Namespace Solstice.Repository

Classes

UnitOfWork<TDbContext>

Interfaces

IUnitOfWork

Interface IUnitOfWork

Namespace: Solstice.Repository

Assembly: Solstice.Repository.dll

```
public interface IUnitOfWork : IDisposable
```

Inherited Members
IDisposable.Dispose()

Methods

GetRepository<TRepository, TEntity>()

```
TRepository GetRepository<TRepository, TEntity>() where TRepository :
ICoreRepository<TEntity> where TEntity : class
```

Returns

TRepository

Type Parameters

`TRepository`

`TEntity`

Class UnitOfWork<TDbContext>

Namespace: Solstice.Repository

Assembly: Solstice.Repository.dll

```
public sealed class UnitOfWork<TDbContext> : IUnitOfWork, IDisposable where
TDbContext : DbContext
```

Type Parameters

`TDbContext`

Inheritance

object ← UnitOfWork<TDbContext>

Implements

IUnitOfWork, IDisposable

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

UnitOfWork(TDbContext, IHttpContextAccessor)

```
public UnitOfWork(TDbContext dbContext, IHttpContextAccessor httpContext)
```

Parameters

`dbContext` TDbContext

`httpContext` IHttpContextAccessor

Methods

Dispose()

Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources.

```
public void Dispose()
```

Namespace Solstice.Repository.Attributes

Classes

QueryAttribute

Represents a custom attribute. This attribute is used to indicate which classes are to be used for database queries. It is a sealed class derived from the Attribute class, hence no other class can inherit from it.

RepositoryAttribute

The 'RepositoryAttribute' class, derived from 'Attribute', is a custom attribute class. It may be used to annotate repository classes. This class is 'sealed' which prevents further derivation.

Class QueryAttribute

Namespace: Solstice.Repository.Attributes

Assembly: Solstice.Repository.dll

Represents a custom attribute. This attribute is used to indicate which classes are to be used for database queries. It is a sealed class derived from the Attribute class, hence no other class can inherit from it.

```
[AttributeUsage(AttributeTargets.Class)]
public sealed class QueryAttribute : Attribute
```

Inheritance

object ← Attribute ← QueryAttribute

Inherited Members

Attribute.Equals(object) , Attribute.GetCustomAttribute(Assembly, Type) ,
Attribute.GetCustomAttribute(Assembly, Type, bool) ,
Attribute.GetCustomAttribute(MemberInfo, Type) ,
Attribute.GetCustomAttribute(MemberInfo, Type, bool) ,
Attribute.GetCustomAttribute(Module, Type) ,
Attribute.GetCustomAttribute(Module, Type, bool) ,
Attribute.GetCustomAttribute(ParameterInfo, Type) ,
Attribute.GetCustomAttribute(ParameterInfo, Type, bool) ,
Attribute.GetCustomAttributes(Assembly) ,
Attribute.GetCustomAttributes(Assembly, bool) ,
Attribute.GetCustomAttributes(Assembly, Type) ,
Attribute.GetCustomAttributes(Assembly, Type, bool) ,
Attribute.GetCustomAttributes(MemberInfo) ,
Attribute.GetCustomAttributes(MemberInfo, bool) ,
Attribute.GetCustomAttributes(MemberInfo, Type) ,
Attribute.GetCustomAttributes(MemberInfo, Type, bool) ,
Attribute.GetCustomAttributes(Module) , Attribute.GetCustomAttributes(Module, bool) ,
Attribute.GetCustomAttributes(Module, Type) ,
Attribute.GetCustomAttributes(Module, Type, bool) ,
Attribute.GetCustomAttributes(ParameterInfo) ,
Attribute.GetCustomAttributes(ParameterInfo, bool) ,
Attribute.GetCustomAttributes(ParameterInfo, Type) ,
Attribute.GetCustomAttributes(ParameterInfo, Type, bool) , Attribute.GetHashCode() ,

Attribute.IsDefaultAttribute() , Attribute.IsDefined(Assembly, Type) , Attribute.IsDefined(Assembly, Type, bool) , Attribute.IsDefined(MemberInfo, Type) , Attribute.IsDefined(MemberInfo, Type, bool) , Attribute.IsDefined(Module, Type) , Attribute.IsDefined(Module, Type, bool) , Attribute.IsDefined(ParameterInfo, Type) , Attribute.IsDefined(ParameterInfo, Type, bool) , Attribute.Match(object) , Attribute.TypeId , object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Class RepositoryAttribute

Namespace: Solstice.Repository.Attributes

Assembly: Solstice.Repository.dll

The 'RepositoryAttribute' class, derived from 'Attribute', is a custom attribute class. It may be used to annotate repository classes. This class is 'sealed' which prevents further derivation.

```
[AttributeUsage(AttributeTargets.Class)]
public sealed class RepositoryAttribute : Attribute
```

Inheritance

object ← Attribute ← RepositoryAttribute

Inherited Members

Attribute.Equals(object) , Attribute.GetCustomAttribute(Assembly, Type) ,
Attribute.GetCustomAttribute(Assembly, Type, bool) ,
Attribute.GetCustomAttribute(MemberInfo, Type) ,
Attribute.GetCustomAttribute(MemberInfo, Type, bool) ,
Attribute.GetCustomAttribute(Module, Type) ,
Attribute.GetCustomAttribute(Module, Type, bool) ,
Attribute.GetCustomAttribute(ParameterInfo, Type) ,
Attribute.GetCustomAttribute(ParameterInfo, Type, bool) ,
Attribute.GetCustomAttributes(Assembly) ,
Attribute.GetCustomAttributes(Assembly, bool) ,
Attribute.GetCustomAttributes(Assembly, Type) ,
Attribute.GetCustomAttributes(Assembly, Type, bool) ,
Attribute.GetCustomAttributes(MemberInfo) ,
Attribute.GetCustomAttributes(MemberInfo, bool) ,
Attribute.GetCustomAttributes(MemberInfo, Type) ,
Attribute.GetCustomAttributes(MemberInfo, Type, bool) ,
Attribute.GetCustomAttributes(Module) , Attribute.GetCustomAttributes(Module, bool) ,
Attribute.GetCustomAttributes(Module, Type) ,
Attribute.GetCustomAttributes(Module, Type, bool) ,
Attribute.GetCustomAttributes(ParameterInfo) ,
Attribute.GetCustomAttributes(ParameterInfo, bool) ,
Attribute.GetCustomAttributes(ParameterInfo, Type) ,
Attribute.GetCustomAttributes(ParameterInfo, Type, bool) , Attribute.GetHashCode() ,

Attribute.IsDefaultAttribute() , Attribute.IsDefined(Assembly, Type) , Attribute.IsDefined(Assembly, Type, bool) , Attribute.IsDefined(MemberInfo, Type) , Attribute.IsDefined(MemberInfo, Type, bool) , Attribute.IsDefined(Module, Type) , Attribute.IsDefined(Module, Type, bool) , Attribute.IsDefined(ParameterInfo, Type) , Attribute.IsDefined(ParameterInfo, Type, bool) , Attribute.Match(object) , Attribute.TypeId , object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Namespace Solstice.Repository.Core

Classes

CoreRepositoryExtension

CoreRepository<T, TContext>

An abstract class for the CoreRepository. Constructs a repository with a given context and Http context.

Interfaces

ICoreRepository<T>

The ICoreRepository interface provides methods for performing CRUD operations, querying, counting, paging, transactions, and other tasks on an underlying data repository in an asynchronous manner. The repository holds objects of a type. The methods in this interface produce or consume tasks that represent ongoing work and are used for structuring asynchronous code.

