Unified Data Accessor by ATheory

[ATheory.UnifiedAccess.Data]

The SDK provides a single set of fluent APIs to access data from different data providers (sql, no-sql) seamlessly as well as exposes functionality to add, update or delete tables/schemas. No need to create or manage DbConext or drivers. Many providers can be used at the same time and the underlying system will manage the infrastructure. Data from one store can be inserted into another store through a data tunnel without any complexities.
Above all its simple and super easy to use.

Supported providers are

1. SqlServer
2. SqlLite
3. Cosmos Db
4. MongoDb
5. DynamoDB
6. MySql (not tested)

Target Framework

1. .net core 3.1 and above

API Reference

Assembly: ATheory.UnifiedAccess.Data.dll

EntityUnifier class

Namespace : ATheory.UnifiedAccess.Data.Infrastructure

Main entry point for the library, a static class acts as a factory manager. Use this class to prepare and start using the SDK functionality.

Methods

public static IGateway Factory()
Gets the object that provides core infrastructure services.

Returns → IGateway
Returns an instance of the gateway object.

Example:

```
var factory = EntityUnifier.Factory();
```

public static void Shutdown()
Cleans-up system, closes context (if single life cycle), etc. Call this method before closing your app.

Example:

```
EntityUnifier. Shutdown ();
```

public static void SetOptions(UnifierOption option)
Provides option to set various settings regarding memory usage; mostly internal cache.

Parameters

Option	: Options to set

Example:

```
EntityUnifier. SetOptions (new UnifierOption { UseCacheForInternalObject = true});
```

IGateway interface

Namespace : ATheory.UnifiedAccess.Data.Infrastructure

Service provider interface. The instance is returned from a call from EntityUnifier.Factory().

Methods [Instance/Extension]

```
public static IGateway RegisterContext(this IGateway _, Expression<Func<IUnifiedContext>> projection,
        string contextName, LifeCycle = LifeCycle.TransientPerAction)
```

Register the context that'll be used to access the database.

Returns → IGateway

The same factory instance; for fluent API.

Parameters

_	: The gateway instance {extension}
projection	: Projection to DbContext implementing the IUnifiedContext.
contextName	: Context name or identifier.
lifeCycle	: Life cycle type

Example:

```
factory. RegisterContext(() => new CustomContext(connection), "my-custom-context", LifeCycle.SingleInstance);
```

```
public static IGateway UseDefaultContext ( this IGateway _, Connection connection, string contextName = null)
```

Create the default context that can be used to access the database instead of implementing the DbContext and then registering it using the RegisterContext method.

Returns → IGateway

The same factory instance; for fluent API.

Parameters

_	: The gateway instance {extension}
connection	: The connection object. Provider specific static methods have been implemented to get the object
contextName	: Context name or identifier.

Example:

```
factory. UseDefaultContext(Connection.CreateSqlServer("server", "database", "user-name", "password"));
```

```
public static IGateway SwitchContext (this IGateway _, string contextName )
```

Use it to make this context the active context when multiple contexts have been registered.

Returns → IGateway

The same factory instance; for fluent API.

Parameters

_	: The gateway instance {extension}
contextName	: Context name or identifier.

Example:

```
factory.SwitchContext("the-other-context-name");
```

public IGateway Register<TEntity>(params Expression<Func<TEntity, object>>[] keys)
Registers the given entity type in the model.

Returns → IGateway
The same factory instance; for fluent API.

Type Parameters
TEntity: Type of entity

Parameters

keys	: Properties that are to be registered as unique key.

Example:

```
factory.Register<Author>();
factory.Register<Book>(b => b.Id);
```

public IGateway Register<TEntity>(string container, params Expression<Func<TEntity, object>>[] keys)
Registers the given entity type in the model.

Returns → IGateway
The same factory instance; for fluent API.

Type Parameters
TEntity: Type of entity

Parameters

container	: Name of the container for No-SQL db (e.g., CosmosDb)
keys	: Properties that are to be registered as unique key.

Example:

```
factory.Register<Author>("my-container");
factory.Register<Book>("my-book", b => b.Id);
```

public IGateway Register<TEntity>(string collectionName, string container = null)
Registers the given entity type in the model, use it for MongoDB, DynamoDb.

Returns → IGateway
The same factory instance; for fluent API.

Type Parameters
TEntity: Type of entity

Parameters

collectionName	: Collection name of this entity.
container	: Name of the Database, if null then it will assume the default database.

