PropertyExpressionAttribute](#)

Data object property plain expression attribute

[PropertyGroupingAttribute](#)

Data object property grouping attribute

[PropertyUpdateWithAttribute](#)

Data object property update source expression attribute

[RangeFilterAttribute](#)

Data object range filter attribute

[ReferenceAttribute](#)

Data object referenced inner view property attribute

[ResultSetAttribute](#)

SQL stored procedure/function result set

[SqlExpressionAttribute](#)

Sql expression name

[SubqueryAttribute](#)

Data object subquery attribute

[SubqueryFilterAttribute](#)

Data object subquery filter attribute

[SubqueryLinkAttribute](#)

Data object link to subquery attribute

[SubqueryVariableAttribute](#)

Data object subquery variable attribute

[WithAttribute](#)

Common-Table-Expression (WITH) Subquery data source

[WithRecursiveAttribute](#)

Common-Table-Expression (WITH) recursive subquery data source

[Xml](#)

Xml field wrapper

[Enums](#)

[HintType](#)

Hint type

Class ActionAttribute

Data action attribute

Inheritance

System.Object
System.Attribute
ActionAttribute

Inherited Members

System.Attribute.Equals(System.Object)
System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type)
System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type, System.Boolean)
System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type)
System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type, System.Boolean)
System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type)
System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type, System.Boolean)
System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type)
System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.Assembly)
System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type)
System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo)
System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type)
System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.Module)
System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type)
System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo)
System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type)
System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type, System.Boolean)
System.Attribute.GetHashCode()
System.Attribute.IsDefaultAttribute()
System.Attribute.IsDefined(System.Reflection.Assembly, System.Type)
System.Attribute.IsDefined(System.Reflection.Assembly, System.Type, System.Boolean)
System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type)
System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type, System.Boolean)
System.Attribute.IsDefined(System.Reflection.Module, System.Type)
System.Attribute.IsDefined(System.Reflection.Module, System.Type, System.Boolean)
System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type)
System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type, System.Boolean)
System.Attribute.Match(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.GetType()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.ToString()

Namespace: [XData.Mapping](#)

Assembly: XData.docfx.dll

Syntax

[AttributeUsage(AttributeTargets.Property, AllowMultiple = true)]
public sealed class ActionAttribute : Attribute

Remarks

Used to assign one or more [DataActionType](#) to [IDataLogic<T>](#) action handler

Hint: Use [ActionOrder](#) to apply execution order of multiple handlers marked [ActionAttribute](#) for same [DataActionType](#)

Hint: Action handlers will executed on server side when using XData Application server

Constructors

ActionAttribute(DataActionType, Int32)

Data action attribute

Declaration

public ActionAttribute(DataActionType type, int order = 0)

Parameters

TYPE	NAME	DESCRIPTION
DataActionType	type	Type of action (see DataActionType)
System.Int32	order	Action execution order

Properties

ActionOrder

Action execution order

Declaration

public int ActionOrder { get; }

Property Value

TYPE	DESCRIPTION
System.Int32	

ActionType

Type of action (see DataActionType)

Declaration

public DataActionType ActionType { get; }

Property Value

TYPE	DESCRIPTION
DataActionType	

See Also

[DataActionType](#)

TypeId

When implemented in a derived class, gets a unique identifier for this System.Attribute.

Declaration

```
public override object TypeId { get; }
```

Property Value

TYPE	DESCRIPTION
System.Object	An System.Object that is a unique identifier for the attribute.

Overrides

System.Attribute.TypeId

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Class ColumnAttribute

Column mapping attribute

Inheritance

System.Object
System.Attribute
ColumnAttribute

Inherited Members

System.Attribute.Equals(System.Object)
System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type)
System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type, System.Boolean)
System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type)
System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type, System.Boolean)
System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type)
System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type, System.Boolean)
System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type)
System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.Assembly)
System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type)
System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo)
System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type)
System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.Module)
System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type)
System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo)
System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type)
System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type, System.Boolean)
System.Attribute.GetHashCode()
System.Attribute.IsDefaultAttribute()
System.Attribute.IsDefined(System.Reflection.Assembly, System.Type)
System.Attribute.IsDefined(System.Reflection.Assembly, System.Type, System.Boolean)
System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type)
System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type, System.Boolean)
System.Attribute.IsDefined(System.Reflection.Module, System.Type)
System.Attribute.IsDefined(System.Reflection.Module, System.Type, System.Boolean)
System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type)
System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type, System.Boolean)
System.Attribute.Match(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.GetType()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.ToString()

Namespace: [XData.Mapping](#)

Assembly: XData.docfx.dll

Syntax

```
[AttributeUsage(AttributeTargets.Class, AllowMultiple = true)]
public sealed class ColumnAttribute : Attribute
```

Remarks

Used to map data without declaring real property

This useful for declare [Id](#), introduce "virtual" property for [PropertyName](#), [ChildProperty](#) or can be used for grouping (see [GroupOrder](#))

Constructors

ColumnAttribute(String, Type, String, String)

Column attribute

Declaration

```
public ColumnAttribute(string propertyName, Type propertyType, string source = "", string fieldName = "")
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	propertyName	Property name
System.Type	propertyType	Property type
System.String	source	Source alias
System.String	fieldName	Field name

Exceptions

TYPE	CONDITION
XDataRuntimeException	Property name cannot be null or empty string
XDataRuntimeException	Field name cannot be null or empty string
XDataRuntimeException	Property type cannot be null

Properties

FieldName

Field name

Declaration

```
public string FieldName { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Flags

Property flags

Declaration

```
public DataPropertyFlag Flags { get; set; }
```

Property Value

TYPE	DESCRIPTION
DataPropertyFlag	

See Also

[DataPropertyFlag](#)

Grouping

Grouping mode

Declaration

```
public DataGrouping Grouping { get; set; }
```

Property Value

TYPE	DESCRIPTION
DataGrouping	

See Also

[DataGrouping](#)

GroupOrder

Grouping order

Declaration

```
public int GroupOrder { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Int32	

Hidden

Hidden column flag

Declaration

```
public bool Hidden { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

Remarks

Used to exclude column from query selected columns

Hidden column can be used as: [PropertyName](#), [ChildProperty](#) or for grouping (see [GroupOrder](#))

NativeSqlType

Native SQL type name

Declaration

```
public string NativeSqlType { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Remarks

Used to specify native SQL column type when conflicted with default type mapping

WARNING! May be not supported by database adapter!

OrderBy

Order by priority

Declaration

```
public int OrderBy { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Int32	

OrderByDesc

Order by desc flag

Declaration

```
public bool OrderByDesc { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

PropertyName

Property name

Declaration

```
public string PropertyName { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

PropertyType

Property type

Declaration

```
public Type PropertyType { get; }
```

Property Value

TYPE	DESCRIPTION
System.Type	

Reference

Referenced property name

Declaration

```
public string Reference { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

ReferenceSource

Referenced property source alias

Declaration

```
public string ReferenceSource { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Source

Data source alias

Declaration

```
public string Source { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

TypeId

When implemented in a derived class, gets a unique identifier for this System.Attribute.

Declaration

```
public override object TypeId { get; }
```

Property Value

TYPE	DESCRIPTION
System.Object	An System.Object that is a unique identifier for the attribute.

Overrides

System.Attribute.TypeId

Extension Methods

[SerializationExtensions.ToXml<T>\(T\)](#)

[CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)

[CollectionsExtensions.AsEnum<T>\(T\)](#)

[DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)

[Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)

[TypeExtensions.ChangeType<T>\(Object\)](#)

[TypeExtensions.ChangeType\(Object, Type\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)

[SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)

[QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Class ColumnDefaultAttribute

Column default value attribute

Inheritance

System.Object

System.Attribute

ColumnDefaultAttribute

Inherited Members

System.Attribute.Equals(System.Object)

System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Module)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.GetHashCode()

System.Attribute.IsDefaultAttribute()

System.Attribute.IsDefined(System.Reflection.Assembly, System.Type)

System.Attribute.IsDefined(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type)

System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.Module, System.Type)

System.Attribute.IsDefined(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type)

System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.Match(System.Object)

System.Object.Equals(System.Object, System.Object)

System.Object.GetType()

System.Object.MemberwiseClone()

System.Object.ReferenceEquals(System.Object, System.Object)

System.Object.ToString()

Namespace: [XData.Mapping](#)

Assembly: XData.docfx.dll

Syntax

```
[AttributeUsage(AttributeTargets.Class, AllowMultiple = true)]
public sealed class ColumnDefaultAttribute : Attribute
```

Remarks

Used to declare default value rule for column

Constructors

ColumnDefaultAttribute(String, DefaultType, Object)

Column default value attribute

Declaration

public ColumnDefaultAttribute(string propertyName, DefaultType source, object val = null)

Parameters

TYPE	NAME	DESCRIPTION
System.String	propertyName	Property name
DefaultType	source	Default value source
System.Object	val	Default value

Exceptions

TYPE	CONDITION
XDataRuntimeException	Property name cannot be null or empty string

Properties

DefaultFeature

Default extended features

Declaration

public DefaultFeature DefaultFeature { get; set; }

Property Value

TYPE	DESCRIPTION
DefaultFeature	

DefaultSource

Default value source

Declaration

```
public DefaultType DefaultSource { get; }
```

Property Value

TYPE	DESCRIPTION
DefaultType	

DefaultValue

Default value

Declaration

```
public object DefaultValue { get; }
```

Property Value

TYPE	DESCRIPTION
System.Object	

PropertyName

Property name

Declaration

```
public string PropertyName { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

See Also

[ColumnAttribute](#)

TypeId

When implemented in a derived class, gets a unique identifier for this System.Attribute.

Declaration

```
public override object TypeId { get; }
```

Property Value

TYPE	DESCRIPTION
System.Object	An System.Object that is a unique identifier for the attribute.

Overrides

System.Attribute.TypeId

Extension Methods

[SerializationExtensions.ToXml<T>\(T\)](#)

[CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)

[CollectionsExtensions.AsEnum<T>\(T\)](#)

[DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)

[Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)

[TypeExtensions.ChangeType<T>\(Object\)](#)

[TypeExtensions.ChangeType\(Object, Type\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)

[SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)

[QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

See Also

[DefaultType](#)

Class ColumnExpressionAttribute

Column expression attribute

Inheritance

System.Object

System.Attribute

ColumnExpressionAttribute

Inherited Members

System.Attribute.Equals(System.Object)

System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Module)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.GetHashCode()

System.Attribute.IsDefaultAttribute()

System.Attribute.IsDefined(System.Reflection.Assembly, System.Type)

System.Attribute.IsDefined(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type)

System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.Module, System.Type)

System.Attribute.IsDefined(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type)

System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.Match(System.Object)

System.Object.Equals(System.Object, System.Object)

System.Object.GetType()

System.Object.MemberwiseClone()

System.Object.ReferenceEquals(System.Object, System.Object)

System.Object.ToString()

Namespace: [XData.Mapping](#)

Assembly: XData.docfx.dll

Syntax

```
[AttributeUsage(AttributeTargets.Class, AllowMultiple = true)]
public sealed class ColumnExpressionAttribute : Attribute
```

Remarks

Used when [ColumnAttribute](#) value is expression ([PlainSql](#) or [SubQuery](#)) according [ExprType](#)

Constructors

ColumnExpressionAttribute(String, String, DataExpressionType, DbType)

Column expression attribute

Declaration

```
public ColumnExpressionAttribute(string propertyName, string exprText, DataExpressionType exprType =
DataExpressionType.PlainSql, DbType type = DbType.String)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	propertyName	Property name
System.String	exprText	Expression text
DataExpressionType	exprType	Expression type
System.Data.DbType	type	Expression DbType

Exceptions

TYPE	CONDITION
XDataRuntimeException	Property name cannot be null or empty string
XDataRuntimeException	Expression text cannot be null or empty string

Properties

DbType

Expression DbType

Declaration

```
public DbType DbType { get; }
```

Property Value

TYPE	DESCRIPTION
System.Data.DbType	

Exists

Exists subquery operation

Declaration

```
public ExistsOperation Exists { get; set; }
```

Property Value

TYPE	DESCRIPTION
ExistsOperation	

ExprScale

Expression scale

Declaration

```
public int ExprScale { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Int32	

ExprSize

Expression size

Declaration

```
public int ExprSize { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Int32	

ExprText

Expression text

Declaration

```
public string ExprText { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Remarks

[PlainSql](#) expression text is Sql expression or [SqlExpressionAttribute](#) name

[SubQuery](#) expression text is [Alias](#)

WARNING! Plain Sql expressions (see [ExprType](#)) can use specific Sql dialect features! Data mapping may be not compatible with other database engines! Use [SqlExpressionAttribute](#) instead.

ExprType

Expression type

Declaration

```
public DataExpressionType ExprType { get; }
```

Property Value

TYPE	DESCRIPTION
DataExpressionType	

Remarks

WARNING! Plain Sql expressions can use specific Sql dialect features! Data mapping may be not compatible with other database engines!

See Also

[DataExpressionType](#)

PropertyName

Property name

Declaration

```
public string PropertyName { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

See Also

[ColumnAttribute](#)

TypeId

When implemented in a derived class, gets a unique identifier for this System.Attribute.

Declaration

```
public override object TypeId { get; }
```

Property Value

TYPE	DESCRIPTION
System.Object	An System.Object that is a unique identifier for the attribute.

Overrides

System.Attribute.TypeId

Extension Methods

SerializationExtensions.ToXml<T>(T)

CollectionsExtensions.SetValue<TKey, T>(TKey, T)

CollectionsExtensions.AsEnum<T>(T)

DataObjectQueryableExtensions.CompareVariable(Object, FilterOperation, String)

Processing.Do<T>(T, Action<IProcess<T>>[])

TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>)

TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>, TResult)

TypeExtensions.ChangeType<T>(Object)

TypeExtensions.ChangeType(Object, Type)

TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)

TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)

SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)

QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Class ColumnUpdateWithAttribute

Column update source expression attribute

Inheritance

System.Object

System.Attribute

ColumnUpdateWithAttribute

Inherited Members

System.Attribute.Equals(System.Object)

System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Module)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.GetHashCode()

System.Attribute.IsDefaultAttribute()

System.Attribute.IsDefined(System.Reflection.Assembly, System.Type)

System.Attribute.IsDefined(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type)

System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.Module, System.Type)

System.Attribute.IsDefined(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type)

System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.Match(System.Object)

System.Object.Equals(System.Object, System.Object)

System.Object.GetType()

System.Object.MemberwiseClone()

System.Object.ReferenceEquals(System.Object, System.Object)

System.Object.ToString()

Namespace: [XData.Mapping](#)

Assembly: XData.docfx.dll

Syntax

```
[AttributeUsage(AttributeTargets.Class, AllowMultiple = true)]
public sealed class ColumnUpdateWithAttribute : Attribute
```

Remarks

Used when [ColumnAttribute](#) value must to be updated with expression ([PlainSql](#) or [SubQuery](#)) result according [ExprType](#)

Constructors

ColumnUpdateWithAttribute(String, String, DataExpressionType)

Column update source expression attribute

Declaration

```
public ColumnUpdateWithAttribute(string propertyName, string exprText, DataExpressionType exprType =
DataExpressionType.PlainSql)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	propertyName	Property name
System.String	exprText	Expression text
DataExpressionType	exprType	Expression type

Exceptions

TYPE	CONDITION
XDataRuntimeException	Property name cannot be null or empty string
XDataRuntimeException	Expression text cannot be null or empty string

Properties

ExprText

Expression text

Declaration

```
public string ExprText { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Remarks

PlainSql expression text is Sql expression or [SqlExpressionAttribute](#) name

SubQuery expression text is [Alias](#)

WARNING! Plain Sql expressions (see [ExprType](#)) can use specific Sql dialect features! Data mapping may be not compatible with other database engines! Use [SqlExpressionAttribute](#) instead.

ExprType

Expression type

Declaration

```
public DataExpressionType ExprType { get; }
```

Property Value

TYPE	DESCRIPTION
DataExpressionType	

Remarks

WARNING! Plain Sql expressions can use specific Sql dialect features! Data mapping may be not compatible with other database engines!

See Also

[DataExpressionType](#)

PropertyName

Property name

Declaration

```
public string PropertyName { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

See Also

[ColumnAttribute](#)

TypeId

When implemented in a derived class, gets a unique identifier for this System.Attribute.

Declaration

```
public override object TypeId { get; }
```

Property Value

TYPE	DESCRIPTION
System.Object	An System.Object that is a unique identifier for the attribute.

Overrides

System.Attribute.TypeId

Extension Methods

SerializationExtensions.ToXml<T>(T)

CollectionsExtensions.SetValue<TKey, T>(TKey, T)

CollectionsExtensions.AsEnum<T>(T)

DataObjectQueryableExtensions.CompareVariable(Object, FilterOperation, String)

Processing.Do<T>(T, Action<IProcess<T>>[])

TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>)

TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>, TResult)

TypeExtensions.ChangeType<T>(Object)

TypeExtensions.ChangeType(Object, Type)

TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)

TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)

SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)

QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Class ConstantFilterAttribute

Data object constant value filter attribute

Inheritance

System.Object

System.Attribute

[FilterBaseAttribute](#)

[FilterAttribute](#)

ConstantFilterAttribute

Inherited Members

[FilterAttribute.Source](#)

[FilterAttribute.FieldName](#)

[FilterAttribute.Operation](#)

[FilterAttribute.Combination](#)

System.Attribute.Equals(System.Object)

System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Module)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.GetHashCode()

System.Attribute.IsDefaultAttribute()

System.Attribute.IsDefined(System.Reflection.Assembly, System.Type)

System.Attribute.IsDefined(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type)

System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.Module, System.Type)

System.Attribute.IsDefined(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type)

System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.Match(System.Object)

System.Object.Equals(System.Object, System.Object)

System.Object.GetType()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.ToString()

Namespace: [XData.Mapping](#)

Assembly: XData.docfx.dll

Syntax

```
[AttributeUsage(AttributeTargets.Class, AllowMultiple = true)]  
public sealed class ConstantFilterAttribute : FilterAttribute
```

Remarks

Filter value can be constant, data object variable value (see [Variables](#)), current date or time

Constructors

ConstantFilterAttribute(String, String, String, Object)

Constant filter attribute

Declaration

```
public ConstantFilterAttribute(string name, string source, string fieldName, object constant)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Filter name
System.String	source	Filter source alias
System.String	fieldName	Filter field name
System.Object	constant	Filter value

Exceptions

TYPE	CONDITION
XDataRuntimeException	Filter name cannot be null or empty string

Properties

ConstantType

Filter value type

Declaration

```
public FilterConstantType ConstantType { get; set; }
```

Property Value

TYPE	DESCRIPTION
FilterConstantType	

See Also

[FilterConstantType](#)

ConstantValue

Filter value

Declaration

```
public object ConstantValue { get; }
```

Property Value

TYPE	DESCRIPTION
System.Object	

See Also

[ConstantType](#)

[FilterConstantType](#)

FilterType

Filter type

Declaration

```
public override FilterType FilterType { get; }
```

Property Value

TYPE	DESCRIPTION
FilterType	Constant

Overrides

[FilterBaseAttribute.FilterType](#)

Name

Filter parameter name

Declaration

```
public string Name { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Remarks

No Sql parameter prefix needed

Nullable

Filter null value handling mode

Declaration

```
public FilterNullable Nullable { get; set; }
```

Property Value

TYPE	DESCRIPTION
FilterNullable	

See Also

[FilterConstantType](#)

TypeId

When implemented in a derived class, gets a unique identifier for this System.Attribute.

Declaration

```
public override object TypeId { get; }
```

Property Value

TYPE	DESCRIPTION
System.Object	An System.Object that is a unique identifier for the attribute.

Overrides

System.Attribute.TypeId

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Class DataObjectAttribute

Data object metadata attribute

Inheritance

System.Object
System.Attribute
DataObjectAttribute

Inherited Members

System.Attribute.Equals(System.Object)
System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type)
System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type, System.Boolean)
System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type)
System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type, System.Boolean)
System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type)
System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type, System.Boolean)
System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type)
System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.Assembly)
System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type)
System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo)
System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type)
System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.Module)
System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type)
System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo)
System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type)
System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type, System.Boolean)
System.Attribute.GetHashCode()
System.Attribute.IsDefaultAttribute()
System.Attribute.IsDefined(System.Reflection.Assembly, System.Type)
System.Attribute.IsDefined(System.Reflection.Assembly, System.Type, System.Boolean)
System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type)
System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type, System.Boolean)
System.Attribute.IsDefined(System.Reflection.Module, System.Type)
System.Attribute.IsDefined(System.Reflection.Module, System.Type, System.Boolean)
System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type)
System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type, System.Boolean)
System.Attribute.Match(System.Object)
System.Attribute.TypeId
System.Object.Equals(System.Object, System.Object)
System.Object.GetType()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.ToString()

Namespace: [XData.Mapping](#)

Syntax

```
[AttributeUsage(AttributeTargets.Class)]
public sealed class DataObjectAttribute : Attribute
```

Remarks

WARNING! Required attribute!

Constructors

DataObjectAttribute(String)

Data object metadata attribute

Declaration

```
public DataObjectAttribute(string baseSource = "")
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	baseSource	Base table alias

Remarks

If Base table is empty - data object has no base table.

WARNING! None based data objects are read only and can not contain any table!

DataObjectAttribute(Type)

Data object metadata attribute

Declaration

```
public DataObjectAttribute(Type baseMappingType)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Type	baseMappingType	Base mapping type

Properties

BaseMappingType

Base mapping type

Declaration

```
public Type BaseMappingType { get; }
```

Property Value

TYPE	DESCRIPTION
System.Type	

BaseSource

Base table alias

Declaration

```
public string BaseSource { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Remarks

Base table is modification allowed data source subtree root of data object

See Also

[Alias](#)

[Parent](#)

Context

Context alias

Declaration

```
public string Context { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

See Also

[Context](#)

Distinct

Distinct flag

Declaration

```
public bool Distinct { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

Flags

Combination of DataObjectFlag values

Declaration

```
public DataStructureFlag Flags { get; set; }
```

Property Value

TYPE	DESCRIPTION
DataStructureFlag	

See Also

[DataStructureFlag](#)

Limit

Limit records

Declaration

```
public int Limit { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Int32	

LogicAssemblyNames

Data logic assembly name

Declaration

```
public string[] LogicAssemblyNames { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String[]	

Remarks

NOTE: Used when no IoC Container available. Otherwise register logic modules through container

See Also

[IDataLogic<T>](#)

Skip

Skip records

Declaration

```
public int Skip { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Int32	

Extension Methods

SerializationExtensions.ToXml<T>(T)

CollectionsExtensions.SetValue<TKey, T>(TKey, T)

CollectionsExtensions.AsEnum<T>(T)

DataObjectQueryableExtensions.CompareVariable(Object, FilterOperation, String)

Processing.Do<T>(T, Action<IProcess<T>>[])

TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>)

TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>, TResult)

TypeExtensions.ChangeType<T>(Object)

TypeExtensions.ChangeType(Object, Type)

TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)

TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)

SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)

QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Class DataSourceAttribute

Base data object source attribute

Inheritance

System.Object
System.Attribute
DataSourceAttribute
[DataTableAttribute](#)
[InnerViewAttribute](#)
[ProcedureAttribute](#)
[SubqueryAttribute](#)
[WithAttribute](#)

Inherited Members

System.Attribute.Equals(System.Object)
System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type)
System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type, System.Boolean)
System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type)
System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type, System.Boolean)
System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type)
System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type, System.Boolean)
System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type)
System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.Assembly)
System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type)
System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo)
System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type)
System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.Module)
System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type)
System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo)
System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type)
System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type, System.Boolean)
System.Attribute.GetHashCode()
System.Attribute.IsDefaultAttribute()
System.Attribute.IsDefined(System.Reflection.Assembly, System.Type)
System.Attribute.IsDefined(System.Reflection.Assembly, System.Type, System.Boolean)
System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type)
System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type, System.Boolean)
System.Attribute.IsDefined(System.Reflection.Module, System.Type)
System.Attribute.IsDefined(System.Reflection.Module, System.Type, System.Boolean)
System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type)
System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type, System.Boolean)
System.Attribute.Match(System.Object)
System.Attribute.TypeId
System.Object.Equals(System.Object, System.Object)

System.Object.GetType()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.ToString()

Namespace: [XData.Mapping](#)

Assembly: XData.docfx.dll

Syntax

```
[AttributeUsage(AttributeTargets.Class, AllowMultiple = true)]  
public abstract class DataSourceAttribute : Attribute
```

Properties

Alias

Data source alias

Declaration

```
public abstract string Alias { get; protected set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Type

Data source type

Declaration

```
public abstract DataSourceType Type { get; }
```

Property Value

TYPE	DESCRIPTION
DataSourceType	

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Class DataTableAttribute

Data object table attribute

Inheritance

System.Object

System.Attribute

[DataSourceAttribute](#)

DataTableAttribute

Inherited Members

System.Attribute.Equals(System.Object)

System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Module)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.GetHashCode()

System.Attribute.IsDefaultAttribute()

System.Attribute.IsDefined(System.Reflection.Assembly, System.Type)

System.Attribute.IsDefined(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type)

System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.Module, System.Type)

System.Attribute.IsDefined(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type)

System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.Match(System.Object)

System.Object.Equals(System.Object, System.Object)

System.Object.GetType()

System.Object.MemberwiseClone()

System.Object.ReferenceEquals(System.Object, System.Object)

System.Object.ToString()

Namespace: [XData.Mapping](#)

Syntax

```
[AttributeUsage(AttributeTargets.Class, AllowMultiple = true)]
public sealed class DataTableAttribute : DataSourceAttribute
```

Constructors

DataTableAttribute(String, String)

Data object table attribute

Declaration

```
public DataTableAttribute(string name, string alias)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Table name
System.String	alias	Table alias

Exceptions

TYPE	CONDITION
System.ArgumentException	Table alias cannot be null or empty string
XDataRuntimeException	Data table alias is null or empty.

Properties

Alias

Table alias

Declaration

```
public override string Alias { get; protected set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Overrides

[DataSourceAttribute.Alias](#)

Name

Table name

Declaration

```
public string Name { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Parent

Parent table alias

Declaration

```
public string Parent { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Remarks

Tables supporting updates must be organized in update hierarchy linked by ParentAlias.

Root table of update hierarchy is the data object base source (see [BaseSource](#))

See Also

[Alias](#)

Type

Data source type

Declaration

```
public override DataSourceType Type { get; }
```

Property Value

TYPE	DESCRIPTION
DataSourceType	Table

Overrides

[DataSourceAttribute.Type](#)

TypeId

When implemented in a derived class, gets a unique identifier for this System.Attribute.

Declaration

```
public override object TypeId { get; }
```

Property Value

TYPE	DESCRIPTION

TYPE	DESCRIPTION
System.Object	An System.Object that is a unique identifier for the attribute.

Overrides

System.Attribute.TypeId

Extension Methods

- SerializationExtensions.ToXml<T>(T)
- CollectionsExtensions.SetValue<TKey, T>(TKey, T)
- CollectionsExtensions.AsEnum<T>(T)
- DataObjectQueryableExtensions.CompareVariable(Object, FilterOperation, String)
- Processing.Do<T>(T, Action<IProcess<T>>[])
- TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>)
- TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>, TResult)
- TypeExtensions.ChangeType<T>(Object)
- TypeExtensions.ChangeType(Object, Type)
- TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)
- TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)
- TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)
- TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)
- SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)
- QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Class DictionaryFilterAttribute

Data object dictionary filter attribute

Inheritance

System.Object

System.Attribute

[FilterBaseAttribute](#)

[FilterAttribute](#)

DictionaryFilterAttribute

Inherited Members

[FilterAttribute.Source](#)

[FilterAttribute.FieldName](#)

[FilterAttribute.Operation](#)

[FilterAttribute.Combination](#)

System.Attribute.Equals(System.Object)

System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Module)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.GetHashCode()

System.Attribute.IsDefaultAttribute()

System.Attribute.IsDefined(System.Reflection.Assembly, System.Type)

System.Attribute.IsDefined(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type)

System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.Module, System.Type)

System.Attribute.IsDefined(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type)

System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.Match(System.Object)

System.Object.Equals(System.Object, System.Object)

System.Object.GetType()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.ToString()

Namespace: [XData.Mapping](#)

Assembly: XData.docfx.dll

Syntax

```
[AttributeUsage(AttributeTargets.Class, AllowMultiple = true)]  
public sealed class DictionaryFilterAttribute : FilterAttribute
```

Remarks

Filter by dictionary code (example: `doc_type_id = (select doc_type_id from t_doc_type where code = 'INVOICE')`)

WARNING! Dictionary restricted by single valued primary key!

WARNING! Dictionary code mast be unique!

Examples

```
DictionaryFilter("T_DOC_TYPE", "doc_type_id", "code", "INVOICE", "D")
```

Constructors

DictionaryFilterAttribute(String, String, String, String, String, String)

Dictionary filter attribute

Declaration

```
public DictionaryFilterAttribute(string dictionaryTable, string dictionaryIdField, string dictionaryCodeField,  
string dictionaryValue, string source, string fieldName = "")
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	dictionaryTable	Dictionary table name
System.String	dictionaryIdField	Dictionary id field
System.String	dictionaryCodeField	Dictionary code field
System.String	dictionaryValue	Dictionary code value
System.String	source	Filter source alias
System.String	fieldName	Filter field name

Exceptions

TYPE	CONDITION
XDataRuntimeException	Dictionary table name cannot be null or empty string
XDataRuntimeException	Dictionary id field cannot be null or empty string
XDataRuntimeException	Dictionary code field cannot be null or empty string
XDataRuntimeException	Dictionary code value cannot be null or empty string

Properties

DictionaryCode

Dictionary code field

Declaration

```
public string DictionaryCode { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

DictionaryId

Dictionary id field

Declaration

```
public string DictionaryId { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

DictionaryTable

Dictionary table name

Declaration

```
public string DictionaryTable { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

DictionaryValue

Dictionary code value

Declaration

```
public string DictionaryValue { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

FilterType

Filter type

Declaration

```
public override FilterType FilterType { get; }
```

Property Value

TYPE	DESCRIPTION
FilterType	Dictionary

Overrides

[FilterBaseAttribute.FilterType](#)

IsVariableValue

Variable name as filter value

Declaration

```
public bool IsVariableValue { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

Remarks

If true [DictionaryValue](#) contains data object variable name which value will be compared with Dictionary code

TypeId

When implemented in a derived class, gets a unique identifier for this System.Attribute.

Declaration

```
public override object TypeId { get; }
```

Property Value

TYPE	DESCRIPTION

TYPE	DESCRIPTION
System.Object	An System.Object that is a unique identifier for the attribute.

Overrides

System.Attribute.TypeId

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

See Also

[Dictionary](#)

Class ExpressionFilterAttribute

Data object plain SQL expression filter attribute

Inheritance

System.Object

System.Attribute

[FilterBaseAttribute](#)

[FilterAttribute](#)

ExpressionFilterAttribute

Inherited Members

[FilterAttribute.Source](#)

[FilterAttribute.FieldName](#)

[FilterAttribute.Operation](#)

[FilterAttribute.Combination](#)

System.Attribute.Equals(System.Object)

System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Module)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.GetHashCode()

System.Attribute.IsDefaultAttribute()

System.Attribute.IsDefined(System.Reflection.Assembly, System.Type)

System.Attribute.IsDefined(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type)

System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.Module, System.Type)

System.Attribute.IsDefined(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type)

System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.Match(System.Object)

System.Object.Equals(System.Object, System.Object)

System.Object.GetType()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.ToString()

Namespace: [XData.Mapping](#)

Assembly: XData.docfx.dll

Syntax

```
[AttributeUsage(AttributeTargets.Class, AllowMultiple = true)]  
public sealed class ExpressionFilterAttribute : FilterAttribute
```

Remarks

Used to filter data by field comparison to plain SQL expression (for example: `some_field = substr(other_field, 1, 2)`)

Hint: Filter field can be represented as expression two by [ColumnExpressionAttribute](#) or [PropertyExpressionAttribute](#)

WARNING! Sql expressions can use specific Sql dialect features! Data mapping may be not compatible with other database engines!

Constructors

ExpressionFilterAttribute(String, String, String)

Expression filter attribute

Declaration

```
public ExpressionFilterAttribute(string source, string fieldName, string exprText)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	source	Primary source alias
System.String	fieldName	Primary field name
System.String	exprText	SQL expression text

Exceptions

TYPE	CONDITION
XDataRuntimeException	SQL expression cannot be null or empty string

Properties

ExprText

SQL expression

Declaration

```
public string ExprText { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Remarks

WARNING! SQL expressions can use specific SQL dialect features! Data mapping may be not compatible with other database engines!

FilterType

Filter type

Declaration

```
public override FilterType FilterType { get; }
```

Property Value

TYPE	DESCRIPTION
FilterType	Expression

Overrides

[FilterBaseAttribute.FilterType](#)

TypeId

When implemented in a derived class, gets a unique identifier for this System.Attribute.

Declaration

```
public override object TypeId { get; }
```

Property Value

TYPE	DESCRIPTION
System.Object	An System.Object that is a unique identifier for the attribute.

Overrides

System.Attribute.TypeId

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)

TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)

SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)

QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Class ExternalLinkAttribute

Data object external link to other (child) data objects attribute

Inheritance

System.Object
System.Attribute
ExternalLinkAttribute

Inherited Members

System.Attribute.Equals(System.Object)
System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type)
System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type, System.Boolean)
System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type)
System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type, System.Boolean)
System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type)
System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type, System.Boolean)
System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type)
System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.Assembly)
System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type)
System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo)
System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type)
System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.Module)
System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type)
System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo)
System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type)
System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type, System.Boolean)
System.Attribute.GetHashCode()
System.Attribute.IsDefaultAttribute()
System.Attribute.IsDefined(System.Reflection.Assembly, System.Type)
System.Attribute.IsDefined(System.Reflection.Assembly, System.Type, System.Boolean)
System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type)
System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type, System.Boolean)
System.Attribute.IsDefined(System.Reflection.Module, System.Type)
System.Attribute.IsDefined(System.Reflection.Module, System.Type, System.Boolean)
System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type)
System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type, System.Boolean)
System.Attribute.Match(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.GetType()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.ToString()

Namespace: [XData.Mapping](#)

Assembly: XData.docfx.dll

Syntax

```
[AttributeUsage(AttributeTargets.Class, AllowMultiple = true)]
public sealed class ExternalLinkAttribute : Attribute
```

Remarks

Used to organize master/detail relation between repositories

Constructors

ExternalLinkAttribute(Type, String, String, String)

External link attribute

Declaration

```
public ExternalLinkAttribute(Type childType, string childProperty, string property = "", string filterName =
    "")
```

Parameters

TYPE	NAME	DESCRIPTION
System.Type	childType	Child data object description type
System.String	childProperty	Child property to link
System.String	property	Parent property contains value to link
System.String	filterName	Filter name

Exceptions

TYPE	CONDITION
XDataRuntimeException	Child data object description type cannot be null
XDataRuntimeException	Child property to link cannot be null or empty string

Properties

ChildLinkType

Child linked property type

Declaration

```
public Type ChildLinkType { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Type	