Class CoreRepositoryExtension

Namespace: Solstice.Repository.Core

Assembly: Solstice.Repository.dll

```
public static class CoreRepositoryExtension
```

Inheritance

object ← CoreRepositoryExtension

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

ToCollectionAsync<T>(IQueryable<T>, CancellationToken)

Converts provided IQueryable of entities into a list of entities in an asynchronous manner, respecting the provided cancellation token.

```
public static Task<ICollection<T>> ToCollectionAsync<T>(this IQueryable<T> query,
CancellationToken cancellationToken)
```

Parameters

`query` IQueryable<T>

The IQueryable of entities to be converted into a list

`cancellationToken` CancellationToken

The cancellation token

Returns

Task<ICollection<T>>

A task that represents the asynchronous operation, with a return value of the list containing the entities

Type Parameters

T

The type of entities

Class CoreRepository<T, TContext>

Namespace: Solstice.Repository.Core

Assembly: Solstice.Repository.dll

An abstract class for the CoreRepository. Constructs a repository with a given context and Http context.

```
public class CoreRepository<T, TContext> : ICoreRepository<T> where T : class where
TContext : DbContext
```

Type Parameters

`T`

A entity type

`TContext`

The DbContext type

Inheritance

object ← CoreRepository<T, TContext>

Implements

ICoreRepository<T>

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

CoreRepository(TContext, IHttpContextAccessor)

Constructs a new instance of CoreRepository with the given context and Http context.

```
protected CoreRepository(TContext dbContext, IHttpContextAccessor httpContext)
```

Parameters

`dbContext` TContext

The DbContext to use.

`httpContext` IHttpContextAccessor

The HttpContext to use for cancellations.

Methods

AddAndSaveAsync(T)

```
public Task AddAndSaveAsync(T entity)
```

Parameters

`entity` T

Returns

Task

AddAndSaveAsync<TEntity>(TEntity)

```
public Task AddAndSaveAsync<TEntity>(TEntity entity) where TEntity : class
```

Parameters

`entity` TEntity

Returns

Task

Type Parameters

`TEntity`

AddAsync(T)

Adds the given entity to the database and saves the changes.

```
public Task AddAsync(T entity)
```

Parameters

`entity` T

The entity to add.

Returns

Task

AddAsync<TEntity>(TEntity)

```
public Task AddAsync<TEntity>(TEntity entity) where TEntity : class
```

Parameters

`entity` TEntity

Returns

Task

Type Parameters

`TEntity`

AddRangeAndSaveAsync(ICollection<T>)

```
public Task AddRangeAndSaveAsync(ICollection<T> entities)
```

Parameters

`entities` ICollection<T>

Returns

Task

AddRangeAndSaveAsync<TEntity>(ICollection<TEntity>)

```
public Task AddRangeAndSaveAsync<TEntity>(ICollection<TEntity> entities) where
TEntity : class
```

Parameters

`entities` ICollection<TEntity>

Returns

Task

Type Parameters

`TEntity`

AddRangeAsync(ICollection<T>)

Adds a range of entities to the database and saves the changes.

```
public Task AddRangeAsync(ICollection<T> entities)
```

Parameters

`entities` ICollection<T>

The entities to add.

Returns

Task

AddRangeAsync<TEntity>(ICollection<TEntity>)

```
public Task AddRangeAsync<TEntity>(ICollection<TEntity> entities) where TEntity
: class
```

Parameters

`entities` ICollection<TEntity>

Returns

Task

Type Parameters

`TEntity`

AnyAsyncBy(Expression<Func<T, bool>>)

Checks if any entities in the database match the given expression.

```
public Task<bool> AnyAsyncBy(Expression<Func<T, bool>> where)
```

Parameters

`where` Expression<Func<T, bool>>

The expression to evaluate.

Returns

Task<bool>

A task that represents the asynchronous operation. The task result contains a boolean indicating whether any entities match the expression.

AnyAsyncBy<TEntity>(Expression<Func<TEntity, bool>>)

```
public Task<bool> AnyAsyncBy<TEntity>(Expression<Func<TEntity, bool>> where) where TEntity : class
```

Parameters

`where` Expression<Func<TEntity, bool>>

Returns

Task<bool>

Type Parameters

`TEntity`

AnyAsyncBy<TEntity>(IQueryable<TEntity>)

```
public Task<bool> AnyAsyncBy<TEntity>(IQueryable<TEntity> queryable) where TEntity : class
```

Parameters

`queryable` IQueryable<TEntity>

Returns

Task<bool>

Type Parameters

TEntity

BeginTransactionAsync()

Begins a new transaction in the database.

```
public Task<IDbContextTransaction> BeginTransactionAsync()
```

Returns

Task<IDbContextTransaction>

A task that represents the asynchronous operation. The task result contains the IDbContextTransaction that encapsulates all changes made to the DbContext within the transaction.

CountAllAsync()

Counts all entities in the database.

```
public Task<decimal> CountAllAsync()
```

Returns

Task<decimal>

A task that represents the asynchronous operation. The task result contains the count of all entities.

CountAllAsyncBy(Expression<Func<T, bool>>?)

Counts all entities in the database that satisfy the given expression.

```
public Task<decimal> CountAllAsyncBy(Expression<Func<T, bool>>? where)
```

Parameters

`where` Expression<Func<T, bool>>

The expression to evaluate.

Returns

Task<decimal>

A task that represents the asynchronous operation. The task result contains the count of entities that satisfy the expression.

CountAllAsyncBy<TEntity>(Expression<Func<TEntity, bool>>)

```
public Task<decimal> CountAllAsyncBy<TEntity>(Expression<Func<TEntity, bool>> where)
where TEntity : class
```

Parameters

`where` Expression<Func<TEntity, bool>>

Returns

Task<decimal>

Type Parameters

`TEntity`

CountAllAsync<TEntity>()

Counts all entities in the database.

```
public Task<decimal> CountAllAsync<TEntity>() where TEntity : class
```

Returns

Task<decimal>

A task that represents the asynchronous operation. The task result contains the count of all entities.

Type Parameters

`TEntity`

ExecuteQuery(string, ICollection<DbParameter>?)

```
public Task ExecuteQuery(string query, ICollection<DbParameter>? dbParameters)
```

Parameters

`query` string

`dbParameters` ICollection<DbParameter>

Returns

Task

FindAsync(int)

```
public Task<T> FindAsync(int id)
```

Parameters

`id` int

Returns

Task<T>

FindAsync<TEntity>(int)

```
public Task<TEntity> FindAsync<TEntity>(int id) where TEntity : class
```

Parameters

`id` int

Returns

Task<TEntity>

Type Parameters

`TEntity`

GetAllAsync()

Gets all entities from the database.

```
public Task<ICollection<T>> GetAllAsync()
```

Returns

Task<ICollection<T>>

A task that represents the asynchronous operation. The task result contains the list of all entities.

GetAllAsync(ICoreSpecifications<T>?)

Gets all entities from the database that satisfy The core specifications.

```
public Task<ICollection<T>> GetAllAsync(ICoreSpecifications<T>? coreSpecifications)
```

Parameters

`coreSpecifications` ICoreSpecifications<T>

The core specifications to evaluate.

Returns

Task<ICollection<T>>

A task that represents the asynchronous operation. The task result contains the list of entities that satisfy the specifications.

GetAllAsync(Expression<Func<T, bool>>)

Gets all entities from the database that satisfy the given expression.

```
public Task<ICollection<T>> GetAllAsync(Expression<Func<T, bool>> where)
```

Parameters

`where` Expression<Func<T, bool>>

The expression to evaluate.

Returns

Task<ICollection<T>>

A task that represents the asynchronous operation. The task result contains the list of entities that satisfy the expression.

GetAllAsync(string, ICoreSpecifications<T>?)

Gets all entities from the database based on the provided query and specifications.

```
public Task<ICollection<T>> GetAllAsync(string query, ICoreSpecifications<T>?
coreSpecifications)
```

Parameters

`query` string

The SQL query to execute.

`coreSpecifications` ICoreSpecifications<T>

The core specifications to evaluate.

Returns

Task<ICollection<T>>

A task that represents the asynchronous operation. The task result contains the list of entities fetched based on the query and specifications.

GetAllAsync(string, ICollection<DbParameter>, ICoreSpecifications<T>?)

Gets all entities from the database based on the provided query, parameters and specifications.

```
public Task<ICollection<T>> GetAllAsync(string query, ICollection<DbParameter>
parameters, ICoreSpecifications<T>? coreSpecifications)
```

Parameters

`query` string

The SQL query to execute.

`parameters` ICollection<DbParameter>

The SQL parameters needed for the query.