Example:

```
factory.Register<Author>(collectionName: "Authors");
```

public IGateway SpecialKey(string key, SpecialKey keyType = TypeCatalogue.SpecialKey.PartitionKey)
Use this method to add special key to the entity registered in the previous Register call. This method must be a continued call, because the entity type is temporarily cached.

Returns → IGateway
The same factory instance; for fluent API.

Parameters

key	: Name of the key.
keyType	: Type of the key.

Example:

```
factory. SpecialKey("PartitionKey", TypeCatalogue.SpecialKey.PartitionKey);
```

public static List<string> GetContextNames (this IGateway _)
Call it to get names of all the registered contexts.

Returns → List<string>
List of context names.

Parameters

_	: The gateway instance {extension}

Example:

```
var contextsRegistered = factory.GetContextNames();
```

public static string GetActiveContext(this IGateway _)
Get the context name that is active at the moment.

Returns → string
Current context name.

Parameters

_	: The gateway instance {extension}

Example:

```
var activeContext = factory.GetActiveContext ();
```

public static IGateway UseErrorCallback (this IGateway _, Action<ErrorPack> callback)
Set a call back function to receive error when it occurs.

Returns → IGateway
The same factory instance; for fluent API.

Parameters

_	: The gateway instance {extension}
callback	: The call back function.

Example:

```
factory.UseErrorCallback (ShowError);

void ShowError(ErrorPack error) {…}
```

public static ErrorPack GetError (this IGateway _)
Get the Error object.

Returns → ErrorPack
Returns the error object.

Parameters

_	: The gateway instance {extension}

Example:

```
var error = factory.GetError ();
```

public static TQuery GetQueryService<TQuery>(this IGateway _)
    where TQuery : IQueryService
Gets the query service interface derived from IQueryService.

Returns → TQuery
Supported interfaces are:

1. IReadQuery<TSource>
2. IWriteQuery<TSource>
3. IMasterDetailQuery<TSource>
4. ISchemaQuery<TSource>
5. ISqlQuery.

Type Parameters

TQuery	: Type of query service.

Parameters

_	: The gateway instance {extension}

Example:

```
var reader = factory.GetQueryService<IReadQuery<Author>>();
var writer = factory.GetQueryService<IWriteQuery<Author>>();
```

public static IReadQuery<TSource> GetReader<TSource>(this IGateway _)
    where TSource : class, new()
Gets the reader query interface for the entity.

Returns → IReadQuery<TSource>
Read query service interface.

Type Parameters
TSource : Type of entity

Parameters
_ : The gateway instance {extension}

Example:

```
var reader = factory.GetReader<Author>();
```

public static IWriteQuery<TSource> GetWriter<TSource>(this IGateway _)
where TSource : class, new()
Gets the reader query interface for the entity.

Returns → IWriteQuery <TSource>
Write query service interface.

Type Parameters
TSource : Type of entity

Parameters
_ : The gateway instance {extension}

Example:

```
var writer = factory.GetWriter<Author>();
```

public static ISchemaQuery<TSource> GetSchema<TSource>(this IGateway _)
where TSource : class, new()
Gets the schema query interface for the entity.

Returns → ISchemaQuery<TSource>
Schema query service interface.

Type Parameters
TSource : Type of entity

Parameters
_ : The gateway instance {extension}

Example:

```
var schema = factory.GetSchema<Author>();
```

public static ISqlQuery GetSql(this IGateway _)
Gets the Sql query interface for the entity.

Returns → ISqlQuery

Sql query service interface.

Parameters

_	: The gateway instance {extension}

Example:

```
var sql = factory.GetSql();
```

public static IBridge **Bridge**(this IGateway _, string **rightContext**)

Creates a bridge (tunnel) between the current context (as left) and the right context.

Returns → IBridge

The instance of IBridge.

Parameters

_	: The gateway instance {extension}
rightContext	: Conext name to establish a data tunnel

Example:

```
var bridge = factory.Bridge("cosmos-context");
```

public static IBridge **Bridge**(this IGateway _, string **leftContext**, string **rightContext**)

Creates a bridge (tunnel) between the left context and the right context.

Returns → IBridge

The instance of IBridge.

Parameters

_	: The gateway instance {extension}
leftContext	: First context name to establish the data tunnel with
rightContext	: Second context name to establish the data tunnel with

Example:

```
var bridge = factory.Bridge("sql-context","cosmos-context");
```

public static void **UnBridge**(this IGateway _)

Remove bridging.

Parameters

_	: The gateway instance {extension}

Example:

```
factory.UnBridge();
```

IReadQuery<TSource> interface

Namespace : ATheory.UnifiedAccess.Data.Core

Service provider interface. The instance is returned from a call from IGateway.GetReader().