Remarks

Used when child repository linked through subquery

ChildProperty

Child property to link

Declaration

```
public string ChildProperty { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

ChildType

Child data object type

Declaration

```
public Type ChildType { get; }
```

Property Value

TYPE	DESCRIPTION
System.Type	

Combination

Filter combination (filter group) name

Declaration

```
public string Combination { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

See Also

[FilterCombinationAttribute](#)

DirectLink

Direct link slave to selected tree node

Declaration

```
public bool DirectLink { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

Remarks

Used to show/hide subtree related items in slave view when master is tree

When ShowSubtree = false - slave contains selected node child items only

When ShowSubtree = true - selected node and it's subtree child items

Works with [ShowSubtree](#): External links marked as DirectLink is filled while ShowSubtree is false and omitted otherwise

FilterName

Filter name

Declaration

```
public string FilterName { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Remarks

If not assigned used FilterBy[ChildProperty](#) pattern

MasterRefresh

Refresh parent object notification on child submit

Declaration

```
public bool MasterRefresh { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

Remarks

Used when parent object contains calculated fields over child objects

Nullable

Filter null value handling

Declaration

```
public FilterNullable Nullable { get; set; }
```

Property Value

TYPE	DESCRIPTION
FilterNullable	

See Also

[FilterNullable](#)

Operation

Filter operation

Declaration

```
public FilterOperation Operation { get; set; }
```

Property Value

TYPE	DESCRIPTION
FilterOperation	

See Also

[FilterOperation](#)

PrimaryFilter

Primary filter flag

Declaration

```
public bool PrimaryFilter { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

See Also

[OptionalFilterAttribute](#)

PropertyName

Property to link

Declaration

```
public string PropertyName { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Remarks

If not assigned used [ChildProperty](#) name

TypeId

When implemented in a derived class, gets a unique identifier for this System Attribute

when implemented in a derived class, gets a unique identifier for this System.Attribute.

Declaration

```
public override object TypeId { get; }
```

Property Value

TYPE	DESCRIPTION
System.Object	An System.Object that is a unique identifier for the attribute.

Overrides

System.Attribute.TypeId

Extension Methods

- SerializationExtensions.ToXml<T>(T)
- CollectionsExtensions.SetValue<TKey, T>(TKey, T)
- CollectionsExtensions.AsEnum<T>(T)
- DataObjectQueryableExtensions.CompareVariable(Object, FilterOperation, String)
- Processing.Do<T>(T, Action<IProcess<T>>[])
- TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>)
- TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>, TResult)
- TypeExtensions.ChangeType<T>(Object)
- TypeExtensions.ChangeType(Object, Type)
- TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)
- TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)
- TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)
- TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)
- SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)
- QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Class FilterAttribute

Base inner filter attribute

Inheritance

System.Object

System.Attribute

[FilterBaseAttribute](#)

FilterAttribute

[ConstantFilterAttribute](#)

[DictionaryFilterAttribute](#)

[ExpressionFilterAttribute](#)

[OptionalFilterAttribute](#)

[RangeFilterAttribute](#)

Inherited Members

[FilterBaseAttribute.FilterType](#)

System.Attribute.Equals(System.Object)

System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Module)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.GetHashCode()

System.Attribute.IsDefaultAttribute()

System.Attribute.IsDefined(System.Reflection.Assembly, System.Type)

System.Attribute.IsDefined(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type)

System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.Module, System.Type)

System.Attribute.IsDefined(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type)

System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.Match(System.Object)

System.Attribute.TypeId
System.Object.Equals(System.Object, System.Object)
System.Object.GetType()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.ToString()

Namespace: [XData.Mapping](#)
Assembly: XData.docfx.dll

Syntax

```
[AttributeUsage(AttributeTargets.Class, AllowMultiple = true)]  
public abstract class FilterAttribute : FilterBaseAttribute
```

Constructors

FilterAttribute(String, String)

Base filter attribute

Declaration

```
protected FilterAttribute(string source, string fieldName)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	source	Primary source alias
System.String	fieldName	Primary field name

Exceptions

TYPE	CONDITION
XDataRuntimeException	Primary source alias cannot be null or empty string
XDataRuntimeException	Primary field name cannot be null or empty string

Properties

Combination

Filter combination (filter group) name

Declaration

```
public string Combination { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

See Also

[FilterCombinationAttribute](#)

FieldName

Primary field name

Declaration

```
public string FieldName { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Operation

Filter operation

Declaration

```
public FilterOperation Operation { get; set; }
```

Property Value

TYPE	DESCRIPTION
FilterOperation	

See Also

[FilterOperation](#)

Source

Primary source alias

Declaration

```
public string Source { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)

[SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)

[QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

See Also

[FilterCombinationAttribute](#)

[LinkAttribute](#)

[SubqueryLinkAttribute](#)

[ConstantFilterAttribute](#)

[DictionaryFilterAttribute](#)

[SubqueryFilterAttribute](#)

[RangeFilterAttribute](#)

[ExpressionFilterAttribute](#)

[RuntimeFilter](#)

Class FilterBaseAttribute

Base filter description attribute

Inheritance

System.Object

System.Attribute

FilterBaseAttribute

[FilterAttribute](#)

[SubqueryLinkAttribute](#)

Inherited Members

System.Attribute.Equals(System.Object)

System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Module)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.GetHashCode()

System.Attribute.IsDefaultAttribute()

System.Attribute.IsDefined(System.Reflection.Assembly, System.Type)

System.Attribute.IsDefined(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type)

System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.Module, System.Type)

System.Attribute.IsDefined(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type)

System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.Match(System.Object)

System.Attribute.TypeId

System.Object.Equals(System.Object, System.Object)

System.Object.GetType()

System.Object.MemberwiseClone()

System.Object.ReferenceEquals(System.Object, System.Object)

System.Object.ToString()

Namespace: [XData.Mapping](#)
Assembly: XData.docfx.dll

Syntax

```
public abstract class FilterBaseAttribute : Attribute
```

Properties

FilterType

Filter type

Declaration

```
public abstract FilterType FilterType { get; }
```

Property Value

TYPE	DESCRIPTION
FilterType	

See Also

[FilterType](#)

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Class FilterCombinationAttribute

Data object filter combination (filter group) attribute

Inheritance

System.Object

System.Attribute

FilterCombinationAttribute

Inherited Members

System.Attribute.Equals(System.Object)

System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Module)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.GetHashCode()

System.Attribute.IsDefaultAttribute()

System.Attribute.IsDefined(System.Reflection.Assembly, System.Type)

System.Attribute.IsDefined(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type)

System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.Module, System.Type)

System.Attribute.IsDefined(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type)

System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.Match(System.Object)

System.Object.Equals(System.Object, System.Object)

System.Object.GetType()

System.Object.MemberwiseClone()

System.Object.ReferenceEquals(System.Object, System.Object)

System.Object.ToString()

Namespace: [XData.Mapping](#)

Assembly: XData.docfx.dll

Syntax

```
[AttributeUsage(AttributeTargets.Class, AllowMultiple = true)]
public sealed class FilterCombinationAttribute : Attribute
```

Remarks

Filter combinations combine filters into named logical groups united by same logical operation (see [Combination](#))

Default filter combination always exists. It have [And](#) combination and named as System.String.Empty

Filter combination tree based on [Parent](#) property

Constructors

FilterCombinationAttribute(String)

Data object filter combination (filter group) attribute constructor

Declaration

```
public FilterCombinationAttribute(string name)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Name of filter combination (filter group)

Exceptions

TYPE	CONDITION
XDataRuntimeException	Name of filter combination (filter group) cannot be null or empty string

Properties

Combination

Filter combination (filter group) logical operation (And or Or)

Declaration

```
public Combination Combination { get; set; }
```

Property Value

TYPE	DESCRIPTION
Combination	

Name

Name of filter combination (filter group)

Declaration

```
public string Name { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Parent

Parent filter combination (filter group) name

Declaration

```
public string Parent { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

TypeId

When implemented in a derived class, gets a unique identifier for this System.Attribute.

Declaration

```
public override object TypeId { get; }
```

Property Value

TYPE	DESCRIPTION
System.Object	An System.Object that is a unique identifier for the attribute.

Overrides

System.Attribute.TypeId

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Class HierarchyAttribute

Hierarchy attribute

Inheritance

System.Object

System.Attribute

HierarchyAttribute

Inherited Members

System.Attribute.Equals(System.Object)

System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Module)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.GetHashCode()

System.Attribute.IsDefaultAttribute()

System.Attribute.IsDefined(System.Reflection.Assembly, System.Type)

System.Attribute.IsDefined(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type)

System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.Module, System.Type)

System.Attribute.IsDefined(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type)

System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.Match(System.Object)

System.Object.Equals(System.Object, System.Object)

System.Object.GetType()

System.Object.MemberwiseClone()

System.Object.ReferenceEquals(System.Object, System.Object)

System.Object.ToString()

Namespace: [XData.Mapping](#)

Assembly: XData.docfx.dll

```
[AttributeUsage(AttributeTargets.Class, AllowMultiple = true)]
public sealed class HierarchyAttribute : Attribute
```

Remarks

Tables can be hierarchically linked to themselves.

On every hierarchical link is possible to build hierarchy table, to simplify tree structure queries.

It's alternative of ORACLE CONNECT BY PRIOR...START WITH construction for other database engines

```
// Table T_TEST has parent_id field tree linked to test_id (T_TEST's pk)

// Table L_TEST has two fields parent_id and child_id all linked to T_TEST.test_id

// Table L_TEST contains all subtree relations inside T_TEST

// For example: for T_TEST (test_id, parent_id): (1, null) : (2, 1) : (3, 2)

// L_TEST will contain (parent_id, child_id): (1, 1) : (1, 2) : (1, 3) : (2, 2) : (2, 3) : (3, 3)

[DataObject("T"),

    DataTable("T_TEST", "T"),

    Hierarchy("T", "L_TEST", "parent_id", "parent_id", "child_id"),

    Column("TestId", "test_id", typeof(long?), "T", Flags = DataPropertyFlag.Id)]
```

WARNING! Hierarchy base table (T_TEST in example above) mast contain non complex primary key!

Hint: Possible to describe multiple hierarchies across one table using different tree links

Constructors

HierarchyAttribute(String, String, String, String, String)

Hierarchy attribute

Declaration

```
public HierarchyAttribute(string source, string treeTableName, string parent, string linkParent, string linkChild)
```

Parameters

TYPE	NAME	DESCRIPTION

TYPE	NAME	DESCRIPTION
System.String	source	Source table alias
System.String	treeTableName	Link table name
System.String	parent	Source table parent column name
System.String	linkParent	Link table parent column name
System.String	linkChild	Link table child column name

Exceptions

TYPE	CONDITION
XDataRuntimeException	Source table alias cannot be null or empty string
XDataRuntimeException	Link table name cannot be null or empty string
XDataRuntimeException	Source table parent column name cannot be null or empty string
XDataRuntimeException	Link table parent column name cannot be null or empty string
XDataRuntimeException	Link table child column name cannot be null or empty string

Properties

LinkChild

Link table child column name

Declaration

```
public string LinkChild { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

LinkParent

Link table parent column name

Declaration

```
public string LinkParent { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Parent

Source table parent column name

Declaration

```
public string Parent { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Source

Source table alias

Declaration

```
public string Source { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

See Also

[Alias](#)

TreeTableName

Link table name

Declaration

```
public string TreeTableName { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Remarks

Tree table contains links from tree node to all subtree nodes.

TypeId

When implemented in a derived class, gets a unique identifier for this System.Attribute.

Declaration

```
public override object TypeId { get; }
```

Property Value

TYPE	DESCRIPTION
System.Object	An System.Object that is a unique identifier for the attribute.

Overrides

System.Attribute.TypeId

Extension Methods

- SerializationExtensions.ToXml<T>(T)
- CollectionsExtensions.SetValue<TKey, T>(TKey, T)
- CollectionsExtensions.AsEnum<T>(T)
- DataObjectQueryableExtensions.CompareVariable(Object, FilterOperation, String)
- Processing.Do<T>(T, Action<IProcess<T>>[])
- TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>)
- TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>, TResult)
- TypeExtensions.ChangeType<T>(Object)
- TypeExtensions.ChangeType(Object, Type)
- TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)
- TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)
- TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)
- TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)
- SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)
- QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Class HintAttribute

Inheritance

System.Object
System.Attribute
HintAttribute

Inherited Members

System.Attribute.Equals(System.Object)
System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type)
System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type, System.Boolean)
System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type)
System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type, System.Boolean)
System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type)
System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type, System.Boolean)
System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type)
System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.Assembly)
System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type)
System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo)
System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type)
System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.Module)
System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type)
System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo)
System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type)
System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type, System.Boolean)
System.Attribute.GetHashCode()
System.Attribute.IsDefaultAttribute()
System.Attribute.IsDefined(System.Reflection.Assembly, System.Type)
System.Attribute.IsDefined(System.Reflection.Assembly, System.Type, System.Boolean)
System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type)
System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type, System.Boolean)
System.Attribute.IsDefined(System.Reflection.Module, System.Type)
System.Attribute.IsDefined(System.Reflection.Module, System.Type, System.Boolean)
System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type)
System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type, System.Boolean)
System.Attribute.Match(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.GetType()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.ToString()

Namespace: [XData.Mapping](#)

Assembly: XData.docfx.dll

Syntax

```
[AttributeUsage(AttributeTargets.Class, AllowMultiple = true)]
public sealed class HintAttribute : Attribute
```

Constructors

HintAttribute(String, String, HintType)

Hint attribute constructor

Declaration

```
public HintAttribute(string alias, string hint, HintType hintType = HintType.Select)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	alias	Data source alias
System.String	hint	Hint text
HintType	hintType	Hint type

Properties

Alias

Data source alias to apply hint

Declaration

```
public string Alias { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Hint

Hint

Declaration

```
public HintInfo Hint { get; }
```

Property Value

TYPE	DESCRIPTION
HintInfo	

TypeId

When implemented in a derived class, gets a unique identifier for this System.Attribute.

Declaration

```
public override object TypeId { get; }
```

Property Value

TYPE	DESCRIPTION
System.Object	An System.Object that is a unique identifier for the attribute.

Overrides

System.Attribute.TypeId

Extension Methods

- SerializationExtensions.ToXml<T>(T)
- CollectionsExtensions.SetValue<TKey, T>(TKey, T)
- CollectionsExtensions.AsEnum<T>(T)
- DataObjectQueryableExtensions.CompareVariable(Object, FilterOperation, String)
- Processing.Do<T>(T, Action<IProcess<T>>[])
- TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>)
- TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>, TResult)
- TypeExtensions.ChangeType<T>(Object)
- TypeExtensions.ChangeType(Object, Type)
- TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)
- TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)
- TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)
- TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)
- SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)
- QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Class HintInfo

Hint description

Inheritance

System.Object

HintInfo

Inherited Members

- System.Object.Equals(System.Object)
- System.Object.Equals(System.Object, System.Object)
- System.Object.GetHashCode()
- System.Object.GetType()
- System.Object.MemberwiseClone()
- System.Object.ReferenceEquals(System.Object, System.Object)
- System.Object.ToString()

Namespace: [XData.Mapping](#)

Assembly: XData.docfx.dll

Syntax

```
public class HintInfo
```

Properties

Hint

Hint text

Declaration

```
public string Hint { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

HintType

Hint type

Declaration

```
public HintType HintType { get; set; }
```

Property Value

TYPE	DESCRIPTION
HintType	

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)

TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>)

TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>, TResult)

TypeExtensions.ChangeType<T>(Object)

TypeExtensions.ChangeType(Object, Type)

TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)

TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)

SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)

QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Enum HintType

Hint type

Namespace: [XData.Mapping](#)

Assembly: XData.docfx.dll

Syntax

```
[Flags]
public enum HintType
```

Fields

NAME	DESCRIPTION
All	
Delete	
Insert	
Select	
Update	

Extension Methods

- [SerializationExtensions.ToXml<HintType>\(\)](#)
- [CollectionsExtensions.SetValue<HintType, T>\(T\)](#)
- [CollectionsExtensions.AsEnum<HintType>\(\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(FilterOperation, String\)](#)
- [Processing.Do<HintType>\(Action<IProcess<HintType>> \[\]\)](#)
- [TypeExtensions.ChangeType<T>\(\)](#)
- [TypeExtensions.ChangeType\(Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.ConvertTo<T>\(\)](#)
- [TypeExtensions.EqualValue\(Enum\)](#)
- [SqlBlockExtensions.SetExpression<HintType, TResult>\(Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<HintType, TResult>\(Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Class InnerViewAttribute

Data object inner view attribute

Inheritance

System.Object
System.Attribute
[DataSourceAttribute](#)
InnerViewAttribute

Inherited Members

System.Attribute.Equals(System.Object)
System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type)
System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type, System.Boolean)
System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type)
System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type, System.Boolean)
System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type)
System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type, System.Boolean)
System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type)
System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.Assembly)
System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type)
System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo)
System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type)
System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.Module)
System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type)
System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo)
System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type)
System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type, System.Boolean)
System.Attribute.GetHashCode()
System.Attribute.IsDefaultAttribute()
System.Attribute.IsDefined(System.Reflection.Assembly, System.Type)
System.Attribute.IsDefined(System.Reflection.Assembly, System.Type, System.Boolean)
System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type)
System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type, System.Boolean)
System.Attribute.IsDefined(System.Reflection.Module, System.Type)
System.Attribute.IsDefined(System.Reflection.Module, System.Type, System.Boolean)
System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type)
System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type, System.Boolean)
System.Attribute.Match(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.GetType()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.ToString()

Namespace: [XData.Mapping](#)

Syntax

```
[AttributeUsage(AttributeTargets.Class, AllowMultiple = true)]
public sealed class InnerViewAttribute : DataSourceAttribute
```

Examples

```
InnerView("H", typeof(DocLastChange))
```

Constructors

InnerViewAttribute(String, Type)

Data object inner view attribute

Declaration

```
public InnerViewAttribute(string alias, Type innerViewType)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	alias	Inner view alias
System.Type	innerViewType	Object type represents inner view structure

Exceptions

TYPE	CONDITION
XDataRuntimeException	Inner view alias cannot be null or empty string
XDataRuntimeException	Object type represents inner view structure cannot be null

Properties

Alias

Inner view alias

Declaration

```
public override string Alias { get; protected set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Overrides

[DataSourceAttribute.Alias](#)

InnerViewType

Object type represents inner view structure

Declaration

```
public Type InnerViewType { get; }
```

Property Value

TYPE	DESCRIPTION
System.Type	

Type

Source type

Declaration

```
public override DataSourceType Type { get; }
```

Property Value

TYPE	DESCRIPTION
DataSourceType	InnerView

Overrides

[DataSourceAttribute.Type](#)

TypeId

When implemented in a derived class, gets a unique identifier for this System.Attribute.

Declaration

```
public override object TypeId { get; }
```

Property Value

TYPE	DESCRIPTION
System.Object	An System.Object that is a unique identifier for the attribute.

Overrides

System.Attribute.TypeId

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)

[SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)

[QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

See Also

[DataTableAttribute](#)

[SubqueryAttribute](#)

[SubqueryLinkAttribute](#)

Class Link<TVal, TSrc>

Dictionary link wrapper

Inheritance

System.Object

Link<TVal, TSrc>

Inherited Members

- System.Object.Equals(System.Object)
- System.Object.Equals(System.Object, System.Object)
- System.Object.GetHashCode()
- System.Object.GetType()
- System.Object.MemberwiseClone()
- System.Object.ReferenceEquals(System.Object, System.Object)
- System.Object.ToString()

Namespace: [XData.Mapping](#)

Assembly: XData.docfx.dll

Syntax

```
public abstract class Link<TVal, TSrc> : ILink where TVal : IComparable<TVal> where TSrc : class, IDataObject, new()
```

Type Parameters

NAME	DESCRIPTION
TVal	Link value type
TSrc	Link source type

Examples

```
// declaration
[Property("name", "S"),
 DictionaryProperty("Name", "DocState"),
 DictionaryProperty("Code", "DocStateCode")]
public Link<string, DocState> DocState { get; set; }
// using
newInvoice.DocState += dataScope.GetDictionaryValue<DocState>(x => x.Code == "CREATED");
// is the same as
newInvoice.DocState.Source = dataScope.GetDictionaryValue<DocState>(x => x.Code == "CREATED");
```

Properties

Source

Dictionary source data object

Declaration

```
public abstract TSrc Source { set; }
```

Property Value

TYPE	DESCRIPTION
TSrc	

Value

Dictionary value

Declaration

```
public TVal Value { get; protected set; }
```

Property Value

TYPE	DESCRIPTION
TVal	

Methods

IsNull()

Check link for null value

Declaration

```
public bool IsNull()
```

Returns

TYPE	DESCRIPTION
System.Boolean	Link is null

Key<TKey>(String)

Declaration

```
public abstract TKey Key<TKey>(string keyProperty)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	keyProperty	

Returns

TYPE	DESCRIPTION
TKey	

Type Parameters

NAME	DESCRIPTION
TKey	

Operators

Addition(Link<TVal, TSrc>, TSrc)

Set link source

Declaration

```
public static Link<TVal, TSrc> operator +(Link<TVal, TSrc> link, TSrc source)
```

Parameters

TYPE	NAME	DESCRIPTION
Link<TVal, TSrc>	link	Link
TSrc	source	Link source

Returns

TYPE	DESCRIPTION
Link<TVal, TSrc>	Updated link

Implicit(Link<TVal, TSrc> to TVal)

Implicitly returns dictionary value

Declaration

```
public static implicit operator TVal(Link<TVal, TSrc> val)
```

Parameters

TYPE	NAME	DESCRIPTION
Link<TVal, TSrc>	val	Dictionary property

Returns

TYPE	DESCRIPTION
TVal	Value

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)

[SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)

[QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

See Also

[LinkPropertyAttribute](#)

[Value](#)

[Source](#)

[Addition\(Link<TVal, TSrc>, TSrc\)](#)

[Implicit\(Link<TVal, TSrc> to TVal\)](#)

[GetDictionaryValue<T>\(Expression<Func<T, Boolean>>, String, ISecuritySession, Variable\[\]\)](#)

[IsNull\(\)](#)

Class LinkAttribute

Base link attribute

Inheritance

System.Object

System.Attribute

[FilterBaseAttribute](#)

[FilterAttribute](#)

[OptionalFilterAttribute](#)

LinkAttribute

Inherited Members

[OptionalFilterAttribute.PrimaryFilter](#)

[FilterAttribute.Source](#)

[FilterAttribute.FieldName](#)

[FilterAttribute.Operation](#)

[FilterAttribute.Combination](#)

System.Attribute.Equals(System.Object)

System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Module)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.GetHashCode()

System.Attribute.IsDefaultAttribute()

System.Attribute.IsDefined(System.Reflection.Assembly, System.Type)

System.Attribute.IsDefined(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type)

System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.Module, System.Type)

System.Attribute.IsDefined(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type)

System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.Match(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.GetType()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.ToString()

Namespace: [XData.Mapping](#)
Assembly: XData.docfx.dll

Syntax

```
[AttributeUsage(AttributeTargets.Class, AllowMultiple = true)]  
public sealed class LinkAttribute : OptionalFilterAttribute
```

Constructors

LinkAttribute(String, String, String, String)

Base link attribute constructor

Declaration

```
public LinkAttribute(string linkedSource, string linkedFieldName, string source, string fieldName = "")
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	linkedSource	Secondary source alias
System.String	linkedFieldName	Secondary field name
System.String	source	Primary source alias
System.String	fieldName	Primary field name

Exceptions

TYPE	CONDITION
XDataRuntimeException	Secondary source alias cannot be null or empty string
XDataRuntimeException	Secondary field name cannot be null or empty string

Properties

FilterType

Filter type

Declaration


```
public override FilterType FilterType { get; }
```

Property Value

TYPE	DESCRIPTION
FilterType	Link

Overrides

[FilterBaseAttribute.FilterType](#)

LinkedFieldName

Secondary field name

Declaration

```
public string LinkedFieldName { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

LinkedSourceAlias

Secondary source alias

Declaration

```
public string LinkedSourceAlias { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

TypeId

When implemented in a derived class, gets a unique identifier for this System.Attribute.

Declaration

```
public override object TypeId { get; }
```

Property Value

TYPE	DESCRIPTION
System.Object	An System.Object that is a unique identifier for the attribute.

Overrides

System.Attribute.TypeId

Extension Methods

[SerializationExtensions.ToXml<T>\(T\)](#)

[CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
[CollectionsExtensions.AsEnum<T>\(T\)](#)
[DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
[Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
[TypeExtensions.ChangeType<T>\(Object\)](#)
[TypeExtensions.ChangeType\(Object, Type\)](#)
[TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
[TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
[TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
[TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
[SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
[QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

See Also

[LinkAttribute](#)

[OptionalFilterAttribute](#)

Class LinkPropertyAttribute

Data object link property attribute

Inheritance

System.Object

System.Attribute

LinkPropertyAttribute

Inherited Members

System.Attribute.Equals(System.Object)

System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Module)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.GetHashCode()

System.Attribute.IsDefaultAttribute()

System.Attribute.IsDefined(System.Reflection.Assembly, System.Type)

System.Attribute.IsDefined(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type)

System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.Module, System.Type)

System.Attribute.IsDefined(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type)

System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.Match(System.Object)

System.Object.Equals(System.Object, System.Object)

System.Object.GetType()

System.Object.MemberwiseClone()

System.Object.ReferenceEquals(System.Object, System.Object)

System.Object.ToString()

Namespace: [XData.Mapping](#)

Assembly: XData.docfx.dll

Syntax

[AttributeUsage(AttributeTargets.Property, AllowMultiple = true)]
public sealed class LinkPropertyAttribute : Attribute

Remarks

Used to declare source and destination properties to copy values when assigned [Source](#)

Every link from [DictSource](#) to [Property](#) must be declared as separated [LinkPropertyAttribute](#)

WARNING! Applicable to [Link<TVal, TSrc>](#) typed properties only!

Constructors

LinkPropertyAttribute(String, String)

Data object link property attribute

Declaration

public LinkPropertyAttribute(string dictSource = null, string property = null)

Parameters

TYPE	NAME	DESCRIPTION
System.String	dictSource	Dictionary source object property
System.String	property	Property to fill

Properties

DictSource

Dictionary source object property

Declaration

public string DictSource { get; }

Property Value

TYPE	DESCRIPTION
System.String	

Property

Property to fill

Declaration

public string Property { get; set; }

Property Value

TYPE	DESCRIPTION
System.String	

TypeId

When implemented in a derived class, gets a unique identifier for this System.Attribute.

Declaration

```
public override object TypeId { get; }
```

Property Value

TYPE	DESCRIPTION
System.Object	An System.Object that is a unique identifier for the attribute.

Overrides

System.Attribute.TypeId

Extension Methods

- SerializationExtensions.ToXml<T>(T)
- CollectionsExtensions.SetValue<TKey, T>(TKey, T)
- CollectionsExtensions.AsEnum<T>(T)
- DataObjectQueryableExtensions.CompareVariable(Object, FilterOperation, String)
- Processing.Do<T>(T, Action<IProcess<T>>[])
- TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>)
- TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>, TResult)
- TypeExtensions.ChangeType<T>(Object)
- TypeExtensions.ChangeType(Object, Type)
- TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)
- TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)
- TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)
- TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)
- SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)
- QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Class Lob

Blob field wrapper

Inheritance

System.Object

Lob

Inherited Members

- System.Object.Equals(System.Object)
- System.Object.Equals(System.Object, System.Object)
- System.Object.GetHashCode()
- System.Object.GetType()
- System.Object.MemberwiseClone()
- System.Object.ReferenceEquals(System.Object, System.Object)
- System.Object.ToString()

Namespace: [XData.Mapping](#)

Assembly: XData.docfx.dll

Syntax

```
public abstract class Lob
```

Examples

```
// declaration
[Property("scan", "SC", Flags = DataPropertyFlag.OuterFlag)]
public Lob Scan { get; set; }
// using
byte[] scan = GetScanFileBody();
newInvoice.Scan += scan;
// is the same as
newInvoice.Scan.Value = scan;
```

Properties

Assigned

Blob is value is changed from null value

Declaration

```
public abstract bool Assigned { get; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

Cleared

Blob is value is cleared

Declaration

```
public abstract bool Cleared { get; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

EmptyKeys

Blob table keys are empty

Declaration

```
public abstract bool EmptyKeys { get; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

Value

Blob value

Declaration

```
public abstract byte[] Value { get; set; }
```

Property Value

TYPE	DESCRIPTION
Byte[]	

Methods

GetSize()

Returns Lob size

Declaration

```
public abstract long GetSize()
```

Returns

TYPE	DESCRIPTION
System.Int64	

Operators

Addition(Lob, Byte[])

Set Lob value

Declaration

```
public static Lob operator +(Lob lob, byte[] body)
```

Parameters

TYPE	NAME	DESCRIPTION
Lob	lob	Lob
Byte[]	body	Lob body

Returns

TYPE	DESCRIPTION
Lob	Updated lob

Implicit(Lob to Byte[])

Implicit Lob body extraction

Declaration

```
public static implicit operator byte[](Lob lob)
```

Parameters

TYPE	NAME	DESCRIPTION
Lob	lob	Lob property

Returns

TYPE	DESCRIPTION
Byte[]	Lob body

Extension Methods

[DataObjectExtensions.Modify\(Lob, Action<Byte\[\]>\)](#)

[SerializationExtensions.ToXml<T>\(T\)](#)

[CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)

[CollectionsExtensions.AsEnum<T>\(T\)](#)

[DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)

[Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)

[TypeExtensions.ChangeType<T>\(Object\)](#)

[TypeExtensions.ChangeType\(Object, Type\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)

[SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)

[QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

See Also

[Value](#)

[Addition\(Lob, Byte\[\]\)](#)

[Implicit\(Lob to Byte\[\]\)](#)

[Modify\(Lob, System.Action<Byte\[\]>\)](#)

[Assigned](#)

[Cleared](#)

[EmptyKeys](#)

Class OptionalFilterAttribute

Base optional filter attribute

Inheritance

System.Object

System.Attribute

[FilterBaseAttribute](#)

[FilterAttribute](#)

OptionalFilterAttribute

[LinkAttribute](#)

[SubqueryFilterAttribute](#)

Inherited Members

[FilterAttribute.Source](#)

[FilterAttribute.FieldName](#)

[FilterAttribute.Operation](#)

[FilterAttribute.Combination](#)

[FilterBaseAttribute.FilterType](#)

System.Attribute.Equals(System.Object)

System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Module)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.GetHashCode()

System.Attribute.IsDefaultAttribute()

System.Attribute.IsDefined(System.Reflection.Assembly, System.Type)

System.Attribute.IsDefined(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type)

System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.Module, System.Type)

System.Attribute.IsDefined(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type)

System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type, System.Boolean)
System.Attribute.Match(System.Object)
System.Attribute.TypeId
System.Object.Equals(System.Object, System.Object)
System.Object.GetType()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.ToString()

Namespace: [XData.Mapping](#)
Assembly: XData.docfx.dll

Syntax

```
[AttributeUsage(AttributeTargets.Class, AllowMultiple = true)]  
public abstract class OptionalFilterAttribute : FilterAttribute
```

Remarks

Optional filters can be used to conditionally omit parts of query

If filter marked by [PrimaryFilter](#) = true has no value assigned - it cause omit it primary source from query

If link marked by [PrimaryFilter](#) = true has one of sources omitted - it cause omit other source from query two

If base source [BaseSource](#) is omitted - all entire query is omitted

Hint: It can be used to build conditional sub queries or inner views

For example:

```
-- if filterBySomeField has PrimaryFilter flag = true, query...  
select ot.one_table_id, ot.name from one_table ot where ot.one_table_id in (select st.one_table_id from  
second_table st where st.some_field = :filterBySomeField)  
-- when filterBySomeField has no value assigned will be transformed into...  
select one_table_id, name from one_table  
-- bat have a full form when filterBySomeField value is assigned
```

Constructors

OptionalFilterAttribute(String, String)

Base optional filter attribute

Declaration

```
protected OptionalFilterAttribute(string sourceAlias, string fieldName)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	sourceAlias	Primary source alias
System.String	fieldName	Primary field name

Properties

PrimaryFilter

Primary filter flag

Declaration

```
public bool PrimaryFilter { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

See Also

- [LinkAttribute](#)
- [SubqueryLinkAttribute](#)
- [SubqueryFilterAttribute](#)
- [RuntimeFilter](#)

Class ParameterAttribute

SQL stored procedure/function parameter

Inheritance

System.Object
System.Attribute
ParameterAttribute

Inherited Members

System.Attribute.Equals(System.Object)
System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type)
System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type, System.Boolean)
System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type)
System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type, System.Boolean)
System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type)
System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type, System.Boolean)
System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type)
System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.Assembly)
System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type)
System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo)
System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type)
System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.Module)
System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type)
System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo)
System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type)
System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type, System.Boolean)
System.Attribute.GetHashCode()
System.Attribute.IsDefaultAttribute()
System.Attribute.IsDefined(System.Reflection.Assembly, System.Type)
System.Attribute.IsDefined(System.Reflection.Assembly, System.Type, System.Boolean)
System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type)
System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type, System.Boolean)
System.Attribute.IsDefined(System.Reflection.Module, System.Type)
System.Attribute.IsDefined(System.Reflection.Module, System.Type, System.Boolean)
System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type)
System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type, System.Boolean)
System.Attribute.Match(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.GetType()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.ToString()

Namespace: [XData.Mapping](#)

Assembly: XData.docfx.dll

```
[AttributeUsage(AttributeTargets.Class, AllowMultiple = true)]
public sealed class ParameterAttribute : Attribute
```

Constructors

ParameterAttribute(String, Int32, String, Type, DbType)

Parameter attribute constructor

Declaration

```
public ParameterAttribute(string alias, int order, string binding, Type type, DbType dbType)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	alias	SQL stored procedure/function alias
System.Int32	order	Parameter order
System.String	binding	Parameter to repository variable binding
System.Type	type	Parameter mapped type
System.Data.DbType	dbType	Parameter database type

Properties

DefaultType

Parameter default value type

Declaration

```
public DefaultType DefaultType { get; set; }
```

Property Value

TYPE	DESCRIPTION
DefaultType	

DefaultValue

Parameter default value

Declaration

```
public object DefaultValue { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Object	

Direction

Parameter direction

Declaration

```
public ParameterDirection Direction { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Data.ParameterDirection	

IsArray

User defined type (UDT) collection is VARARRAY

Declaration

```
public bool IsArray { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

Name

Parameter name

Declaration

```
public string Name { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

NativeSqlType

Native SQL type name

Declaration

```
public string NativeSqlType { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Remarks

Used to specify native SQL column type when conflicted with default type mapping

WARNING! May be not supported by database adapter!

Scale

Parameter data scale (if applicable)

Declaration

```
public int Scale { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Int32	

Size

Parameter data size (if applicable)

Declaration

```
public int Size { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Int32	

TypeId

When implemented in a derived class, gets a unique identifier for this System.Attribute.

Declaration

```
public override object TypeId { get; }
```

Property Value

TYPE	DESCRIPTION
System.Object	An System.Object that is a unique identifier for the attribute.

Overrides

System.Attribute.TypeId

UdtElementTypeName

User defined type (UDT) element type name

Declaration