`coreSpecifications` ICoreSpecifications<T>

The core specifications to evaluate.

Returns

Task<ICollection<T>>

A task that represents the asynchronous operation. The task result contains the list of entities fetched based on the query, parameters, and specifications.

GetAllAsync<TEntity>()

```
public Task<ICollection<TEntity>> GetAllAsync<TEntity>() where TEntity : class
```

Returns

Task<ICollection<TEntity>>

Type Parameters

`TEntity`

GetAllAsync<TEntity>(ICoreSpecifications<TEntity>?)

```
public Task<ICollection<TEntity>> GetAllAsync<TEntity>(ICoreSpecifications<TEntity>?
coreSpecifications) where TEntity : class
```

Parameters

`coreSpecifications` ICoreSpecifications<TEntity>

Returns

Task<ICollection<TEntity>>

Type Parameters

`TEntity`

GetAllAsync<TEntity>(Expression<Func<TEntity, bool>>)

```
public Task<ICollection<TEntity>> GetAllAsync<TEntity>(Expression<Func<TEntity,
bool>> where) where TEntity : class
```

Parameters

`where` Expression<Func<TEntity, bool>>

Returns

Task<ICollection<TEntity>>

Type Parameters

`TEntity`

GetAllAsync<TEntity>(string)

```
public Task<ICollection<TEntity>> GetAllAsync<TEntity>(string query) where TEntity
: class
```

Parameters

`query` string

Returns

Task<ICollection<TEntity>>

Type Parameters

`TEntity`

GetAllAsync<TEntity>(string, ICoreSpecifications<TEntity>?)

```
public Task<ICollection<TEntity>> GetAllAsync<TEntity>(string query,
ICoreSpecifications<TEntity>? coreSpecifications) where TEntity : class
```

Parameters

`query` string

`coreSpecifications` ICoreSpecifications<TEntity>

Returns

Task<ICollection<TEntity>>

Type Parameters

`TEntity`

GetAllAsync<TEntity>(string, ICollection<DbParameter>)

```
public Task<ICollection<TEntity>> GetAllAsync<TEntity>(string query,
ICollection<DbParameter> parameters) where TEntity : class
```

Parameters

`query` string

`parameters` ICollection<DbParameter>

Returns

Task<ICollection<TEntity>>

Type Parameters

`TEntity`

GetAllAsync<TEntity>(string, ICollection<DbParameter>, ICoreSpecifications<TEntity>?)

```
public Task<ICollection<TEntity>> GetAllAsync<TEntity>(string query,
ICollection<DbParameter> parameters, ICoreSpecifications<TEntity>?
coreSpecifications) where TEntity : class
```

Parameters

`query` string

`parameters` ICollection<DbParameter>

`coreSpecifications` ICoreSpecifications<TEntity>

Returns

Task<ICollection<TEntity>>

Type Parameters

`TEntity`

GetAllByQueryable<TEntity>(IQueryable<TEntity>)

Gets all entities from the database by executing the provided IQueryable query.

```
public Task<ICollection<TEntity>> GetAllByQueryable<TEntity>
(IQueryable<TEntity> query)
```

Parameters

`query` IQueryable<TEntity>

The IQueryable query to execute.

Returns

Task<ICollection<TEntity>>

A task that represents the asynchronous operation. The task result contains the list of entities fetched by executing the query.

Type Parameters

`TEntity`

GetAllQueryable()

Get all entities as IQueryable.

```
public IQueryable<T> GetAllQueryable()
```

Returns

IQueryable<T>

IQueryable of all entities in the database.

GetAllQueryable(ICoreSpecifications<T>?)

Gets all entities from the database satisfying the specifications provided.

```
public IQueryable<T> GetAllQueryable(ICoreSpecifications<T>? coreSpecifications)
```

Parameters

`coreSpecifications` ICoreSpecifications<T>

The core specifications to evaluate.

Returns

IQueryable<T>

IQueryable of entities satisfying the specifications.

GetAllQueryable(Expression<Func<T, bool>>)

Gets all entities from the database that matches the given expression as IQueryable.

```
public IQueryable<T> GetAllQueryable(Expression<Func<T, bool>> where)
```

Parameters

`where` Expression<Func<T, bool>>

The expression to evaluate.

Returns

IQueryable<T>

IQueryable of entities that match the expression.

GetAllQueryable(string)

Gets entities from the database based on the SQL query and specifications provided.

```
public IQueryable<T> GetAllQueryable(string query)
```

Parameters

`query` string

The SQL query to execute.

Returns

IQueryable<T>

IQueryable of entities fetched based on the query and specifications.

GetAllQueryable(string, ICoreSpecifications<T>?)

Gets entities from the database based on the SQL query and specifications provided.

```
public IQueryable<T> GetAllQueryable(string query, ICoreSpecifications<T>?
coreSpecifications)
```

Parameters

`query` string

The SQL query to execute.

`coreSpecifications` ICoreSpecifications<T>

The core specifications to evaluate.

Returns

IQueryable<T>

IQueryable of entities fetched based on the query and specifications.

GetAllQueryable(string, ICollection<DbParameter>)

Gets entities from the database based on the SQL query, parameters, and specifications provided.

```
public IQueryable<T> GetAllQueryable(string query, ICollection<DbParameter>
parameters)
```

Parameters

`query` string

The SQL query to execute.

`parameters` ICollection<DbParameter>

The SQL parameters needed for the query.

Returns

IQueryable<T>

IQueryable of entities fetched based on the query, parameters, and specifications.

GetAllQueryable(string, ICollection<DbParameter>, ICoreSpecifications<T>?)

Gets entities from the database based on the SQL query, parameters, and specifications provided.

```
public IQueryable<T> GetAllQueryable(string query, ICollection<DbParameter>
parameters, ICoreSpecifications<T>? coreSpecifications)
```

Parameters

`query` string

The SQL query to execute.

`parameters` ICollection<DbParameter>

The SQL parameters needed for the query.

`coreSpecifications` ICoreSpecifications<T>

The core specifications to evaluate.

Returns

IQueryable<T>

IQueryable of entities fetched based on the query, parameters, and specifications.

GetAllQueryable<TEntity>()

Get all entities as IQueryable.

```
public IQueryable<TEntity> GetAllQueryable<TEntity>() where TEntity : class
```

Returns

IQueryable<TEntity>

IQueryable of all entities in the database.

Type Parameters

`TEntity`

GetAllQueryable<TEntity>(ICoreSpecifications<TEntity>?)

Gets all entities from the database satisfying the specifications provided.

```
public IQueryable<TEntity> GetAllQueryable<TEntity>(ICoreSpecifications<TEntity>?
coreSpecifications) where TEntity : class
```

Parameters

`coreSpecifications` ICoreSpecifications<TEntity>

The core specifications to evaluate.

Returns

IQueryable<TEntity>

IQueryable of entities satisfying the specifications.

Type Parameters

`TEntity`

GetAllQueryable<TEntity>(Expression<Func<TEntity, bool>>)

Gets all entities from the database that matches the given expression as IQueryable.

```
public IQueryable<TEntity> GetAllQueryable<TEntity>(Expression<Func<TEntity, bool>>
where) where TEntity : class
```

Parameters

`where` Expression<Func<TEntity, bool>>

The expression to evaluate.

Returns

IQueryable<TEntity>

IQueryable of entities that match the expression.

Type Parameters

`TEntity`

GetAllQueryable<TEntity>(string)

Gets entities from the database based on the SQL query and specifications provided.

```
public IQueryable<TEntity> GetAllQueryable<TEntity>(string query) where TEntity
: class
```

Parameters

`query` string

The SQL query to execute.

Returns

IQueryable<TEntity>

IQueryable of entities fetched based on the query and specifications.

Type Parameters

`TEntity`

GetAllQueryable<TEntity>(string, ICoreSpecifications<TEntity>?)

Gets entities from the database based on the SQL query and specifications provided.

```
public IQueryable<TEntity> GetAllQueryable<TEntity>(string query,
ICoreSpecifications<TEntity>? coreSpecifications) where TEntity : class
```

Parameters

`query` string

The SQL query to execute.