Methods [Instance/Extension]

```
public static TResult ExecQueryable<TSource, TResult>(this IReadQuery<TSource> _,
        Func<IQueryable<TSource>, TResult> func)
        where TSource : class, new()
```

Use it to execute any function supported by IQueryable.

Returns → TResult

Result of type TResult.

Type Parameters

TSource	: Type of entity
TResult	: Type of the returned result

Parameters

_	: The service instance {extension}.
func	: A linq function expression

Example:

```
var result = reader. ExecQueryable(q => q.FirstOrDefault(a => a.Id == 9));
```

```
public static TSource GetFirst<TSource>( this IReadQuery<TSource> _,  Expression<Func<TSource, bool>> predicate)
        where TSource : class, new()
```

Fetches the first record in the sequence.

Returns → TSource

The entity object.

Type Parameters

TSource	: Type of entity

Parameters

_	: The service instance {extension}.
predicate	: A function for a condition to filter elements. (s => s.Id).

Example:

```
var result = reader.GetFirst(a => a.Id == 88);
```

```
public static TSelect GetFirst<TSource, TSelect>( this IReadQuery<TSource> _,
        Expression<Func<TSource, bool>> predicate, Expression<Func<TSource, TSelect>> selector)
        where TSource : class, new()
```

Fetches the first record in the sequence and returns a DTO of TSelect type.

Returns → TSelect

The selector object.

Type Parameters

TSource	: Type of entity
TSelect	: Type of the DTO

Parameters

_	: The service instance {extension}.
predicate	: A function for a condition to filter elements. (s => s.Id).
selector	: A function to copy TSource elements to TSelect element.

Example:

```
var result = reader.GetFirst(a => a.Id == 88, s => new SelectType{Identifier = s.Id, Name = s.UserName});
```

public static TSource GetLast<TSource>(this IReadQuery<TSource> _, Expression<Func<TSource, bool>> predicate)
    where TSource : class, new()

Fetches the last record in the sequence.

Returns → TSource

The entity object.

Type Parameters

TSource	: Type of entity

Parameters

_	: The service instance {extension}.
predicate	: A function for a condition to filter elements. (s => s.Id).

Example:

```
var result = reader.GetLast(a => a.Id == 88);
```

public static TSelect GetLast<TSource, TSelect>(this IReadQuery<TSource> _,
    Expression<Func<TSource, bool>> predicate, Expression<Func<TSource, TSelect>> selector)
    where TSource : class, new()

Fetches the last record in the sequence and returns a DTO of TSelect type.

Returns → TSelect

The selector object.

Type Parameters

TSource	: Type of entity
TSelect	: Type of the DTO

Parameters

_	: The service instance {extension}.
predicate	: A function for a condition to filter elements. (s => s.Id).
selector	: A function to copy TSource elements to TSelect element.

Example:

```
var result = reader.GetLast(a => a.Id == 88, s => new SelectType{Identifier = s.Id, Name = s.UserName});
```

```
public static TSource GetTop<TSource, TKey>( this IReadQuery<TSource> _,
        Expression<Func<TSource, bool>> predicate,  Expression<Func<TSource, TKey>> keySelector)
        where TSource : class, new()
```
Fetches the first record in the sequence ordered by the key in descending order.

Returns → TSource

The entity object.

Type Parameters

TSource	: Type of entity
TKey	: Type of the selector key

Parameters

_	: The service instance {extension}.
predicate	: A function for a condition to filter elements. (s => s.Id).
keySelector	: A function for a property to order by.

Example:

```
var result = reader.GetTop(a => a.Id >= 5 && a.Id < 12, k => k.Id);
```

```
public static TSelect GetTop<TSource, TKey, TSelect>( this IReadQuery<TSource> _,
        Expression<Func<TSource, bool>> predicate,   Expression<Func<TSource, TKey>> keySelector,
        Expression<Func<TSource, TSelect>> selector)
        where TSource : class, new()
```
Fetches the first record in the sequence ordered by the key in descending order.

Returns → TSelect

The selector object.

Type Parameters

TSource	: Type of entity
TKey	: Type of the selector key
TSelect	: Type of the DTO

Parameters

_	: The service instance {extension}.
predicate	: A function for a condition to filter elements. (s => s.Id).
keySelector	: A function for a property to order by.
selector	: A function to copy TSource elements to TSelect element

Example:

```
var result = reader.GetTop(a => a.Id >= 5 && a.Id < 12, k => k.Id,
        s => new SelectType{Identifier = s.Id, Name = s.UserName});
```

```
public static TSource GetBottom<TSource, TKey>(this IReadQuery<TSource> _,
        Expression<Func<TSource, bool>> predicate,  Expression<Func<TSource, TKey>> keySelector)
        where TSource : class, new()
```
Fetches the first record in the sequence ordered by the key in ascending order.