```
public string UdtElementTypeName { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

UdtTypeName

User defined type (UDT) name

Declaration

```
public string UdtTypeName { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Extension Methods

- SerializationExtensions.ToXml<T>(T)
- CollectionsExtensions.SetValue<TKey, T>(TKey, T)
- CollectionsExtensions.AsEnum<T>(T)
- DataObjectQueryableExtensions.CompareVariable(Object, FilterOperation, String)
- Processing.Do<T>(T, Action<IProcess<T>>[])
- TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>)
- TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>, TResult)
- TypeExtensions.ChangeType<T>(Object)
- TypeExtensions.ChangeType(Object, Type)
- TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)
- TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)
- TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)
- TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)
- SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)
- QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Class ProcedureAttribute

Data object SQL procedure / function attribute

Inheritance

System.Object

System.Attribute

[DataSourceAttribute](#)

ProcedureAttribute

Inherited Members

System.Attribute.Equals(System.Object)

System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Module)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.GetHashCode()

System.Attribute.IsDefaultAttribute()

System.Attribute.IsDefined(System.Reflection.Assembly, System.Type)

System.Attribute.IsDefined(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type)

System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.Module, System.Type)

System.Attribute.IsDefined(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type)

System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.Match(System.Object)

System.Attribute.TypeId

System.Object.Equals(System.Object, System.Object)

System.Object.GetType()

System.Object.MemberwiseClone()

System.Object.ReferenceEquals(System.Object, System.Object)

System.Object.ToString()

Namespace: [XData.Mapping](#)

Assembly: XData.docfx.dll

Syntax

```
[AttributeUsage(AttributeTargets.Class)]
public sealed class ProcedureAttribute : DataSourceAttribute
```

Examples

```
Procedure("C", "dbo.GetCustomers", ProcedureType.Function)
```

Constructors

[ProcedureAttribute](#)(String, String, ProcedureType)

Data object SQL procedure / function attribute

Declaration

```
public ProcedureAttribute(string alias, string name, ProcedureType type)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	alias	Procedure / function alias
System.String	name	Procedure / function name
ProcedureType	type	Stored procedure type

Exceptions

TYPE	CONDITION
XDataRuntimeException	Procedure alias cannot be null or empty string
XDataRuntimeException	Procedure name cannot be null or empty string

Properties

Alias

Procedure / function alias

Set procedure alias as [BaseSource](#)

Hint: Alias used with table-valued functions as from-structure alias

Declaration

```
public override string Alias { get; protected set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Overrides

[DataSourceAttribute.Alias](#)

Name

SQL stored procedure / function name

WARNING! Specify full qualified name of procedure!

Declaration

```
public string Name { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

ProcedureType

Stored procedure type

Declaration

```
public ProcedureType ProcedureType { get; }
```

Property Value

TYPE	DESCRIPTION
ProcedureType	

See Also

[ProcedureType](#)

Type

Data source type

Declaration

```
public override DataSourceType Type { get; }
```

Property Value

TYPE	DESCRIPTION
DataSourceType	Procedure

Overrides

[DataSourceAttribute.Type](#)

Extension Methods

[SerializationExtensions.ToXml<T>\(T\)](#)

[CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)

[CollectionsExtensions.AsEnum<T>\(T\)](#)
[DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
[Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
[TypeExtensions.ChangeType<T>\(Object\)](#)
[TypeExtensions.ChangeType\(Object, Type\)](#)
[TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
[TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
[TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
[TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
[SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
[QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

See Also

[DataTableAttribute](#)
[InnerViewAttribute](#)
[SubqueryAttribute](#)
[ParameterAttribute](#)
[ResultSetAttribute](#)

Class PropertyAttribute

Data object property attribute

Inheritance

System.Object

System.Attribute

[PropertyBaseAttribute](#)

PropertyAttribute

Inherited Members

[PropertyBaseAttribute.Source](#)

[PropertyBaseAttribute.Flags](#)

[PropertyBaseAttribute.NativeSqlType](#)

[PropertyBaseAttribute.OrderBy](#)

[PropertyBaseAttribute.OrderByDesc](#)

System.Attribute.Equals(System.Object)

System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Module)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.GetHashCode()

System.Attribute.IsDefaultAttribute()

System.Attribute.IsDefined(System.Reflection.Assembly, System.Type)

System.Attribute.IsDefined(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type)

System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.Module, System.Type)

System.Attribute.IsDefined(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type)

System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.Match(System.Object)

System.Attribute.TypeId

System.Object.Equals(System.Object, System.Object)
System.Object.GetType()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.ToString()

Namespace: [XData.Mapping](#)
Assembly: XData.docfx.dll

Syntax

```
[AttributeUsage(AttributeTargets.Property)]  
public sealed class PropertyAttribute : PropertyBaseAttribute
```

Constructors

PropertyAttribute(String, String)

Data object property attribute

Declaration

```
public PropertyAttribute(string source = "", string fieldName = "")
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	source	Source alias
System.String	fieldName	Field name

Exceptions

TYPE	CONDITION
XDataRuntimeException	Field name cannot be null or empty string

Properties

FieldName

Field name

Declaration

```
public string FieldName { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

IsLink

Property link flag

Declaration

```
public override bool IsLink { get; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	False

Overrides

[PropertyBaseAttribute.IsLink](#)

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Class PropertyBaseAttribute

Data property base attribute

Inheritance

System.Object

System.Attribute

PropertyBaseAttribute

[PropertyAttribute](#)

[ReferenceAttribute](#)

Inherited Members

System.Attribute.Equals(System.Object)

System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Module)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.GetHashCode()

System.Attribute.IsDefaultAttribute()

System.Attribute.IsDefined(System.Reflection.Assembly, System.Type)

System.Attribute.IsDefined(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type)

System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.Module, System.Type)

System.Attribute.IsDefined(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type)

System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.Match(System.Object)

System.Attribute.TypeId

System.Object.Equals(System.Object, System.Object)

System.Object.GetType()

System.Object.MemberwiseClone()

System.Object.ReferenceEquals(System.Object, System.Object)

System.Object.ToString()

Namespace: [XData.Mapping](#)
Assembly: XData.docfx.dll

Syntax

```
public abstract class PropertyBaseAttribute : Attribute
```

Properties

Flags

Property flags

Declaration

```
public DataPropertyFlag Flags { get; set; }
```

Property Value

TYPE	DESCRIPTION
DataPropertyFlag	

See Also

[DataPropertyFlag](#)

IsLink

Property link flag

Declaration

```
public abstract bool IsLink { get; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

NativeSqlType

Native SQL type name

Declaration

```
public string NativeSqlType { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Remarks

Used to specify native SQL column type when conflicted with default type mapping

WARNING! May be not supported by database adapter!

OrderBy

Order by priority

Declaration

```
public int OrderBy { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Int32	

OrderByDesc

Order by desc flag

Declaration

```
public bool OrderByDesc { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

Source

Source alias

Declaration

```
public string Source { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

See Also

[Alias](#)

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)

[QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

See Also

[PropertyAttribute](#)

[ReferenceAttribute](#)

Class PropertyDefaultAttribute

Data object property default value attribute

Inheritance

System.Object

System.Attribute

PropertyDefaultAttribute

Inherited Members

System.Attribute.Equals(System.Object)

System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Module)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.GetHashCode()

System.Attribute.IsDefaultAttribute()

System.Attribute.IsDefined(System.Reflection.Assembly, System.Type)

System.Attribute.IsDefined(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type)

System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.Module, System.Type)

System.Attribute.IsDefined(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type)

System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.Match(System.Object)

System.Attribute.TypeId

System.Object.Equals(System.Object, System.Object)

System.Object.GetType()

System.Object.MemberwiseClone()

System.Object.ReferenceEquals(System.Object, System.Object)

System.Object.ToString()

Namespace: [XData.Mapping](#)

Syntax

```
[AttributeUsage(AttributeTargets.Property)]
public sealed class PropertyDefaultAttribute : Attribute
```

Remarks

Used to declare default value rule for property

Constructors

PropertyDefaultAttribute(DefaultType, Object)

Data object property default value attribute

Declaration

```
public PropertyDefaultAttribute(DefaultType source, object val = null)
```

Parameters

TYPE	NAME	DESCRIPTION
DefaultType	source	Default value source
System.Object	val	Default value

Properties

DefaultFeature

Default extended features

Declaration

```
public DefaultFeature DefaultFeature { get; set; }
```

Property Value

TYPE	DESCRIPTION
DefaultFeature	

DefaultSource

Default value source

Declaration

```
public DefaultType DefaultSource { get; }
```

Property Value

TYPE	DESCRIPTION
DefaultType	

See Also

[DefaultType](#)

DefaultValue

Default value

Declaration

```
public object DefaultValue { get; }
```

Property Value

TYPE	DESCRIPTION
System.Object	

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

See Also

[DefaultType](#)

Class PropertyExpressionAttribute

Data object property plain expression attribute

Inheritance

System.Object

System.Attribute

PropertyExpressionAttribute

Inherited Members

System.Attribute.Equals(System.Object)

System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Module)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.GetHashCode()

System.Attribute.IsDefaultAttribute()

System.Attribute.IsDefined(System.Reflection.Assembly, System.Type)

System.Attribute.IsDefined(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type)

System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.Module, System.Type)

System.Attribute.IsDefined(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type)

System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.Match(System.Object)

System.Attribute.TypeId

System.Object.Equals(System.Object, System.Object)

System.Object.GetType()

System.Object.MemberwiseClone()

System.Object.ReferenceEquals(System.Object, System.Object)

System.Object.ToString()

Namespace: [XData.Mapping](#)

Syntax

```
[AttributeUsage(AttributeTargets.Property)]
public sealed class PropertyExpressionAttribute : Attribute
```

Remarks

Used when [PropertyAttribute](#) value is expression ([PlainSql](#) or [SubQuery](#)) according [ExprType](#)

Constructors

[PropertyExpressionAttribute](#)(String, [DataExpressionType](#), [DbType](#))

Data object property plain expression attribute

Declaration

```
public PropertyExpressionAttribute(string exprText, DataExpressionType exprType = DataExpressionType.PlainSql,
DbType type = DbType.String)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	exprText	Expression text
DataExpressionType	exprType	Expression type
System.Data.DbType	type	Expression DbType

Exceptions

TYPE	CONDITION
XDataRuntimeException	Expression text cannot be null or empty string

Properties

[DbType](#)

Expression DbType

Declaration

```
public DbType DbType { get; }
```

Property Value

TYPE	DESCRIPTION
System.Data.DbType	

Exists

Exists subquery operation

Declaration

```
public ExistsOperation Exists { get; set; }
```

Property Value

TYPE	DESCRIPTION
ExistsOperation	

ExprScale

Expression scale

Declaration

```
public int ExprScale { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Int32	

ExprSize

Expression size

Declaration

```
public int ExprSize { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Int32	

ExprText

Expression text

Declaration

```
public string ExprText { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Remarks

[PlainSql](#) expression text is Sql expression or [SqlExpressionAttribute](#) name

[SubQuery](#) expression text is [Alias](#)

WARNING! Plain Sql expressions (see [ExprType](#)) can use specific SQL dialect features!

Data mapping may be not compatible with other database engines! Use [SqlExpressionAttribute](#) instead.

ExprType

Expression type

Declaration

```
public DataExpressionType ExprType { get; }
```

Property Value

TYPE	DESCRIPTION
DataExpressionType	

Remarks

WARNING! Plain Sql expressions can use specific Sql dialect features! Data mapping may be not compatible with other database engines! Use [SqlExpressionAttribute](#) instead.

See Also

[DataExpressionType](#)

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Class PropertyGroupingAttribute

Data object property grouping attribute

Inheritance

System.Object

System.Attribute

PropertyGroupingAttribute

Inherited Members

System.Attribute.Equals(System.Object)

System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Module)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.GetHashCode()

System.Attribute.IsDefaultAttribute()

System.Attribute.IsDefined(System.Reflection.Assembly, System.Type)

System.Attribute.IsDefined(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type)

System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.Module, System.Type)

System.Attribute.IsDefined(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type)

System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.Match(System.Object)

System.Attribute.TypeId

System.Object.Equals(System.Object, System.Object)

System.Object.GetType()

System.Object.MemberwiseClone()

System.Object.ReferenceEquals(System.Object, System.Object)

System.Object.ToString()

Namespace: [XData.Mapping](#)

Syntax

```
[AttributeUsage(AttributeTargets.Property)]
public sealed class PropertyGroupingAttribute : Attribute
```

Constructors

PropertyGroupingAttribute()

Data object property grouping attribute

Declaration

```
public PropertyGroupingAttribute()
```

Properties

Grouping

Property data aggregate function

Declaration

```
public DataGrouping Grouping { get; set; }
```

Property Value

TYPE	DESCRIPTION
DataGrouping	

See Also

[DataGrouping](#)

GroupOrder

Property group order

Declaration

```
public int GroupOrder { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Int32	

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)

[SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)

[QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

See Also

[DataGrouping](#)

Class PropertyUpdateWithAttribute

Data object property update source expression attribute

Inheritance

System.Object

System.Attribute

PropertyUpdateWithAttribute

Inherited Members

System.Attribute.Equals(System.Object)

System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Module)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.GetHashCode()

System.Attribute.IsDefaultAttribute()

System.Attribute.IsDefined(System.Reflection.Assembly, System.Type)

System.Attribute.IsDefined(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type)

System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.Module, System.Type)

System.Attribute.IsDefined(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type)

System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.Match(System.Object)

System.Attribute.TypeId

System.Object.Equals(System.Object, System.Object)

System.Object.GetType()

System.Object.MemberwiseClone()

System.Object.ReferenceEquals(System.Object, System.Object)

System.Object.ToString()

Namespace: [XData.Mapping](#)

Syntax

```
[AttributeUsage(AttributeTargets.Property)]
public sealed class PropertyUpdateWithAttribute : Attribute
```

Remarks

Used when [PropertyAttribute](#) value must to be updated with expression ([PlainSql](#) or [SubQuery](#)) result according [ExprType](#)

Constructors

[PropertyUpdateWithAttribute](#)(String, [DataExpressionType](#))

Data object property plain expression attribute

Declaration

```
public PropertyUpdateWithAttribute(string exprText, DataExpressionType exprType = DataExpressionType.PlainSql)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	exprText	Expression text
DataExpressionType	exprType	Expression type

Exceptions

TYPE	CONDITION
XDataRuntimeException	Expression text cannot be null or empty string

Properties

[ExprText](#)

Expression text

Declaration

```
public string ExprText { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Remarks

[PlainSql](#) expression text is Sql expression or [SqlExpressionAttribute](#) name

[SubQuery](#) expression text is [Alias](#)

WARNING! Plain Sql expressions (see [ExprType](#)) can use specific SQL dialect features!

Data mapping may be not compatible with other database engines! Use [SqlExpressionAttribute](#) instead.

ExprType

Expression type

Declaration

```
public DataExpressionType ExprType { get; }
```

Property Value

TYPE	DESCRIPTION
DataExpressionType	

Remarks

WARNING! Plain Sql expressions can use specific Sql dialect features! Data mapping may be not compatible with other database engines! Use [SqlExpressionAttribute](#) instead.

See Also

[DataExpressionType](#)

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Class RangeFilterAttribute

Data object range filter attribute

Inheritance

System.Object

System.Attribute

[FilterBaseAttribute](#)

[FilterAttribute](#)

RangeFilterAttribute

Inherited Members

[FilterAttribute.Source](#)

[FilterAttribute.FieldName](#)

[FilterAttribute.Operation](#)

[FilterAttribute.Combination](#)

System.Attribute.Equals(System.Object)

System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Module)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.GetHashCode()

System.Attribute.IsDefaultAttribute()

System.Attribute.IsDefined(System.Reflection.Assembly, System.Type)

System.Attribute.IsDefined(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type)

System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.Module, System.Type)

System.Attribute.IsDefined(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type)

System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.Match(System.Object)

System.Object.Equals(System.Object, System.Object)

System.Object.GetType()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.ToString()

Namespace: [XData.Mapping](#)

Assembly: XData.docfx.dll

Syntax

```
[AttributeUsage(AttributeTargets.Class, AllowMultiple = true)]
public sealed class RangeFilterAttribute : FilterAttribute
```

Remarks

Used to filter data by range of values (for example: `some_field in (1, 2, 3)` or `some_field not in (1, 2, 3)`)

Hint: Using dynamic LINQ range filters will increase code flexibility (see example).

Examples

```
var states = new List<int>();
// fill states from somewhere
dataScope.GetRepository<Invoice>().Where(x => states.Contains(x.State));
```

Constructors

RangeFilterAttribute(String, String, Object[])

Range filter attribute

Declaration

```
public RangeFilterAttribute(string source, string fieldName, params object[] range)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	source	Primary source alias
System.String	fieldName	Primary field name
System.Object[]	range	Range values

Exceptions

TYPE	CONDITION
XDataRuntimeException	Range values cannot be null or empty array

Properties

FilterType

Filter type

Declaration

```
public override FilterType FilterType { get; }
```

Property Value

TYPE	DESCRIPTION
FilterType	Range

Overrides

[FilterBaseAttribute.FilterType](#)

Range

Range values

Declaration

```
public object[] Range { get; }
```

Property Value

TYPE	DESCRIPTION
System.Object[]	

TypeId

When implemented in a derived class, gets a unique identifier for this System.Attribute.

Declaration

```
public override object TypeId { get; }
```

Property Value

TYPE	DESCRIPTION
System.Object	An System.Object that is a unique identifier for the attribute.

Overrides

[System.Attribute.TypeId](#)

Extension Methods

[SerializationExtensions.ToXml<T>\(T\)](#)

[CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)

[CollectionsExtensions.AsEnum<T>\(T\)](#)

[DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)

[Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)

[TypeExtensions.ChangeType<T>\(Object\)](#)

[TypeExtensions.ChangeType\(Object, Type\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)

[SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)

```
QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)
```

Class ReferenceAttribute

Data object referenced inner view property attribute

Inheritance

System.Object

System.Attribute

[PropertyBaseAttribute](#)

ReferenceAttribute

Inherited Members

[PropertyBaseAttribute.Source](#)

[PropertyBaseAttribute.Flags](#)

[PropertyBaseAttribute.NativeSqlType](#)

[PropertyBaseAttribute.OrderBy](#)

[PropertyBaseAttribute.OrderByDesc](#)

System.Attribute.Equals(System.Object)

System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Module)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.GetHashCode()

System.Attribute.IsDefaultAttribute()

System.Attribute.IsDefined(System.Reflection.Assembly, System.Type)

System.Attribute.IsDefined(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type)

System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.Module, System.Type)

System.Attribute.IsDefined(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type)

System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.Match(System.Object)

System.Attribute.TypeId

System.Object.Equals(System.Object, System.Object)
System.Object.GetType()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.ToString()

Namespace: [XData.Mapping](#)
Assembly: XData.docfx.dll

Syntax

```
[AttributeUsage(AttributeTargets.Property)]  
public sealed class ReferenceAttribute : PropertyBaseAttribute
```

Remarks

Used to map property to data of inner view declared [PropertyAttribute](#) or [ColumnAttribute](#)

Constructors

[ReferenceAttribute](#)(String, String)

Data object referenced inner view property attribute

Declaration

```
public ReferenceAttribute(string source, string propertyName = "")
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	source	Source alias
System.String	propertyName	Linked property name

Exceptions

TYPE	CONDITION
XDataRuntimeException	Linked property name cannot be null or empty string

Properties

[IsLink](#)

Property reference flag

Declaration

```
public override bool IsLink { get; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	True

Overrides

[PropertyBaseAttribute.IsLink](#)

PropertyName

Referenced property (or column) name

Declaration

```
public string PropertyName { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Class ResultSetAttribute

SQL stored procedure/function result set

Inheritance

System.Object
System.Attribute
ResultSetAttribute

Inherited Members

System.Attribute.Equals(System.Object)
System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type)
System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type, System.Boolean)
System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type)
System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type, System.Boolean)
System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type)
System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type, System.Boolean)
System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type)
System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.Assembly)
System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type)
System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo)
System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type)
System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.Module)
System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type)
System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo)
System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Boolean)
System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type)
System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type, System.Boolean)
System.Attribute.GetHashCode()
System.Attribute.IsDefaultAttribute()
System.Attribute.IsDefined(System.Reflection.Assembly, System.Type)
System.Attribute.IsDefined(System.Reflection.Assembly, System.Type, System.Boolean)
System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type)
System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type, System.Boolean)
System.Attribute.IsDefined(System.Reflection.Module, System.Type)
System.Attribute.IsDefined(System.Reflection.Module, System.Type, System.Boolean)
System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type)
System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type, System.Boolean)
System.Attribute.Match(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.GetType()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.ToString()

Namespace: [XData.Mapping](#)

Assembly: XData.docfx.dll

Syntax

[AttributeUsage(AttributeTargets.Class, AllowMultiple = true)]
public sealed class ResultSetAttribute : Attribute

Remarks

Result sets can be accessed by [GetResultSet<T, TRes>\(IRepository<T>, Expression<Func<T, IEnumerable<TRes>>>\)](#) repository method

Constructors

ResultSetAttribute(String)

Result set attribute constructor

Declaration

public ResultSetAttribute(string alias)

Parameters

TYPE	NAME	DESCRIPTION
System.String	alias	SQL procedure/function alias

ResultSetAttribute(String, String)

Result set attribute constructor

Declaration

public ResultSetAttribute(string alias, string name)

Parameters

TYPE	NAME	DESCRIPTION
System.String	alias	SQL procedure/function alias
System.String	name	Result set name

Properties

IsDefault

Result set is used to fill procedure output data object

Declaration

public bool IsDefault { get; set; }

Property Value

TYPE	DESCRIPTION
System.Boolean	

Order

Result set order

Declaration

```
public int Order { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Int32	

ResultType

Result set mapped type

Declaration

```
public Type ResultType { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Type	

TypeId

When implemented in a derived class, gets a unique identifier for this System.Attribute.

Declaration

```
public override object TypeId { get; }
```

Property Value

TYPE	DESCRIPTION
System.Object	An System.Object that is a unique identifier for the attribute.

Overrides

System.Attribute.TypeId

Extension Methods

[SerializationExtensions.ToXml<T>\(T\)](#)

[CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)

[CollectionsExtensions.AsEnum<T>\(T\)](#)

[DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)

[Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)

[TypeExtensions.ChangeType<T>\(Object\)](#)

[TypeExtensions.ChangeType\(Object, Type\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)

TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)

SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)

QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Class SqlExpressionAttribute

Sql expression name

Inheritance

System.Object

System.Attribute

SqlExpressionAttribute

Inherited Members

System.Attribute.Equals(System.Object)

System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Module)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.GetHashCode()

System.Attribute.IsDefaultAttribute()

System.Attribute.IsDefined(System.Reflection.Assembly, System.Type)

System.Attribute.IsDefined(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type)

System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.Module, System.Type)

System.Attribute.IsDefined(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type)

System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.Match(System.Object)

System.Attribute.TypeId

System.Object.Equals(System.Object, System.Object)

System.Object.GetType()

System.Object.MemberwiseClone()

System.Object.ReferenceEquals(System.Object, System.Object)

System.Object.ToString()

Namespace: [XData.Mapping](#)

Syntax

```
[AttributeUsage(AttributeTargets.Field)]  
public sealed class SqlExpressionAttribute : Attribute
```

Remarks

Mark static field of type [Calculate<T>](#) to use in

[ExpressionFilterAttribute](#) or [PropertyExpressionAttribute](#) as expression text.

Hint: This is native way to use expressions

Extension Methods

[SerializationExtensions.ToXml<T>\(T\)](#)

[CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)

[CollectionsExtensions.AsEnum<T>\(T\)](#)

[DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)

[Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)

[TypeExtensions.ChangeType<T>\(Object\)](#)

[TypeExtensions.ChangeType\(Object, Type\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)

[SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)

[QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

See Also

[ExpressionFilterAttribute](#)

[PropertyExpressionAttribute](#)

Class SubqueryAttribute

Data object subquery attribute

Inheritance

System.Object

System.Attribute

[DataSourceAttribute](#)

SubqueryAttribute

Inherited Members

System.Attribute.Equals(System.Object)

System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Module)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.GetHashCode()

System.Attribute.IsDefaultAttribute()

System.Attribute.IsDefined(System.Reflection.Assembly, System.Type)

System.Attribute.IsDefined(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type)

System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.Module, System.Type)

System.Attribute.IsDefined(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type)

System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.Match(System.Object)

System.Object.Equals(System.Object, System.Object)

System.Object.GetType()

System.Object.MemberwiseClone()

System.Object.ReferenceEquals(System.Object, System.Object)

System.Object.ToString()

Namespace: [XData.Mapping](#)

Syntax

```
[AttributeUsage(AttributeTargets.Class, AllowMultiple = true)]
public sealed class SubqueryAttribute : DataSourceAttribute
```

Examples

```
Subquery("A", typeof(DocSpecAmounts), "Amount", Grouping = DataGrouping.Sum)
```

Constructors

SubqueryAttribute(String, Type, String)

Data object subquery attribute

Declaration

```
public SubqueryAttribute(string alias, Type subqueryType, string propertyName)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	alias	Subquery alias
System.Type	subqueryType	Object type represents subquery structure
System.String	propertyName	Subquery selected property

Exceptions

TYPE	CONDITION
XDataRuntimeException	Subquery alias cannot be null or empty string
XDataRuntimeException	Object type represents subquery structure cannot be null
XDataRuntimeException	Subquery selected property cannot be null or empty string

Properties

Alias

Subquery alias

Declaration

```
public override string Alias { get; protected set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Overrides

[DataSourceAttribute.Alias](#)

Grouping

Subquery grouping mode

Declaration

```
public DataGrouping Grouping { get; set; }
```

Property Value

TYPE	DESCRIPTION
DataGrouping	

Remarks

Subquery selected property (see [PropertyName](#)) aggregate function

Grouping in subquery mast to be defined by [GroupOrder](#)

WARNING! Do not use properties inside grouped subqueries! Use [ColumnAttribute](#) instead!

PropertyName

Subquery selected property

Declaration

```
public string PropertyName { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Remarks

Only one property (or column) can be selected from subquery

SubqueryType

Object type represents subquery structure

Declaration

```
public Type SubqueryType { get; }
```

Property Value

TYPE	DESCRIPTION
System.Type	

Type

Data source type

Declaration

```
public override DataSourceType Type { get; }
```

Property Value

TYPE	DESCRIPTION
DataSourceType	Subquery

Overrides

[DataSourceAttribute.Type](#)

TypeId

When implemented in a derived class, gets a unique identifier for this System.Attribute.

Declaration

```
public override object TypeId { get; }
```

Property Value

TYPE	DESCRIPTION
System.Object	An System.Object that is a unique identifier for the attribute.

Overrides

[System.Attribute.TypeId](#)

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

See Also

- [DataTableAttribute](#)
- [InnerViewAttribute](#)
- [SubqueryLinkAttribute](#)
- [SubqueryFilterAttribute](#)

SubQuery
ColumnExpressionAttribute
PropertyExpressionAttribute
ExpressionFilterAttribute

Class SubqueryFilterAttribute

Data object subquery filter attribute

Inheritance

System.Object

System.Attribute

[FilterBaseAttribute](#)

[FilterAttribute](#)

[OptionalFilterAttribute](#)

SubqueryFilterAttribute

Inherited Members

[OptionalFilterAttribute.PrimaryFilter](#)

[FilterAttribute.Source](#)

[FilterAttribute.FieldName](#)

[FilterAttribute.Operation](#)

[FilterAttribute.Combination](#)

System.Attribute.Equals(System.Object)

System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Module)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.GetHashCode()

System.Attribute.IsDefaultAttribute()

System.Attribute.IsDefined(System.Reflection.Assembly, System.Type)

System.Attribute.IsDefined(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type)

System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.Module, System.Type)

System.Attribute.IsDefined(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type)

System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.Match(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.GetType()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.ToString()

Namespace: [XData.Mapping](#)
Assembly: XData.docfx.dll

Syntax

```
[AttributeUsage(AttributeTargets.Class, AllowMultiple = true)]  
public sealed class SubqueryFilterAttribute : OptionalFilterAttribute
```

Remarks

Filter data by subquery:

```
some_field = (select ...)
```

```
some_field != (select ...)
```

```
some_field in (select ...)
```

```
some_field not in (select ...)
```

```
exists (select ...)
```

```
not exists (select ...)
```

Constructors

SubqueryFilterAttribute(String, String, String)

Subquery filter attribute

Declaration

```
public SubqueryFilterAttribute(string subquery, string source, string fieldName)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	subquery	Subquery alias
System.String	source	Primary source alias
System.String	fieldName	Primary field name

Exceptions

TYPE	CONDITION
XDataRuntimeException	Subquery alias cannot be null or empty string

Properties

FilterType

Filter type

Declaration

```
public override FilterType FilterType { get; }
```

Property Value

TYPE	DESCRIPTION
FilterType	Subquery

Overrides

[FilterBaseAttribute.FilterType](#)

Subquery

Subquery alias

Declaration

```
public string Subquery { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

See Also

[SubqueryAttribute](#)

TypeId

When implemented in a derived class, gets a unique identifier for this System.Attribute.

Declaration

```
public override object TypeId { get; }
```

Property Value

TYPE	DESCRIPTION
System.Object	An System.Object that is a unique identifier for the attribute.

Overrides

System.Attribute.TypeId

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)

[TypeExtensions.ChangeType<T>\(Object\)](#)

[TypeExtensions.ChangeType\(Object, Type\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)

[SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)

[QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

See Also

[SubqueryAttribute](#)

Class SubqueryLinkAttribute

Data object link to subquery attribute

Inheritance

System.Object

System.Attribute

[FilterBaseAttribute](#)

SubqueryLinkAttribute

Inherited Members

System.Attribute.Equals(System.Object)

System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Module)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.GetHashCode()

System.Attribute.IsDefaultAttribute()

System.Attribute.IsDefined(System.Reflection.Assembly, System.Type)

System.Attribute.IsDefined(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type)

System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.Module, System.Type)

System.Attribute.IsDefined(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type)

System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.Match(System.Object)

System.Object.Equals(System.Object, System.Object)

System.Object.GetType()

System.Object.MemberwiseClone()

System.Object.ReferenceEquals(System.Object, System.Object)

System.Object.ToString()

Namespace: [XData.Mapping](#)

Syntax

```
[AttributeUsage(AttributeTargets.Class, AllowMultiple = true)]
public sealed class SubqueryLinkAttribute : FilterBaseAttribute
```

Constructors

SubqueryLinkAttribute(String, String, String)

Subquery link attribute

Declaration

```
public SubqueryLinkAttribute(string subquery, string subqueryProperty, string property = "")
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	subquery	Child source alias
System.String	subqueryProperty	Child property to link
System.String	property	Parent property contains value to link

Exceptions

TYPE	CONDITION
XDataRuntimeExpection	Child source alias cannot be null or empty string
XDataRuntimeExpection	Child property to link cannot be null or empty string

Properties

Combination

Filter combination (filter group) name

Declaration

```
public string Combination { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

See Also

[FilterCombinationAttribute](#)

FilterType

Filter type

Declaration

```
public override FilterType FilterType { get; }
```

Property Value

TYPE	DESCRIPTION
FilterType	

Overrides

[FilterBaseAttribute.FilterType](#)

See Also

[FilterType](#)

Operation

Filter operation

Declaration

```
public FilterOperation Operation { get; set; }
```

Property Value

TYPE	DESCRIPTION
FilterOperation	

See Also

[FilterOperation](#)

PrimaryFilter

Primary filter flag

Declaration

```
public bool PrimaryFilter { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

See Also

[OptionalFilterAttribute](#)

PropertyName

Parent property

Declaration

```
public string PropertyName { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Remarks

If not assigned used [SubqueryProperty](#) name

Subquery

Child source alias

Declaration

```
public string Subquery { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

See Also

[Alias](#)

SubqueryProperty

Child property to link

Declaration

```
public string SubqueryProperty { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

TypeId

When implemented in a derived class, gets a unique identifier for this System.Attribute.

Declaration

```
public override object TypeId { get; }
```

Property Value

TYPE	DESCRIPTION
System.Object	An System.Object that is a unique identifier for the attribute.