`coreSpecifications` ICoreSpecifications<TEntity>

The core specifications to evaluate.

Returns

IQueryable<TEntity>

IQueryable of entities fetched based on the query and specifications.

Type Parameters

`TEntity`

GetAllQueryable<TEntity>(string, ICollection<DbParameter>)

Gets entities from the database based on the SQL query, parameters, and specifications provided.

```
public IQueryable<TEntity> GetAllQueryable<TEntity>(string query,
ICollection<DbParameter> parameters) where TEntity : class
```

Parameters

`query` string

The SQL query to execute.

`parameters` ICollection<DbParameter>

The SQL parameters needed for the query.

Returns

IQueryable<TEntity>

IQueryable of entities fetched based on the query, parameters, and specifications.

Type Parameters

`TEntity`

GetAllQueryable<TEntity>(string, ICollection<DbParameter>, ICoreSpecifications<TEntity>?)

Gets entities from the database based on the SQL query, parameters, and specifications provided.

```
public IQueryable<TEntity> GetAllQueryable<TEntity>(string query,
ICollection<DbParameter> parameters, ICoreSpecifications<TEntity>?
coreSpecifications) where TEntity : class
```

Parameters

`query` string

The SQL query to execute.

`parameters` ICollection<DbParameter>

The SQL parameters needed for the query.

`coreSpecifications` ICoreSpecifications<TEntity>

The core specifications to evaluate.

Returns

IQueryable<TEntity>

IQueryable of entities fetched based on the query, parameters, and specifications.

Type Parameters

`TEntity`

GetBy(ICoreSpecifications<T>?)

Gets the first entity that satisfies the provided specifications.

```
public Task<T?> GetBy(ICoreSpecifications<T>? coreSpecifications)
```

Parameters

`coreSpecifications` ICoreSpecifications<T>

The specifications to evaluate.

Returns

Task<T>

A task that represents the asynchronous operation. The task result contains the first entity that satisfies the specifications or null if no such entity exists.

GetBy(Expression<Func<T, bool>>)

Gets the first entity that satisfies the provided expression.

```
public Task<T?> GetBy(Expression<Func<T, bool>> where)
```

Parameters

`where` Expression<Func<T, bool>>

The expression to evaluate.

Returns

Task<T>

A task that represents the asynchronous operation. The task result contains the first entity that satisfies the expression or null if no such entity exists.

GetBy<TEntity>(ICoreSpecifications<TEntity>?)

```
public Task<TEntity?> GetBy<TEntity>(ICoreSpecifications<TEntity>?
coreSpecifications) where TEntity : class
```

Parameters

`coreSpecifications` ICoreSpecifications<TEntity>

Returns

Task<TEntity>

Type Parameters

`TEntity`

GetBy<TEntity>(Expression<Func<TEntity, bool>>)

```
public Task<TEntity?> GetBy<TEntity>(Expression<Func<TEntity, bool>> where) where
TEntity : class
```

Parameters

`where` Expression<Func<TEntity, bool>>

Returns

Task<TEntity>

Type Parameters

`TEntity`

GetBy<TEntity>(string, ICollection<DbParameter>, ICoreSpecifications<TEntity>?)

```
public Task<TEntity?> GetBy<TEntity>(string query, ICollection<DbParameter>
```

```
parameters, ICoreSpecifications<TEntity>? coreSpecifications) where TEntity : class
```

Parameters

`query` string

`parameters` ICollection<DbParameter>

`coreSpecifications` ICoreSpecifications<TEntity>

Returns

Task<TEntity>

Type Parameters

`TEntity`

GetPagedResult(Page)

Retrieves a paginated list of entities from the repository that conforms to the specified Radiant page.

```
public Task<Paged<T>> GetPagedResult(Page page)
```

Parameters

`page` Page

The page number and size of the entities to be retrieved from the repository.

Returns

Task<Paged<T>>

A task that represents the asynchronous operation. The task result is a collection of entities that meet The core specifications, paginated based on the given Radiant page object. If no entities meet the specifications, the task result is an empty collection.

GetPagedResult(Page, ICoreSpecifications<T>?)

```
public Task<Paged<T>> GetPagedResult(Page page, ICoreSpecifications<T>?
coreSpecifications)
```

Parameters

`page` Page

`coreSpecifications` ICoreSpecifications<T>

Returns

Task<Paged<T>>

GetPagedResult(Page, string, ICollection<DbParameter>)

```
public Task<Paged<T>> GetPagedResult(Page page, string query,
ICollection<DbParameter> parameters)
```

Parameters

`page` Page

`query` string

`parameters` ICollection<DbParameter>

Returns

Task<Paged<T>>

GetPagedResult(Page, string, ICollection<DbParameter>, ICoreSpecifications<T>?)

```
public Task<Paged<T>> GetPagedResult(Page page, string query,
```

```
ICollection<DbParameter> parameters, ICoreSpecifications<T>? coreSpecifications)
```

Parameters

`page` Page

`query` string

`parameters` ICollection<DbParameter>

`coreSpecifications` ICoreSpecifications<T>

Returns

Task<Paged<T>>

GetPagedResult<TEntity>(Page)

```
public Task<Paged<TEntity>> GetPagedResult<TEntity>(Page page) where TEntity : class
```

Parameters

`page` Page

Returns

Task<Paged<TEntity>>

Type Parameters

`TEntity`

GetPagedResult<TEntity>(Page, ICoreSpecifications<TEntity>?)

```
public Task<Paged<TEntity>> GetPagedResult<TEntity>(Page page,
ICoreSpecifications<TEntity>? coreSpecifications) where TEntity : class
```

Parameters

`page` Page

`coreSpecifications` ICoreSpecifications<TEntity>

Returns

Task<Paged<TEntity>>

Type Parameters

`TEntity`

GetPagedResult<TEntity>(Page, string, ICollection<DbParameter>)

```
public Task<Paged<TEntity>> GetPagedResult<TEntity>(Page page, string query,
ICollection<DbParameter> parameters) where TEntity : class
```

Parameters

`page` Page

`query` string

`parameters` ICollection<DbParameter>

Returns

Task<Paged<TEntity>>

Type Parameters

`TEntity`

GetPagedResult<TEntity>(Page, string, ICollection<DbParameter>,

ICoreSpecifications<TEntity>?)

```
public Task<Paged<TEntity>> GetPagedResult<TEntity>(Page page, string query,
ICollection<DbParameter> parameters, ICoreSpecifications<TEntity>?
coreSpecifications) where TEntity : class
```

Parameters

`page` Page

`query` string

`parameters` ICollection<DbParameter>

`coreSpecifications` ICoreSpecifications<TEntity>

Returns

Task<Paged<TEntity>>

Type Parameters

`TEntity`

PageAllAsync(Page, ICoreSpecifications<T>?)

Pages all entities based on the provided page information and specifications.

```
public Task<ICollection<T>> PageAllAsync(Page page, ICoreSpecifications<T>?
coreSpecifications)
```

Parameters

`page` Page

The page information.

`coreSpecifications` ICoreSpecifications<T>

The specifications to evaluate.

Returns

Task<ICollection<T>>

A task that represents the asynchronous operation. The task result contains a list of entities paged according to the provided information and specifications.

PageAllQueryable(Page, ICoreSpecifications<T>?)

Returns an IQueryable of all paged entities based on the provided page information and specifications.

```
public IQueryable<T> PageAllQueryable(Page page, ICoreSpecifications<T>?
coreSpecifications)
```

Parameters

`page` Page

The page information.

`coreSpecifications` ICoreSpecifications<T>

The specifications to evaluate.

Returns

IQueryable<T>

IQueryable of entities paged according to the provided page information and specifications.

Remove(T)

Removes the given entity from the database and saves the changes.

```
public void Remove(T entity)
```

Parameters

`entity` T

The entity to remove.

RemoveAndSaveAsync(T)

```
public Task RemoveAndSaveAsync(T entity)
```

Parameters

`entity` T

Returns

Task

RemoveAndSaveAsync<TEntity>(TEntity)

```
public Task RemoveAndSaveAsync<TEntity>(TEntity entity) where TEntity : class
```

Parameters

`entity` TEntity

Returns

Task

Type Parameters

`TEntity`

RemoveRange(ICollection<T>)

Removes a range of entities from the database and saves the changes.

```
public void RemoveRange(ICollection<T> entities)
```

Parameters

`entities` ICollection<T>

The entities to removal.