Returns → TSource

The entity object.

Type Parameters

TSource	: Type of entity
TKey	: Type of the selector key

Parameters

_	: The service instance {extension}.
predicate	: A function for a condition to filter elements. (s => s.Id).
keySelector	: A function for a property to order by.

Example:

```
var result = reader.GetBottom(a => a.Id >= 5 && a.Id < 12, k => k.Id);
```

public static TSelect GetBottom<TSource, TKey, TSelect>(this IReadQuery<TSource> _,
Expression<Func<TSource, bool>> predicate, Expression<Func<TSource, TKey>> keySelector,
Expression<Func<TSource, TSelect>> selector)
where TSource : class, new()

Fetches the first record in the sequence ordered by the key in ascending order.

Returns → TSelect

The selector object.

Type Parameters

TSource	: Type of entity
TKey	: Type of the selector key
TSelect	: Type of the DTO

Parameters

_	: The service instance {extension}.
predicate	: A function for a condition to filter elements. (s => s.Id).
keySelector	: A function for a property to order by.
selector	: A function to copy TSource elements to TSelect element

Example:

```
var result = reader.GetBottom(a => a.Id >= 5 && a.Id < 12, k => k.Id,
        s => new SelectType{Identifier = s.Id, Name = s.UserName});
```

public static IList<TSource> GetList<TSource>(this IReadQuery<TSource> _,
Expression<Func<TSource, bool>> predicate = null)
where TSource : class, new()

Filters a sequence of TSource elements based on the predicate if one is provided, otherwise all.

Returns → IList<TSource>

The list of entity objects.

Type Parameters

TSource	: Type of entity

Parameters

_	: The service instance {extension}.
predicate	: A function for a condition to filter elements (s => s.Id) .

Example:

```
var result = reader.GetList(a => a.Id > 0 && a.Id <= 6);
```

**public static IList<TSelect> GetList<TSource, TSelect>(this IReadQuery<TSource> _,
Expression<Func<TSource, TSelect>> selector, Expression<Func<TSource, bool>> predicate = null)
where TSource : class, new()**

Filters a sequence of TSource elements based on the predicate if one is provided, otherwise all.

Returns → IList< TSelect>

The list of selector objects.

Type Parameters

TSource	: Type of entity

Parameters

_	: The service instance {extension}.
selector	: A function to copy TSource elements to TSelect element
predicate	: A function for a condition to filter elements (s => s.Id) .

Example:

```
var result = reader.GetList(s => new SelectType{Identifier = s.Id}, a => a.Id > 0 && a.Id <= 6);
```

**public static IList<TSource> GetOrderedList<TSource, TKey>(this IReadQuery<TSource> _,
Expression<Func<TSource, TKey>> keySelector, Expression<Func<TSource, bool>> predicate = null)
where TSource : class, new()**

Filters a sequence of TSource elements based on the predicate if one is provided, otherwise all. Orders by ascending order.

Returns → IList<TSource>

The list of entity objects.

Type Parameters

TSource	: Type of entity
TKey	: Type of the selector key

Parameters

_	: The service instance {extension}.
keySelector	: A function for a property to order by.
predicate	: A function for a condition to filter elements. (s => s.Id).

Example:

```
var result = reader.GetOrderedList(k => k.Name, a => a.Id > 0 && a.Id <= 6);
```

**public static IList<TSelect> GetOrderedList<TSource, TKey, TSelect>(this IReadQuery<TSource> _,
Expression<Func<TSource, TKey>> keySelector, Expression<Func<TSource, TSelect>> selector,
Expression<Func<TSource, bool>> predicate = null)**
where TSource : class, new()

Filters a sequence of TSource elements based on the predicate if one is provided, otherwise all. Orders by ascending order.

Returns → IList<TSelect>
The list of selector objects.

Type Parameters

TSource	: Type of entity
TKey	: Type of the selector key
TSelect	: Type of the DTO

Parameters

_	: The service instance {extension}.
keySelector	: A function for a property to order by.
selector	: A function to copy TSource elements to TSelect element
predicate	: A function for a condition to filter elements. (s => s.Id).