Overrides

System.Attribute.TypeId

Extension Methods

[SerializationExtensions.ToXml<T>\(T\)](#)

[CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)

[CollectionsExtensions.AsEnum<T>\(T\)](#)

[DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)

[Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)

[TypeExtensions.ChangeType<T>\(Object\)](#)

[TypeExtensions.ChangeType\(Object, Type\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)

[SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)

[QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

See Also

[SubqueryAttribute](#)

[InnerViewAttribute](#)

Class SubqueryVariableAttribute

Data object subquery variable attribute

Inheritance

System.Object

System.Attribute

SubqueryVariableAttribute

Inherited Members

System.Attribute.Equals(System.Object)

System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Module)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.GetHashCode()

System.Attribute.IsDefaultAttribute()

System.Attribute.IsDefined(System.Reflection.Assembly, System.Type)

System.Attribute.IsDefined(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type)

System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.Module, System.Type)

System.Attribute.IsDefined(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type)

System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.Match(System.Object)

System.Object.Equals(System.Object, System.Object)

System.Object.GetType()

System.Object.MemberwiseClone()

System.Object.ReferenceEquals(System.Object, System.Object)

System.Object.ToString()

Namespace: [XData.Mapping](#)

Assembly: XData.docfx.dll

Syntax

```
[AttributeUsage(AttributeTargets.Class, AllowMultiple = true)]
public class SubqueryVariableAttribute : Attribute
```

Examples

```
SubqueryVariable("A", "TestVar", 1)
```

Constructors

SubqueryVariableAttribute(String, String, Object)

Declaration

```
public SubqueryVariableAttribute(string alias, string name, object value)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	alias	
System.String	name	
System.Object	value	

Properties

Alias

Subquery alias

Declaration

```
public string Alias { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Name

Variable name

Declaration

```
public string Name { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

TypeId

When implemented in a derived class, gets a unique identifier for this System.Attribute.

Declaration

```
public override object TypeId { get; }
```

Property Value

TYPE	DESCRIPTION
System.Object	An System.Object that is a unique identifier for the attribute.

Overrides

System.Attribute.TypeId

Value

Variable value

Declaration

```
public object Value { get; }
```

Property Value

TYPE	DESCRIPTION
System.Object	

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

See Also

[SubqueryAttribute](#)

Class WithAttribute

Common-Table-Expression (WITH) Subquery data source

Inheritance

System.Object

System.Attribute

[DataSourceAttribute](#)

WithAttribute

Inherited Members

System.Attribute.Equals(System.Object)

System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Module)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.GetHashCode()

System.Attribute.IsDefaultAttribute()

System.Attribute.IsDefined(System.Reflection.Assembly, System.Type)

System.Attribute.IsDefined(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type)

System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.Module, System.Type)

System.Attribute.IsDefined(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type)

System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.Match(System.Object)

System.Object.Equals(System.Object, System.Object)

System.Object.GetType()

System.Object.MemberwiseClone()

System.Object.ReferenceEquals(System.Object, System.Object)

System.Object.ToString()

Namespace: [XData.Mapping](#)

Syntax

```
public sealed class WithAttribute : DataSourceAttribute
```

Constructors

WithAttribute(String, Type, WithRecursiveType, String[])

Common-Table-Expression (WITH) Subquery data source constructor

Declaration

```
public WithAttribute(string alias, Type subqueryType, WithRecursiveType withType = WithRecursiveType.RecursiveUnion, params string[] properties)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	alias	Common-Table-Expression alias
System.Type	subqueryType	Subquery type
WithRecursiveType	withType	Recursive connection type (if applicable)
System.String[]	properties	Common-Table-Expression (WITH) Subquery property order

Properties

Alias

Data source alias

Declaration

```
public override string Alias { get; protected set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Overrides

[DataSourceAttribute.Alias](#)

Properties

Common-Table-Expression (WITH) Subquery property order

Declaration

```
public string[] Properties { get; }
```

Property Value

TYPE	DESCRIPTION
System.String[]	

SubqueryType

Subquery type

Declaration

```
public Type SubqueryType { get; }
```

Property Value

TYPE	DESCRIPTION
System.Type	

Type

Data source type

Declaration

```
public override DataSourceType Type { get; }
```

Property Value

TYPE	DESCRIPTION
DataSourceType	

Overrides

[DataSourceAttribute.Type](#)

TypeId

When implemented in a derived class, gets a unique identifier for this System.Attribute.

Declaration

```
public override object TypeId { get; }
```

Property Value

TYPE	DESCRIPTION
System.Object	An System.Object that is a unique identifier for the attribute.

Overrides

System.Attribute.TypeId

WithType

Common-Table-Expression (WITH) Subquery type

Declaration

```
public WithRecursiveType WithType { get; }
```

Property Value

TYPE	DESCRIPTION
WithRecursiveType	

Extension Methods

- SerializationExtensions.ToXml<T>(T)
- CollectionsExtensions.SetValue<TKey, T>(TKey, T)
- CollectionsExtensions.AsEnum<T>(T)
- DataObjectQueryableExtensions.CompareVariable(Object, FilterOperation, String)
- Processing.Do<T>(T, Action<IProcess<T>>[])
- TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>)
- TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>, TResult)
- TypeExtensions.ChangeType<T>(Object)
- TypeExtensions.ChangeType(Object, Type)
- TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)
- TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)
- TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)
- TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)
- SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)
- QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Class WithRecursiveAttribute

Common-Table-Expression (WITH) recursive subquery data source

Inheritance

System.Object

System.Attribute

WithRecursiveAttribute

Inherited Members

System.Attribute.Equals(System.Object)

System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type)

System.Attribute.GetCustomAttribute(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Module)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Boolean)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type)

System.Attribute.GetCustomAttributes(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.GetHashCode()

System.Attribute.IsDefaultAttribute()

System.Attribute.IsDefined(System.Reflection.Assembly, System.Type)

System.Attribute.IsDefined(System.Reflection.Assembly, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type)

System.Attribute.IsDefined(System.Reflection.MemberInfo, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.Module, System.Type)

System.Attribute.IsDefined(System.Reflection.Module, System.Type, System.Boolean)

System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type)

System.Attribute.IsDefined(System.Reflection.ParameterInfo, System.Type, System.Boolean)

System.Attribute.Match(System.Object)

System.Object.Equals(System.Object, System.Object)

System.Object.GetType()

System.Object.MemberwiseClone()

System.Object.ReferenceEquals(System.Object, System.Object)

System.Object.ToString()

Namespace: [XData.Mapping](#)

Assembly: XData.docfx.dll

Syntax

```
[AttributeUsage(AttributeTargets.Class, AllowMultiple = true)]  
public sealed class WithRecursiveAttribute : Attribute
```

Constructors

WithRecursiveAttribute(String, Type, String)

Common-Table-Expression (WITH) Subquery recursive data source constructor

Declaration

```
public WithRecursiveAttribute(string alias, Type subqueryType, string initialAlias)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	alias	Subquery alias
System.Type	subqueryType	Subquery type
System.String	initialAlias	Common-Table-Expression alias

Properties

Alias

Data source alias

Declaration

```
public string Alias { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

InitialAlias

Initial Common-Table-Expression (WITH) Subquery alias

Declaration

```
public string InitialAlias { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

SubqueryType

Subquery type

Declaration

```
public Type SubqueryType { get; }
```

Property Value

TYPE	DESCRIPTION
System.Type	

TypeId

When implemented in a derived class, gets a unique identifier for this System.Attribute.

Declaration

```
public override object TypeId { get; }
```

Property Value

TYPE	DESCRIPTION
System.Object	An System.Object that is a unique identifier for the attribute.

Overrides

System.Attribute.TypeId

Extension Methods

[SerializationExtensions.ToXml<T>\(T\)](#)

[CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)

[CollectionsExtensions.AsEnum<T>\(T\)](#)

[DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)

[Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)

[TypeExtensions.ChangeType<T>\(Object\)](#)

[TypeExtensions.ChangeType\(Object, Type\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)

[SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)

[QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Class Xml

Xml field wrapper

Inheritance

System.Object
Xml

Inherited Members

System.Object.Equals(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.GetHashCode()
System.Object.GetType()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.ToString()

Namespace: [XData.Mapping](#)
Assembly: XData.docfx.dll

Syntax

```
public abstract class Xml
```

Examples

```
// declaration
[Property("source", "SR", Flags = DataPropertyFlag.OuterFlag)]
public Xml Source { get; set; }
// using
newInvoice.Source += new XDocument(new XElement("invoice", new XAttribute("number", number), new
XAttribute("state", "CREATED")));
// is the same as
newInvoice.Source.Document = new XDocument(new XElement("invoice", new XAttribute("number", number),
new XAttribute("state", "CREATED")));
```

Properties

Assigned

Xml is value is changed from null value

Declaration

```
public abstract bool Assigned { get; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

Cleared

Xml is value is cleared

Declaration

```
public abstract bool Cleared { get; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

Document

Xml document

Declaration

```
public abstract XDocument Document { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Xml.Linq.XDocument	

EmptyKeys

Xml table keys are empty

Declaration

```
public abstract bool EmptyKeys { get; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

Methods

Extract(String, KeyValuePair<String, String>[])

Extract value by xpath expression

Declaration

```
public abstract string Extract(string path, params KeyValuePair<string, string>[] namespaces)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	path	XPath expression
System.Collections.Generic.KeyValuePair<System.String, System.String> []	namespaces	Xml namespaces

Returns

TYPE	DESCRIPTION
System.String	Value

Operators

Addition(Xml, XDocument)

Set Xml document

Declaration

```
public static Xml operator +(Xml xml, XDocument document)
```

Parameters

TYPE	NAME	DESCRIPTION
Xml	xml	Xml property
System.Xml.Linq.XDocument	document	Xml document

Returns

TYPE	DESCRIPTION
Xml	Updated xml property

Implicit(Xml to XDocument)

Implicit Xml document conversion

Declaration

```
public static implicit operator XDocument(Xml xml)
```

Parameters

TYPE	NAME	DESCRIPTION
Xml	xml	Xml property

Returns

TYPE	DESCRIPTION
System.Xml.Linq.XDocument	Xml document

Extension Methods

[DataObjectExtensions.Modify\(Xml, Action<XDocument>\)](#)

[SerializationExtensions.ToXml<T>\(T\)](#)

[CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)

[CollectionsExtensions.AsEnum<T>\(T\)](#)

[DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)

[Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)

[TypeExtensions.ChangeType<T>\(Object\)](#)

[TypeExtensions.ChangeType\(Object, Type\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)

[SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)

[QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

See Also

[XData.Mapping.Xml.Value](#)

[Addition\(Xml, XDocument\)](#)

[Implicit\(Xml to XDocument\)](#)

[Modify\(Xml, System.Action<System.Xml.Linq.XDocument>\)](#)

[Assigned](#)

[Cleared](#)

[EmptyKeys](#)

[Extract\(String, KeyValuePair<String, String>\[\]\)](#)

Namespace XData.Mapping.Dynamic

Classes

[ParameterDescription](#)

SQL procedure / function parameter description

[QueryDescriptionExtensions](#)

Query description extensions

[TaggedExpression<TTag, TResult>](#)

Tagged expression

[XDataMapping](#)

Interfaces

[IExpressionPropertyAdapter](#)

SQL expression property mapping adapter

[IExternalLinkAdapter<T, TChild>](#)

External link mapping adapter

[IExternalLinkDefinition](#)

Dynamic structure external link definition interface

[IExternalLinkDefinitionAdapter<T>](#)

External link definition adapter

[IExternalLinkFlag](#)

Dynamic structure external link flag interface

[IFilterAdapter](#)

Filter description adapter

[IFilterDescription](#)

Dynamic query filter description

[IInheritedStructure<T>](#)

Inherited repository structure descriptor

[IInnerFilterAdapter](#)

Inner filter description adapter

[ILinkPropertyMappingAdapter<T>](#)

Link property mapping adapter

[IParameterDescription](#)

Dynamic stored procedure parameter description

[IProcedureParameterAdapter](#)

Procedure parameter mapping adapter

[IProcedureParameterFlag](#)

Procedure parameter mapping flag

[IPropertyMappingAdapter](#)

Basic dynamic structure adapter

[IQueryDescription](#)

Dynamic query description basic interface

[IQueryDescription<T>](#)

Dynamic query description with result type specified interface

[IQueryStructure](#)

Dynamic query structure interface

[IQueryStructureAdapter](#)

Query structure mapping adapter interface

[IQueryWithAdapter<TRoot>](#)

Common-Table-Expression (WITH) mapping attributes

[IQueryWithFlag](#)

Common-Table-Expression (WITH) flag

[IRepositoryDescription<T>](#)

Dynamic repository structure description interface

[IRepositoryExpressionMappingAdapter](#)

Repository SQL expression property mapping adapter

[IRepositoryPropertyMappingAdapter<T, TValue>](#)

Repository property mapping adapter

[IRepositoryReadOnlyPropertyMappingAdapter](#)

Repository read only property mapping adapter

[IRepositoryStructure<T>](#)

Repository structure descriptor

[IRepositoryStructureAdapter](#)

Repository structure adapter

[IRepositoryStructureAdapter<T>](#)

Repository structure mapping adapter interface

[IResultSetDescription](#)

Dynamic stored procedure result set description

[IStructureFlag](#)

Dynamic structure flag interface

[ISubqueryLinkAdapter](#)

Subquery link filter description adapter

[ISubqueryLinkAdapter<TDObj>](#)

Subquery link filter description adapter

[ISubqueryLinkAdapter<T, TDObj>](#)

Subquery link filter description adapter

[ITaggedExpression](#)

Object tagged expression interface

Interface IExpressionPropertyAdapter

SQL expression property mapping adapter

Inherited Members

- [IPropertyMappingAdapter.Key\(\)](#)
- [IPropertyMappingAdapter.Group\(DataGrouping\)](#)
- [IPropertyMappingAdapter.Group\(Int32\)](#)
- [IPropertyMappingAdapter.Hidden\(\)](#)
- [IPropertyMappingAdapter.OrderBy\(Int32, Boolean\)](#)
- [IPropertyMappingAdapter.OrderBy\(Int32\)](#)

Namespace: `XData.Mapping.Dynamic`

Assembly: `XData.docfx.dll`

Syntax

```
public interface IExpressionPropertyAdapter : IPropertyMappingAdapter
```

Methods

Exists()

SQL subquery expression exists operation

Declaration

```
IStructureFlag Exists()
```

Returns

TYPE	DESCRIPTION
IStructureFlag	SQL expression mapping flag

Exists(ExistsOperation)

SQL subquery expression exists operation

Declaration

```
IStructureFlag Exists(ExistsOperation exists)
```

Parameters

TYPE	NAME	DESCRIPTION
ExistsOperation	exists	Exists operation

Returns

TYPE	DESCRIPTION
IStructureFlag	SQL expression mapping flag

NativeSqlType(String)

SQL expression result native SQL type name

Declaration

```
IStructureFlag NativeSqlType(string nativeSqlType)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	nativeSqlType	Native SQL type name

Returns

TYPE	DESCRIPTION
IStructureFlag	SQL expression mapping flag

Size(Int32)

Size of SQL expression result value

Declaration

```
IStructureFlag Size(int size)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Int32	size	Size

Returns

TYPE	DESCRIPTION
IStructureFlag	SQL expression mapping flag

Size(Int32, Int32)

Size of SQL expression result value

Declaration

```
IStructureFlag Size(int size, int scale)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Int32	size	Size

TYPE	NAME	DESCRIPTION
System.Int32	scale	Scale

Returns

TYPE	DESCRIPTION
IStructureFlag	SQL expression mapping flag

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Interface IExternalLinkAdapter<T, TChild>

External link mapping adapter

Namespace: [XData.Mapping.Dynamic](#)

Assembly: XData.docfx.dll

Syntax

```
public interface IExternalLinkAdapter<T, TChild>
    where T : class, ISqlObject, new()
    where TChild : class, ISqlObject, new()
```

Type Parameters

NAME	DESCRIPTION
T	Parent date object type
TChild	Child date object type

Methods

DirectLink()

External link direct link flag

Declaration

```
IExternalLinkFlag DirectLink()
```

Returns

TYPE	DESCRIPTION
IExternalLinkFlag	External link flag

See Also

[ShowSubtree](#)

Filter(Expression<Func<TChild, RuntimeFilter>>)

External link runtime filter

Declaration

```
IExternalLinkFlag Filter(Expression<Func<TChild, RuntimeFilter>> filter)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func<TChild, RuntimeFilter >>	filter	Runtime filter

Returns

TYPE	DESCRIPTION
IExternalLinkFlag	External link flag

FilterCombination(String)

External link filter combination flag

Declaration

IExternalLinkFlag FilterCombination(string combinationName)

Parameters

TYPE	NAME	DESCRIPTION
System.String	combinationName	Filter combination name

Returns

TYPE	DESCRIPTION
IExternalLinkFlag	

MasterRefresh()

External link master refresh flag

Declaration

IExternalLinkFlag MasterRefresh()

Returns

TYPE	DESCRIPTION
IExternalLinkFlag	External link flag

Nullable(FilterNullable)

External link filter null value assignment mode flag

Declaration

IExternalLinkFlag Nullable(FilterNullable nullable)

Parameters

TYPE	NAME	DESCRIPTION
FilterNullable	nullable	Filter null value assignment mode

Returns

TYPE	DESCRIPTION
IExternalLinkFlag	External link flag

See Also

[FilterNullable](#)

Operation(FilterOperation)

External link filter operation flag

Declaration

IExternalLinkFlag Operation(FilterOperation operation)

Parameters

TYPE	NAME	DESCRIPTION
FilterOperation	operation	Filter operation

Returns

TYPE	DESCRIPTION
IExternalLinkFlag	External link flag

See Also

[FilterOperation](#)

PrimaryFilter()

External link optional filter flag

Declaration

IExternalLinkFlag PrimaryFilter()

Returns

TYPE	DESCRIPTION
IExternalLinkFlag	External link flag

See Also

[IOptionalFilter](#)

Property(Expression<Func<TChild, Object>>)

External link child property

Declaration

IExternalLinkFlag Property(Expression<Func<TChild, object>> property)

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func<TChild, System.Object>>	property	Child type property

Returns

TYPE	DESCRIPTION
IExternalLinkFlag	External link flag

Property(String)

External link child property

Declaration

IExternalLinkFlag Property(string property)

Parameters

TYPE	NAME	DESCRIPTION
System.String	property	Child type property

Returns

TYPE	DESCRIPTION
IExternalLinkFlag	External link flag

Property<TChildInner>(Expression<Func<TChildInner, Object>>)

External link child property

Declaration

IExternalLinkFlag Property<TChildInner>(Expression<Func<TChildInner, object>> property)

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func<TChildInner, System.Object>>	property	Child inner type property

Returns

TYPE	DESCRIPTION
IExternalLinkFlag	External link flag

Type Parameters

NAME	DESCRIPTION
TChildInner	Child inner type

Property<TChildInner>(String)

External link child property

Declaration

IExternalLinkFlag Property<TChildInner>(string property)

Parameters

TYPE	NAME	DESCRIPTION
System.String	property	Child inner type property

Returns

TYPE	DESCRIPTION
IExternalLinkFlag	External link flag

Type Parameters

NAME	DESCRIPTION
TChildInner	Child inner type

TreeFilter()

External link filter tree root condition (START WITH) flag

Declaration

IExternalLinkFlag TreeFilter()

Returns

TYPE	DESCRIPTION
IExternalLinkFlag	External link flag

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)

TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>, TResult)

TypeExtensions.ChangeType<T>(Object)

TypeExtensions.ChangeType(Object, Type)

TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)

TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)

SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)

QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Interface IExternalLinkDefinition

Dynamic structure external link definition interface

Namespace: [XData.Mapping.Dynamic](#)

Assembly: [XData.docfx.dll](#)

Syntax

```
public interface IExternalLinkDefinition
```

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Interface IExternalLinkDefinitionAdapter<T>

External link definition adapter

Namespace: [XData.Mapping.Dynamic](#)

Assembly: XData.docfx.dll

Syntax

```
public interface IExternalLinkDefinitionAdapter<T>
    where T : class, ISqlObject, new()
```

Type Parameters

NAME	DESCRIPTION
T	Parent data object type

Methods

ExternalLink<TChild>(Expression<Func<T, Object>>, Expression<Func<IExternalLinkAdapter<T, TChild>, IExternalLinkFlag>>[])

External link definition

Declaration

```
IExternalLinkDefinition ExternalLink<TChild>(Expression<Func<T, object>> parent, params
Expression<Func<IExternalLinkAdapter<T, TChild>, IExternalLinkFlag>>[] linkFlags)
    where TChild : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func<T, System.Object>>	parent	Link parent object property
System.Linq.Expressions.Expression<System.Func< IExternalLinkAdapter <T, TChild>, IExternalLinkFlag >>[]	linkFlags	External link flags

Returns

TYPE	DESCRIPTION
IExternalLinkDefinition	External link definition

Type Parameters

NAME	DESCRIPTION
TChild	External link repository data object type

ExternalLink<TChild>(Expression<Func<IExternalLinkAdapter<T, TChild>, IExternalLinkFlag>>[])

External link definition

Declaration

```
IEternalLinkDefinition ExternalLink<TChild>(params Expression<Func<IEternalLinkAdapter<T, TChild>,
IEternalLinkFlag>>[] linkFlags)
    where TChild : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func< IEternalLinkAdapter <T, TChild>, IEternalLinkFlag >>[]	linkFlags	External link flags

Returns

TYPE	DESCRIPTION
IEternalLinkDefinition	External link definition

Type Parameters

NAME	DESCRIPTION
TChild	External link repository data object type

ExternalLink<TChild>(String, Expression<Func<IEternalLinkAdapter<T, TChild>, IEternalLinkFlag>>[])

External link definition

Declaration

```
IEternalLinkDefinition ExternalLink<TChild>(string property, params Expression<Func<IEternalLinkAdapter<T,
TChild>, IEternalLinkFlag>>[] linkFlags)
    where TChild : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	property	Link property name
System.Linq.Expressions.Expression<System.Func< IEternalLinkAdapter <T, TChild>, IEternalLinkFlag >>[]	linkFlags	External link flags

Returns

TYPE	DESCRIPTION
IEternalLinkDefinition	External link definition

Type Parameters

NAME	DESCRIPTION
TChild	External link repository data object type

Extension Methods

SerializationExtensions.ToXml<T>(T)

CollectionsExtensions.SetValue<TKey, T>(TKey, T)

CollectionsExtensions.AsEnum<T>(T)

DataObjectQueryableExtensions.CompareVariable(Object, FilterOperation, String)

Processing.Do<T>(T, Action<IProcess<T>>[])

TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>)

TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>, TResult)

TypeExtensions.ChangeType<T>(Object)

TypeExtensions.ChangeType(Object, Type)

TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)

TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)

SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)

QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Interface IExternalLinkFlag

Dynamic structure external link flag interface

Namespace: [XData.Mapping.Dynamic](#)

Assembly: [XData.docfx.dll](#)

Syntax

```
public interface IExternalLinkFlag
```

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Interface IFilterAdapter

Filter description adapter

Namespace: [XData.Mapping.Dynamic](#)

Assembly: XData.docfx.dll

Syntax

```
public interface IFilterAdapter
```

Methods

Combination(String, Combination)

Filter combination description

Declaration

```
IFilterDescription Combination(string name, Combination combination)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Filter combination name
Combination	combination	Filter combination logic

Returns

TYPE	DESCRIPTION
IFilterDescription	Filter description

Combination(String, Combination, String)

Filter combination description

Declaration

```
IFilterDescription Combination(string name, Combination combination, string parent)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Filter combination name
Combination	combination	Filter combination logic
System.String	parent	Parent filter combination name

Returns

TYPE	DESCRIPTION
IFilterDescription	Filter description

Extension Methods

[SerializationExtensions.ToXml<T>\(T\)](#)
[CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
[CollectionsExtensions.AsEnum<T>\(T\)](#)
[DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
[Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
[TypeExtensions.ChangeType<T>\(Object\)](#)
[TypeExtensions.ChangeType\(Object, Type\)](#)
[TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
[TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
[TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
[TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
[SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
[QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Interface IFilterDescription

Dynamic query filter description

Namespace: [XData.Mapping.Dynamic](#)

Assembly: [XData.docfx.dll](#)

Syntax

```
public interface IFilterDescription
```

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetOperation<T>\(T, FilterOperation\)](#)
- [QueryDescriptionExtensions.AsPrimary<T>\(T\)](#)
- [QueryDescriptionExtensions.SetCombination<T>\(T, String\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Interface IInheritedStructure<T>

Inherited repository structure descriptor

Namespace: [XData.Mapping.Dynamic](#)

Assembly: XData.docfx.dll

Syntax

```
public interface IInheritedStructure<T>
    where T : class, ISqlObject, new()
```

Type Parameters

NAME	DESCRIPTION
T	Data object type

Methods

AddInnerFilters(String, Expression<Func<IInnerFilterAdapter, IFilterDescription>>>[])

Add inner filters

Declaration

```
IInheritedStructure<T> AddInnerFilters(string alias, params Expression<Func<IInnerFilterAdapter, IFilterDescription>>>[] filters)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	alias	Data source alias
System.Linq.Expressions.Expression<System.Func< IInnerFilterAdapter , IFilterDescription >>>[]	filters	Filters to add

Returns

TYPE	DESCRIPTION
IInheritedStructure <T>	Inherited repository structure

AddSubqueryLinks(String, Expression<Func<ISubqueryLinkAdapter<T>, IFilterDescription>>>[])

Add subquery links

Declaration

```
IInheritedStructure<T> AddSubqueryLinks(string alias, params Expression<Func<ISubqueryLinkAdapter<T>, IFilterDescription>>>[] filters)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	alias	Subquery alias
System.Linq.Expressions.Expression<System.Func<ISubqueryLinkAdapter<T>, IFilterDescription>>[]	filters	Links to add

Returns

TYPE	DESCRIPTION
IIheritedStructure<T>	Inherited repository structure

AddSubqueryLinks<TSubquery>(String, Expression<Func<ISubqueryLinkAdapter<T, TSubquery>, IFilterDescription>>[])

Add subquery links

Declaration

```
IIheritedStructure<T> AddSubqueryLinks<TSubquery>(string alias, params
Expression<Func<ISubqueryLinkAdapter<T, TSubquery>, IFilterDescription>>[] filters)
    where TSubquery : class
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	alias	Subquery alias
System.Linq.Expressions.Expression<System.Func<ISubqueryLinkAdapter<T, TSubquery>, IFilterDescription>>[]	filters	Links to add

Returns

TYPE	DESCRIPTION
IIheritedStructure<T>	Inherited repository structure

Type Parameters

NAME	DESCRIPTION
TSubquery	

Column<TResult>(String, Expression<Func<IRepositoryStructureAdapter<T>, TResult>>)

Dynamic repository hidden property mapping

Declaration

```
IIheritedStructure<T> Column<TResult>(string name, Expression<Func<IRepositoryStructureAdapter<T>, TResult>>
mapper)
```


Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Hidden property name
System.Linq.Expressions.Expression<System.Func< IRepositoryStructureAdapter <T>, TResult>>	mapper	Property mapping expression

Returns

TYPE	DESCRIPTION
IInheritedStructure <T>	Inherited repository structure

Type Parameters

NAME	DESCRIPTION
TResult	Hidden property type

See Also

- [GetStructure\(String, String, DataStructureFlag, Variable\[\]\)](#)
- [IQueryStructure](#)
- [IQueryStructureAdapter](#)

`DataTable(String, String, Expression<Func<IInnerFilterAdapter, IFilterDescription>>[])`

Dynamic repository table mapping

Declaration

```
IInheritedStructure<T> DataTable(string name, string alias, params Expression<Func<IInnerFilterAdapter, IFilterDescription>>[] filters)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Table name
System.String	alias	Table alias
System.Linq.Expressions.Expression<System.Func< IInnerFilterAdapter , IFilterDescription >>[]	filters	Filters and links applied to table

Returns

TYPE	DESCRIPTION

TYPE	DESCRIPTION
IInheritedStructure <T>	Inherited repository structure

See Also

[GetStructure\(String, String, DataStructureFlag, Variable\[\]\)](#)

[IQueryStructure](#)

[IFilterDescription](#)

[DataTable\(String, String, String, Expression<Func<IInnerFilterAdapter, IFilterDescription>>\[\]\)](#)

Dynamic repository table mapping

Declaration

```
IInheritedStructure<T> DataTable(string name, string alias, string parent = null, params
Expression<Func<IInnerFilterAdapter, IFilterDescription>>[] filters)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Table name
System.String	alias	Table alias
System.String	parent	Parent table alias
System.Linq.Expressions.Expression<System.Func< IInnerFilterAdapter , IFilterDescription >>[]	filters	Filters and links applied to table

Returns

TYPE	DESCRIPTION
IInheritedStructure <T>	Inherited repository structure

See Also

[GetStructure\(String, String, DataStructureFlag, Variable\[\]\)](#)

[IQueryStructure](#)

[IFilterDescription](#)

[Hint\(String, String, HintType\)](#)

Dynamic repository data source hint

Declaration

```
IInheritedStructure<T> Hint(string alias, string hint, HintType hintType = HintType.Select)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	alias	Data source alias
System.String	hint	Hint
HintType	hintType	Hint type

Returns

TYPE	DESCRIPTION
IInheritedStructure <T>	Inherited repository structure

See Also

[GetStructure\(String, String, DataStructureFlag, Variable\[\]\)](#)

[InnerView\(String, Type, Expression<Func<ISubqueryLinkAdapter<T>, IFilterDescription>>\[\]\)](#)

Dynamic repository inner view mapping

Declaration

```
IInheritedStructure<T> InnerView(string alias, Type innerViewType, params
Expression<Func<ISubqueryLinkAdapter<T>, IFilterDescription>>[] filters)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	alias	Inner view alias
System.Type	innerViewType	Inner view statically mapped type
System.Linq.Expressions.Expression<System.Func< ISubqueryLinkAdapter <T>, IFilterDescription >>[]	filters	Filters and links applied to inner view

Returns

TYPE	DESCRIPTION
IInheritedStructure <T>	Inherited repository structure

See Also

[GetStructure\(String, String, DataStructureFlag, Variable\[\]\)](#)

[IQueryStructure](#)

[IFilterDescription](#)

InnerView<TInnerView>(String, Expression<Func<ISubqueryLinkAdapter<T, TInnerView>, IFilterDescription>>[])

Dynamic repository inner view mapping

Declaration

```
IInheritedStructure<T> InnerView<TInnerView>(string alias, params Expression<Func<ISubqueryLinkAdapter<T, TInnerView>, IFilterDescription>>[] filters)
    where TInnerView : class
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	alias	Inner view alias
System.Linq.Expressions.Expression<System.Func< ISubqueryLinkAdapter <T, TInnerView>, IFilterDescription >>[]	filters	Filters and links applied to inner view

Returns

TYPE	DESCRIPTION
IInheritedStructure <T>	Inherited repository structure

Type Parameters

NAME	DESCRIPTION
TInnerView	Inner view statically mapped type

See Also

[GetStructure\(String, String, DataStructureFlag, Variable\[\]\)](#)

[IQueryStructure](#)

[IFilterDescription](#)

InnerView<TInnerView>(String, IQueryDescription<TInnerView>, Expression<Func<ISubqueryLinkAdapter<T, TInnerView>, IFilterDescription>>[])

Dynamic repository inner view mapping

Declaration

```
IInheritedStructure<T> InnerView<TInnerView>(string alias, IQueryDescription<TInnerView> sub, params Expression<Func<ISubqueryLinkAdapter<T, TInnerView>, IFilterDescription>>[] filters)
    where TInnerView : class
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	alias	Inner view alias

TYPE	NAME	DESCRIPTION
IQueryDescription <TInnerView>	sub	Inner view query structure
System.Linq.Expressions.Expression<System.Func< ISubqueryLinkAdapter <T, TInnerView>, IFilterDescription >> []	filters	Filters and links applied to inner view

Returns

TYPE	DESCRIPTION
IInheritedStructure <T>	Inherited repository structure

Type Parameters

NAME	DESCRIPTION
TInnerView	

See Also

[GetStructure\(String, String, DataStructureFlag, Variable\[\]\)](#)

[IQueryStructure](#)

[IQueryDescription](#)

[IFilterDescription](#)

InnerView<TInnerView>(String, Variable[], Expression<Func<[ISubqueryLinkAdapter](#)<T, TInnerView>, [IFilterDescription](#)>> [])

Dynamic repository inner view mapping

Declaration

<pre>IInheritedStructure<T> InnerView<TInnerView>(string alias, Variable[] variables, params Expression<Func<ISubqueryLinkAdapter<T, TInnerView>, IFilterDescription>>[] filters) where TInnerView : class</pre>

Parameters

TYPE	NAME	DESCRIPTION
System.String	alias	Inner view alias
Variable []	variables	Inner view variables
System.Linq.Expressions.Expression<System.Func< ISubqueryLinkAdapter <T, TInnerView>, IFilterDescription >> []	filters	Filters and links applied to inner view

Returns

TYPE	DESCRIPTION
IInheritedStructure <T>	Inherited repository structure

Type Parameters

NAME	DESCRIPTION
TInnerView	Inner view statically mapped type

See Also

[GetStructure\(String, String, DataStructureFlag, Variable\[\]\)](#)

[IQueryStructure](#)

[Variable](#)

[IFilterDescription](#)

Map(Expression<Func<IExternalLinkDefinitionAdapter<T>, IExternalLinkDefinition>>[])

Dynamic repository properties mapping

Declaration

```
IRepositoryDescription<T> Map(params Expression<Func<IExternalLinkDefinitionAdapter<T>,
IExternalLinkDefinition>>[] externalLinks)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func< IExternalLinkDefinitionAdapter <T>, IExternalLinkDefinition >>[]	externalLinks	External link descriptions

Returns

TYPE	DESCRIPTION
IRepositoryDescription <T>	Dynamic repository description

See Also

[GetStructure\(String, String, DataStructureFlag, Variable\[\]\)](#)

[IQueryStructure](#)

[IQueryDescription](#)<T>

[IQueryStructureAdapter](#)

Map(Expression<Func<IRepositoryStructureAdapter<T>, T>>, Expression<Func<IExternalLinkDefinitionAdapter<T>, IExternalLinkDefinition>>[])

Dynamic repository properties mapping

Declaration

```
IRepositoryDescription<T> Map(Expression<Func<IRepositoryStructureAdapter<T>, T>> mapper, params
Expression<Func<IExternalLinkDefinitionAdapter<T>, IExternalLinkDefinition>>[] externalLinks)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func< IRepositoryStructureAdapter <T>, T>>	mapper	Mapping expression
System.Linq.Expressions.Expression<System.Func< IExternalLinkDefinitionAdapter <T>, IExternalLinkDefinition >> []	externalLinks	External link descriptions

Returns

TYPE	DESCRIPTION
IRepositoryDescription <T>	Dynamic repository description

See Also

- [GetStructure\(String, String, DataStructureFlag, Variable\[\]\)](#)
- [IQueryStructure](#)
- [IQueryDescription](#)<T>
- [IQueryStructureAdapter](#)

[Parameter](#)<TResult>(String, Int32, String, DbType, Expression<Func<[IProcedureParameterAdapter](#), [IProcedureParameterFlag](#)>>[])

Dynamic repository procedure parameter mapping

Declaration

```
IInheritedStructure<T> Parameter<TResult>(string alias, int order, string binding, DbType type, params Expression<Func<IProcedureParameterAdapter, IProcedureParameterFlag>>[] attributes)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	alias	Procedure alias
System.Int32	order	Parameter order
System.String	binding	Parameter value binding
System.Data.DbType	type	Parameter SQL data type
System.Linq.Expressions.Expression<System.Func< IProcedureParameterAdapter , IProcedureParameterFlag >> []	attributes	Parameter attributes

Returns

TYPE	DESCRIPTION
IInheritedStructure<T>	Inherited repository structure

Type Parameters

NAME	DESCRIPTION
TResult	Parameter data type

Procedure(String, String, ProcedureType, Expression<Func<IInnerFilterAdapter, IFilterDescription>>[])

Dynamic repository procedure mapping

Declaration

```
IInheritedStructure<T> Procedure(string alias, string name, ProcedureType type, params
Expression<Func<IInnerFilterAdapter, IFilterDescription>>[] filters)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	alias	Procedure alias
System.String	name	Procedure name
ProcedureType	type	Procedure type
System.Linq.Expressions.Expression<System.Func< IInnerFilterAdapter , IFilterDescription >>[]	filters	Filters and links applied to table

Returns

TYPE	DESCRIPTION
IInheritedStructure<T>	Inherited repository structure

See Also

[GetStructure\(String, String, DataStructureFlag, Variable\[\]\)](#)

[IQueryStructure](#)

[IFilterDescription](#)

ReadOnlyProperty<TResult>(Expression<Func<T, TResult>>, Expression<Func<IRepositoryStructureAdapter<T>, TResult>>)

Dynamic repository read only properties mapping

Declaration


```
IInheritedStructure<T> ReadOnlyProperty<TResult>(Expression<Func<T, TResult>> property,
Expression<Func<IRepositoryStructureAdapter<T>, TResult>> mapper)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func<T, TResult>>	property	Property name expression
System.Linq.Expressions.Expression<System.Func<IRepositoryStructureAdapter<T>, TResult>>	mapper	Property mapping

Returns

TYPE	DESCRIPTION
IInheritedStructure<T>	Inherited repository structure

Type Parameters

NAME	DESCRIPTION
TResult	Property type

Return(String)

Dynamic repository procedure default result set mapping

Declaration

```
IInheritedStructure<T> Return(string alias)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	alias	Procedure alias

Returns

TYPE	DESCRIPTION
IInheritedStructure<T>	Inherited repository structure

Return<TResult>(String, String, Int32, Expression<Func<IQueryStructureAdapter, T>>)

Dynamic repository procedure result set mapping

Declaration

```
IInheritedStructure<T> Return<TResult>(string alias, string name, int order,
Expression<Func<IQueryStructureAdapter, T>> mapper = null)
    where TResult : class
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	alias	Procedure alias
System.String	name	Result set name
System.Int32	order	Result set order
System.Linq.Expressions.Expression<System.Func< IQueryStructureAdapter , T>>	mapper	Result set mapper