RemoveRangeAndSaveAsync(ICollection<T>)

```
public Task RemoveRangeAndSaveAsync(ICollection<T> entities)
```

Parameters

`entities` ICollection<T>

Returns

Task

RemoveRangeAndSaveAsync<TEntity>(ICollection<TEntity>)

```
public Task RemoveRangeAndSaveAsync<TEntity>(ICollection<TEntity> entities) where
TEntity : class
```

Parameters

`entities` ICollection<TEntity>

Returns

Task

Type Parameters

TEntity

RemoveRange<TEntity>(ICollection<TEntity>)

```
public void RemoveRange<TEntity>(ICollection<TEntity> entities) where TEntity
: class
```

Parameters

entities ICollection<TEntity>

Type Parameters

TEntity

Remove<TEntity>(TEntity)

```
public void Remove<TEntity>(TEntity entity) where TEntity : class
```

Parameters

entity TEntity

Type Parameters

TEntity

SaveAsync()

Saves changes in the DbContext to the database.

```
public Task SaveAsync()
```

Returns

Task

A task represents the asynchronous operation for saving changes to the database.

Update(T)

Updates the provided entity in the DbContext and saves the changes to the database.

```
public void Update(T entity)
```

Parameters

`entity` T

The entity to update.

UpdateAndSaveAsync(T)

```
public Task UpdateAndSaveAsync(T entity)
```

Parameters

`entity` T

Returns

Task

UpdateAndSaveAsync<TEntity>(TEntity)

```
public Task UpdateAndSaveAsync<TEntity>(TEntity entity) where TEntity : class
```

Parameters

`entity` TEntity

Returns

Task

Type Parameters

`TEntity`

UpdateRange(ICollection<T>)

Updates the range of entities in the DbContext and saves the changes to the database.

```
public void UpdateRange(ICollection<T> entities)
```

Parameters

`entities` ICollection<T>

The entities to update.

UpdateRangeAndSaveAsync(ICollection<T>)

```
public Task UpdateRangeAndSaveAsync(ICollection<T> entities)
```

Parameters

`entities` ICollection<T>

Returns

Task

UpdateRangeAndSaveAsync<TEntity>(ICollection<TEntity>)

```
public Task UpdateRangeAndSaveAsync<TEntity>(ICollection<TEntity> entities) where
TEntity : class
```

Parameters

`entities` ICollection<TEntity>

Returns

Task

Type Parameters

`TEntity`

UpdateRange<TEntity>(ICollection<TEntity>)

```
public void UpdateRange<TEntity>(ICollection<TEntity> entities) where TEntity
: class
```

Parameters

`entities` ICollection<TEntity>

Type Parameters

`TEntity`

Update<TEntity>(TEntity)

```
public void Update<TEntity>(TEntity entity) where TEntity : class
```

Parameters

`entity` TEntity

Type Parameters

`TEntity`

Interface ICoreRepository<T>

Namespace: Solstice.Repository.Core

Assembly: Solstice.Repository.dll

The ICoreRepository interface provides methods for performing CRUD operations, querying, counting, paging, transactions, and other tasks on an underlying data repository in an asynchronous manner. The repository holds objects of a type. The methods in this interface produce or consume tasks that represent ongoing work and are used for structuring asynchronous code.

```
public interface ICoreRepository<T> where T : class
```

Type Parameters

T

A type parameter. This type parameter is used to define the type of objects managed by the repository.

Remarks

- The methods grouped in the 'Create, Update, Delete' region are for managing entities within the repository.
- The 'Actions' region contains methods for counting entities in the repository or check its status (AnyAsyncBy).
- The 'Get By' region provides methods to retrieve an entity by a specific criteria.
- The 'Get all' region offers APIs to get collections of entities based on different criteria/patterns.
- In 'Queryable' area, you find methods for searching within the repository, but the operations are not executed right away and more criteria can be added later in the operation chain.
- 'Pageable' region offers the option to receive the data in chunks, good for large data sets to retrieve and process in smaller parts.
- 'Transactions' region provides a mechanism for batch of operations to be executed together and in an atomic manner.
- 'Others' region provides methods that handle various other tasks not covered by the previously described groupings.

Methods

AddAndSaveAsync(T)

```
Task AddAndSaveAsync(T entity)
```

Parameters

`entity` T

Returns

Task

AddAndSaveAsync<TEntity>(TEntity)

```
Task AddAndSaveAsync<TEntity>(TEntity entity) where TEntity : class
```

Parameters

`entity` TEntity

Returns

Task

Type Parameters

`TEntity`

AddAsync(T)

Add entity to repository

```
Task AddAsync(T entity)
```

Parameters

`entity` T

The entity object

Returns

Task

AddAsync<TEntity>(TEntity)

```
Task AddAsync<TEntity>(TEntity entity) where TEntity : class
```

Parameters

`entity` TEntity

Returns

Task

Type Parameters

`TEntity`

AddRangeAndSaveAsync(ICollection<T>)

```
Task AddRangeAndSaveAsync(ICollection<T> entities)
```

Parameters

`entities` ICollection<T>

Returns

Task

AddRangeAndSaveAsync<TEntity>(ICollection<TEntity>)

```
Task AddRangeAndSaveAsync<TEntity>(ICollection<TEntity> entities) where TEntity
: class
```

Parameters

`entities` ICollection<TEntity>

Returns

Task

Type Parameters

`TEntity`

AddRangeAsync(ICollection<T>)

Add range of entities to repository

```
Task AddRangeAsync(ICollection<T> entities)
```

Parameters

`entities` ICollection<T>

The entities list

Returns

Task

AddRangeAsync<TEntity>(ICollection<TEntity>)

```
Task AddRangeAsync<TEntity>(ICollection<TEntity> entities) where TEntity : class
```

Parameters

`entities` ICollection<TEntity>

Returns

Task

Type Parameters

`TEntity`

AnyAsyncBy(Expression<Func<T, bool>>)

Checks if any entity in the repository matches the provided expression

```
Task<bool> AnyAsyncBy(Expression<Func<T, bool>> where)
```

Parameters

`where` Expression<Func<T, bool>>

The expression that describes the condition to match

Returns

Task<bool>

True if any entity matches the condition, False otherwise

AnyAsyncBy<TEntity>(Expression<Func<TEntity, bool>>)

```
Task<bool> AnyAsyncBy<TEntity>(Expression<Func<TEntity, bool>> where) where TEntity
: class
```

Parameters

`where` Expression<Func<TEntity, bool>>

Returns

Task<bool>

Type Parameters

`TEntity`

AnyAsyncBy<TEntity>(IQueryable<TEntity>)

```
Task<bool> AnyAsyncBy<TEntity>(IQueryable<TEntity> queryable) where TEntity : class
```

Parameters

`queryable` IQueryable<TEntity>

Returns

Task<bool>

Type Parameters

`TEntity`

BeginTransactionAsync()

Begins a new transaction asynchronously.

```
Task<IDbContextTransaction> BeginTransactionAsync()
```

Returns

Task<IDbContextTransaction>

A task that represents the asynchronous operation. The task result is an IDbContextTransaction object which encapsulates all information about the transaction.