Example:

```
var result = reader.GetOrderedList(k => k.Name, s => new SelectType{Id = s.Id}, a => a.Id > 0 && a.Id <= 6);
```

```
public static IList<TSource> GetDescendingOrderedList<TSource, TKey>(this IReadQuery<TSource> _,
        Expression<Func<TSource, TKey>> keySelector,  Expression<Func<TSource, bool>> predicate = null)
        where TSource : class, new()
```
Filters a sequence of TSource elements based on the predicate if one is provided, otherwise all. Orders by desc.

Returns → IList<TSource>
The list of entity objects.

Type Parameters

TSource	: Type of entity
TKey	: Type of the selector key

Parameters

_	: The service instance {extension}.
keySelector	: A function for a property to order by.
predicate	: A function for a condition to filter elements. (s => s.Id).

Example:

```
var result = reader. GetDescendingOrderedList (k => k.Name, a => a.Id > 0 && a.Id <= 6);
```

```
public static IList<TSelect> GetDescendingOrderedList<TSource, TKey, TSelect>(this IReadQuery<TSource> _,
        Expression<Func<TSource, TKey>> keySelector, Expression<Func<TSource, TSelect>> selector,
        Expression<Func<TSource, bool>> predicate = null)
        where TSource : class, new()
```
Filters a sequence of TSource elements based on the predicate if one is provided, otherwise all. Orders by desc.

Returns → IList<TSelect>
The list of selector objects.

Type Parameters

TSource	: Type of entity

TKey	: Type of the selector key
TSelect	: Type of the DTO

Parameters

_	: The service instance {extension}.
keySelector	: A function for a property to order by.
selector	: A function to copy TSource elements to TSelect element
predicate	: A function for a condition to filter elements. (s => s.Id).

Example:

```
var result = reader. GetDescendingOrderedList (k => k.Name, s => new SelectType{Id = s.Id}, a => a.Id > 0 && a.Id <= 6);
```

```
public IList<TSource> GetRange<TSource>(this IReadQuery<TSource> _, (int from, int count) range,
        Expression<Func<TSource, bool>> predicate = null)
        where TSource : class, new()
```

Filters a sequence of TSource elements based on the predicate if one is provided otherwise all and returns only the elements within the range.

Returns → IList< TSource >
The list of entity objects.

Type Parameters

TSource	: Type of entity

Parameters

_	: The service instance {extension}.
range	: Range: from = 0 based element in the sequence; count = total number of elements.
predicate	: A function for a condition to filter elements. (s => s.Id).

Example:

```
var result = reader.GetRange((2, 3), a => a.Id > 0 && a.Id <= 6);
```

```
public IList<TSelect> GetRange<TSource, TSelect>(this IReadQuery<TSource> _, (int from, int count) range,
        Expression<Func<TSource, TSelect>> selector, Expression<Func<TSource, bool>> predicate = null)
        where TSource : class, new()
```

Filters a sequence of TSource elements based on the predicate if one is provided otherwise all and returns only the elements within the range.

Returns → IList< TSelect >
The list of selector objects.

Type Parameters

TSource	: Type of entity
TSelect	: Type of DTO

Parameters

_	: The service instance {extension}.
range	: Range: from = 0 based element in the sequence; count = total number of elements.
selector	: A function to copy TSource elements to TSelect element
predicate	: A function for a condition to filter elements. (s => s.Id).

Example:

```
var result = reader.GetRange((2, 3), s => new SelectType {Id = s.Id}, a => a.Id > 0 && a.Id <= 6);
```

**public IList<TSource> GetRangeOrderBy<TSource, TKey>(this IReadQuery<TSource> _, (int from, int count) range,
Expression<Func<TSource, TKey>> keySelector, Expression<Func<TSource, bool>> predicate = null)
where TSource : class, new()**

Filters a sequence of TSource elements based on the predicate if one is provided otherwise all and returns only the elements within the range.

Returns → IList< TSource >

The list of entity objects.

Type Parameters

TSource	: Type of entity

Parameters

_	: The service instance {extension}.
range	: Range: from = 0 based element in the sequence; count = total number of elements.
keySelector	: A function for a property to order by.
predicate	: A function for a condition to filter elements. (s => s.Id).

Example:

```
var result = reader.GetRangeOrderBy((2, 3), k => k.Id, a => a.Id > 0 && a.Id <= 6);
```

**public IList<TSelect> GetRangeOrderBy<TSource, TKey, TSelect>(this IReadQuery<TSource> _,
(int from, int count) range, Expression<Func<TSource, TKey>> keySelector,
Expression<Func<TSource, TSelect>> selector, Expression<Func<TSource, bool>> predicate = null)
where TSource : class, new()**

Filters a sequence of TSource elements based on the predicate if one is provided otherwise all and returns only the elements within the range.