Returns

TYPE	DESCRIPTION
IInheritedStructure <T>	Inherited repository structure

Type Parameters

NAME	DESCRIPTION
TResult	Result set data type

SetSubqueryVariables(String, Variable[])

Set subquery variable values

Declaration

```
IInheritedStructure<T> SetSubqueryVariables(string alias, params Variable[] variables)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	alias	Subquery alias
Variable []	variables	Variables to set

Returns

TYPE	DESCRIPTION
IInheritedStructure <T>	Inherited repository structure

Subquery(String, Type, String, DataGrouping, Expression<Func<ISubqueryLinkAdapter<T>, IFilterDescription>>[])

Dynamic repository subquery mapping

Declaration

```
IInheritedStructure<T> Subquery(string alias, Type subqueryType, string property, DataGrouping grouping = DataGrouping.None, params Expression<Func<ISubqueryLinkAdapter<T>, IFilterDescription>>[] filters)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	alias	Subquery alias
System.Type	subqueryType	Subquery statically mapped type
System.String	property	Subquery selected property
DataGrouping	grouping	Subquery selected property aggregate function
System.Linq.Expressions.Expression<System.Func< ISubqueryLinkAdapter <T>, IFilterDescription >>[]	filters	Filters and links applied to subquery

Returns

TYPE	DESCRIPTION
IInheritedStructure <T>	Inherited repository structure

See Also

[GetStructure\(String, String, DataStructureFlag, Variable\[\]\)](#)

[IQueryStructure](#)

[IFilterDescription](#)

Subquery<TSubquery>(String, Expression<Func<TSubquery, Object>>, DataGrouping, Expression<Func<ISubqueryLinkAdapter<T, TSubquery>, IFilterDescription>>[])

Dynamic repository subquery mapping

Declaration

```
IInheritedStructure<T> Subquery<TSubquery>(string alias, Expression<Func<TSubquery, object>> property, DataGrouping grouping = DataGrouping.None, params Expression<Func<ISubqueryLinkAdapter<T, TSubquery>, IFilterDescription>>[] filters)
    where TSubquery : class
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	alias	Subquery alias
System.Linq.Expressions.Expression<System.Func<TSubquery, System.Object>>	property	Subquery selected property expression
DataGrouping	grouping	Subquery selected property aggregate function
System.Linq.Expressions.Expression<System.Func< ISubqueryLinkAdapter <T, TSubquery>, IFilterDescription >>[]	filters	Filters and links applied to subquery

Returns

TYPE	DESCRIPTION
IInheritedStructure <T>	Inherited repository structure

Type Parameters

NAME	DESCRIPTION
TSubquery	Subquery statically mapped type

See Also

- [GetStructure\(String, String, DataStructureFlag, Variable\[\]\)](#)
- [IQueryStructure](#)
- [IFilterDescription](#)

Subquery<TSubquery>(String, Expression<Func<TSubquery, Object>>, DataGrouping, Variable[], Expression<Func<[ISubqueryLinkAdapter](#)<T, TSubquery>, [IFilterDescription](#)>>[])

Dynamic repository subquery mapping

Declaration

```
IInheritedStructure<T> Subquery<TSubquery>(string alias, Expression<Func<TSubquery, object>> property,
DataGrouping grouping, Variable[] variables, params Expression<Func<ISubqueryLinkAdapter<T, TSubquery>,
IFilterDescription>>[] filters)
    where TSubquery : class
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	alias	Subquery alias

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func<TSubquery, System.Object> >	property	Subquery selected property expression
DataGrouping	grouping	Subquery selected property aggregate function
Variable[]	variables	Subquery variables
System.Linq.Expressions.Expression<System.Func< ISubqueryLinkAdapter <T, TSubquery>, IFilterDescription > > []	filters	Filters and links applied to subquery

Returns

TYPE	DESCRIPTION
IInheritedStructure <T>	Inherited repository structure

Type Parameters

NAME	DESCRIPTION
TSubquery	Subquery statically mapped type

See Also

- [GetStructure\(String, String, DataStructureFlag, Variable\[\]\)](#)
- [IQueryStructure](#)
- [IFilterDescription](#)
- [Variable](#)

Subquery<TSubquery>(String, IQueryDescription<TSubquery>, String, DataGrouping, Expression<Func<ISubqueryLinkAdapter<T, TSubquery>, IFilterDescription>>[])

Dynamic repository subquery mapping

Declaration

```
IInheritedStructure<T> Subquery<TSubquery>(string alias, IQueryDescription<TSubquery> sub, string property,
DataGrouping grouping = DataGrouping.None, params Expression<Func<ISubqueryLinkAdapter<T, TSubquery>,
IFilterDescription>>[] filters)
    where TSubquery : class
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	alias	Subquery alias

TYPE	NAME	DESCRIPTION
IQueryDescription <TSubquery>	sub	Subquery query structure
System.String	property	Subquery selected property
DataGrouping	grouping	Subquery selected property aggregate function
System.Linq.Expressions.Expression<System.Func< ISubqueryLinkAdapter <T, TSubquery>, IFilterDescription >>[]	filters	Filters and links applied to subquery

Returns

TYPE	DESCRIPTION
IInheritedStructure <T>	Inherited repository structure

Type Parameters

NAME	DESCRIPTION
TSubquery	

See Also

- [GetStructure\(String, String, DataStructureFlag, Variable\[\]\)](#)
- [IQueryStructure](#)
- [IQueryDescription](#)
- [DataGrouping](#)
- [IFilterDescription](#)

With<TRoot>(String, IQueryDescription<TRoot>, Expression<Func<IQueryWithAdapter<TRoot>, IQueryWithFlag>>[])

Dynamic repository Common-Table-Expression (WITH) subquery mapping

Declaration

```
IInheritedStructure<T> With<TRoot>(string alias, IQueryDescription<TRoot> sub, params
Expression<Func<IQueryWithAdapter<TRoot>, IQueryWithFlag>>[] attributes)
    where TRoot : class
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	alias	Common-Table-Expression (WITH) subquery alias

TYPE	NAME	DESCRIPTION
IQueryDescription <TRoot>	sub	Common-Table-Expression (WITH) initial subquery description
System.Linq.Expressions.Expression<System.Func< IQueryWithAdapter <TRoot>, IQueryWithFlag >> []	attributes	Common-Table-Expression (WITH) attributes

Returns

TYPE	DESCRIPTION
IInheritedStructure <T>	Inherited repository structure

Type Parameters

NAME	DESCRIPTION
TRoot	Common-Table-Expression (WITH) initial subquery mapped type

Extension Methods

- [SerializationExtensions.ToXml](#)<T>(T)
- [CollectionsExtensions.SetValue](#)<TKey, T>(TKey, T)
- [CollectionsExtensions.AsEnum](#)<T>(T)
- [DataObjectQueryableExtensions.CompareVariable](#)(Object, FilterOperation, String)
- [Processing.Do](#)<T>(T, Action<IProcess<T>> [])
- [TypeExtensions.TryConvert](#)<T, TResult>(T, Func<T, TResult>)
- [TypeExtensions.TryConvert](#)<T, TResult>(T, Func<T, TResult>, TResult)
- [TypeExtensions.ChangeType](#)<T>(Object)
- [TypeExtensions.ChangeType](#)(Object, Type)
- [TypeExtensions.InvokeMethod](#)<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)
- [TypeExtensions.InvokeMethod](#)(Object, Type[], Expression<Action>, BindingFlags)
- [TypeExtensions.InvokeMethod](#)<TRet>(Object, Expression<Func<TRet>>, BindingFlags)
- [TypeExtensions.InvokeMethod](#)(Object, Expression<Action>, BindingFlags)
- [SqlBlockExtensions.SetExpression](#)<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)
- [QueryDescriptionExtensions.SetExpression](#)<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Interface IInnerFilterAdapter

Inner filter description adapter

Inherited Members

[IFilterAdapter.Combination\(String, Combination, String\)](#)
[IFilterAdapter.Combination\(String, Combination\)](#)

Namespace: [XData.Mapping.Dynamic](#)
Assembly: [XData.docfx.dll](#)

Syntax

```
public interface IInnerFilterAdapter : IFilterAdapter
```

Methods

ConstFilter(String, String, Object)

Constant valued filter description

Declaration

```
IFilterDescription ConstFilter(string name, string field, object constantValue)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Filter name
System.String	field	Field name to filter on
System.Object	constantValue	Constant value

Returns

TYPE	DESCRIPTION
IFilterDescription	Filter description

ConstFilter(String, String, FilterConstantType)

Constant valued filter description

Declaration

```
IFilterDescription ConstFilter(string name, string field, FilterConstantType constantType)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Filter name

TYPE	NAME	DESCRIPTION
System.String	field	Field name to filter on
FilterConstantType	constantType	Constant type

Returns

TYPE	DESCRIPTION
IFilterDescription	Filter description

ConstFilter(String, String, FilterConstantType, Object)

Constant valued filter description

Declaration

```
IFilterDescription ConstFilter(string name, string field, FilterConstantType constantType, object
constantValue)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Filter name
System.String	field	Field name to filter on
FilterConstantType	constantType	Constant type
System.Object	constantValue	Constant value

Returns

TYPE	DESCRIPTION
IFilterDescription	Filter description

ConstFilter(String, String, FilterConstantType, Object, FilterNullable)

Constant valued filter description

Declaration

```
IFilterDescription ConstFilter(string name, string field, FilterConstantType constantType, object
constantValue, FilterNullable nullable)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Filter name
System.String	field	Field name to filter on
FilterConstantType	constantType	Constant type
System.Object	constantValue	Constant value
FilterNullable	nullable	Filter null value handling mode

Returns

TYPE	DESCRIPTION
IFilterDescription	Filter description

DictFilter(String, String, String, String)

Dictionary code filter description

Declaration

```
IFilterDescription DictFilter(string dictTable, string dictId, string dictCode, string dictValue)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	dictTable	Dictionary table name
System.String	dictId	Dictionary table identity column
System.String	dictCode	Dictionary table code column
System.String	dictValue	Dictionary code value to filter on

Returns

TYPE	DESCRIPTION

TYPE	DESCRIPTION
IFilterDescription	Filter description

DictFilter(String, String, String, String, Boolean)

Dictionary code filter description

Declaration

```
IFilterDescription DictFilter(string dictTable, string dictId, string dictCode, string dictValue, bool
valueIsVariable)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	dictTable	Dictionary table name
System.String	dictId	Dictionary table identity column
System.String	dictCode	Dictionary table code column
System.String	dictValue	Dictionary code value to filter on
System.Boolean	valuesVariable	Dictionary code value is a variable name flag

Returns

TYPE	DESCRIPTION
IFilterDescription	Filter description

DictFilter(String, String, String, String, String)

Dictionary code filter description

Declaration

```
IFilterDescription DictFilter(string dictTable, string dictId, string dictCode, string dictValue, string
field)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	dictTable	Dictionary table name

TYPE	NAME	DESCRIPTION
System.String	dictId	Dictionary table identity column
System.String	dictCode	Dictionary table code column
System.String	dictValue	Dictionary code value to filter on
System.String	field	Field name to filter on

Returns

TYPE	DESCRIPTION
IFilterDescription	Filter description

DictFilter(String, String, String, String, String, Boolean)

Dictionary code filter description

Declaration

```
IFilterDescription DictFilter(string dictTable, string dictId, string dictCode, string dictValue, string field, bool valueIsVariable)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	dictTable	Dictionary table name
System.String	dictId	Dictionary table identity column
System.String	dictCode	Dictionary table code column
System.String	dictValue	Dictionary code value to filter on
System.String	field	Field name to filter on
System.Boolean	valuesVariable	Dictionary code value is a variable name flag

Returns

TYPE	DESCRIPTION
IFilterDescription	Filter description

ExprFilter(String, String)

Expression filter description

Declaration

IFilterDescription ExprFilter(string field, string exprText)

Parameters

TYPE	NAME	DESCRIPTION
System.String	field	Field name to filter on
System.String	exprText	SQL expression text

Returns

TYPE	DESCRIPTION
IFilterDescription	Filter description

ExprFilter<T>(String, Expression<Func<IQueryStructureAdapter, T>>)

Expression filter description

Declaration

IFilterDescription ExprFilter<T>(string field, Expression<Func<IQueryStructureAdapter, T>> expression)

Parameters

TYPE	NAME	DESCRIPTION
System.String	field	Field name to filter on
System.Linq.Expressions.Expression<System.Func< IQueryStructureAdapter , T>>	expression	SQL expression description

Returns

TYPE	DESCRIPTION
IFilterDescription	Filter description

Type Parameters

NAME	DESCRIPTION
T	

Link(String, String)

Inner link filter description

Declaration

```
IFilterDescription Link(string linkedAlias, string linkedField)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	linkedAlias	Linked source alias
System.String	linkedField	Linked source field name to link

Returns

TYPE	DESCRIPTION
IFilterDescription	Filter description

Link(String, String, String)

Inner link filter description

Declaration

```
IFilterDescription Link(string linkedAlias, string linkedField, string field)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	linkedAlias	Linked source alias
System.String	linkedField	Linked source field name to link
System.String	field	Current field name to link

Returns

TYPE	DESCRIPTION
IFilterDescription	Filter description

RangeFilter(String, Object[])

Range filter description

Declaration

```
IFilterDescription RangeFilter(string field, params object[] range)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	field	Field name to filter on
System.Object[]	range	Values range

Returns

TYPE	DESCRIPTION
IFilterDescription	Filter description

SubqueryFilter(String, String)

Subquery filter (exists, not exists, in select, not in select) description

Declaration

```
IFilterDescription SubqueryFilter(string field, string subqueryAlias)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	field	Field name to filter on
System.String	subqueryAlias	Subquery alias

Returns

TYPE	DESCRIPTION
IFilterDescription	Filter description

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)

TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>, TResult)

TypeExtensions.ChangeType<T>(Object)

TypeExtensions.ChangeType(Object, Type)

TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)

TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)

SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)

QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Interface ILinkPropertyMappingAdapter<T>

Link property mapping adapter

Inherited Members

- [IRepositoryReadOnlyPropertyMappingAdapter.ConcurrencyToken\(\)](#)
- [IRepositoryReadOnlyPropertyMappingAdapter.OuterFlag\(\)](#)
- [IPropertyMappingAdapter.Key\(\)](#)
- [IPropertyMappingAdapter.Group\(DataGrouping\)](#)
- [IPropertyMappingAdapter.Group\(Int32\)](#)
- [IPropertyMappingAdapter.Hidden\(\)](#)
- [IPropertyMappingAdapter.OrderBy\(Int32, Boolean\)](#)
- [IPropertyMappingAdapter.OrderBy\(Int32\)](#)

Namespace: [XData.Mapping.Dynamic](#)
Assembly: [XData.docfx.dll](#)

Syntax

```
public interface ILinkPropertyMappingAdapter<T> : IRepositoryReadOnlyPropertyMappingAdapter,
IPropertyMappingAdapter where T : class, ISqlObject, new()
```

Type Parameters

NAME	DESCRIPTION
T	Data object type

Methods

LinkProperty()

Link property data association property

Declaration

```
IStructureFlag LinkProperty()
```

Returns

TYPE	DESCRIPTION
IStructureFlag	Property link description

LinkProperty<TSource>(Expression<Func<TSource, Object>>)

Link property data association property

Declaration

```
IStructureFlag LinkProperty<TSource>(Expression<Func<TSource, object>> source)
where TSource : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func<TSource, System.Object>>	source	Source dictionary property

Returns

TYPE	DESCRIPTION
IStructureFlag	Property link description

Type Parameters

NAME	DESCRIPTION
TSource	Source dictionary type

LinkProperty<TSource>(Expression<Func<TSource, Object>>, Expression<Func<T, Object>>)

Link property data association property

Declaration

```

IStructureFlag LinkProperty<TSource>(Expression<Func<TSource, object>> source, Expression<Func<T, object>>
property)
    where TSource : class, IDataObject, new()

```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func<TSource, System.Object>>	source	Source dictionary property
System.Linq.Expressions.Expression<System.Func<T, System.Object>>	property	Property

Returns

TYPE	DESCRIPTION
IStructureFlag	Property link description

Type Parameters

NAME	DESCRIPTION
TSource	Source dictionary type

LinkProperty<TSource>(Expression<Func<T, Object>>)

Link property data association property

Declaration

```
IStructureFlag LinkProperty<TSource>(Expression<Func<T, object>> property)
    where TSource : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func<T, System.Object>>	property	Property

Returns

TYPE	DESCRIPTION
IStructureFlag	Property link description

Type Parameters

NAME	DESCRIPTION
TSource	Source dictionary type

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Interface IParameterDescription

Dynamic stored procedure parameter description

Namespace: [XData.Mapping.Dynamic](#)

Assembly: XData.docfx.dll

Syntax

```
public interface IParameterDescription
```

Properties

Alias

Procedure source alias

Declaration

```
string Alias { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Binding

Parameter value binding

Declaration

```
string Binding { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

DbType

Parameter database type

Declaration

```
DbType DbType { get; }
```

Property Value

TYPE	DESCRIPTION
System.Data.DbType	

DefaultValue

Default parameter value

Declaration

```
object DefaultValue { get; }
```

Property Value

TYPE	DESCRIPTION
System.Object	

DefaultValueType

Default parameter value type

Declaration

```
DefaultType DefaultValue { get; }
```

Property Value

TYPE	DESCRIPTION
DefaultType	

Direction

Parameter direction

Declaration

```
ParameterDirection Direction { get; }
```

Property Value

TYPE	DESCRIPTION
System.Data.ParameterDirection	

Name

Parameter name

Declaration

```
string Name { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

NativeSqlType

Native SQL type name

Declaration

```
string NativeSqlType { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Remarks

Used to specify native SQL column type when conflicted with default type mapping

WARNING! May be not supported by database adapter!

Order

Parameter order

Declaration

```
int Order { get; }
```

Property Value

TYPE	DESCRIPTION
System.Int32	

Scale

Parameter scale

Declaration

```
int Scale { get; }
```

Property Value

TYPE	DESCRIPTION
System.Int32	

Size

Parameter size

Declaration

```
int Size { get; }
```

Property Value

TYPE	DESCRIPTION
System.Int32	

Type

Parameter mapped type

Declaration

```
Type Type { get; }
```

Property Value

TYPE	DESCRIPTION
System.Type	

TypeIsArray

Parameter mapped type is array

Declaration

```
bool TypeIsArray { get; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

UdtSqlType

Parameter SQL user defined type (UDT) description

Declaration

```
UdtSqlType UdtSqlType { get; }
```

Property Value

TYPE	DESCRIPTION
UdtSqlType	

Value

Parameter value

Declaration

```
object Value { get; }
```

Property Value

TYPE	DESCRIPTION
System.Object	

Methods

Clone()

Clone parameter description

Declaration

```
IParame terDescription Clone()
```

Returns

TYPE	DESCRIPTION
IParame terDescription	Cloned parameter

GetValue<T>()

Returns parameter value

Declaration

T GetValue<T>()

Returns

TYPE	DESCRIPTION
T	Parameter value

Type Parameters

NAME	DESCRIPTION
T	Parameter mapped type

Extension Methods

- SerializationExtensions.ToXml<T>(T)
- CollectionsExtensions.SetValue<TKey, T>(TKey, T)
- CollectionsExtensions.AsEnum<T>(T)
- DataObjectQueryableExtensions.CompareVariable(Object, FilterOperation, String)
- Processing.Do<T>(T, Action<IProcess<T>>[])
- TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>)
- TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>, TResult)
- TypeExtensions.ChangeType<T>(Object)
- TypeExtensions.ChangeType(Object, Type)
- TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)
- TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)
- TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)
- TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)
- SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)
- QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Interface IProcedureParameterAdapter

Procedure parameter mapping adapter

Namespace: [XData.Mapping.Dynamic](#)

Assembly: XData.docfx.dll

Syntax

```
public interface IProcedureParameterAdapter
```

Methods

Array()

Parameter is simple type array flag

Declaration

```
IProcedureParameterFlag Array()
```

Returns

TYPE	DESCRIPTION
IProcedureParameterFlag	Procedure parameter mapping flag

Default(Object)

Parameter default value

Declaration

```
IProcedureParameterFlag Default(object defaultValue)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Object	defaultValue	Parameter default value

Returns

TYPE	DESCRIPTION
IProcedureParameterFlag	Procedure parameter mapping flag

Default(DefaultType)

Parameter default value

Declaration

```
IProcedureParameterFlag Default(DefaultType type)
```

Parameters

TYPE	NAME	DESCRIPTION
DefaultType	type	Parameter default value type

Returns

TYPE	DESCRIPTION
IProcedureParameterFlag	Procedure parameter mapping flag

Default(DefaultType, Object)

Parameter default value

Declaration

```
IProcedureParameterFlag Default(DefaultType type, object defaultValue)
```

Parameters

TYPE	NAME	DESCRIPTION
DefaultType	type	Parameter default value type
System.Object	defaultValue	Parameter default value

Returns

TYPE	DESCRIPTION
IProcedureParameterFlag	Procedure parameter mapping flag

Direction(ParameterDirection)

Parameter direction

Declaration

```
IProcedureParameterFlag Direction(ParameterDirection direction)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Data.ParameterDirection	direction	Parameter direction

Returns

TYPE	DESCRIPTION

TYPE	DESCRIPTION
IProcedureParameterFlag	Procedure parameter mapping flag

NativeSqlType(String)

Native SQL type name

Declaration

```
IProcedureParameterFlag NativeSqlType(string nativeSqlType)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	nativeSqlType	Native SQL type name

Returns

TYPE	DESCRIPTION
IProcedureParameterFlag	Procedure parameter mapping flag

ParameterName(String)

Parameter mapping name

Declaration

```
IProcedureParameterFlag ParameterName(string parameterName)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	parameterName	Parameter mapping name

Returns

TYPE	DESCRIPTION
IProcedureParameterFlag	Procedure parameter mapping flag

Size(Int32)

Size of SQL expression result value

Declaration

```
IProcedureParameterFlag Size(int size)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Int32	size	Size

Returns

TYPE	DESCRIPTION
IProcedureParameterFlag	SQL expression mapping flag

Size(Int32, Int32)

Size of SQL expression result value

Declaration

```
IProcedureParameterFlag Size(int size, int scale)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Int32	size	Size
System.Int32	scale	Scale

Returns

TYPE	DESCRIPTION
IProcedureParameterFlag	SQL expression mapping flag

UdtDataType(String)

User defined type mapping

Declaration

```
IProcedureParameterFlag UdtDataType(string sqlTypeName)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	sqlTypeName	UDT SQL type name

Returns

TYPE	DESCRIPTION

TYPE	DESCRIPTION
IProcedureParameterFlag	Procedure parameter mapping flag

UdtDataType(String, String)

User defined type mapping

Declaration

<code>IProcedureParameterFlag UdtDataType(string sqlTypeName, string elementSqlTypeName)</code>

Parameters

TYPE	NAME	DESCRIPTION
System.String	sqlTypeName	UDT SQL type name
System.String	elementSqlTypeName	UDT type element type name

Returns

TYPE	DESCRIPTION
IProcedureParameterFlag	Procedure parameter mapping flag

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Interface IProcedureParameterFlag

Procedure parameter mapping flag

Namespace: [XData.Mapping.Dynamic](#)

Assembly: XData.docfx.dll

Syntax

```
public interface IProcedureParameterFlag
```

Extension Methods

[SerializationExtensions.ToXml<T>\(T\)](#)

[CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)

[CollectionsExtensions.AsEnum<T>\(T\)](#)

[DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)

[Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)

[TypeExtensions.ChangeType<T>\(Object\)](#)

[TypeExtensions.ChangeType\(Object, Type\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)

[SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)

[QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Interface IPropertyMappingAdapter

Basic dynamic structure adapter

Namespace: [XData.Mapping.Dynamic](#)

Assembly: XData.docfx.dll

Syntax

```
public interface IPropertyMappingAdapter
```

Methods

Group(Int32)

Property grouping modifier

Declaration

```
IStructureFlag Group(int groupOrder)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Int32	groupOrder	Property grouping option

Returns

TYPE	DESCRIPTION
IStructureFlag	Property grouping description

Group(DataGrouping)

Property grouping modifier

Declaration

```
IStructureFlag Group(DataGrouping grouping)
```

Parameters

TYPE	NAME	DESCRIPTION
DataGrouping	grouping	Property grouping option

Returns

TYPE	DESCRIPTION
IStructureFlag	Property key description

Remarks

Used as property description attribute

Hidden()

Column is hidden column flag

Declaration

```
IStructureFlag Hidden()
```

Returns

TYPE	DESCRIPTION
IStructureFlag	

Key()

Property is key modifier

Declaration

```
IStructureFlag Key()
```

Returns

TYPE	DESCRIPTION
IStructureFlag	Property key description

Remarks

Used as property description attribute

OrderBy(Int32)

Property order by modifier

Declaration

```
IStructureFlag OrderBy(int order)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Int32	order	Property oder by option

Returns

TYPE	DESCRIPTION
IStructureFlag	Property oder by description

OrderBy(Int32, Boolean)

Property order by modifier

Declaration

IStructureFlag OrderBy(int order, bool desc)

Parameters

TYPE	NAME	DESCRIPTION
System.Int32	order	Property oder by option
System.Boolean	desc	Descent order flag

Returns

TYPE	DESCRIPTION
IStructureFlag	Property oder by description

Extension Methods

[SerializationExtensions.ToXml<T>\(T\)](#)

[CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)

[CollectionsExtensions.AsEnum<T>\(T\)](#)

[DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)

[Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)

[TypeExtensions.ChangeType<T>\(Object\)](#)

[TypeExtensions.ChangeType\(Object, Type\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)

[SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)

[QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Interface IQueryDescription

Dynamic query description basic interface

Namespace: [XData.Mapping.Dynamic](#)

Assembly: [XData.docfx.dll](#)

Syntax

```
public interface IQueryDescription
```

Properties

Type

Dynamic query description element type

Declaration

```
Type Type { get; }
```

Property Value

TYPE	DESCRIPTION
System.Type	

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Interface IQueryDescription<T>

Dynamic query description with result type specified interface

Inherited Members

[IQueryDescription.Type](#)

Namespace: [XData.Mapping.Dynamic](#)

Assembly: XData.docfx.dll

Syntax

```
public interface IQueryDescription<out T> : IQueryDescription
```

Type Parameters

NAME	DESCRIPTION
T	Query description element type

Methods

[AsQuery\(IDataScope, Variable\[\]\)](#)

Returns queryable data source based on description

Declaration

```
IDataQuery<T> AsQuery(IDataScope dataScope, params Variable[] variables)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataScope	dataScope	Data scope
Variable[]	variables	Object variables values

Returns

TYPE	DESCRIPTION
IDataQuery<T>	Data source

Extension Methods

[SerializationExtensions.ToXml<T>\(T\)](#)

[CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)

[CollectionsExtensions.AsEnum<T>\(T\)](#)

[DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)

[Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)

[TypeExtensions.ChangeType<T>\(Object\)](#)

[TypeExtensions.ChangeType\(Object, Type\)](#)

TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)

TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)

SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)

QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Interface IQueryStructure

Dynamic query structure interface

Namespace: [XData.Mapping.Dynamic](#)

Assembly: XData.docfx.dll

Syntax

```
public interface IQueryStructure
```

Methods

AddFilters(String, Expression<Func<IInnerFilterAdapter, IFilterDescription>>[])

Add dynamic filters

Declaration

```
IQueryStructure AddFilters(string alias, params Expression<Func<IInnerFilterAdapter, IFilterDescription>>[] filters)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	alias	Source alias
System.Linq.Expressions.Expression<System.Func< IInnerFilterAdapter , IFilterDescription >>[]	filters	Dynamic filters

Returns

TYPE	DESCRIPTION
IQueryStructure	

Call(IDataScope, Variable[])

Dynamic procedure call with no result set

Declaration

```
IDataQuery Call(IDataScope dataScope, params Variable[] variables)
```

Parameters

TYPE	NAME	DESCRIPTION
IDataScope	dataScope	Data scope
Variable[]	variables	Data object variables

Returns

TYPE	DESCRIPTION
IDataQuery	Procedure call result

Remarks

Hint: Use variables as parameter values

Hint: To pass binary parameters use Base64 string value

Hint: To pass UDT parameters use Xml-serialized value of mapped type

See Also

[GetStructure\(String, String, DataStructureFlag, Variable\[\]\)](#)

[IQueryStructure](#)

[IQueryDescription<T>](#)

[IQueryStructureAdapter](#)

[Call<TResult>\(IDataScope, String, Variable\[\]\)](#)

Dynamic procedure call

Declaration

```

IDataQuery<TResult> Call<TResult>(IDataScope dataScope, string resultSetName, params Variable[] variables)
    where TResult : class

```

Parameters

TYPE	NAME	DESCRIPTION
IDataScope	dataScope	Data scope
System.String	resultSetNames	Base result set name
Variable[]	variables	Data object variables

Returns

TYPE	DESCRIPTION
IDataQuery<TResult>	Procedure call result

Type Parameters

NAME	DESCRIPTION
TResult	Dynamic query result type

Remarks

Hint: Use variables as parameter values

Hint: To pass binary parameters use Base64 string value

Hint: To pass UDT parameters use Xml-serialized value of mapped type

See Also

[GetStructure\(String, String, DataStructureFlag, Variable\[\]\)](#)

[IQueryStructure](#)

[IQueryDescription<T>](#)

[IQueryStructureAdapter](#)

Column<TResult>(String, Expression<Func<IQueryStructureAdapter, TResult>>)

Dynamic query hidden property mapping

Declaration

```
IQueryStructure Column<TResult>(string name, Expression<Func<IQueryStructureAdapter, TResult>> mapper)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Hidden property name
System.Linq.Expressions.Expression<System.Func< IQueryStructureAdapter , TResult>>	mapper	Property mapping expression

Returns

TYPE	DESCRIPTION
IQueryStructure	Query structure

Type Parameters

NAME	DESCRIPTION
TResult	

See Also

[GetStructure\(String, String, DataStructureFlag, Variable\[\]\)](#)

[IQueryStructure](#)

[IQueryStructureAdapter](#)

DataTable(String, String, Expression<Func<IInnerFilterAdapter, IFilterDescription>>[])

Dynamic query table mapping

Declaration

```
IQueryStructure DataTable(string name, string alias, params Expression<Func<IInnerFilterAdapter, IFilterDescription>>[] filters)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Table name
System.String	alias	Table alias
System.Linq.Expressions.Expression<System.Func< IInnerFilterAdapter , IFilterDescription >>[]	filters	Filters and links applied to table

Returns

TYPE	DESCRIPTION
IQueryStructure	Query structure

See Also

- [GetStructure\(String, String, DataStructureFlag, Variable\[\]\)](#)
- [IQueryStructure](#)
- [IFilterDescription](#)

Distinct()

Dynamic query distinct modifier

Declaration

```
IQueryStructure Distinct()
```

Returns

TYPE	DESCRIPTION
IQueryStructure	Query structure

Hint(String, String, HintType)

Dynamic query data source hint

Declaration

```
IQueryStructure Hint(string alias, string hint, HintType hintType = HintType.Select)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	alias	Data source alias
System.String	hint	Hint

TYPE	NAME	DESCRIPTION
HintType	hintType	Hint type

Returns

TYPE	DESCRIPTION
IQueryStructure	Query structure

See Also

[GetStructure\(String, String, DataStructureFlag, Variable\[\]\)](#)

[InnerView\(String, Type, Expression<Func<ISubqueryLinkAdapter, IFilterDescription>>\[\]\)](#)

Dynamic query inner view mapping

Declaration

```
IQueryStructure InnerView(string alias, Type innerViewType, params Expression<Func<ISubqueryLinkAdapter,
IFilterDescription>>[] filters)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	alias	Inner view alias
System.Type	innerViewType	Inner view statically mapped type
System.Linq.Expressions.Expression<System.Func< ISubqueryLinkAdapter , IFilterDescription >>[]	filters	Filters and links applied to inner view

Returns

TYPE	DESCRIPTION
IQueryStructure	Query structure

See Also

[GetStructure\(String, String, DataStructureFlag, Variable\[\]\)](#)

[IQueryStructure](#)

[IFilterDescription](#)

[InnerView<TInnerView>\(String, Expression<Func<ISubqueryLinkAdapter<TInnerView>, IFilterDescription>>\[\]\)](#)

Dynamic query inner view mapping

Declaration

```
IQueryStructure InnerView<TInnerView>(string alias, params Expression<Func<ISubqueryLinkAdapter<TInnerView>,
IFilterDescription>>[] filters)
    where TInnerView : class
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	alias	Inner view alias
System.Linq.Expressions.Expression<System.Func< ISubqueryLinkAdapter <TInnerView>, IFilterDescription >>[]	filters	Filters and links applied to inner view

Returns

TYPE	DESCRIPTION
IQueryStructure	Query structure

Type Parameters

NAME	DESCRIPTION
TInnerView	Inner view statically mapped type

See Also

- [GetStructure\(String, String, DataStructureFlag, Variable\[\]\)](#)
- [IQueryStructure](#)
- [IFilterDescription](#)

```
InnerView<TInnerView>(String, IQueryDescription<TInnerView>,
Expression<Func<ISubqueryLinkAdapter<TInnerView>, IFilterDescription>>[])
```

Dynamic query inner view mapping

Declaration

```
IQueryStructure InnerView<TInnerView>(string alias, IQueryDescription<TInnerView> sub, params
Expression<Func<ISubqueryLinkAdapter<TInnerView>, IFilterDescription>>[] filters)
    where TInnerView : class
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	alias	Inner view alias
IQueryDescription <TInnerView>	sub	Inner view query structure

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func< ISubqueryLinkAdapter <TInnerView>, IFilterDescription >>[]	filters	Filters and links applied to inner view

Returns

TYPE	DESCRIPTION
IQueryStructure	Query structure

Type Parameters

NAME	DESCRIPTION
TInnerView	

See Also

[GetStructure\(String, String, DataStructureFlag, Variable\[\]\)](#)
[IQueryStructure](#)
[IQueryDescription](#)
[IFilterDescription](#)

Limit(Int32)

Dynamic query limit count modifier

Declaration

IQueryStructure Limit(int count)

Parameters

TYPE	NAME	DESCRIPTION
System.Int32	count	Limit records count

Returns

TYPE	DESCRIPTION
IQueryStructure	Query structure

Map<TResult>(Expression<Func<[IRepositoryStructureAdapter](#)<TResult>, TResult>>, Expression<Func<[IRepositoryStructureAdapter](#)<TResult>, [IExternalLinkDefinition](#)>>>[])

Dynamic repository mapping

Declaration

IRepositoryDescription<TResult> Map<TResult>(Expression<Func< IRepositoryStructureAdapter <TResult>, TResult>> mapper, params Expression<Func< IRepositoryStructureAdapter <TResult>, IExternalLinkDefinition >>>[] externalLinks) where TResult : class, IDataObject , new()

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func< IRepositoryStructureAdapter <TResult>, TResult>>	mapper	Mapping expression
System.Linq.Expressions.Expression<System.Func< IRepositoryStructureAdapter <TResult>, IExternalLinkDefinition >>[]	externalLinks	External link descriptions