CountAllAsync()

Counts all entities in repository

```
Task<decimal> CountAllAsync()
```

Returns

Task<decimal>

Total count of all entities

CountAllAsyncBy(Expression<Func<T, bool>>)

Counts the total entities that match the provided expression

```
Task<decimal> CountAllAsyncBy(Expression<Func<T, bool>> where)
```

Parameters

`where` Expression<Func<T, bool>>

The expression that describes the condition to match

Returns

Task<decimal>

Total matched entities count

CountAllAsyncBy<TEntity>(Expression<Func<TEntity, bool>>)

```
Task<decimal> CountAllAsyncBy<TEntity>(Expression<Func<TEntity, bool>> where) where
TEntity : class
```

Parameters

`where` Expression<Func<TEntity, bool>>

Returns

Task<decimal>

Type Parameters

`TEntity`

ExecuteQuery(string, ICollection<DbParameter>?)

```
Task ExecuteQuery(string query, ICollection<DbParameter>? dbParameters)
```

Parameters

`query` string

`dbParameters` ICollection<DbParameter>

Returns

Task

FindAsync(int)

```
Task<T> FindAsync(int id)
```

Parameters

`id` int

Returns

Task<T>

FindAsync<TEntity>(int)

```
Task<TEntity> FindAsync<TEntity>(int id) where TEntity : class
```

Parameters

`id` int

Returns

Task<TEntity>

Type Parameters

`TEntity`

GetAllAsync()

```
Task<ICollection<T>> GetAllAsync()
```

Returns

Task<ICollection<T>>

GetAllAsync(ICoreSpecifications<T>?)

Retrieves a collection of all entities from the repository asynchronously.

```
Task<ICollection<T>> GetAllAsync(ICoreSpecifications<T>? coreSpecifications)
```

Parameters

`coreSpecifications` ICoreSpecifications<T>

The specifications that entities must meet to be retrieved from the repository.

Returns

Task<ICollection<T>>

A task that represents the asynchronous operation. The task result contains a collection of entities satisfying The core specifications or an empty collection if no matches.

GetAllAsync(Expression<Func<T, bool>>)

Retrieves a collection of entities from the repository that satisfy the specified condition asynchronously.

```
Task<ICollection<T>> GetAllAsync(Expression<Func<T, bool>> where)
```

Parameters

`where` Expression<Func<T, bool>>

An expression representing a condition to be matched by entities in the repository.

Returns

Task<ICollection<T>>

A task that represents the asynchronous operation. The task result contains a collection of entities satisfying the condition or an empty collection if there are no matches.

GetAllAsync(string, ICoreSpecifications<T>?)

Retrieves a collection of all entities from the repository asyncronously based on a provided SQL-like query and specifications.

```
Task<ICollection<T>> GetAllAsync(string query, ICoreSpecifications<T>?
coreSpecifications)
```

Parameters

`query` string

A SQL-like query that retrieves entities from the repository.

`coreSpecifications` ICoreSpecifications<T>

The specifications that entities must meet to be retrieved from the repository.

Returns

Task<ICollection<T>>

A task that represents the asynchronous operation. The task results contains a collection od entities satisfying the query and The core specifications or an empty collection if no matches.

GetAllAsync(string, ICollection<DbParameter>, ICoreSpecifications<T>?)

Retrieves a collection of all entities from the repository asyncronously based on a provided SQL-like query, parameters and specifications.

```
Task<ICollection<T>> GetAllAsync(string query, ICollection<DbParameter> parameters,
ICoreSpecifications<T>? coreSpecifications)
```

Parameters

`query` string

A SQL-like query that retrieves entities from the repository.

`parameters` ICollection<DbParameter>

A collection of database parameters used in the query

`coreSpecifications` ICoreSpecifications<T>

The specifications that entities must meet to be retrieved from the repository.

Returns

Task<ICollection<T>>

A task that represents the asynchronous operation. The task results contains a collection od entities satisfying the query, parameters and The core specifications or an empty collection if no matches.

GetAllAsync<TEntity>()

```
Task<ICollection<TEntity>> GetAllAsync<TEntity>() where TEntity : class
```

Returns

Task<ICollection<TEntity>>

Type Parameters

`TEntity`

GetAllAsync<TEntity>(ICoreSpecifications<TEntity>)

```
Task<ICollection<TEntity>> GetAllAsync<TEntity>(ICoreSpecifications<TEntity>
coreSpecifications) where TEntity : class
```

Parameters

`coreSpecifications` ICoreSpecifications<TEntity>

Returns

Task<ICollection<TEntity>>

Type Parameters

`TEntity`

GetAllAsync<TEntity>(Expression<Func<TEntity, bool>>)

```
Task<ICollection<TEntity>> GetAllAsync<TEntity>(Expression<Func<TEntity, bool>>
where) where TEntity : class
```

Parameters

where Expression<Func<TEntity, bool>>

Returns

Task<ICollection<TEntity>>

Type Parameters

TEntity

GetAllAsync<TEntity>(string)

```
Task<ICollection<TEntity>> GetAllAsync<TEntity>(string query) where TEntity : class
```

Parameters

query string

Returns

Task<ICollection<TEntity>>

Type Parameters

TEntity

GetAllAsync<TEntity>(string, ICoreSpecifications<TEntity>)

```
Task<ICollection<TEntity>> GetAllAsync<TEntity>(string query,
ICoreSpecifications<TEntity> coreSpecifications) where TEntity : class
```

Parameters

query string

coreSpecifications ICoreSpecifications<TEntity>

Returns

Task<ICollection<TEntity>>

Type Parameters

TEntity

GetAllAsync<TEntity>(string, ICollection<DbParameter>)

```
Task<ICollection<TEntity>> GetAllAsync<TEntity>(string query,
ICollection<DbParameter> parameters) where TEntity : class
```

Parameters

query string

parameters ICollection<DbParameter>

Returns

Task<ICollection<TEntity>>

Type Parameters

TEntity

GetAllAsync<TEntity>(string, ICollection<DbParameter>, ICoreSpecifications<TEntity>)

```
Task<ICollection<TEntity>> GetAllAsync<TEntity>(string query,
ICollection<DbParameter> parameters, ICoreSpecifications<TEntity>
coreSpecifications) where TEntity : class
```

Parameters

`query` string

`parameters` ICollection<DbParameter>

`coreSpecifications` ICoreSpecifications<TEntity>

Returns

Task<ICollection<TEntity>>

Type Parameters

`TEntity`

GetAllByQueryable<TEntity>(IQueryable<TEntity>)

Retrieves a collection of entities from a specified IQueryable.

```
Task<ICollection<TEntity>> GetAllByQueryable<TEntity>(IQueryable<TEntity> query)
```

Parameters

`query` IQueryable<TEntity>

An IQueryable that retrieves entities from the repository.

Returns

Task<ICollection<TEntity>>

A task that represents the asynchronous operation. The task result contains a collection of entities that satisfy the query or an empty collection if no matches.

Type Parameters

`TEntity`

GetAllQueryable()

Retrieves all entities from the repository asynchronously.

```
IQueryable<T> GetAllQueryable()
```

Returns

IQueryable<T>

A queryable collection of all entities in the repository.

GetAllQueryable(ICoreSpecifications<T>?)

Retrieves entities that match the specified Radiant specifications from the repository asynchronously.

```
IQueryable<T> GetAllQueryable(ICoreSpecifications<T>? coreSpecifications)
```

Parameters

`coreSpecifications` ICoreSpecifications<T>

The specifications that entities must meet to be retrieved from the repository.

Returns

IQueryable<T>

A queryable collection of entities that satisfy The core specifications.

GetAllQueryable(Expression<Func<T, bool>>)

Retrieves entities that match the specified condition from the repository asynchronously.

```
IQueryable<T> GetAllQueryable(Expression<Func<T, bool>> where)
```

Parameters

`where` Expression<Func<T, bool>>

An expression representing a condition to be matched by entities in the repository.

Returns

IQueryable<T>

A queryable collection of entities matching the condition.

GetAllQueryable(string, ICoreSpecifications<T>?)

Retrieves entities from the repository based on a provided SQL-like query and specifications asynchronously.

```
IQueryable<T> GetAllQueryable(string query, ICoreSpecifications<T>?
coreSpecifications)
```

Parameters

`query` string

A SQL-like query that retrieves entities from the repository.

`coreSpecifications` ICoreSpecifications<T>

The specifications that entities must meet to be retrieved from the repository.

Returns

IQueryable<T>

A queryable collection of entities satisfying the query and The core specifications.

GetAllQueryable(string, ICollection<DbParameter>, ICoreSpecifications<T>?)

Retrieves entities from the repository based on a provided SQL-like query, parameters and specifications asynchronously.

```
IQueryable<T> GetAllQueryable(string query, ICollection<DbParameter> parameters,
ICoreSpecifications<T>? coreSpecifications)
```

Parameters

`query` string

A SQL-like query that retrieves entities from the repository.