Returns → IList< TSelect >

The list of entity objects.

Type Parameters

TSource	: Type of entity
TSelect	: Type of DTO

Parameters

_	: The service instance {extension}.
range	: Range: from = 0 based element in the sequence; count = total number of elements.
keySelector	: A function for a property to order by.
selector	: A function to copy TSource elements to TSelect element
predicate	: A function for a condition to filter elements. (s => s.Id).

Example:

```
var result = reader.GetRangeOrderBy((2, 3), k => k.Id, s => new SelectType{Id =s.Id}, a => a.Id > 0 && a.Id <= 6);
```

IWriteQuery<TSource> interface

Namespace : ATheory.UnifiedAccess.Data.Core

Service provider interface. The instance is returned from a call from IGateway. GetWriter ().

Methods [Instance/Extension]

```
public static bool Insert<TSource>(this IWriteQuery<TSource> _, TSource entity)
        where TSource : class, new()
```

Inserts a new entity in the database.

Returns → bool

Success or failure.

Type Parameters

TSource	: Type of entity

Parameters

_	: The service instance {extension}.
entity	: Entity to be pushed in to the database.

Example:

```
var result = writer.Insert(new Author { Id = 99, Name = "James Patterson", Description = "Crime - Thriller", Index = 99 });
```

```
public bool Update<TSource>(this IWriteQuery<TSource> _, TSource entity,
        params Expression<Func<TSource, object>>[] properties)
        where TSource : class, new()
```

Updates the entity, affects only the columns if specified in properties otherwise all. Not to be used for MongoDB.

Returns → bool

Success or failure.

Type Parameters

TSource	: Type of entity

Parameters

_	: The service instance {extension}.
entity	: Entity to be pushed in to the database.
properties	: Array of properties that would be updated, if none is provided the whole entity will be updated

Example:

```
author.Name =”Heinlein”;
author.Description =”Heinlein”;

//Update all
var result = writer.Update(author);
//Update only ‘Description’
var result = writer.Update(author, p=>p. Description);
```

public bool Update<TSource>(this IWriteQuery<TSource> _, Expression<Func<TSource, bool>> predicate,
TSource entity)
where TSource : class, new()

Updates the entity when the predicate matches.

Returns → bool

Success or failure.

Type Parameters

TSource	: Type of entity

Parameters

_	: The service instance {extension}.
predicate	: A function for a condition to update elements.
entity	: Entity to be pushed in to the database.

Example:

```
var result = writer.Update(a => a.Id == 3, new Author { Name = "Peter F. Hamilton", Index = 3 });
```

public bool Delete<TSource>(this IWriteQuery<TSource> _, TSource entity)
where TSource : class, new()

Deletes the entity from the database,

Returns → bool

Success or failure.

Type Parameters

TSource	: Type of entity

Parameters

_	: The service instance {extension}.
entity	: The entity to delete.

Example:

```
var result = writer.Delete(author);
```

public bool Delete<TSource>(this IWriteQuery<TSource> _, Expression<Func<TSource, bool>> predicate)
where TSource : class, new()

Delete the entity(s) from the database.

Returns → bool

Success or failure.

Type Parameters

TSource	: Type of entity

Parameters

_	: The service instance {extension}.
predicate	: A function for a condition to delete elements.

Example:

```
var result = writer.Delete(a => a.Id == 99);
```

public bool InsertBulk<TSource>(this IWriteQuery<TSource> _, IList<TSource> sources)
    where TSource : class, new()

Inserts in bulk, the entire list of entities.

Returns → bool

Success or failure.

Type Parameters

TSource	: Type of entity

Parameters

_	: The service instance {extension}.
sources	: List of TSource elements that'll be inserted in to the table.

Example:

```
var result = writer.InsertBulk( new List<Author> {
        new Author { Id = "55", Name = "Greg Bear", Description="Sci-fi", Index = 55 },
        new Author { Id = "66", Name = "Clark", Index = 66 },
        new Author { Id = "77", Name = "Gregory Benford", Description ="Science flick", Index = 77 },
        new Author { Id = "88", Name = "David Drake", Index = 88 },
        new Author { Id = "99", Name = "James Patterson", Index = 99 }
    });
```

ISqlQuery interface

Namespace : ATheory.UnifiedAccess.Data.Core

Service provider interface. The instance is returned from a call from IGateway. GetSql ().

Methods [Instance/Extension]

public static IList<TSource> GetList<TSource>(this ISqlQuery _, string sql, params object[] parameters)
Filters a sequence of TSource elements based on the sql statement, otherwise all.