Returns

TYPE	DESCRIPTION
IRepositoryDescription <TResult>	Dynamic repository description

Type Parameters

NAME	DESCRIPTION
TResult	Dynamic repository type

See Also

- [GetStructure\(String, String, DataStructureFlag, Variable\[\]\)](#)
- [IQueryStructure](#)
- [IQueryDescription<T>](#)
- [IQueryStructureAdapter](#)

Parameter<T>(String, Int32, String, DbType, Expression<Func<IProcedureParameterAdapter, IProcedureParameterFlag>>[])

Dynamic query procedure parameter mapping

Declaration

```
IQueryStructure Parameter<T>(string alias, int order, string binding, DbType type, params Expression<Func<IProcedureParameterAdapter, IProcedureParameterFlag>>[] attributes)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	alias	Procedure alias
System.Int32	order	Parameter order
System.String	binding	Parameter value binding

TYPE	NAME	DESCRIPTION
System.Data.DbType	type	Parameter SQL data type
System.Linq.Expressions.Expression<System.Func< IProcedureParameterAdapter , IProcedureParameterFlag >> []	attributes	Parameter mapping attributes

Returns

TYPE	DESCRIPTION
IQueryStructure	Query structure

Type Parameters

NAME	DESCRIPTION
T	Parameter data type

Procedure(String, String, ProcedureType, Expression<Func<IInnerFilterAdapter, IFilterDescription>>[])

Dynamic query procedure mapping

Declaration

IQueryStructure Procedure(string alias, string name, ProcedureType type, params Expression<Func<IInnerFilterAdapter, IFilterDescription>>[] filters)

Parameters

TYPE	NAME	DESCRIPTION
System.String	alias	Procedure alias
System.String	name	Procedure name
ProcedureType	type	Procedure type
System.Linq.Expressions.Expression<System.Func< IInnerFilterAdapter , IFilterDescription >> []	filters	Filters and links applied to table

Returns

TYPE	DESCRIPTION
IQueryStructure	Query structure

See Also

[GetStructure\(String, String, DataStructureFlag, Variable\[\]\)](#)

[IQueryStructure](#)

[IFilterDescription](#)

Return<T>(String, String, Int32, Expression<Func<IQueryStructureAdapter, T>>)

Dynamic query procedure result set mapping

Declaration

```
IQueryStructure Return<T>(string alias, string name, int order, Expression<Func<IQueryStructureAdapter, T>>
mapper = null)
    where T : class
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	alias	Procedure alias
System.String	name	Result set name
System.Int32	order	Result set order
System.Linq.Expressions.Expression<System.Func< IQueryStructureAdapter , T>>	mapper	Result set mapper

Returns

TYPE	DESCRIPTION
IQueryStructure	Query structure

Type Parameters

NAME	DESCRIPTION
T	Parameter data type

Select<TResult>()

Dynamic query result mapping using statically mapped data

Declaration

```
IQueryDescription<TResult> Select<TResult>()
    where TResult : class, IDataObject, new()
```

Returns

TYPE	DESCRIPTION
IQueryDescription <TResult>	Dynamic query description

Type Parameters

NAME	DESCRIPTION
TResult	Dynamic query result type

See Also

[GetStructure\(String, String, DataStructureFlag, Variable\[\]\)](#)

[IQueryStructure](#)

[IQueryDescription](#)<T>

[IQueryStructureAdapter](#)

Select<TResult>(Expression<Func<IQueryStructureAdapter, TResult>>)

Dynamic query result mapping

Declaration

<pre>IQueryDescription<TResult> Select<TResult>(Expression<Func<IQueryStructureAdapter, TResult>> mapper) where TResult : class</pre>

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func< IQueryStructureAdapter , TResult>>	mapper	Mapping expression

Returns

TYPE	DESCRIPTION
IQueryDescription <TResult>	Dynamic query description

Type Parameters

NAME	DESCRIPTION
TResult	Dynamic query result type

See Also

[GetStructure\(String, String, DataStructureFlag, Variable\[\]\)](#)

[IQueryStructure](#)

[IQueryDescription](#)<T>

[IQueryStructureAdapter](#)

Skip(Int32)

Dynamic query skip count modifier

Declaration

```
IQueryStructure Skip(int count)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Int32	count	Skip records count

Returns

TYPE	DESCRIPTION
IQueryStructure	Query structure

Subquery(String, Type, String, DataGrouping, Expression<Func<ISubqueryLinkAdapter, IFilterDescription>>[])

Dynamic query subquery mapping

Declaration

```
IQueryStructure Subquery(string alias, Type subqueryType, string property, DataGrouping grouping = DataGrouping.None, params Expression<Func<ISubqueryLinkAdapter, IFilterDescription>>[] filters)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	alias	Subquery alias
System.Type	subqueryType	Subquery statically mapped type
System.String	property	Subquery selected property
DataGrouping	grouping	Subquery selected property aggregate function
System.Linq.Expressions.Expression<System.Func< ISubqueryLinkAdapter , IFilterDescription >>[]	filters	Filters and links applied to subquery

Returns

TYPE	DESCRIPTION
IQueryStructure	Query structure

See Also

[GetStructure\(String, String, DataStructureFlag, Variable\[\]\)](#)

[IQueryStructure](#)
[IFilterDescription](#)

`Subquery<TSubquery>(String, Expression<Func<TSubquery, Object>>, DataGrouping, Expression<Func<ISubqueryLinkAdapter<TSubquery>, IFilterDescription>>[])`

Dynamic query subquery mapping

Declaration

```
IQueryStructure Subquery<TSubquery>(string alias, Expression<Func<TSubquery, object>> property, DataGrouping grouping = DataGrouping.None, params Expression<Func<ISubqueryLinkAdapter<TSubquery>, IFilterDescription>>[] filters)
    where TSubquery : class
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	alias	Subquery alias
System.Linq.Expressions.Expression<System.Func<TSubquery, System.Object>>	property	Subquery selected property expression
DataGrouping	grouping	Subquery selected property aggregate function
System.Linq.Expressions.Expression<System.Func< ISubqueryLinkAdapter <TSubquery>, IFilterDescription >>[]	filters	Filters and links applied to subquery

Returns

TYPE	DESCRIPTION
IQueryStructure	Query structure

Type Parameters

NAME	DESCRIPTION
TSubquery	Subquery statically mapped type

See Also

[GetStructure\(String, String, DataStructureFlag, Variable\[\]\)](#)
[IQueryStructure](#)
[IFilterDescription](#)

`Subquery<TSubquery>(String, IQueryDescription<TSubquery>, String, DataGrouping, Expression<Func<ISubqueryLinkAdapter<TSubquery>, IFilterDescription>>[])`

Dynamic query subquery mapping

Declaration

```
IQueryStructure Subquery<TSubquery>(string alias, IQueryDescription<TSubquery> sub, string property,
DataGrouping grouping = DataGrouping.None, params Expression<Func<ISubqueryLinkAdapter<TSubquery>,
IFilterDescription>>[] filters)
    where TSubquery : class
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	alias	Subquery alias
IQueryDescription <TSubquery>	sub	Subquery query structure
System.String	property	Subquery selected property
DataGrouping	grouping	Subquery selected property aggregate function
System.Linq.Expressions.Expression<System.Func< ISubqueryLinkAdapter <TSubquery>, IFilterDescription >>[]	filters	Filters and links applied to subquery

Returns

TYPE	DESCRIPTION
IQueryStructure	Query structure

Type Parameters

NAME	DESCRIPTION
TSubquery	

See Also

[GetStructure\(String, String, DataStructureFlag, Variable\[\]\)](#)

[IQueryStructure](#)

[IQueryDescription](#)

[DataGrouping](#)

[IFilterDescription](#)

[TempTable\(String, String, Expression<Func<IInnerFilterAdapter, IFilterDescription>>\[\]\)](#)

Dynamic query temporary table mapping

Declaration

```
IQueryStructure TempTable(string name, string alias, params Expression<Func<IInnerFilterAdapter,
IFilterDescription>>[] filters)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Temporary table data variable name
System.String	alias	Temporary table data source alias
System.Linq.Expressions.Expression<System.Func< IInnerFilterAdapter , IFilterDescription >>[]	filters	Filters and links applied to temporary table data source

Returns

TYPE	DESCRIPTION
IQueryStructure	Query structure

Remarks

Data passed to query as variable with name equals name of temporary table data source

Variable value must to be System.Data.DataTable

See Also

[GetStructure\(String, String, DataStructureFlag, Variable\[\]\)](#)

[IQueryStructure](#)

[IFilterDescription](#)

With<TRoot>(String, IQueryDescription<TRoot>, Expression<Func<IQueryWithAdapter<TRoot>, IQueryWithFlag>>[])

Dynamic query Common-Table-Expression (WITH) subquery mapping

Declaration

<pre>IQueryStructure With<TRoot>(string alias, IQueryDescription<TRoot> sub, params Expression<Func<IQueryWithAdapter<TRoot>, IQueryWithFlag>>[] attributes) where TRoot : class</pre>

Parameters

TYPE	NAME	DESCRIPTION
System.String	alias	Common-Table-Expression (WITH) subquery alias
IQueryDescription <TRoot>	sub	Common-Table-Expression (WITH) initial subquery description
System.Linq.Expressions.Expression<System.Func< IQueryWithAdapter <TRoot>, IQueryWithFlag >>[]	attributes	Common-Table-Expression (WITH) attributes

Returns

TYPE	DESCRIPTION
IQueryStructure	Query structure

Type Parameters

NAME	DESCRIPTION
TRoot	Common-Table-Expression (WITH) initial subquery mapped type

XmlSource(String, String, String, Expression<Func<IInnerFilterAdapter, IFilterDescription>>[])

Dynamic query XML data source mapping

Declaration

```
IQueryStructure XmlSource(string name, string alias, string xmlRoot, params
Expression<Func<IInnerFilterAdapter, IFilterDescription>>[] filters)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	XML data variable name
System.String	alias	XML data source alias
System.String	xmlRoot	XML data root XPath expression
System.Linq.Expressions.Expression<System.Func< IInnerFilterAdapter , IFilterDescription >> []	filters	Filters and links applied to XML data source

Returns

TYPE	DESCRIPTION
IQueryStructure	Query structure

Remarks

XML data passed to query as variable with name equals name of XML data source

Variable value must to be a System.String

See Also

- [GetStructure\(String, String, DataStructureFlag, Variable\[\]\)](#)
- [IQueryStructure](#)
- [IFilterDescription](#)

Extension Methods

[SerializationExtensions.ToXml<T>\(T\)](#)

[CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)

[CollectionsExtensions.AsEnum<T>\(T\)](#)

[DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)

[Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)

[TypeExtensions.ChangeType<T>\(Object\)](#)

[TypeExtensions.ChangeType\(Object, Type\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)

[SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)

[QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

See Also

[GetStructure\(String, String, DataStructureFlag, Variable\[\]\)](#)

Interface IQueryStructureAdapter

Query structure mapping adapter interface

Namespace: [XData.Mapping.Dynamic](#)

Assembly: XData.docfx.dll

Syntax

```
public interface IQueryStructureAdapter
```

Methods

CastTo<T>()

Type cast modifier

Declaration

```
IQueryStructureAdapter CastTo<T>()
```

Returns

TYPE	DESCRIPTION
IQueryStructureAdapter	

Type Parameters

NAME	DESCRIPTION
T	Type to cast

Expr<TValue>(String, Expression<Func<IQueryStructureAdapter, TValue>>, DbType, Expression<Func<IExpressionPropertyAdapter, IStructureFlag>>[])

SQL expression mapping

Declaration

```
TValue Expr<TValue>(string fieldName, Expression<Func<IQueryStructureAdapter, TValue>> expr, DbType dbType, params Expression<Func<IExpressionPropertyAdapter, IStructureFlag>>[] attributes)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	fieldName	Expression field name
System.Linq.Expressions.Expression<System.Func< IQueryStructureAdapter , TValue>>	expr	Expression
System.Data.DbType	dbType	Expression database type

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func< IExpressionPropertyAdapter , IStructureFlag >>[]	attributes	Property mapping attributes

Returns

TYPE	DESCRIPTION
TValue	Expression value

Type Parameters

NAME	DESCRIPTION
TValue	Expression result type

Remarks

WARNING! Can be used in dynamic query mapping only!

See Also

[DataExpressionType](#)
System.Data.DbType
[Select<TResult>\(Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Expr<TValue>(String, DataExpressionType, String, DbType, Expression<Func<IExpressionPropertyAdapter, IStructureFlag>>[])

SQL expression mapping

Declaration

```
TValue Expr<TValue>(string fieldName, DataExpressionType type, string exprText, DbType dbType, params Expression<Func<IExpressionPropertyAdapter, IStructureFlag>>[] attributes)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	fieldName	Expression field name
DataExpressionType	type	Expression source type
System.String	exprText	Expression text
System.Data.DbType	dbType	Expression database type

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func< IExpressionPropertyAdapter , IStructureFlag >>[]	attributes	Property mapping attributes

Returns

TYPE	DESCRIPTION
TValue	Expression value

Type Parameters

NAME	DESCRIPTION
TValue	Expression result type

Remarks

WARNING! Can be used in dynamic query mapping only!

See Also

[DataExpressionType](#)

System.Data.DbType

[Select<TResult>\(Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

[Field<TValue>\(Expression<Func<IPropertyMappingAdapter, IStructureFlag>>\[\]\)](#)

Field mapping

Declaration

TValue Field<TValue>(params Expression<Func<IPropertyMappingAdapter, IStructureFlag>>[] attributes)

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func< IPropertyMappingAdapter , IStructureFlag >>[]	attributes	Property mapping attributes

Returns

TYPE	DESCRIPTION
TValue	Field value

Type Parameters

NAME	DESCRIPTION

NAME	DESCRIPTION
TValue	Field type

Remarks

WARNING! Can be used in dynamic query mapping only!

See Also

[Select<TResult>\(Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

[Field<TValue>\(String, String, DbType, Expression<Func<IExpressionPropertyAdapter, IStructureFlag>>\[\]\)](#)

Field mapping

Declaration

```
TValue Field<TValue>(string sourceAlias, string fieldName, DbType dbType, params
Expression<Func<IExpressionPropertyAdapter, IStructureFlag>>[] attributes)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	sourceAlias	Source alias
System.String	fieldName	Field name
System.Data.DbType	dbType	Field database type
System.Linq.Expressions.Expression<System.Func< IExpressionPropertyAdapter , IStructureFlag >>[]	attributes	Property mapping attributes

Returns

TYPE	DESCRIPTION
TValue	Field value

Type Parameters

NAME	DESCRIPTION
TValue	Field type

See Also

[Select<TResult>\(Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

[Field<TValue>\(String, String, Expression<Func<IPropertyMappingAdapter, IStructureFlag>>\[\]\)](#)

Field mapping

Declaration

```
TValue Field<TValue>(string sourceAlias, string fieldName, params Expression<Func<IPropertyMappingAdapter, IStructureFlag>>[] attributes)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	sourceAlias	Source alias
System.String	fieldName	Field name
System.Linq.Expressions.Expression<System.Func< IPropertyMappingAdapter , IStructureFlag >>[]	attributes	Property mapping attributes

Returns

TYPE	DESCRIPTION
TValue	Field value

Type Parameters

NAME	DESCRIPTION
TValue	Field type

Remarks

WARNING! Can be used in dynamic query mapping only!

See Also

[Select<TResult>\(Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

[Ref<TValue>\(String, String, Expression<Func<IPropertyMappingAdapter, IStructureFlag>>\[\]\)](#)

Inner view property reference mapping

Declaration

```
TValue Ref<TValue>(string sourceAlias, string property, params Expression<Func<IPropertyMappingAdapter, IStructureFlag>>[] attributes)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	sourceAlias	Referenced property subquery alias

TYPE	NAME	DESCRIPTION
System.String	property	Referenced property name
System.Linq.Expressions.Expression<System.Func< IPropertyMappingAdapter , IStructureFlag >>[]	attributes	Property mapping attributes

Returns

TYPE	DESCRIPTION
TValue	Referenced property value

Type Parameters

NAME	DESCRIPTION
TValue	Referenced property type

Remarks

WARNING! Can be used in dynamic query mapping only!

See Also

[Select<TResult>\(Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Ref<TValue, TInnerView>(String, Expression<Func<TInnerView, Object>>, Expression<Func<IPropertyMappingAdapter, IStructureFlag>>[] attributes)

Inner view property reference mapping

Declaration

<pre>TValue Ref<TValue, TInnerView>(string sourceAlias, Expression<Func<TInnerView, object>> property, params Expression<Func<IPropertyMappingAdapter, IStructureFlag>>[] attributes) where TInnerView : class</pre>

Parameters

TYPE	NAME	DESCRIPTION
System.String	sourceAlias	Referenced property subquery alias
System.Linq.Expressions.Expression<System.Func<TInnerView, System.Object>>	property	Referenced property name
System.Linq.Expressions.Expression<System.Func< IPropertyMappingAdapter , IStructureFlag >>[]	attributes	Property mapping attributes

Returns

TYPE	DESCRIPTION
TValue	Referenced property value

Type Parameters

NAME	DESCRIPTION
TValue	Referenced property type
TInnerView	Referenced inner view type

Remarks

WARNING! Can be used in dynamic query mapping only!

See Also

[Select<TResult>\(Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

[XmlField<TValue>\(String, String, String, DbType, Expression<Func<IExpressionPropertyAdapter, IStructureFlag>>\[\]\)](#)

XML field mapping

Declaration

TValue XmlField<TValue>(string sourceAlias, string fieldName, string xPath, DbType dbType, params Expression<Func<IExpressionPropertyAdapter, IStructureFlag>>[] attributes)

Parameters

TYPE	NAME	DESCRIPTION
System.String	sourceAlias	XML data source alias
System.String	fieldName	XML field name
System.String	xPath	XML field XPath expression
System.Data.DbType	dbType	XML field database type
System.Linq.Expressions.Expression<System.Func< IExpressionPropertyAdapter , IStructureFlag >>[]	attributes	Property mapping attributes

Returns

TYPE	DESCRIPTION

TYPE	DESCRIPTION
TValue	Field value

Type Parameters

NAME	DESCRIPTION
TValue	XML Field type

Remarks

WARNING! Can be used in dynamic query mapping for [XmlSource\(String, String, String, Expression<Func<IInnerFilterAdapter, IFilterDescription>>\[\]\)](#) data sources only!

See Also

[Select<TResult>\(Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Extension Methods

[SerializationExtensions.ToXml<T>\(T\)](#)

[CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)

[CollectionsExtensions.AsEnum<T>\(T\)](#)

[DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)

[Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)

[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)

[TypeExtensions.ChangeType<T>\(Object\)](#)

[TypeExtensions.ChangeType\(Object, Type\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)

[TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)

[SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)

[QueryDescriptionExtensions.Exists\(IQueryStructureAdapter, String\)](#)

[QueryDescriptionExtensions.Case<T, TResult>\(IQueryStructureAdapter, Expression<Func<IQueryStructureAdapter, T>>, Expression<Func<IQueryStructureAdapter, TResult>>, TaggedExpression<T, TResult>\[\]\)](#)

[QueryDescriptionExtensions.DateDiff<T>\(IQueryStructureAdapter, Expression<Func<IQueryStructureAdapter, DateTime>>, Expression<Func<IQueryStructureAdapter, DateTime>>, DateDiffKind\)](#)

[QueryDescriptionExtensions.HasFlag<T>\(IQueryStructureAdapter, Expression<Func<IQueryStructureAdapter, T>>, Expression<Func<IQueryStructureAdapter, T>>, Expression<Func<IQueryStructureAdapter, T>>, Expression<Func<IQueryStructureAdapter, T>>\)](#)

[QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

[QueryDescriptionExtensions.SqlFn<TResult>\(IQueryStructureAdapter, String, Expression<Func<IQueryStructureAdapter, Object>>\[\]\)](#)

[QueryDescriptionExtensions.SqlParam<TResult>\(IQueryStructureAdapter, String, DbType, Expression<Func<IProcedureParameterAdapter, IProcedureParameterFlag>>\[\]\)](#)

[QueryDescriptionExtensions.SqlExpr<TResult>\(IQueryStructureAdapter, String, Expression<Func<IQueryStructureAdapter, Object>>\[\]\)](#)

[QueryDescriptionExtensions.GetVariable<TResult>\(IQueryStructureAdapter, String\)](#)

See Also

[Select<TResult>\(Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Interface IQueryWithAdapter<TRoot>

Common-Table-Expression (WITH) mapping attributes

Namespace: [XData.Mapping.Dynamic](#)

Assembly: XData.docfx.dll

Syntax

```
public interface IQueryWithAdapter<TRoot>
    where TRoot : class
```

Type Parameters

NAME	DESCRIPTION
TRoot	Common-Table-Expression (WITH) initial subquery mapped type

Methods

Properties(Expression<Func<TRoot, Object>>[])

Common-Table-Expression (WITH) properties order

Declaration

```
IQueryWithFlag Properties(params Expression<Func<TRoot, object>>[] properties)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func<TRoot, System.Object>>[]	properties	Common-Table-Expression (WITH) properties expressions

Returns

TYPE	DESCRIPTION
IQueryWithFlag	Common-Table-Expression (WITH) flag

RecursiveType(WithRecursiveType)

Common-Table-Expression (WITH) recursive link type attribute

Declaration

```
IQueryWithFlag RecursiveType(WithRecursiveType withType)
```

Parameters

TYPE	NAME	DESCRIPTION
WithRecursiveType	withType	Common-Table-Expression (WITH) recursive link type

Returns

TYPE	DESCRIPTION
IQueryWithFlag	Common-Table-Expression (WITH) flag

WithRecursive<TSubquery>(String, IQueryDescription<TSubquery>, Expression<Func<ISubqueryLinkAdapter<TRoot, TSubquery>, IFilterDescription>>[])

Common-Table-Expression (WITH) recursive subqueries

Declaration

```
IQueryWithFlag WithRecursive<TSubquery>(string alias, IQueryDescription<TSubquery> query, params
Expression<Func<ISubqueryLinkAdapter<TRoot, TSubquery>, IFilterDescription>>[] filters)
    where TSubquery : class
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	alias	Common-Table-Expression (WITH) recursive subquery alias
IQueryDescription <TSubquery>	query	Common-Table-Expression (WITH) recursive subquery description
System.Linq.Expressions.Expression<System.Func< ISubqueryLinkAdapter <TRoot, TSubquery>, IFilterDescription >>[]	filters	Common-Table-Expression (WITH) recursive subquery links

Returns

TYPE	DESCRIPTION
IQueryWithFlag	Common-Table-Expression (WITH) flag

Type Parameters

NAME	DESCRIPTION
TSubquery	Common-Table-Expression (WITH) recursive subquery type

Extension Methods

- [SerializationExtensions.ToXml](#)<T>(T)
- [CollectionsExtensions.SetValue](#)<TKey, T>(TKey, T)
- [CollectionsExtensions.AsEnum](#)<T>(T)
- [DataObjectQueryableExtensions.CompareVariable](#)(Object, FilterOperation, String)
- [Processing.Do](#)<T>(T, Action<IProcess<T>>[])
- [TypeExtensions.TryConvert](#)<T, TResult>(T, Func<T, TResult>)
- [TypeExtensions.TryConvert](#)<T, TResult>(T, Func<T, TResult>, TResult)
- [TypeExtensions.ChangeType](#)<T>(Object)

TypeExtensions.ChangeType(Object, Type)

TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)

TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)

SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)

QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Interface IQueryWithFlag

Common-Table-Expression (WITH) flag

Namespace: [XData.Mapping.Dynamic](#)

Assembly: [XData.docfx.dll](#)

Syntax

```
public interface IQueryWithFlag
```

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Interface IRepositoryDescription<T>

Dynamic repository structure description interface

Inherited Members

[IQueryDescription.Type](#)

Namespace: [XData.Mapping.Dynamic](#)

Assembly: XData.docfx.dll

Syntax

```
public interface IRepositoryDescription<out T> : IQueryDescription where T : class, ISqlObject, new()
```

Type Parameters

NAME	DESCRIPTION
T	

Methods

SetBaseTable(String)

Set base table alias to repository dynamic mapping

Declaration

```
IRepositoryDescription<T> SetBaseTable(string baseTableAlias)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	baseTableAlias	Base table alias

Returns

TYPE	DESCRIPTION
IRepositoryDescription<T>	Dynamic repository description

SetContext(String)

Set context to repository dynamic mapping

Declaration

```
IRepositoryDescription<T> SetContext(string context)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	context	Database context name

Returns

TYPE	DESCRIPTION
IRepositoryDescription<T>	Dynamic repository description

SetFlags(DataStructureFlag)

Set flags to repository dynamic mapping

Declaration

<code>IRepositoryDescription<T> SetFlags(DataStructureFlag flags)</code>

Parameters

TYPE	NAME	DESCRIPTION
DataStructureFlag	flags	Data repository flags

Returns

TYPE	DESCRIPTION
IRepositoryDescription<T>	Dynamic repository description

SetLogicAssemblies(String[])

Set server data logic assembly name to repository dynamic mapping

Declaration

<code>IRepositoryDescription<T> SetLogicAssemblies(params string[] logicAssemblyNames)</code>

Parameters

TYPE	NAME	DESCRIPTION
System.String[]	logicAssemblyNames	Server data logic assembly names

Returns

TYPE	DESCRIPTION
IRepositoryDescription<T>	Dynamic repository description

Remarks

NOTE: Used when no IoC Container available. Otherwise register logic modules through container

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)

Processing.Do<T>(T, Action<IProcess<T>>[])

TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>)

TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>, TResult)

TypeExtensions.ChangeType<T>(Object)

TypeExtensions.ChangeType(Object, Type)

TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)

TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)

SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)

QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Interface IRepositoryExpressionMappingAdapter

Repository SQL expression property mapping adapter

Inherited Members

[IExpressionPropertyAdapter.Size\(Int32\)](#)
[IExpressionPropertyAdapter.Size\(Int32, Int32\)](#)
[IExpressionPropertyAdapter.NativeSqlType\(String\)](#)
[IExpressionPropertyAdapter.Exists\(\)](#)
[IExpressionPropertyAdapter.Exists\(ExistsOperation\)](#)
[IPropertyMappingAdapter.Key\(\)](#)
[IPropertyMappingAdapter.Group\(DataGrouping\)](#)
[IPropertyMappingAdapter.Group\(Int32\)](#)
[IPropertyMappingAdapter.Hidden\(\)](#)
[IPropertyMappingAdapter.OrderBy\(Int32, Boolean\)](#)
[IPropertyMappingAdapter.OrderBy\(Int32\)](#)

Namespace: [XData.Mapping.Dynamic](#)

Assembly: [XData.docfx.dll](#)

Syntax

```
public interface IRepositoryExpressionMappingAdapter : IExpressionPropertyAdapter, IPropertyMappingAdapter
```

Extension Methods

[SerializationExtensions.ToXml<T>\(T\)](#)
[CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
[CollectionsExtensions.AsEnum<T>\(T\)](#)
[DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
[Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
[TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
[TypeExtensions.ChangeType<T>\(Object\)](#)
[TypeExtensions.ChangeType\(Object, Type\)](#)
[TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
[TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
[TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
[TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
[SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
[QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Interface IRepositoryPropertyMappingAdapter<T, TValue>

Repository property mapping adapter

Inherited Members

- [IRepositoryReadOnlyPropertyMappingAdapter.ConcurrencyToken\(\)](#)
- [IRepositoryReadOnlyPropertyMappingAdapter.OuterFlag\(\)](#)
- [IPropertyMappingAdapter.Key\(\)](#)
- [IPropertyMappingAdapter.Group\(DataGrouping\)](#)
- [IPropertyMappingAdapter.Group\(Int32\)](#)
- [IPropertyMappingAdapter.Hidden\(\)](#)
- [IPropertyMappingAdapter.OrderBy\(Int32, Boolean\)](#)
- [IPropertyMappingAdapter.OrderBy\(Int32\)](#)

Namespace: [XData.Mapping.Dynamic](#)

Assembly: [XData.docfx.dll](#)

Syntax

```
public interface IRepositoryPropertyMappingAdapter<T, TValue> : IRepositoryReadOnlyPropertyMappingAdapter,
IPropertyMappingAdapter where T : class, ISqlObject, new()
```

Type Parameters

NAME	DESCRIPTION
T	
TValue	

Methods

Default(DefaultType)

Default value description

Declaration

```
IStructureFlag Default(DefaultType source)
```

Parameters

TYPE	NAME	DESCRIPTION
DefaultType	source	Default value algorithm

Returns

TYPE	DESCRIPTION
IStructureFlag	Property default value description

Default(DefaultType, TValue)

Default value description

Declaration

```
IStructureFlag Default(DefaultType source, TValue value)
```

Parameters

TYPE	NAME	DESCRIPTION
DefaultType	source	Default value algorithm
TValue	value	Default value

Returns

TYPE	DESCRIPTION
IStructureFlag	Property default value description

Default(DefaultType, TValue, DefaultFeature)

Default value description

Declaration

```
IStructureFlag Default(DefaultType source, TValue value, DefaultFeature features)
```

Parameters

TYPE	NAME	DESCRIPTION
DefaultType	source	Default value algorithm
TValue	value	Default value
DefaultFeature	features	Default extended features

Returns

TYPE	DESCRIPTION
IStructureFlag	Property default value description

Default(DefaultType, DefaultFeature)

Default value description

Declaration

```
IStructureFlag Default(DefaultType source, DefaultFeature features)
```

Parameters

TYPE	NAME	DESCRIPTION
DefaultType	source	Default value algorithm
DefaultFeature	features	Default extended features

Returns

TYPE	DESCRIPTION
IStructureFlag	Property default value description

UpdateWith(Expression<Func<IQueryStructureAdapter, TValue>>)

Update value source description

Declaration

```
IStructureFlag UpdateWith(Expression<Func<IQueryStructureAdapter, TValue>> expr)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func< IQueryStructureAdapter , TValue>>	expr	Update value expression definition

Returns

TYPE	DESCRIPTION
IStructureFlag	Update value source description

UpdateWith(String, Type, String, DataGrouping, Expression<Func<ISubqueryLinkAdapter, IStructureFlag>>[])

Update value source description

Declaration

```
IStructureFlag UpdateWith(string alias, Type subqueryType, string property, DataGrouping grouping = DataGrouping.None, params Expression<Func<ISubqueryLinkAdapter, IStructureFlag>>[] attributes)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	alias	Subquery alias
System.Type	subqueryType	Subquery type

TYPE	NAME	DESCRIPTION
System.String	property	Subquery selected property
DataGrouping	grouping	Subquery selected property aggregate function
System.Linq.Expressions.Expression<System.Func<ISubqueryLinkAdapter, IStructureFlag>>[]	attributes	Subquery links

Returns

TYPE	DESCRIPTION
IStructureFlag	Update value source description

UpdateWith(DataExpressionType, String)

Update value source description

Declaration

```
IStructureFlag UpdateWith(DataExpressionType type, string exprText)
```

Parameters

TYPE	NAME	DESCRIPTION
DataExpressionType	type	Update value source expression type
System.String	exprText	Expression text alias

Returns

TYPE	DESCRIPTION
IStructureFlag	Update value source description

UpdateWith<TDObj>(String, IQueryDescription<TDObj>, Expression<Func<TDObj, TValue>>, DataGrouping, Expression<Func<ISubqueryLinkAdapter<T, TDObj>, IStructureFlag>>[])

Update value source description

Declaration

```
IStructureFlag UpdateWith<TDObj>(string alias, IQueryDescription<TDObj> sub, Expression<Func<TDObj, TValue>>
property, DataGrouping grouping = DataGrouping.None, params Expression<Func<ISubqueryLinkAdapter<T, TDObj>,
IStructureFlag>>[] attributes)
    where TDObj : class
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	alias	Subquery alias
IQueryDescription <TObj>	sub	Subquery definition
System.Linq.Expressions.Expression<System.Func<TObj, TValue> >	property	Subquery selected property expression
DataGrouping	grouping	Subquery selected property aggregate function
System.Linq.Expressions.Expression<System.Func< ISubqueryLinkAdapter <T, TObj>, IStructureFlag > > []	attributes	Subquery links

Returns

TYPE	DESCRIPTION
IStructureFlag	Update value source description

Type Parameters

NAME	DESCRIPTION
TObj	

UpdateWith<TObj>(String, IQueryDescription<TObj>, String, DataGrouping, Expression<Func<ISubqueryLinkAdapter<T, TObj>, IStructureFlag>>[])

Update value source description

Declaration

```
IStructureFlag UpdateWith<TObj>(string alias, IQueryDescription<TObj> sub, string property, DataGrouping
grouping = DataGrouping.None, params Expression<Func<ISubqueryLinkAdapter<T, TObj>, IStructureFlag>>[]
attributes)
    where TObj : class
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	alias	Subquery alias
IQueryDescription <TObj>	sub	Subquery definition

TYPE	NAME	DESCRIPTION
System.String	property	Subquery selected property
DataGrouping	grouping	Subquery selected property aggregate function
System.Linq.Expressions.Expression<System.Func<ISubqueryLinkAdapter<T, TDObj>, IStructureFlag>>[]	attributes	Subquery links

Returns

TYPE	DESCRIPTION
IStructureFlag	Update value source description

Type Parameters

NAME	DESCRIPTION
TDObj	

Extension Methods

- SerializationExtensions.ToXml<T>(T)
- CollectionsExtensions.SetValue<TKey, T>(TKey, T)
- CollectionsExtensions.AsEnum<T>(T)
- DataObjectQueryableExtensions.CompareVariable(Object, FilterOperation, String)
- Processing.Do<T>(T, Action<IProcess<T>>[])
- TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>)
- TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>, TResult)
- TypeExtensions.ChangeType<T>(Object)
- TypeExtensions.ChangeType(Object, Type)
- TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)
- TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)
- TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)
- TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)
- SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)
- QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Interface IRepositoryReadOnlyPropertyMappingAdapter

Repository read only property mapping adapter

Inherited Members

- [IPropertyMappingAdapter.Key\(\)](#)
- [IPropertyMappingAdapter.Group\(DataGrouping\)](#)
- [IPropertyMappingAdapter.Group\(Int32\)](#)
- [IPropertyMappingAdapter.Hidden\(\)](#)
- [IPropertyMappingAdapter.OrderBy\(Int32, Boolean\)](#)
- [IPropertyMappingAdapter.OrderBy\(Int32\)](#)

Namespace: `XData.Mapping.Dynamic`

Assembly: `XData.docfx.dll`

Syntax

```
public interface IRepositoryReadOnlyPropertyMappingAdapter : IPropertyMappingAdapter
```

Methods

ConcurrencyToken()

Property is concurrency token flag

Declaration

```
IStructureFlag ConcurrencyToken()
```

Returns

TYPE	DESCRIPTION
IStructureFlag	

OuterFlag()

Property is outer join flag

Declaration

```
IStructureFlag OuterFlag()
```

Returns

TYPE	DESCRIPTION
IStructureFlag	

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)

TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)

TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)

SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)

QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Interface IRepositoryStructure<T>

Repository structure descriptor

Namespace: [XData.Mapping.Dynamic](#)

Assembly: XData.docfx.dll

Syntax