`parameters` ICollection<DbParameter>

A collection of database parameters used in the query

`coreSpecifications` ICoreSpecifications<T>

The specifications that entities must meet to be retrieved from the repository.

Returns

IQueryable<T>

A queryable collection of entities satisfying the query, parameters and The core specifications.

GetAllQueryable<TEntity>()

Retrieves all entities from the repository asynchronously.

```
IQueryable<TEntity> GetAllQueryable<TEntity>() where TEntity : class
```

Returns

IQueryable<TEntity>

A queryable collection of all entities in the repository.

Type Parameters

`TEntity`

GetAllQueryable<TEntity>(ICoreSpecifications<TEntity>?)

Retrieves entities that match the specified Radiant specifications from the repository asynchronously.

```
IQueryable<TEntity> GetAllQueryable<TEntity>(ICoreSpecifications<TEntity>?
coreSpecifications) where TEntity : class
```

Parameters

`coreSpecifications` ICoreSpecifications<TEntity>

The specifications that entities must meet to be retrieved from the repository.

Returns

IQueryable<TEntity>

A queryable collection of entities that satisfy The core specifications.

Type Parameters

`TEntity`

GetAllQueryable<TEntity>(Expression<Func<TEntity, bool>>)

Retrieves entities that match the specified condition from the repository asynchronously.

```
IQueryable<TEntity> GetAllQueryable<TEntity>(Expression<Func<TEntity, bool>> where)
where TEntity : class
```

Parameters

`where` Expression<Func<TEntity, bool>>

An expression representing a condition to be matched by entities in the repository.

Returns

IQueryable<TEntity>

A queryable collection of entities matching the condition.

Type Parameters

`TEntity`

GetAllQueryable<TEntity>(string, ICoreSpecifications<TEntity>?)

Retrieves entities from the repository based on a provided SQL-like query and specifications asynchronously.

```
IQueryable<TEntity> GetAllQueryable<TEntity>(string query,
ICoreSpecifications<TEntity>? coreSpecifications) where TEntity : class
```

Parameters

`query` string

A SQL-like query that retrieves entities from the repository.

`coreSpecifications` ICoreSpecifications<TEntity>

The specifications that entities must meet to be retrieved from the repository.

Returns

IQueryable<TEntity>

A queryable collection of entities satisfying the query and The core specifications.

Type Parameters

`TEntity`

GetAllQueryable<TEntity>(string, ICollection<DbParameter>, ICoreSpecifications<TEntity>?)

Retrieves entities from the repository based on a provided SQL-like query, parameters and specifications asynchronously.

```
IQueryable<TEntity> GetAllQueryable<TEntity>(string query, ICollection<DbParameter>
parameters, ICoreSpecifications<TEntity>? coreSpecifications) where TEntity : class
```

Parameters

`query` string

A SQL-like query that retrieves entities from the repository.

`parameters` ICollection<DbParameter>

A collection of database parameters used in the query

`coreSpecifications` ICoreSpecifications<TEntity>

The specifications that entities must meet to be retrieved from the repository.

Returns

IQueryable<TEntity>

A queryable collection of entities satisfying the query, parameters and The core specifications.

Type Parameters

`TEntity`

GetBy(ICoreSpecifications<T>)

Retrieves an entity from the repository that meets the criteria specified by the given Radiant specification.

```
Task<T?> GetBy(ICoreSpecifications<T> coreSpecifications)
```

Parameters

`coreSpecifications` ICoreSpecifications<T>

The specifications that an entity must meet to be retrieved from the repository.

Returns

Task<T>

A task that represents the asynchronous operation. The task result is the first entity that satisfies the specified Radiant specifications. If no entity satisfies the specifications, the task result is null.

GetBy(Expression<Func<T, bool>>)

Retrieves an entity that matches the specified condition from the repository.

```
Task<T?> GetBy(Expression<Func<T, bool>> where)
```

Parameters

`where` Expression<Func<T, bool>>

An expression representing a condition to be matched by entities in the repository.

Returns

Task<T>

A task that represents the asynchronous operation. The task result contains the first matching entity or null if no entity matches the condition.

GetBy<TEntity>(ICoreSpecifications<TEntity>)

```
Task<TEntity?> GetBy<TEntity>(ICoreSpecifications<TEntity> coreSpecifications) where
TEntity : class
```

Parameters

`coreSpecifications` ICoreSpecifications<TEntity>

Returns

Task<TEntity>

Type Parameters

`TEntity`

GetBy<TEntity>(Expression<Func<TEntity, bool>>)

```
Task<TEntity?> GetBy<TEntity>(Expression<Func<TEntity, bool>> where) where TEntity
: class
```

Parameters

`where` Expression<Func<TEntity, bool>>

Returns

Task<TEntity>

Type Parameters

`TEntity`

GetBy<TEntity>(string, ICollection<DbParameter>, ICoreSpecifications<TEntity>)

```
Task<TEntity?> GetBy<TEntity>(string query, ICollection<DbParameter> parameters,
ICoreSpecifications<TEntity> coreSpecifications) where TEntity : class
```

Parameters

`query` string

`parameters` ICollection<DbParameter>

`coreSpecifications` ICoreSpecifications<TEntity>

Returns

Task<TEntity>

Type Parameters

`TEntity`

GetPagedResult(Page)

Retrieves a paginated list of entities from the repository that conforms to the specified Radiant page.

```
Task<Paged<T>> GetPagedResult(Page page)
```

Parameters

`page` Page

The page number and size of the entities to be retrieved from the repository.

Returns

Task<Paged<T>>

A task that represents the asynchronous operation. The task result is a collection of entities that meet The core specifications, paginated based on the given Radiant page object. If no entities meet the specifications, the task result is an empty collection.

GetPagedResult(Page, ICoreSpecifications<T>)

```
Task<Paged<T>> GetPagedResult(Page page, ICoreSpecifications<T> coreSpecifications)
```

Parameters

`page` Page

`coreSpecifications` ICoreSpecifications<T>

Returns

Task<Paged<T>>

GetPagedResult(Page, string, ICollection<DbParameter>)

```
Task<Paged<T>> GetPagedResult(Page page, string query, ICollection<DbParameter>
parameters)
```

Parameters

`page` Page

`query` string

`parameters` ICollection<DbParameter>

Returns

Task<Paged<T>>

GetPagedResult(Page, string, ICollection<DbParameter>, ICoreSpecifications<T>)

```
Task<Paged<T>> GetPagedResult(Page page, string query, ICollection<DbParameter>
parameters, ICoreSpecifications<T> coreSpecifications)
```

Parameters

`page` Page

`query` string

`parameters` ICollection<DbParameter>

`coreSpecifications` ICoreSpecifications<T>

Returns

Task<Paged<T>>

GetPagedResult<TEntity>(Page)

```
Task<Paged<TEntity>> GetPagedResult<TEntity>(Page page) where TEntity : class
```

Parameters

`page` Page

Returns

Task<Paged<TEntity>>

Type Parameters

`TEntity`

GetPagedResult<TEntity>(Page, ICoreSpecifications<TEntity>)

```
Task<Paged<TEntity>> GetPagedResult<TEntity>(Page page, ICoreSpecifications<TEntity>
coreSpecifications) where TEntity : class
```

Parameters

`page` Page

`coreSpecifications` ICoreSpecifications<TEntity>

Returns

Task<Paged<TEntity>>

Type Parameters

`TEntity`

GetPagedResult<TEntity>(Page, string, ICollection<DbParameter>)

```
Task<Paged<TEntity>> GetPagedResult<TEntity>(Page page, string query,
ICollection<DbParameter> parameters) where TEntity : class
```

Parameters

`page` Page

`query` string

`parameters` ICollection<DbParameter>

Returns

Task<Paged<TEntity>>

Type Parameters

`TEntity`

GetPagedResult<TEntity>(Page, string, ICollection<DbParameter>, ICoreSpecifications<TEntity>)

```
Task<Paged<TEntity>> GetPagedResult<TEntity>(Page page, string query,
ICollection<DbParameter> parameters, ICoreSpecifications<TEntity>
coreSpecifications) where TEntity : class
```

Parameters

`page` Page

`query` string

`parameters` ICollection<DbParameter>

`coreSpecifications` ICoreSpecifications<TEntity>

Returns

Task<Paged<TEntity>>

Type Parameters

`TEntity`

PageAllAsync(Page, ICoreSpecifications<T>?)