Returns → IList<TSource>
List of TSource elements.

Type Parameters

TSource	: Type of entity

Parameters

_	: The service instance {extension}.
sql	: Select statement
parameters	: Parameters used in the query as sql param.

Example:

```
var result = sql.GetList<Author>("select Id, name from author where Id >= 3 and Id <= 12");
```

public static TSource GetFirst<TSource>(this ISqlQuery _, string sql, params object[] parameters)
Fetches the first record in the sequence.

Returns → TSource
TSource instance.

Type Parameters

TSource	: Type of entity

Parameters

_	: The service instance {extension}.
sql	: Select statement
parameters	: Parameters used in the query as sql param.

Example:

```
var result = sql.GetFirst<Author>("select Id, name from author where Id >= @idMin and Id <= @idMax",
        SqlHelper.Parameters.Get("@idMin",5), SqlHelper.Parameters.Get@idMax", 12));
```

public static TSource GetLast<TSource>(this ISqlQuery _, string sql, params object[] parameters)
Fetches the last record in the sequence.

Returns → TSource
TSource instance.

Type Parameters

TSource	: Type of entity

Parameters

_	: The service instance {extension}.
sql	: Select statement
parameters	: Parameters used in the query as sql param.

Example:

```
var result = sql.GetLast<Author>("select Id, name from author where Id >= @idMin and Id <= @idMax",
        SqlHelper.Parameters.Get("@idMin",5), SqlHelper.Parameters.Get@idMax", 12));
```

public static bool Execute(this ISqlQuery _, string sql, params object[] parameters)
Non-Select queries: Insert/Update/Delete.

Returns → bool
Success or failure.

Parameters

_	: The service instance {extension}.
sql	: Select statement
parameters	: Parameters used in the query as sql param.

Example:

```
var result = sql. Execute("insert into author (name, description) values (@name, @description)",
        SqlHelper.Parameters.Get("@name","Isaac"), SqlHelper.Parameters.Get("@description", "sci-fi"));
```

public static DataTable GetTableForBulkInsertion(this ISqlQuery _, string tableName)
Creates the DataTable instance that would be used to populate with data for bulk insertion. InsertBulk(dataTable) method.

Returns → DataTable
Instance of the DataTable.

Parameters

_	: The service instance {extension}.
tableName	: Name of table, schema must be included

Example:

```
var result = sql.GetTableForBulkInsertion ("author"));
```

public static bool InsertBulk(this ISqlQuery _, DataTable dataTable)
Bulk inserts in to the table.

Returns → bool
Success or failure.

Parameters

_	: The service instance {extension}.
dataTable	: DataTable instance.

Example:

```
var result = sql.InsertBulk(dataTable);
```

ISchemaQuery<TSource> interface

Namespace : ATheory.UnifiedAccess.Data.Core

Service provider interface. The instance is returned from a call from IGateway. GetSchema ().

Methods [Instance/Extension]

```
public static bool CreateSchema<TSource>(this ISchemaQuery<TSource> _)
        where TSource : class, new()
```

Deletes table (sql) or schema (non-sql) based on the entity.

Returns → bool

Success or failure.

Type Parameters

TSource	: Type of entity

Parameters

_	: The service instance {extension}.

Example:

```
var result = schema.CreateSchema();
```

```
public static bool DeleteSchema<TSource>(this ISchemaQuery<TSource> _)
        where TSource : class, new()
```

Creates table (sql) or schema (non-sql) based on the entity.

Returns → bool

Success or failure.

Type Parameters

TSource	: Type of entity

Parameters

_	: The service instance {extension}.

Example:

```
var result = schema.DeleteSchema();
```

```
public static bool UpdateSchema<TSource>(this ISchemaQuery<TSource> _)
        where TSource : class, new()
```

Updates the schema (add/delete) column/attribute.

Returns → bool

Success or failure.

Type Parameters

TSource	: Type of entity

Parameters

_ : The service instance {extension}.

Example:

```
var result = schema.UpdateSchema();
```

IBridge interface

Namespace : ATheory.UnifiedAccess.Data.Core

Service provider interface. The instance is returned from a call from IGateway. Bridge ().

Methods [Instance/Extension]

```
public BridgeResult Push<TLeft, TRight>(this IBridge _, Expression<Func<TLeft, bool>> predicate,
        Expression<Func<TLeft, TRight>> projection)
        where TLeft : class, new()
        where TRight : class, new()
```

Push the first result to the right context.

Returns → BridgeResult
One of BridgeResult.