```
public interface IRepositoryStructure<T>
    where T : class, ISqlObject, new()
```

Type Parameters

NAME	DESCRIPTION
T	Data object type

Methods

Column<TResult>(String, Expression<Func<IRepositoryStructureAdapter<T>, TResult>>)

Dynamic repository hidden property mapping

Declaration

```
IRepositoryStructure<T> Column<TResult>(string name, Expression<Func<IRepositoryStructureAdapter<T>, TResult>> mapper)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Hidden property name
System.Linq.Expressions.Expression<System.Func< IRepositoryStructureAdapter <T>, TResult>>	mapper	Property mapping expression

Returns

TYPE	DESCRIPTION
IRepositoryStructure <T>	Repository structure

Type Parameters

NAME	DESCRIPTION
TResult	Hidden property type

See Also

[GetStructure\(String, String, DataStructureFlag, Variable\[\]\)](#)

[IQueryStructure](#)

[IQueryStructureAdapter](#)

DataTable(String, String, Expression<Func<IInnerFilterAdapter, IFilterDescription>>[])

Dynamic repository table mapping

Declaration

```
IRepositoryStructure<T> DataTable(string name, string alias, params Expression<Func<IInnerFilterAdapter, IFilterDescription>>[] filters)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Table name
System.String	alias	Table alias
System.Linq.Expressions.Expression<System.Func< IInnerFilterAdapter , IFilterDescription >>[]	filters	Filters and links applied to table

Returns

TYPE	DESCRIPTION
IRepositoryStructure <T>	Query structure

See Also

- [GetStructure\(String, String, DataStructureFlag, Variable\[\]\)](#)
- [IQueryStructure](#)
- [IFilterDescription](#)

DataTable(String, String, String, Expression<Func<IInnerFilterAdapter, IFilterDescription>>[])

Dynamic repository table mapping

Declaration

```
IRepositoryStructure<T> DataTable(string name, string alias, string parent = null, params Expression<Func<IInnerFilterAdapter, IFilterDescription>>[] filters)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Table name
System.String	alias	Table alias
System.String	parent	Parent table alias

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func< IInnerFilterAdapter , IFilterDescription >>[]	filters	Filters and links applied to table

Returns

TYPE	DESCRIPTION
IRepositoryStructure <T>	Repository structure

See Also

[GetStructure\(String, String, DataStructureFlag, Variable\[\]\)](#)

[IQueryStructure](#)

[IFilterDescription](#)

Distinct()

Dynamic repository distinct modificator

Declaration

<code>IRepositoryStructure<T> Distinct()</code>

Returns

TYPE	DESCRIPTION
IRepositoryStructure <T>	Query structure

Hint(String, String, HintType)

Dynamic repository data source hint

Declaration

<code>IRepositoryStructure<T> Hint(string alias, string hint, HintType hintType = HintType.Select)</code>

Parameters

TYPE	NAME	DESCRIPTION
System.String	alias	Data source alias
System.String	hint	Hint
HintType	hintType	Hint type

Returns

TYPE	DESCRIPTION
IRepositoryStructure<T>	Repository structure

See Also

[GetStructure\(String, String, DataStructureFlag, Variable\[\]\)](#)

[InnerView\(String, Type, Expression<Func<ISubqueryLinkAdapter<T>, IFilterDescription>>\[\]\)](#)

Dynamic repository inner view mapping

Declaration

```
IRepositoryStructure<T> InnerView(string alias, Type innerViewType, params
Expression<Func<ISubqueryLinkAdapter<T>, IFilterDescription>>[] filters)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	alias	Inner view alias
System.Type	innerViewType	Inner view statically mapped type
System.Linq.Expressions.Expression<System.Func< ISubqueryLinkAdapter<T> , IFilterDescription >>[]	filters	Filters and links applied to inner view

Returns

TYPE	DESCRIPTION
IRepositoryStructure<T>	Repository structure

See Also

[GetStructure\(String, String, DataStructureFlag, Variable\[\]\)](#)

[IQueryStructure](#)

[IFilterDescription](#)

[InnerView<TInnerView>\(String, Expression<Func<ISubqueryLinkAdapter<T, TInnerView>, IFilterDescription>>\[\]\)](#)

Dynamic repository inner view mapping

Declaration

```
IRepositoryStructure<T> InnerView<TInnerView>(string alias, params Expression<Func<ISubqueryLinkAdapter<T,
TInnerView>, IFilterDescription>>[] filters)
    where TInnerView : class
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	alias	Inner view alias
System.Linq.Expressions.Expression<System.Func< ISubqueryLinkAdapter <T, TInnerView>, IFilterDescription >>[]	filters	Filters and links applied to inner view

Returns

TYPE	DESCRIPTION
IRepositoryStructure <T>	Query structure

Type Parameters

NAME	DESCRIPTION
TInnerView	Inner view statically mapped type

See Also

[GetStructure\(String, String, DataStructureFlag, Variable\[\]\)](#)
[IQueryStructure](#)
[IFilterDescription](#)

InnerView<TInnerView>(String, IQueryDescription<TInnerView>, Expression<Func<[ISubqueryLinkAdapter](#)<T, TInnerView>, [IFilterDescription](#)>>[])

Dynamic repository inner view mapping

Declaration

```

IRepositoryStructure<T> InnerView<TInnerView>(string alias, IQueryDescription<TInnerView> sub, params
Expression<Func<ISubqueryLinkAdapter<T, TInnerView>, IFilterDescription>>[] filters)
    where TInnerView : class

```

Parameters

TYPE	NAME	DESCRIPTION
System.String	alias	Inner view alias
IQueryDescription <TInnerView>	sub	Inner view query structure
System.Linq.Expressions.Expression<System.Func< ISubqueryLinkAdapter <T, TInnerView>, IFilterDescription >>[]	filters	Filters and links applied to inner view

Returns

TYPE	DESCRIPTION
IRepositoryStructure<T>	Repository structure

Type Parameters

NAME	DESCRIPTION
TInnerView	

See Also

[GetStructure\(String, String, DataStructureFlag, Variable\[\]\)](#)

[IQueryStructure](#)

[IQueryDescription](#)

[IFilterDescription](#)

InnerView<TInnerView>(String, Variable[], Expression<Func<ISubqueryLinkAdapter<T, TInnerView>, IFilterDescription>>[])

Dynamic repository inner view mapping

Declaration

```
IRepositoryStructure<T> InnerView<TInnerView>(string alias, Variable[] variables, params
Expression<Func<ISubqueryLinkAdapter<T, TInnerView>, IFilterDescription>>[] filters)
    where TInnerView : class
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	alias	Inner view alias
Variable[]	variables	Inner view variables
System.Linq.Expressions.Expression<System.Func< ISubqueryLinkAdapter <T, TInnerView>, IFilterDescription >>[]	filters	Filters and links applied to inner view

Returns

TYPE	DESCRIPTION
IRepositoryStructure<T>	Query structure

Type Parameters

NAME	DESCRIPTION
TInnerView	Inner view statically mapped type

See Also

[GetStructure\(String, String, DataStructureFlag, Variable\[\]\)](#)
[IQueryStructure](#)
[Variable](#)
[IFilterDescription](#)

Limit(Int32)

Dynamic repository limit count modifier

Declaration

```
IRepositoryStructure<T> Limit(int count)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Int32	count	Limit records count

Returns

TYPE	DESCRIPTION
IRepositoryStructure<T>	Query structure

Map(Expression<Func<IExternalLinkDefinitionAdapter<T>, IExternalLinkDefinition>>[])

Dynamic repository properties mapping

Declaration

```
IRepositoryDescription<T> Map(params Expression<Func<IExternalLinkDefinitionAdapter<T>, IExternalLinkDefinition>>[] externalLinks)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func< IExternalLinkDefinitionAdapter<T> , IExternalLinkDefinition >>[]	externalLinks	External link descriptions

Returns

TYPE	DESCRIPTION
IRepositoryDescription<T>	Dynamic repository description

See Also

[GetStructure\(String, String, DataStructureFlag, Variable\[\]\)](#)
[IQueryStructure](#)
[IQueryDescription<T>](#)
[IQueryStructureAdapter](#)

Map(Expression<Func<IRepositoryStructureAdapter<T>, T>>, Expression<Func<IExternalLinkDefinitionAdapter<T>, IExternalLinkDefinition>>[])

Dynamic repository properties mapping

Declaration

```
 IRepositoryDescription<T> Map(Expression<Func< IRepositoryStructureAdapter<T>, T>> mapper, params
 Expression<Func< IExternalLinkDefinitionAdapter<T>, IExternalLinkDefinition>>[] externalLinks)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func< IRepositoryStructureAdapter <T>, T>>	mapper	Mapping expression
System.Linq.Expressions.Expression<System.Func< IExternalLinkDefinitionAdapter <T>, IExternalLinkDefinition >>[]	externalLinks	External link descriptions

Returns

TYPE	DESCRIPTION
IRepositoryDescription <T>	Dynamic repository description

See Also

- [GetStructure\(String, String, DataStructureFlag, Variable\[\]\)](#)
- [IQueryStructure](#)
- [IQueryDescription](#)<T>
- [IQueryStructureAdapter](#)

[Parameter<TResult>\(String, Int32, String, DbType, Expression<Func<IProcedureParameterAdapter, IProcedureParameterFlag>>\[\]\)](#)

Dynamic repository procedure parameter mapping

Declaration

```
 IRepositoryStructure<T> Parameter<TResult>(string alias, int order, string binding, DbType type, params
 Expression<Func<IProcedureParameterAdapter, IProcedureParameterFlag>>[] attributes)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	alias	Procedure alias
System.Int32	order	Parameter order
System.String	binding	Parameter value binding

TYPE	NAME	DESCRIPTION
System.Data.DbType	type	Parameter SQL data type
System.Linq.Expressions.Expression<System.Func< IProcedureParameterAdapter , IProcedureParameterFlag >> []	attributes	Parameter attributes

Returns

TYPE	DESCRIPTION
IRepositoryStructure <T>	Repository structure

Type Parameters

NAME	DESCRIPTION
TResult	Parameter data type

Procedure(String, String, ProcedureType, Expression<Func<IInnerFilterAdapter, IFilterDescription>>>[])

Dynamic repository procedure mapping

Declaration

<pre>IRepositoryStructure<T> Procedure(string alias, string name, ProcedureType type, params Expression<Func<IInnerFilterAdapter, IFilterDescription>>[] filters)</pre>

Parameters

TYPE	NAME	DESCRIPTION
System.String	alias	Procedure alias
System.String	name	Procedure name
ProcedureType	type	Procedure type
System.Linq.Expressions.Expression<System.Func< IInnerFilterAdapter , IFilterDescription >> []	filters	Filters and links applied to table

Returns

TYPE	DESCRIPTION
IRepositoryStructure <T>	Repository structure

See Also

GetStructure(String, String, DataStructureFlag, Variable[])
IQueryStructure
IFilterDescription

ReadOnlyProperty<TResult>(Expression<Func<T, TResult>>, Expression<Func<IRepositoryStructureAdapter<T>, TResult>>)

Dynamic repository read only properties mapping

Declaration

IRepositoryStructure<T> ReadOnlyProperty<TResult>(Expression<Func<T, TResult>> property, Expression<Func<IRepositoryStructureAdapter<T>, TResult>> mapper)

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func<T, TResult>>	property	Property name expression
System.Linq.Expressions.Expression<System.Func<IRepositoryStructureAdapter<T>, TResult>>	mapper	Property mapping

Returns

TYPE	DESCRIPTION
IRepositoryStructure<T>	Repository structure

Type Parameters

NAME	DESCRIPTION
TResult	Property type

Return(String)

Dynamic repository procedure default result set mapping

Declaration

IRepositoryStructure<T> Return(string alias)

Parameters

TYPE	NAME	DESCRIPTION
System.String	alias	Procedure alias

Returns

TYPE	DESCRIPTION

TYPE	DESCRIPTION
IRepositoryStructure<T>	Repository structure

Return<TResult>(String, String, Int32, Expression<Func<IQueryStructureAdapter, T>>)

Dynamic repository procedure result set mapping

Declaration

```
IRepositoryStructure<T> Return<TResult>(string alias, string name, int order,
Expression<Func<IQueryStructureAdapter, T>> mapper = null)
    where TResult : class
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	alias	Procedure alias
System.String	name	Result set name
System.Int32	order	Result set order
System.Linq.Expressions.Expression<System.Func< IQueryStructureAdapter , T>>	mapper	Result set mapper

Returns

TYPE	DESCRIPTION
IRepositoryStructure<T>	Repository structure

Type Parameters

NAME	DESCRIPTION
TResult	Result set data type

Skip(Int32)

Dynamic repository skip count modifier

Declaration

```
IRepositoryStructure<T> Skip(int count)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Int32	count	Skip records count

Returns

TYPE	DESCRIPTION
IRepositoryStructure<T>	Query structure

Subquery(String, Type, String, DataGrouping, Expression<Func<ISubqueryLinkAdapter<T>, IFilterDescription>>[])

Dynamic repository subquery mapping

Declaration

```
IRepositoryStructure<T> Subquery(string alias, Type subqueryType, string property, DataGrouping grouping =
DataGrouping.None, params Expression<Func<ISubqueryLinkAdapter<T>, IFilterDescription>>[] filters)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	alias	Subquery alias
System.Type	subqueryType	Subquery statically mapped type
System.String	property	Subquery selected property
DataGrouping	grouping	Subquery selected property aggregate function
System.Linq.Expressions.Expression<System.Func< ISubqueryLinkAdapter<T> , IFilterDescription >>[]	filters	Filters and links applied to subquery

Returns

TYPE	DESCRIPTION
IRepositoryStructure<T>	Repository structure

See Also

[GetStructure\(String, String, DataStructureFlag, Variable\[\]\)](#)

[IQueryStructure](#)

[IFilterDescription](#)

Subquery<TSubquery>(String, Expression<Func<TSubquery, Object>>, DataGrouping, Expression<Func<ISubqueryLinkAdapter<T, TSubquery>, IFilterDescription>>[])

Dynamic repository subquery mapping

Declaration

```
IRepositoryStructure<T> Subquery<TSubquery>(string alias, Expression<Func<TSubquery, object>> property,
DataGrouping grouping = DataGrouping.None, params Expression<Func<ISubqueryLinkAdapter<T, TSubquery>,
IFilterDescription>>[] filters)
    where TSubquery : class
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	alias	Subquery alias
System.Linq.Expressions.Expression<System.Func<TSubquery, System.Object> >	property	Subquery selected property expression
DataGrouping	grouping	Subquery selected property aggregate function
System.Linq.Expressions.Expression<System.Func< ISubqueryLinkAdapter <T, TSubquery>, IFilterDescription >>[]	filters	Filters and links applied to subquery

Returns

TYPE	DESCRIPTION
IRepositoryStructure <T>	Repository structure

Type Parameters

NAME	DESCRIPTION
TSubquery	Subquery statically mapped type

See Also

- [GetStructure\(String, String, DataStructureFlag, Variable\[\]\)](#)
- [IQueryStructure](#)
- [IFilterDescription](#)

Subquery<TSubquery>(String, Expression<Func<TSubquery, Object>>, DataGrouping, Variable[], Expression<Func<ISubqueryLinkAdapter<T, TSubquery>, IFilterDescription>>[])

Dynamic repository subquery mapping

Declaration

```
IRepositoryStructure<T> Subquery<TSubquery>(string alias, Expression<Func<TSubquery, object>> property,
DataGrouping grouping, Variable[] variables, params Expression<Func<ISubqueryLinkAdapter<T, TSubquery>,
IFilterDescription>>[] filters)
    where TSubquery : class
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	alias	Subquery alias
System.Linq.Expressions.Expression<System.Func<TSubquery, System.Object>>	property	Subquery selected property expression
DataGrouping	grouping	Subquery selected property aggregate function
Variable[]	variables	Subquery variables
System.Linq.Expressions.Expression<System.Func< ISubqueryLinkAdapter <T, TSubquery>, IFilterDescription >>[]	filters	Filters and links applied to subquery

Returns

TYPE	DESCRIPTION
IRepositoryStructure <T>	Repository structure

Type Parameters

NAME	DESCRIPTION
TSubquery	Subquery statically mapped type

See Also

- [GetStructure\(String, String, DataStructureFlag, Variable\[\]\)](#)
- [IQueryStructure](#)
- [IFilterDescription](#)
- [Variable](#)

Subquery<TSubquery>(String, IQueryDescription<TSubquery>, String, DataGrouping, Expression<Func<ISubqueryLinkAdapter<T, TSubquery>, IFilterDescription>>[])

Dynamic repository subquery mapping

Declaration

```
IRepositoryStructure<T> Subquery<TSubquery>(string alias, IQueryDescription<TSubquery> sub, string property, DataGrouping grouping = DataGrouping.None, params Expression<Func<ISubqueryLinkAdapter<T, TSubquery>, IFilterDescription>>[] filters)
    where TSubquery : class
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	alias	Subquery alias
IQueryDescription <TSubquery>	sub	Subquery query structure
System.String	property	Subquery selected property
DataGrouping	grouping	Subquery selected property aggregate function
System.Linq.Expressions.Expression<System.Func< ISubqueryLinkAdapter <T, TSubquery>, IFilterDescription >>[]	filters	Filters and links applied to subquery

Returns

TYPE	DESCRIPTION
IRepositoryStructure <T>	Repository structure

Type Parameters

NAME	DESCRIPTION
TSubquery	

See Also

[GetStructure\(String, String, DataStructureFlag, Variable\[\]\)](#)

[IQueryStructure](#)

[IQueryDescription](#)

[DataGrouping](#)

[IFilterDescription](#)

[With<TRoot>\(String, IQueryDescription<TRoot>, Expression<Func<IQueryWithAdapter<TRoot>, IQueryWithFlag>>\[\]\)](#)

Dynamic repository Common-Table-Expression (WITH) subquery mapping

Declaration

```

IRepositoryStructure<T> With<TRoot>(string alias, IQueryDescription<TRoot> sub, params
Expression<Func<IQueryWithAdapter<TRoot>, IQueryWithFlag>>[] attributes)
    where TRoot : class

```

Parameters

TYPE	NAME	DESCRIPTION

TYPE	NAME	DESCRIPTION
System.String	alias	Common-Table-Expression (WITH) subquery alias
IQueryDescription <TRoot>	sub	Common-Table-Expression (WITH) initial subquery description
System.Linq.Expressions.Expression<System.Func< IQueryWithAdapter <TRoot>, IQueryWithFlag >> []	attributes	Common-Table-Expression (WITH) attributes

Returns

TYPE	DESCRIPTION
IRepositoryStructure <T>	Repository structure

Type Parameters

NAME	DESCRIPTION
TRoot	Common-Table-Expression (WITH) initial subquery mapped type

Extension Methods

- [SerializationExtensions.ToXml](#)<T>(T)
- [CollectionsExtensions.SetValue](#)<TKey, T>(TKey, T)
- [CollectionsExtensions.AsEnum](#)<T>(T)
- [DataObjectQueryableExtensions.CompareVariable](#)(Object, FilterOperation, String)
- [Processing.Do](#)<T>(T, Action<IProcess<T>> [])
- [TypeExtensions.TryConvert](#)<T, TResult>(T, Func<T, TResult>)
- [TypeExtensions.TryConvert](#)<T, TResult>(T, Func<T, TResult>, TResult)
- [TypeExtensions.ChangeType](#)<T>(Object)
- [TypeExtensions.ChangeType](#)(Object, Type)
- [TypeExtensions.InvokeMethod](#)<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)
- [TypeExtensions.InvokeMethod](#)(Object, Type[], Expression<Action>, BindingFlags)
- [TypeExtensions.InvokeMethod](#)<TRet>(Object, Expression<Func<TRet>>, BindingFlags)
- [TypeExtensions.InvokeMethod](#)(Object, Expression<Action>, BindingFlags)
- [SqlBlockExtensions.SetExpression](#)<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)
- [QueryDescriptionExtensions.SetExpression](#)<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Interface IRepositoryStructureAdapter

Repository structure adapter

Namespace: [XData.Mapping.Dynamic](#)

Assembly: XData.docfx.dll

Syntax

```
public interface IRepositoryStructureAdapter
```

Methods

Expr<TValue>(String, Expression<Func<IQueryStructureAdapter, TValue>>, DbType, Expression<Func<IRepositoryExpressionMappingAdapter, IStructureFlag>>[])

SQL expression mapping

Declaration

```
TValue Expr<TValue>(string fieldName, Expression<Func<IQueryStructureAdapter, TValue>> expr, DbType dbType, params Expression<Func<IRepositoryExpressionMappingAdapter, IStructureFlag>>[] attributes)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	fieldName	Expression field name
System.Linq.Expressions.Expression<System.Func< IQueryStructureAdapter , TValue>>	expr	Expression
System.Data.DbType	dbType	Expression database type
System.Linq.Expressions.Expression<System.Func< IRepositoryExpressionMappingAdapter , IStructureFlag >>[]	attributes	Property mapping attributes

Returns

TYPE	DESCRIPTION
TValue	Expression value

Type Parameters

NAME	DESCRIPTION
TValue	Expression result type

Remarks

WARNING! Can be used in dynamic query mapping only!

See Also

[DataExpressionType](#)

System.Data.DbType

[Select<TResult>\(Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Expr<TValue>(String, DataExpressionType, String, DbType, Expression<Func<IRepositoryExpressionMappingAdapter, IStructureFlag>>[])

SQL expression mapping

Declaration

```
TValue Expr<TValue>(string fieldName, DataExpressionType type, string exprText, DbType dbType, params Expression<Func<IRepositoryExpressionMappingAdapter, IStructureFlag>>[] attributes)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	fieldName	Expression field name
DataExpressionType	type	Expression source type
System.String	exprText	Expression text
System.Data.DbType	dbType	Expression database type
System.Linq.Expressions.Expression<System.Func< IRepositoryExpressionMappingAdapter , IStructureFlag >>[]	attributes	Property mapping attributes

Returns

TYPE	DESCRIPTION
TValue	Expression value

Type Parameters

NAME	DESCRIPTION
TValue	Expression result type

Remarks

WARNING! Can be used in dynamic query mapping only!

See Also

[DataExpressionType](#)

System.Data.DbType

Select<TResult>(Expression<Func<IQueryStructureAdapter, TResult>>)

Ref<TValue>(String, String, Expression<Func<IRepositoryReadOnlyPropertyMappingAdapter, IStructureFlag>>[])

Inner view property reference mapping

Declaration

```
TValue Ref<TValue>(string sourceAlias, string property, params
Expression<Func<IRepositoryReadOnlyPropertyMappingAdapter, IStructureFlag>>[] attributes)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	sourceAlias	Referenced property subquery alias
System.String	property	Referenced property name
System.Linq.Expressions.Expression<System.Func<IRepositoryReadOnlyPropertyMappingAdapter, IStructureFlag>>[]	attributes	Property mapping attributes

Returns

TYPE	DESCRIPTION
TValue	Referenced property value

Type Parameters

NAME	DESCRIPTION
TValue	Referenced property type

Remarks

WARNING! Can be used in dynamic query mapping only!

See Also

Select<TResult>(Expression<Func<IQueryStructureAdapter, TResult>>)

Ref<TValue, TInnerView>(String, Expression<Func<TInnerView, Object>>, Expression<Func<IRepositoryReadOnlyPropertyMappingAdapter, IStructureFlag>>[])

Inner view property reference mapping

Declaration

```
TValue Ref<TValue, TInnerView>(string sourceAlias, Expression<Func<TInnerView, object>> property, params
Expression<Func<IRepositoryReadOnlyPropertyMappingAdapter, IStructureFlag>>[] attributes)
    where TInnerView : class
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	sourceAlias	Referenced property subquery alias
System.Linq.Expressions.Expression<System.Func<TInnerView, System.Object>>	property	Referenced property name
System.Linq.Expressions.Expression<System.Func<IRepositoryReadOnlyPropertyMappingAdapter, IStructureFlag>>[]	attributes	Property mapping attributes

Returns

TYPE	DESCRIPTION
TValue	Referenced property value

Type Parameters

NAME	DESCRIPTION
TValue	Referenced property type
TInnerView	Referenced inner view type

Remarks

WARNING! Can be used in dynamic query mapping only!

See Also

[Select<TResult>\(Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Interface IRepositoryStructureAdapter<T>

Repository structure mapping adapter interface

Inherited Members

[IRepositoryStructureAdapter.Expr<TValue>\(String, DataExpressionType, String, DbType, Expression<Func<IRepositoryExpressionMappingAdapter, IStructureFlag>>\[\]\)](#)
[IRepositoryStructureAdapter.Expr<TValue>\(String, Expression<Func<IQueryStructureAdapter, TValue>>, DbType, Expression<Func<IRepositoryExpressionMappingAdapter, IStructureFlag>>\[\]\)](#)
[IRepositoryStructureAdapter.Ref<TValue>\(String, String, Expression<Func<IRepositoryReadOnlyPropertyMappingAdapter, IStructureFlag>>\[\]\)](#)
[IRepositoryStructureAdapter.Ref<TValue, TInnerView>\(String, Expression<Func<TInnerView, Object>>, Expression<Func<IRepositoryReadOnlyPropertyMappingAdapter, IStructureFlag>>\[\]\)](#)

Namespace: [XData.Mapping.Dynamic](#)

Assembly: [XData.docfx.dll](#)

Syntax

```
public interface IRepositoryStructureAdapter<T> : IRepositoryStructureAdapter where T : class, ISqlObject, new()
```

Type Parameters

NAME	DESCRIPTION
T	

Methods

[Field<TValue>\(Expression<Func<IRepositoryPropertyMappingAdapter<T, TValue>, IStructureFlag>>\[\]\)](#)

Field mapping

Declaration

```
TValue Field<TValue>(params Expression<Func<IRepositoryPropertyMappingAdapter<T, TValue>, IStructureFlag>>[] attributes)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func<IRepositoryPropertyMappingAdapter<T, TValue>, IStructureFlag>>[]	attributes	Property mapping attributes

Returns

TYPE	DESCRIPTION
TValue	Field value

Type Parameters

NAME	DESCRIPTION

NAME	DESCRIPTION
TValue	Field type

Remarks

WARNING! Can be used in dynamic query mapping only!

See Also

[Select<TResult>\(Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

[Field<TValue>\(String, Expression<Func<IRepositoryPropertyMappingAdapter<T, TValue>, IStructureFlag>>\[\]\)](#)

Field mapping

Declaration

```
TValue Field<TValue>(string fieldName, params Expression<Func<IRepositoryPropertyMappingAdapter<T, TValue>, IStructureFlag>>[] attributes)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	fieldName	Field name
System.Linq.Expressions.Expression<System.Func< IRepositoryPropertyMappingAdapter <T, TValue>, IStructureFlag >>[]	attributes	Property mapping attributes

Returns

TYPE	DESCRIPTION
TValue	Field value

Type Parameters

NAME	DESCRIPTION
TValue	Field type

Remarks

WARNING! Can be used in dynamic query mapping only!

See Also

[Select<TResult>\(Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

[Field<TValue>\(String, String, Expression<Func<IRepositoryPropertyMappingAdapter<T, TValue>, IStructureFlag>>\[\]\)](#)

Field mapping

Declaration

```
TValue Field<TValue>(string sourceAlias, string fieldName, params
Expression<Func<IRepositoryPropertyMappingAdapter<T, TValue>, IStructureFlag>>[] attributes)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	sourceAlias	Source alias
System.String	fieldName	Field name
System.Linq.Expressions.Expression<System.Func< IRepositoryPropertyMappingAdapter <T, TValue>, IStructureFlag >>[]	attributes	Property mapping attributes

Returns

TYPE	DESCRIPTION
TValue	Field value

Type Parameters

NAME	DESCRIPTION
TValue	Field type

Remarks

WARNING! Can be used in dynamic query mapping only!

See Also

[Select<TResult>\(Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

[Link<TValue, TSource>\(String, Expression<Func<ILinkPropertyMappingAdapter<T>, IStructureFlag>>\[\]\)](#)

Link property mapping

Declaration

```
Link<TValue, TSource> Link<TValue, TSource>(string sourceAlias, params
Expression<Func<ILinkPropertyMappingAdapter<T>, IStructureFlag>>[] attributes)
    where TValue : IComparable<TValue> where TSource : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	sourceAlias	Property source alias

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func< ILinkPropertyMappingAdapter <T>, IStructureFlag >>[]	attributes	Property mapping attributes

Returns

TYPE	DESCRIPTION
Link <TValue, TSource>	Link property

Type Parameters

NAME	DESCRIPTION
TValue	Link value type
TSource	Link source type

[Link](#)<TValue, TSource>(String, String, Expression<Func<[ILinkPropertyMappingAdapter](#)<T>, [IStructureFlag](#)>>[])

Link property mapping

Declaration

<pre> Link<TValue, TSource> Link<TValue, TSource>(string sourceAlias, string fieldName, params Expression<Func<ILinkPropertyMappingAdapter<T>, IStructureFlag>>[] attributes) where TValue : IComparable<TValue> where TSource : class, IDataObject, new()</pre>

Parameters

TYPE	NAME	DESCRIPTION
System.String	sourceAlias	Property source alias
System.String	fieldName	Property field name
System.Linq.Expressions.Expression<System.Func< ILinkPropertyMappingAdapter <T>, IStructureFlag >>[]	attributes	Property mapping attributes

Returns

TYPE	DESCRIPTION
Link <TValue, TSource>	Link property

Type Parameters

NAME	DESCRIPTION
TValue	Link value type
TSource	Link source type

Lob(String, Expression<Func<IRepositoryReadOnlyPropertyMappingAdapter, IStructureFlag>>[])

Lob property mapping

Declaration

```
Lob Lob(string sourceAlias, params Expression<Func<IRepositoryReadOnlyPropertyMappingAdapter, IStructureFlag>>
[] attributes)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	sourceAlias	Property source alias
System.Linq.Expressions.Expression<System.Func< IRepositoryReadOnlyPropertyMappingAdapter , IStructureFlag >>[]	attributes	Property mapping attributes

Returns

TYPE	DESCRIPTION
Lob	Lob property

Lob(String, String, Expression<Func<IRepositoryReadOnlyPropertyMappingAdapter, IStructureFlag>>[])

Lob property mapping

Declaration

```
Lob Lob(string sourceAlias, string fieldName, params
Expression<Func<IRepositoryReadOnlyPropertyMappingAdapter, IStructureFlag>>[] attributes)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	sourceAlias	Property source alias
System.String	fieldName	Property field name

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func< IRepositoryReadOnlyPropertyMappingAdapter , IStructureFlag >> []	attributes	Property mapping attributes

Returns

TYPE	DESCRIPTION
Lob	Lob property

Xml(String, Expression<Func<[IRepositoryReadOnlyPropertyMappingAdapter](#), [IStructureFlag](#)>> [])

Xml property mapping

Declaration

```
Xml Xml(string sourceAlias, params Expression<Func<IRepositoryReadOnlyPropertyMappingAdapter, IStructureFlag>> [] attributes)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	sourceAlias	Property source alias
System.Linq.Expressions.Expression<System.Func< IRepositoryReadOnlyPropertyMappingAdapter , IStructureFlag >> []	attributes	Property mapping attributes

Returns

TYPE	DESCRIPTION
Xml	Xml property

Xml(String, String, Expression<Func<[IRepositoryReadOnlyPropertyMappingAdapter](#), [IStructureFlag](#)>> [])

Xml property mapping

Declaration

```
Xml Xml(string sourceAlias, string fieldName, params Expression<Func<IRepositoryReadOnlyPropertyMappingAdapter, IStructureFlag>> [] attributes)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	sourceAlias	Property source alias

TYPE	NAME	DESCRIPTION
System.String	fieldName	Property field name
System.Linq.Expressions.Expression<System.Func<IRepositoryReadOnlyPropertyMappingAdapter, IStructureFlag>>[]	attributes	Property mapping attributes

Returns

TYPE	DESCRIPTION
Xml	Xml property

Extension Methods

- SerializationExtensions.ToXml<T>(T)
- CollectionsExtensions.SetValue<TKey, T>(TKey, T)
- CollectionsExtensions.AsEnum<T>(T)
- DataObjectQueryableExtensions.CompareVariable(Object, FilterOperation, String)
- Processing.Do<T>(T, Action<IProcess<T>>[])
- TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>)
- TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>, TResult)
- TypeExtensions.ChangeType<T>(Object)
- TypeExtensions.ChangeType(Object, Type)
- TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)
- TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)
- TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)
- TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)
- SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)
- QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Interface IResultSetDescription

Dynamic stored procedure result set description

Namespace: [XData.Mapping.Dynamic](#)

Assembly: XData.docfx.dll

Syntax

```
public interface IResultSetDescription
```

Properties

Alias

Stored procedure alias

Declaration

```
string Alias { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Name

Result set name

Declaration

```
string Name { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Order

Result set order

Declaration

```
int Order { get; }
```

Property Value

TYPE	DESCRIPTION
System.Int32	

Methods

Clone()

Clone result set description

Declaration

IResultSetDescription Clone()

Returns

TYPE	DESCRIPTION
IResultSetDescription	

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

See Also

- [Return<T>\(String, String, Int32, Expression<Func<IQueryStructureAdapter, T>>\)](#)

Interface IStructureFlag

Dynamic structure flag interface

Namespace: [XData.Mapping.Dynamic](#)

Assembly: [XData.docfx.dll](#)

Syntax

```
public interface IStructureFlag
```

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Interface ISubqueryLinkAdapter

Subquery link filter description adapter

Inherited Members

[IFilterAdapter.Combination\(String, Combination, String\)](#)

[IFilterAdapter.Combination\(String, Combination\)](#)

Namespace: [XData.Mapping.Dynamic](#)

Assembly: XData.docfx.dll

Syntax

```
public interface ISubqueryLinkAdapter : IFilterAdapter
```

Methods

SubqueryLink(String)

Subquery link filter description

Declaration

```
IFilterDescription SubqueryLink(string subqueryProperty)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	subqueryProperty	Subquery property name to link with

Returns

TYPE	DESCRIPTION
IFilterDescription	Filter description

SubqueryLink(String, String)

Subquery link filter description

Declaration

```
IFilterDescription SubqueryLink(string subqueryProperty, string property)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	subqueryProperty	Subquery property name to link with
System.String	property	Base query property name to link with

Returns

TYPE	DESCRIPTION
IFilterDescription	Filter description

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Interface ISubqueryLinkAdapter<T, TDObj>

Subquery link filter description adapter

Inherited Members

- [ISubqueryLinkAdapter<T>.SubqueryLink\(Expression<Func<T, Object>>\)](#)
- [ISubqueryLinkAdapter<T>.SubqueryLink\(Expression<Func<T, Object>>, String\)](#)
- [ISubqueryLinkAdapter.SubqueryLink\(String\)](#)
- [ISubqueryLinkAdapter.SubqueryLink\(String, String\)](#)
- [IFilterAdapter.Combination\(String, Combination, String\)](#)
- [IFilterAdapter.Combination\(String, Combination\)](#)

Namespace: [XData.Mapping.Dynamic](#)

Assembly: [XData.docfx.dll](#)

Syntax

```
public interface ISubqueryLinkAdapter<T, TDObj> : ISubqueryLinkAdapter<T>, ISubqueryLinkAdapter,
IFilterAdapter where T : class where TDObj : class
```

Type Parameters

NAME	DESCRIPTION
T	Repository data object type
TDObj	Subquery mapped type

Methods

[SubqueryLink\(Expression<Func<TDObj, Object>>\)](#)

Subquery link filter description

Declaration