Type Parameters

TLeft	: Type of entity used in the left context
TRight	: Type of entity used in the right context

Parameters

_	: The service instance {extension}.
predicate	: A function for a condition to filter elements. (s => s.Id)
projection	: A function to copy convert TLeft element to TRight element.

Example:

```
var result = bridge.Push<Author, AuthorMongo>(a => a.Id == 8,
            (s) => new AuthorMongo { Name = s.Name, Description = s.description })
```

```
public BridgeResult PushMany<TLeft, TRight>(this IBridge _, Expression<Func<TLeft, bool>> predicate,
        Expression<Func<TLeft, TRight>> projection)
        where TLeft : class, new()
        where TRight : class, new()
```

Push the list result to the right context.

Returns → BridgeResult
One of BridgeResult.

Type Parameters

TLeft	: Type of entity used in the left context
TRight	: Type of entity used in the right context

Parameters

_	: The service instance {extension}.
predicate	: A function for a condition to filter elements. (s => s.Id)
projection	: A function to copy convert TLeft element to TRight element.

Example:

```
var result = bridge.PushMany<Author, AuthorMongo>(a => a.Id  < 8,
            (s) => new AuthorMongo { Name = s.Name, Description = s.description });
```

```
public BridgeResult Pull<TLeft, TRight>(this IBridge _, Expression<Func<TRight, bool>> predicate,
        Expression<Func<TRight, TLeft>> projection)
        where TLeft : class, new()
        where TRight : class, new()
```
Push the first result from the right context and pushes it to the left context.

Returns → BridgeResult
One of BridgeResult.

Type Parameters

TLeft	: Type of entity used in the left context
TRight	: Type of entity used in the right context

Parameters

_	: The service instance {extension}.
predicate	: A function for a condition to filter elements. (s => s.Id)
projection	: A function to copy convert TRight element to TLeft element.

Example:

```
var result = bridge.Pull<Author, AuthorMongo>(a => a.Index == 4,
            (s) => new Author { Name = s.Name, description = s.Description,index = s.Index + 20 });
```

```
public BridgeResult PullMany<TLeft, TRight>(this IBridge _, Expression<Func<TRight, bool>> predicate,
        Expression<Func<TRight, TLeft>> projection)
        where TLeft : class, new()
        where TRight : class, new()
```
Push the all the results from the right context and pushes them to the left context.

Returns → BridgeResult
One of BridgeResult.

Type Parameters

TLeft	: Type of entity used in the left context
TRight	: Type of entity used in the right context

Parameters

_	: The service instance {extension}.
predicate	: A function for a condition to filter elements. (s => s.Id)
projection	: A function to copy convert TRight element to TLeft element.

Example:

```
var result = bridge.PullMany<Author, AuthorMongo>(a => a.Index < 4,
            (s) => new Author { Name = s.Name, description = s.Description,index = s.Index + 20 });
```

Complete Sample

Provider settings and registration

```
IGateway factory;        // member var

//Server settings and registering models, that’s all. For other providers, the only different would be
//connection and nothing else.
factory = EntityUnifier.Factory()
        .UseDefaultContext(Connection.CreateMongo("localhost", "bookdb"))
        .Register<Author>()
        .Register<Book>(b => b.Id);
```

Schema

```
var auther = factory.GetSchema<Author>();
var result = auther.CreateSchema();
```

Reading

```
var auther = factory.GetReader<Author>();
var result = auther.GetFirst(a => a.Index > 2);

/* For a raw SQL */
var sql = factory.GetSql();
var result = sql.GetLast<Author>("select Id, name from author where Id >= @idMin and Id <= @idMax",
SqlHelper.Parameters.Get("@idMin",5), SqlHelper.Parameters.Get@idMax", 12))
```

Writing

```
var auther = factory.GetWriter<Author>();
var result = auther. Insert(new Author { Name = "Isaac Asimov", Description = "Sci-fi", Index = 4 });
```

Bridging

```
var bridge = factory.Bridge(“cosmos-context”);
var result = bridge.Push<Author, AuthorMongo>(a => a.Id == 8,
 (s) => new AuthorMongo { Name = s.Name, Description = s.description })
```

Raw SQL (Sql Server/Sql express)

```
var sql = factory.GetSql();

//Read
var result = sql.GetLast<Author>("select Id, name from author where Id >= @idMin and Id <= @idMax",
SqlHelper.Parameters.Get("@idMin",5), SqlHelper.Parameters.Get@idMax", 12));

//Write
var result = sql.Execute("insert into author (name, description) values (@name, @description)",
SqlHelper.Parameters.Get("@name","Isaac"), SqlHelper.Parameters.Get("@description", "sci-fi"));
```