```
IFilterDescription SubqueryLink(Expression<Func<TDObj, object>> subqueryProperty)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func<TDObj, System.Object>>	subqueryProperty	Subquery property name expression to link with

Returns

TYPE	DESCRIPTION
IFilterDescription	Filter description

[SubqueryLink\(Expression<Func<TDObj, Object>>, Expression<Func<T, Object>>\)](#)

Subquery link filter description

Declaration

```
IFilterDescription SubqueryLink(Expression<Func<TDObj, object>> subqueryProperty, Expression<Func<T, object>> property)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func<TDObj, System.Object>>	subqueryProperty	Subquery property name expression to link with
System.Linq.Expressions.Expression<System.Func<T, System.Object>>	property	Base query property name expression to link with

Returns

TYPE	DESCRIPTION
IFilterDescription	Filter description

SubqueryLink(Expression<Func<TDObj, Object>>, String)

Subquery link filter description

Declaration

```
IFilterDescription SubqueryLink(Expression<Func<TDObj, object>> subqueryProperty, string property)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func<TDObj, System.Object>>	subqueryProperty	Subquery property name expression to link with
System.String	property	Base query property name to link with

Returns

TYPE	DESCRIPTION
IFilterDescription	Filter description

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)

TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>, TResult)

TypeExtensions.ChangeType<T>(Object)

TypeExtensions.ChangeType(Object, Type)

TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)

TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)

SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)

QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Interface ISubqueryLinkAdapter<TDObj>

Subquery link filter description adapter

Inherited Members

- [ISubqueryLinkAdapter.SubqueryLink\(String\)](#)
- [ISubqueryLinkAdapter.SubqueryLink\(String, String\)](#)
- [IFilterAdapter.Combination\(String, Combination, String\)](#)
- [IFilterAdapter.Combination\(String, Combination\)](#)

Namespace: [XData.Mapping.Dynamic](#)
Assembly: [XData.docfx.dll](#)

Syntax

```
public interface ISubqueryLinkAdapter<TDObj> : ISubqueryLinkAdapter, IFilterAdapter where TDObj : class
```

Type Parameters

NAME	DESCRIPTION
TDObj	Subquery mapped type

Methods

[SubqueryLink\(Expression<Func<TDObj, Object>>\)](#)

Subquery link filter description

Declaration

```
IFilterDescription SubqueryLink(Expression<Func<TDObj, object>> subqueryProperty)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func<TDObj, System.Object>>	subqueryProperty	Subquery property name expression to link with

Returns

TYPE	DESCRIPTION
IFilterDescription	Filter description

[SubqueryLink\(Expression<Func<TDObj, Object>>, String\)](#)

Subquery link filter description

Declaration

```
IFilterDescription SubqueryLink(Expression<Func<TDObj, object>> subqueryProperty, string property)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func<TObj, System.Object>>	subqueryProperty	Subquery property name expression to link with
System.String	property	Base query property name to link with

Returns

TYPE	DESCRIPTION
IFilterDescription	Filter description

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)
- [Processing.Do<T>\(T, Action<IProcess<T>>\[\]\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>\)](#)
- [TypeExtensions.TryConvert<T, TResult>\(T, Func<T, TResult>, TResult\)](#)
- [TypeExtensions.ChangeType<T>\(Object\)](#)
- [TypeExtensions.ChangeType\(Object, Type\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Type\[\], Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Type\[\], Expression<Action>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod<TRet>\(Object, Expression<Func<TRet>>, BindingFlags\)](#)
- [TypeExtensions.InvokeMethod\(Object, Expression<Action>, BindingFlags\)](#)
- [SqlBlockExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IBlockQueryAdapter, TResult>>\)](#)
- [QueryDescriptionExtensions.SetExpression<TTag, TResult>\(TTag, Expression<Func<IQueryStructureAdapter, TResult>>\)](#)

Interface ITaggedExpression

Object tagged expression interface

Namespace: [XData.Mapping.Dynamic](#)

Assembly: XData.docfx.dll

Syntax

```
public interface ITaggedExpression
```

Properties

Key

Key value

Declaration

```
object Key { get; }
```

Property Value

TYPE	DESCRIPTION
System.Object	

KeyType

Key type

Declaration

```
Type KeyType { get; }
```

Property Value

TYPE	DESCRIPTION
System.Type	

Value

Value type

Declaration

```
Expression Value { get; }
```

Property Value

TYPE	DESCRIPTION
System.Linq.Expressions.Expression	

Extension Methods

- [SerializationExtensions.ToXml<T>\(T\)](#)
- [CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)
- [CollectionsExtensions.AsEnum<T>\(T\)](#)
- [DataObjectQueryableExtensions.CompareVariable\(Object, FilterOperation, String\)](#)

Processing.Do<T>(T, Action<IProcess<T>>[])

TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>)

TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>, TResult)

TypeExtensions.ChangeType<T>(Object)

TypeExtensions.ChangeType(Object, Type)

TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)

TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)

TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)

SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)

QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Class ParameterDescription

SQL procedure / function parameter description

Inheritance

System.Object
ParameterDescription

Implements

[IParameterDescription](#)

Inherited Members

System.Object.Equals(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.GetHashCode()
System.Object.GetType()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.ToString()

Namespace: [XData.Mapping.Dynamic](#)
Assembly: XData.docfx.dll

Syntax

```
public class ParameterDescription : IParameterDescription
```

Properties

Alias

Procedure alias

Declaration

```
public string Alias { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Binding

Parameter XData binding name

Declaration

```
public string Binding { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

DbType

Parameter database type

Declaration

```
public DbType DbType { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Data.DbType	

DefaultValue

Parameter default value

Declaration

```
public object DefaultValue { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Object	

DefaultValueType

Parameter default value type

Declaration

```
public DefaultType DefaultValueType { get; set; }
```

Property Value

TYPE	DESCRIPTION
DefaultType	

Direction

Parameter direction

Declaration

```
public ParameterDirection Direction { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Data.ParameterDirection	

Name

Parameter name

Declaration

```
public string Name { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

NativeSqlType

Parameter native SQL type

Declaration

```
public string NativeSqlType { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Order

Parameter order

Declaration

```
public int Order { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Int32	

Scale

Parameter scale

Declaration

```
public int Scale { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Int32	

Size

Parameter size

Declaration

```
public int Size { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Int32	

Type

Parameter mapped type

Declaration

```
public Type Type { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Type	

TypesToArray

Parameter is VARARRAY

Declaration

```
public bool TypeIsArray { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

UdtSqlType

Parameter user-defined-type definition

Declaration

```
public UdtSqlType UdtSqlType { get; set; }
```

Property Value

TYPE	DESCRIPTION
UdtSqlType	

Value

Parameter value

Declaration

```
public object Value { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Object	

Methods

Clone()

Parameter clone

Declaration

```
public IParameterDescription Clone()
```

Returns

TYPE	DESCRIPTION
IParameterDescription	

Copy(Object)

Parameter copy

Declaration

```
public IParameterDescription Copy(object value)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Object	value	Parameter value

Returns

TYPE	DESCRIPTION
IParameterDescription	Parameter description

GetValue<T>()

Parameter conversion

Declaration

```
public T GetValue<T>()
```

Returns

TYPE	DESCRIPTION
T	

Type Parameters

NAME	DESCRIPTION
T	Conversion result type

Implements

[IParameterDescription](#)

Extension Methods

[SerializationExtensions.ToXml<T>\(T\)](#)

[CollectionsExtensions.SetValue<TKey, T>\(TKey, T\)](#)

CollectionsExtensions.AsEnum<T>(T)
DataObjectQueryableExtensions.CompareVariable(Object, FilterOperation, String)
Processing.Do<T>(T, Action<IProcess<T>>[])
TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>)
TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>, TResult)
TypeExtensions.ChangeType<T>(Object)
TypeExtensions.ChangeType(Object, Type)
TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)
TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)
TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)
TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)
SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)
QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Class QueryDescriptionExtensions

Query description extensions

Inheritance

System.Object

QueryDescriptionExtensions

Inherited Members

System.Object.Equals(System.Object)

System.Object.Equals(System.Object, System.Object)

System.Object.GetHashCode()

System.Object.GetType()

System.Object.MemberwiseClone()

System.Object.ReferenceEquals(System.Object, System.Object)

System.Object.ToString()

Namespace: [XData.Mapping.Dynamic](#)

Assembly: XData.docfx.dll

Syntax

```
public static class QueryDescriptionExtensions
```

Methods

AsPrimary<T>(T)

Set primary filter flag to filter description

Declaration

```
public static T AsPrimary<T>(this T desc)
    where T : IFilterDescription
```

Parameters

TYPE	NAME	DESCRIPTION
T	desc	Filter description

Returns

TYPE	DESCRIPTION
T	Filter description

Type Parameters

NAME	DESCRIPTION
T	Filter description type

Exceptions

TYPE	CONDITION
XDataRuntimeException	desc argument can not be null

Case<T, TResult>(IQueryStructureAdapter, Expression<Func<IQueryStructureAdapter, T>>, Expression<Func<IQueryStructureAdapter, TResult>>, TaggedExpression<T, TResult>[])

Case

Declaration

```
public static TResult Case<T, TResult>(this IQueryStructureAdapter adapter,
Expression<Func<IQueryStructureAdapter, T>> expression, Expression<Func<IQueryStructureAdapter, TResult>>
elseValue, params TaggedExpression<T, TResult>[] switchValues)
```

Parameters

TYPE	NAME	DESCRIPTION
IQueryStructureAdapter	adapter	Source query structure adapter
System.Linq.Expressions.Expression<System.Func< IQueryStructureAdapter , T>>	expression	Case expression
System.Linq.Expressions.Expression<System.Func< IQueryStructureAdapter , TResult>>	elseValue	Else value
TaggedExpression <T, TResult>[]	switchValues	Case switch values

Returns

TYPE	DESCRIPTION
TResult	Case result

Type Parameters

NAME	DESCRIPTION
T	
TResult	

Exceptions

TYPE	CONDITION
XDataRuntimeException	Function accessible in XData LINQ expressions only

DateDiff<T>(IQueryStructureAdapter, Expression<Func<IQueryStructureAdapter, DateTime>>, Expression<Func<IQueryStructureAdapter, DateTime>>, DateDiffKind)

Dates difference

Declaration

```
public static T DateDiff<T>(this IQueryStructureAdapter adapter, Expression<Func<IQueryStructureAdapter, DateTime>> finalDate, Expression<Func<IQueryStructureAdapter, DateTime>> startDate, DateDiffKind kind)
```

Parameters

TYPE	NAME	DESCRIPTION
IQueryStructureAdapter	adapter	Source query structure adapter
System.Linq.Expressions.Expression<System.Func< IQueryStructureAdapter , System.DateTime>>	finalDate	Final date
System.Linq.Expressions.Expression<System.Func< IQueryStructureAdapter , System.DateTime>>	startDate	Start date
DateDiffKind	kind	Dates differences kind

Returns

TYPE	DESCRIPTION
T	Dates difference

Type Parameters

NAME	DESCRIPTION
T	

Exceptions

TYPE	CONDITION
XDataRuntimeException	Function accessible in XData LINQ expressions only

Exists(IQueryStructureAdapter, String)

Exists

Declaration

```
public static bool Exists(this IQueryStructureAdapter adapter, string subQueryAlias)
```

Parameters

TYPE	NAME	DESCRIPTION

TYPE	NAME	DESCRIPTION
IQueryStructureAdapter	adapter	Source query structure adapter
System.String	subQueryAlias	Subquery alias

Returns

TYPE	DESCRIPTION
System.Boolean	

Exceptions

TYPE	CONDITION
XDataRuntimeException	Function accessible in XData LINQ expressions only

GetProperty(ISqlObject, String)

Property get handler used in XData mapping

Declaration

```
public static object GetProperty(this ISqlObject obj, string property)
```

Parameters

TYPE	NAME	DESCRIPTION
ISqlObject	obj	Object
System.String	property	Property name

Returns

TYPE	DESCRIPTION
System.Object	Property

GetVariable<TResult>(IQueryStructureAdapter, String)

Declaration

```
public static TResult GetVariable<TResult>(this IQueryStructureAdapter adapter, string variableName)
```

Parameters

TYPE	NAME	DESCRIPTION
IQueryStructureAdapter	adapter	

TYPE	NAME	DESCRIPTION
System.String	variableName	

Returns

TYPE	DESCRIPTION
TResult	

Type Parameters

NAME	DESCRIPTION
TResult	

HasFlag<T>(IQueryStructureAdapter, Expression<Func<IQueryStructureAdapter, T>>, Expression<Func<IQueryStructureAdapter, T>>, Expression<Func<IQueryStructureAdapter, T>>, Expression<Func<IQueryStructureAdapter, T>>)

Bit mask has flag

Declaration

```
public static T HasFlag<T>(this IQueryStructureAdapter adapter, Expression<Func<IQueryStructureAdapter, T>>
expression, Expression<Func<IQueryStructureAdapter, T>> flagValue, Expression<Func<IQueryStructureAdapter, T>>
trueValue, Expression<Func<IQueryStructureAdapter, T>> falseValue)
```

Parameters

TYPE	NAME	DESCRIPTION
IQueryStructureAdapter	adapter	Source query structure adapter
System.Linq.Expressions.Expression<System.Func<IQueryStructureAdapter, T>>	expression	Expression to test
System.Linq.Expressions.Expression<System.Func<IQueryStructureAdapter, T>>	flagValue	Flag value
System.Linq.Expressions.Expression<System.Func<IQueryStructureAdapter, T>>	trueValue	Result if flag exists
System.Linq.Expressions.Expression<System.Func<IQueryStructureAdapter, T>>	falseValue	Result if flag not exists

Returns

TYPE	DESCRIPTION
T	Bit mask has flag than trueValue else falseValue

Type Parameters

NAME	DESCRIPTION
T	

Exceptions

TYPE	CONDITION
XDataRuntimeException	Function accessible in XData LINQ expressions only

SetCombination<T>(T, String)

Set combination name to filter description

Declaration

<pre>public static T SetCombination<T>(this T desc, string combinationName) where T : IFilterDescription</pre>

Parameters

TYPE	NAME	DESCRIPTION
T	desc	Filter description
System.String	combinationName	Filter combination name

Returns

TYPE	DESCRIPTION
T	Filter description

Type Parameters

NAME	DESCRIPTION
T	Filter description type

Exceptions

TYPE	CONDITION
XDataRuntimeException	desc argument can not be null

SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Set expression to tag

Declaration


```
public static TaggedExpression<TTag, TResult> SetExpression<TTag, TResult>(this TTag key,
Expression<Func<IQueryStructureAdapter, TResult>> expression)
```

Parameters

TYPE	NAME	DESCRIPTION
TTag	key	Tag value
System.Linq.Expressions.Expression<System.Func< IQueryStructureAdapter , TResult>>	expression	Expression

Returns

TYPE	DESCRIPTION
TaggedExpression <TTag, TResult>	Tagged expression

Type Parameters

NAME	DESCRIPTION
TTag	Tag type
TResult	Expression result

SetOperation<T>(T, FilterOperation)

Set filter or link operation

Declaration

```
public static T SetOperation<T>(this T desc, FilterOperation operation)
    where T : IFilterDescription
```

Parameters

TYPE	NAME	DESCRIPTION
T	desc	Filter description
FilterOperation	operation	Filter operation

Returns

TYPE	DESCRIPTION
T	Filter description

Type Parameters

NAME	DESCRIPTION
T	Filter description type

Exceptions

TYPE	CONDITION
XDataRuntimeException	desc argument can not be null or empty

See Also

[FilterOperation](#)

SqlExpr<TResult>(IQueryStructureAdapter, String, Expression<Func<IQueryStructureAdapter, Object>>[])

Plain SQL expression with XData LINQ expression placeholders

Declaration

```
public static TResult SqlExpr<TResult>(this IQueryStructureAdapter adapter, string format, params
Expression<Func<IQueryStructureAdapter, object>>[] expressions)
```

Parameters

TYPE	NAME	DESCRIPTION
IQueryStructureAdapter	adapter	Source query structure adapter
System.String	format	SQL expression body in System.String.Format(String, Object[]) style
System.Linq.Expressions.Expression<System.Func< IQueryStructureAdapter , System.Object>>[]	expressions	XData LINQ parameter expressions

Returns

TYPE	DESCRIPTION
TResult	SQL expression result

Type Parameters

NAME	DESCRIPTION
TResult	

Remarks

WARNING! Plain SQL expressions can use specific SQL dialect features!

Data mapping may be not compatible with other database engines!

Exceptions

TYPE	CONDITION
XDataRuntimeException	Function accessible in XData LINQ expressions only

SqlFn<TResult>(IQueryStructureAdapter, String, Expression<Func<IQueryStructureAdapter, Object>>[])

SQL function call

Declaration

<pre>public static TResult SqlFn<TResult>(this IQueryStructureAdapter adapter, string name, params Expression<Func<IQueryStructureAdapter, object>>[] parameters)</pre>

Parameters

TYPE	NAME	DESCRIPTION
IQueryStructureAdapter	adapter	Source query structure adapter
System.String	name	SQL function full qualified name
System.Linq.Expressions.Expression<System.Func< IQueryStructureAdapter , System.Object>>[]	parameters	Function parameters descriptions

Returns

TYPE	DESCRIPTION
TResult	Function call result

Type Parameters

NAME	DESCRIPTION
TResult	

Exceptions

TYPE	CONDITION
XDataRuntimeException	Function accessible in XData LINQ expressions only

SqlParameter<TResult>(IQueryStructureAdapter, String, DbType, Expression<Func<IProcedureParameterAdapter, IProcedureParameterFlag>>[])

SQL parameter definition

Declaration

```
public static TResult SqlParam<TResult>(this IQueryStructureAdapter adapter, string binding, DbType type,
params Expression<Func<IProcedureParameterAdapter, IProcedureParameterFlag>>[] attributes)
```

Parameters

TYPE	NAME	DESCRIPTION
IQueryStructureAdapter	adapter	Query structure adapter
System.String	binding	Parameter data binding name
System.Data.DbType	type	Parameter database type
System.Linq.Expressions.Expression<System.Func< IProcedureParameterAdapter , IProcedureParameterFlag >> []	attributes	Parameter mapping attributes

Returns

TYPE	DESCRIPTION
TResult	Parameter

Type Parameters

NAME	DESCRIPTION
TResult	Parameter mapped type

Exceptions

TYPE	CONDITION
XDataRuntimeException	Function accessible in XData LINQ expressions only

ToDataSet<T>(IEnumerable<T>, String, ISqlBuilder)

Composes data source text from sequence

Declaration

```
public static string ToDataSet<T>(this IEnumerable<T> source, string prefix, ISqlBuilder builder)
```

Parameters

TYPE	NAME	DESCRIPTION

TYPE	NAME	DESCRIPTION
System.Collections.Generic.IEnumerable<T>	source	Source sequence
System.String	prefix	Source column prefix
ISqlBuilder	builder	SQL dialect adapter

Returns

TYPE	DESCRIPTION
System.String	Data source text

Type Parameters

NAME	DESCRIPTION
T	Source object type

Remarks

Used to compose synthetic data source from object sequence

```

var ds = Enumerable.Range(1,3).ToDataSet("val", XDataManager.GetContext().DialectAdapter);
// for MSSQL dialect ds will contains "(select 1 as val union select 2 as val union select 3 as val)"
// for Oracle dialect ds will contains "(select 1 as val from dual union select 2 as val from dual union
select 3 as val from dual)"
// Enumeration types can be used as
var ds = Enum.GetValues(typeof(SomeEnumType)).Cast<int>().ToDataSet("val",
XDataManager.GetContext().DialectAdapter);
// Object types properties are represented as fields
var test = new []{new {Id=1, Name="Test"}, new {Id=2, Name="Another"}};
var ds = test.ToDataSet("X", XDataManager.GetContext().DialectAdapter);
// for MSSQL dialect ds will contains "(select 1 as X_Id, 'Test' as X_Name union select 2 as X_Id, 'Another'
as X_Name)"

```

Class TaggedExpression<TTag, TResult>

Tagged expression

Inheritance

System.Object
TaggedExpression<TTag, TResult>

Implements

[ITaggedExpression](#)

Inherited Members

System.Object.Equals(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.GetHashCode()
System.Object.GetType()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.ToString()

Namespace: [XData.Mapping.Dynamic](#)
Assembly: XData.docfx.dll

Syntax

```
public class TaggedExpression<TTag, TResult> : ITaggedExpression
```

Type Parameters

NAME	DESCRIPTION
TTag	Tag type
TResult	Expression

Properties

Key

Tag key

Declaration

```
public object Key { get; }
```

Property Value

TYPE	DESCRIPTION
System.Object	

KeyType

Tag key type

Declaration

```
public Type KeyType { get; }
```

Property Value

TYPE	DESCRIPTION
System.Type	

Value

Tagged expression

Declaration

```
public Expression Value { get; }
```

Property Value

TYPE	DESCRIPTION
System.Linq.Expressions.Expression	

Operators

Implicit(KeyValuePair<TTag, Expression<Func<IBlockQueryAdapter, TResult>>> to TaggedExpression<TTag, TResult>)

Tagged expression implicit conversion operator

Declaration

```
public static implicit operator TaggedExpression<TTag, TResult>(KeyValuePair<TTag, Expression<Func<IBlockQueryAdapter, TResult>>> pair)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Collections.Generic.KeyValuePair<TTag, System.Linq.Expressions.Expression<System.Func<IBlockQueryAdapter, TResult>>>	pair	Key value pair

Returns

TYPE	DESCRIPTION
TaggedExpression<TTag, TResult>	Tagged expression

Implicit(KeyValuePair<TTag, Expression<Func<IQueryStructureAdapter, TResult>>> to TaggedExpression<TTag, TResult>)

Tagged expression implicit conversion operator

Declaration

```
public static implicit operator TaggedExpression<TTag, TResult>(KeyValuePair<TTag, Expression<Func<IQueryStructureAdapter, TResult>>> pair)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Collections.Generic.KeyValuePair<TTag, System.Linq.Expressions.Expression<System.Func< IQueryStructureAdapter , TResult>>>	pair	Key value pair

Returns

TYPE	DESCRIPTION
TaggedExpression <TTag, TResult>	Tagged expression

Implements

[ITaggedExpression](#)

Extension Methods

- [SerializationExtensions.ToXml](#)<T>(T)
- [CollectionsExtensions.SetValue](#)<TKey, T>(TKey, T)
- [CollectionsExtensions.AsEnum](#)<T>(T)
- [DataObjectQueryableExtensions.CompareVariable](#)(Object, FilterOperation, String)
- [Processing.Do](#)<T>(T, Action<IProcess<T>>[])
- [TypeExtensions.TryConvert](#)<T, TResult>(T, Func<T, TResult>)
- [TypeExtensions.TryConvert](#)<T, TResult>(T, Func<T, TResult>, TResult)
- [TypeExtensions.ChangeType](#)<T>(Object)
- [TypeExtensions.ChangeType](#)(Object, Type)
- [TypeExtensions.InvokeMethod](#)<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)
- [TypeExtensions.InvokeMethod](#)(Object, Type[], Expression<Action>, BindingFlags)
- [TypeExtensions.InvokeMethod](#)<TRet>(Object, Expression<Func<TRet>>, BindingFlags)
- [TypeExtensions.InvokeMethod](#)(Object, Expression<Action>, BindingFlags)
- [SqlBlockExtensions.SetExpression](#)<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)
- [QueryDescriptionExtensions.SetExpression](#)<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)

Class XDataMapping

Inheritance

System.Object
XDataMapping

Inherited Members

System.Object.Equals(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.GetHashCode()
System.Object.GetType()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.ToString()

Namespace: [XData.Mapping.Dynamic](#)
Assembly: XData.docfx.dll

Syntax

```
public static class XDataMapping
```

Methods

CustomMapping<T>()

Returns dynamic repository structure

Declaration

```
public static IRepositoryStructure<T> CustomMapping<T>()  
    where T : class, ISqlObject, new()
```

Returns

TYPE	DESCRIPTION
IRepositoryStructure<T>	Dynamic query structure

Type Parameters

NAME	DESCRIPTION
T	Repository element mapped type

Exceptions

TYPE	CONDITION
XDataRuntimeException	

See Also

[DataStructureFlag](#)
[Variable](#)
[IQueryStructure](#)

GetStructure()

Returns dynamic query structure

Declaration

```
public static IQueryStructure GetStructure()
```

Returns

TYPE	DESCRIPTION
IQueryStructure	Dynamic query structure

See Also

- [DataStructureFlag](#)
- [IQueryStructure](#)

GetStructure(String)

Returns dynamic query structure

Declaration

```
public static IQueryStructure GetStructure(string baseTableAlias)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	baseTableAlias	Base table alias

Returns

TYPE	DESCRIPTION
IQueryStructure	Dynamic query structure

See Also

- [DataStructureFlag](#)
- [IQueryStructure](#)

GetStructure(String, String, DataStructureFlag, Variable[])

Returns dynamic query structure

Declaration

```
public static IQueryStructure GetStructure(string baseTableAlias, string context, DataStructureFlag flags = DataStructureFlag.ReadOnly, params Variable[] variables)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	baseTableAlias	Base table alias

TYPE	NAME	DESCRIPTION
System.String	context	Context name
DataStructureFlag	flags	Data object flags
Variable[]	variables	Variables

Returns

TYPE	DESCRIPTION
IQueryStructure	Dynamic query structure

See Also

- [DataStructureFlag](#)
- [Variable](#)
- [IQueryStructure](#)

GetStructure(String, DataStructureFlag)

Returns dynamic query structure

Declaration

```
public static IQueryStructure GetStructure(string baseTableAlias, DataStructureFlag flags)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	baseTableAlias	Base table alias
DataStructureFlag	flags	Data object flags

Returns

TYPE	DESCRIPTION
IQueryStructure	Dynamic query structure

See Also

- [DataStructureFlag](#)
- [IQueryStructure](#)

GetStructure<T>(String)

Returns dynamic query structure based on static mapping

Declaration

```
public static IQueryStructure GetStructure<T>(string context = null)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	context	Context name

Returns

TYPE	DESCRIPTION
IQueryStructure	Dynamic query structure

Type Parameters

NAME	DESCRIPTION
T	

See Also

[DataStructureFlag](#)

[Variable](#)

[IQueryStructure](#)

UseMapping<T>()

Returns dynamic repository inherited structure

Declaration

```
public static IInheritedStructure<T> UseMapping<T>()
    where T : class, ISqlObject, new()
```

Returns

TYPE	DESCRIPTION
IInheritedStructure<T>	Dynamic query structure

Type Parameters

NAME	DESCRIPTION
T	Repository element mapped type

Exceptions

TYPE	CONDITION
XDataRuntimeException	

See Also

[DataStructureFlag](#)
[Variable](#)
[IQueryStructure](#)

UseMapping<T>(Expression<Func<IExternalLinkDefinitionAdapter<T>, IExternalLinkDefinition>>[])

Returns dynamic repository inherited structure

Declaration

```
public static IRepositoryDescription<T> UseMapping<T>(params
Expression<Func<IExternalLinkDefinitionAdapter<T>, IExternalLinkDefinition>>[] externalLinks)
    where T : class, ISqlObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func< IExternalLinkDefinitionAdapter <T>, IExternalLinkDefinition >>[]	externalLinks	

Returns

TYPE	DESCRIPTION
IRepositoryDescription <T>	Dynamic query structure

Type Parameters

NAME	DESCRIPTION
T	Repository element mapped type

Exceptions

TYPE	CONDITION
XDataRuntimeException	

See Also

[DataStructureFlag](#)
[Variable](#)
[IQueryStructure](#)

UseMapping<T, TParent>()

Returns dynamic repository inherited structure

Declaration

```
public static IInheritedStructure<T> UseMapping<T, TParent>()
    where T : class, ISqlObject, new()
    where TParent : class, ISqlObject, new()
```

Returns

TYPE	DESCRIPTION
InheritedStructure<T>	Dynamic query structure

Type Parameters

NAME	DESCRIPTION
T	Repository element mapped type
TParent	Parent mapping structure type

Exceptions

TYPE	CONDITION
XDataRuntimeException	

See Also

- [DataStructureFlag](#)
- [Variable](#)
- [IQueryStructure](#)

UseMapping<T, TParent>(Expression<Func<IExternalLinkDefinitionAdapter<T>, IExternalLinkDefinition>>[])

Returns dynamic repository inherited structure

Declaration

```
public static IRepositoryDescription<T> UseMapping<T, TParent>(params
Expression<Func<IExternalLinkDefinitionAdapter<T>, IExternalLinkDefinition>>[] externalLinks)
    where T : class, TParent, new()
    where TParent : class, ISqlObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func< IExternalLinkDefinitionAdapter<T> , IExternalLinkDefinition >>[]	externalLinks	

Returns

TYPE	DESCRIPTION
IRepositoryDescription<T>	Dynamic query structure

Type Parameters

NAME	DESCRIPTION
T	Repository element mapped type

NAME	DESCRIPTION
TParent	Parent mapping structure type

Exceptions

TYPE	CONDITION
XDataRuntimeException	

See Also

- [DataStructureFlag](#)
- [Variable](#)
- [IQueryStructure](#)

Namespace XData.Security

Classes

[SecuritySession](#)

Security session base class

Class SecuritySession

Security session base class

Inheritance

System.Object

SecuritySession

Namespace: [XData.Security](#)
Assembly: XData.Security.docfx.dll

Syntax

```
public abstract class SecuritySession : ISecuritySession
```

Constructors

SecuritySession(String, String[], Boolean)

Security session constructor

Declaration

```
public SecuritySession(string userName, string[] roles, bool isSupervisor = false)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	userName	User name
System.String[]	roles	User roles
System.Boolean	isSupervisor	Supervisor flag

Properties

IsSupervisor

Current user is supervisor flag

Declaration

```
public bool IsSupervisor { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

Roles

Current user roles list

Declaration

```
public string[] Roles { get; protected set; }
```

Property Value

TYPE	DESCRIPTION
System.String[]	

UserName

Current user name

Declaration

```
public string UserName { get; protected set; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Methods

Action<T>(Expression<Func<CustomLogic<T>>>)

Custom action name access helper

Declaration

```
protected static string Action<T>(Expression<Func<CustomLogic<T>>> expr)  
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func<CustomLogic<T>>>	expr	Custom logic expression

Returns

TYPE	DESCRIPTION
System.String	Custom action name

Type Parameters

NAME	DESCRIPTION
T	Data object type

FilterGrantedObjects<T>(String, Expression)

Converts initial IQueryable expression filtering granted objects only

Declaration

```
public Expression FilterGrantedObjects<T>(string context, Expression allDataExpr)  
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	context	Context name
System.Linq.Expressions.Expression	allDataExpr	Initial IQueryable expression

Returns

TYPE	DESCRIPTION
System.Linq.Expressions.Expression	Modified IQueryable expression

Type Parameters

NAME	DESCRIPTION
T	Data object type

GetGrantedActions<T>(String)

Returns list of granted actions for current user

Declaration

<pre>public string[] GetGrantedActions<T>(string context = null) where T : class, IDataObject, new()</pre>

Parameters

TYPE	NAME	DESCRIPTION
System.String	context	Context name

Returns

TYPE	DESCRIPTION
System.String[]	Actions list

Type Parameters

NAME	DESCRIPTION
T	Data object type

GetRestrictedProperties<T>(String)

Returns list of restricted properties for current user

Declaration

```
public string[] GetRestrictedProperties<T>(string context = null)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	context	Context name

Returns

TYPE	DESCRIPTION
System.String[]	Properties name list

Type Parameters

NAME	DESCRIPTION
T	Data object type

InitializeSession()

Session initialization

Declaration

```
protected abstract void InitializeSession()
```

See Also

[RegisterGrantedActions\(String, Type, IEnumerable<KeyValuePair<String, String\[\]>>\)](#)

[RegisterRestrictedProperties\(String, Type, IEnumerable<KeyValuePair<String, String\[\]>>\)](#)

[RegisterSecurityFilters\(String, Type, IEnumerable<KeyValuePair<String, Expression>>\)](#)

Property<T>(Expression<Func<T, Object>>)

Property name access helper

Declaration

```
protected static string Property<T>(Expression<Func<T, object>> expr)
    where T : class, IDataObject, new()
```

Parameters

TYPE	NAME	DESCRIPTION
System.Linq.Expressions.Expression<System.Func<T, System.Object>>	expr	Property expression

Returns

TYPE	DESCRIPTION
System.String	Property name

Type Parameters

NAME	DESCRIPTION
T	Data object type

RegisterGrantedActions(String, Type, IEnumerable<KeyValuePair<String, String[]>>)

Granted actions registration

Declaration

```
protected void RegisterGrantedActions(string context, Type type, IEnumerable<KeyValuePair<string, string[]>> actions)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	context	Context name
System.Type	type	Data type to grant actions
System.Collections.Generic.IEnumerable<System.Collections.Generic.KeyValuePair<System.String, System.String[]>>	actions	Granted actions

RegisterRestrictedProperties(String, Type, IEnumerable<KeyValuePair<String, String[]>>)

Restricted properties registration

Declaration

```
protected void RegisterRestrictedProperties(string context, Type type, IEnumerable<KeyValuePair<string, string[]>> restricted)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	context	Context name
System.Type	type	Data type to restrict property access

TYPE	NAME	DESCRIPTION
System.Collections.Generic.IEnumerable<System.Collections.Generic.KeyValuePair<System.String, System.String[]>>	restricted	Restricted properties

RegisterSecurityFilters(String, Type, IEnumerable<KeyValuePair<String, Expression>>)

Security filters registration

Declaration

```
protected void RegisterSecurityFilters(string context, Type type, IEnumerable<KeyValuePair<string, Expression>> filters)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	context	Context name
System.Type	type	Data type to filter data
System.Collections.Generic.IEnumerable<System.Collections.Generic.KeyValuePair<System.String, System.Linq.Expressions.Expression>>	filters	Security filters

Extension Methods

- SerializationExtensions.ToXml<T>(T)
- CollectionsExtensions.SetValue<TKey, T>(TKey, T)
- CollectionsExtensions.AsEnum<T>(T)
- DataObjectQueryableExtensions.CompareVariable(Object, FilterOperation, String)
- Processing.Do<T>(T, Action<IProcess<T>>[])
- TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>)
- TypeExtensions.TryConvert<T, TResult>(T, Func<T, TResult>, TResult)
- TypeExtensions.ChangeType<T>(Object)
- TypeExtensions.ChangeType(Object, Type)
- TypeExtensions.InvokeMethod<TRet>(Object, Type[], Expression<Func<TRet>>, BindingFlags)
- TypeExtensions.InvokeMethod(Object, Type[], Expression<Action>, BindingFlags)
- TypeExtensions.InvokeMethod<TRet>(Object, Expression<Func<TRet>>, BindingFlags)
- TypeExtensions.InvokeMethod(Object, Expression<Action>, BindingFlags)
- SqlBlockExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IBlockQueryAdapter, TResult>>)
- QueryDescriptionExtensions.SetExpression<TTag, TResult>(TTag, Expression<Func<IQueryStructureAdapter, TResult>>)