Retrieves a collection of entities from the repository that satisfy the specified condition asyncronously, and do paging on them.

```
Task<ICollection<T>> PageAllAsync(Page page, ICoreSpecifications<T>?
coreSpecifications)
```

Parameters

`page` Page

The page number and size of the entities to be retrieved from the repository.

`coreSpecifications` ICoreSpecifications<T>

The specifications that entities must meet to be retrieved from the repository.

Returns

Task<ICollection<T>>

A task that represents the asynchronous operation. The task result contains a collection of entities satisfying the condition. or an empty collection if no matches, paged by the given information in RadiantPage object.

PageAllQueryable(Page, ICoreSpecifications<T>?)

Retrieves a queryable collection of entities from the repository that satisfy the specified condition asyncronously, and do paging on them.

```
IQueryable<T> PageAllQueryable(Page page, ICoreSpecifications<T>?
coreSpecifications)
```

Parameters

`page` Page

The page number and size of the entities to be retrieved from the repository.

`coreSpecifications` ICoreSpecifications<T>

The specifications that entities must meet to be retrieved from the repository.

Returns

IQueryable<T>

A queryable collection of all entities in the repository that fulfill the condition, paged by the given information in RadiantPage object.

Remove(T)

Remove entity from repository

```
void Remove(T entity)
```

Parameters

`entity` T

The entity object

RemoveAndSaveAsync(T)

```
Task RemoveAndSaveAsync(T entity)
```

Parameters

`entity` T

Returns

Task

RemoveAndSaveAsync<TEntity>(TEntity)

```
Task RemoveAndSaveAsync<TEntity>(TEntity entity) where TEntity : class
```

Parameters

`entity` TEntity

Returns

Task

Type Parameters

`TEntity`

RemoveRange(ICollection<T>)

Remove range of entities from repository

```
void RemoveRange(ICollection<T> entities)
```

Parameters

`entities` ICollection<T>

The entities list

RemoveRangeAndSaveAsync(ICollection<T>)

```
Task RemoveRangeAndSaveAsync(ICollection<T> entities)
```

Parameters

`entities` ICollection<T>

Returns

Task

RemoveRangeAndSaveAsync<TEntity>(ICollection<TEntity>)

```
Task RemoveRangeAndSaveAsync<TEntity>(ICollection<TEntity> entities) where TEntity
: class
```

Parameters

`entities` ICollection<TEntity>

Returns

Task

Type Parameters

`TEntity`

RemoveRange<TEntity>(ICollection<TEntity>)

```
void RemoveRange<TEntity>(ICollection<TEntity> entities) where TEntity : class
```

Parameters

`entities` ICollection<TEntity>

Type Parameters

`TEntity`

Remove<TEntity>(TEntity)

```
void Remove<TEntity>(TEntity entity) where TEntity : class
```

Parameters

`entity` TEntity

Type Parameters

`TEntity`

SaveAsync()

Save changes in repository

```
Task SaveAsync()
```

Returns

Task

Update(T)

Update entity in repository

```
void Update(T entity)
```

Parameters

`entity` T

The entity object

UpdateAndSaveAsync(T)

```
Task UpdateAndSaveAsync(T entity)
```

Parameters

`entity` T

Returns

Task

UpdateAndSaveAsync<TEntity>(TEntity)

```
Task UpdateAndSaveAsync<TEntity>(TEntity entity) where TEntity : class
```

Parameters

`entity` TEntity

Returns

Task

Type Parameters

`TEntity`

UpdateRange(ICollection<T>)

Update range of entities in repository

```
void UpdateRange(ICollection<T> entities)
```

Parameters

`entities` ICollection<T>

The entities list

UpdateRangeAndSaveAsync(ICollection<T>)

```
Task UpdateRangeAndSaveAsync(ICollection<T> entities)
```

Parameters

`entities` ICollection<T>

Returns

Task

UpdateRangeAndSaveAsync<TEntity>(ICollection<TEntity>)

```
Task UpdateRangeAndSaveAsync<TEntity>(ICollection<TEntity> entities) where TEntity
: class
```

Parameters

`entities` ICollection<TEntity>

Returns

Task

Type Parameters

TEntity

UpdateRange<TEntity>(ICollection<TEntity>)

```
void UpdateRange<TEntity>(ICollection<TEntity> entities) where TEntity : class
```

Parameters

entities ICollection<TEntity>

Type Parameters

TEntity

Update<TEntity>(TEntity)

```
void Update<TEntity>(TEntity entity) where TEntity : class
```

Parameters

entity TEntity

Type Parameters

TEntity

Namespace Solstice.Repository.Extensions

Classes

CoreQueryableExtension

Represents a utility class that provides extension methods to IQueryable interface objects.

Class CoreQueryableExtension

Namespace: Solstice.Repository.Extensions

Assembly: Solstice.Repository.dll

Represents a utility class that provides extension methods to IQueryable interface objects.

```
public static class CoreQueryableExtension
```

Inheritance

object ← CoreQueryableExtension

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Remarks

CoreQueryableExtension offers pagination for queries.

Methods

Pageable<TSource>(IQueryable<TSource>, Page)

Returns a paged source from the provided IQueryable object using the given CorePage object parameters.

```
public static IQueryable<TSource> Pageable<TSource>(this IQueryable<TSource>
queryable, Page page)
```

Parameters

`queryable` IQueryable<TSource>

The System.Linq.IQueryable{T} to create a paged source from.

`page` Page

CorePage object that determines the number of elements to bypass and the size of the page.

Returns

IQueryable<TSource>

An System.Linq.IQueryable{T} that contains elements from the input sequence that occur after the specified index and has the specified page size.

Type Parameters

`TSource`

The type of the elements of source.

Namespace Solstice.Repository.Injections

Classes

RepositoryInjections

The RepositoryInjections static class contains extension methods for ModelBuilder and IServiceCollection instances. These extension methods add convenient functionality for database and service-related configurations.

Class RepositoryInjections

Namespace: Solstice.Repository.Injections

Assembly: Solstice.Repository.dll

The RepositoryInjections static class contains extension methods for ModelBuilder and IServiceCollection instances. These extension methods add convenient functionality for database and service-related configurations.

```
public static class RepositoryInjections
```

Inheritance

object ← RepositoryInjections

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

AddQueryDtoToDbContext(ModelBuilder)

This extension method for the ModelBuilder class scans all assemblies in the current domain for types marked with the QueryAttribute. Each located type is then added to the ModelBuilder as an entity, with no key, and set to be excluded from migrations.

```
public static void AddQueryDtoToDbContext(this ModelBuilder modelBuilder)
```

Parameters

`modelBuilder` ModelBuilder

The ModelBuilder instance to which the types are added.

Exceptions

CoreException

Thrown when no types with the QueryAttribute are found.

AddRepositories<TDbContext>(IServiceCollection)

The AddRepositories extension method for the IServiceCollection, which scans all assemblies in the current domain for types marked with the RepositoryAttribute.

```
public static void AddRepositories<TDbContext>(this IServiceCollection services)
where TDbContext : DbContext
```

Parameters

`services` IServiceCollection

The IServiceCollection instance to which the types are added.

Type Parameters

`TDbContext`

Remarks

The located type(s) that have the RepositoryAttribute are added to the IServiceCollection as a Scope. If no such types are found, it throws a CoreInjectionsException.

Exceptions

CoreException

Thrown when no types with the RepositoryAttribute are found.

AddRepositories<TDbContext>(IServiceCollection, string)

```
[Obsolete("AddRepositories with namespace is deprecated, please use AddRepositories
with Repository Attribute instead.")]
public static void AddRepositories<TDbContext>(this IServiceCollection services,
string assemblyName) where TDbContext : DbContext
```

Parameters

`services` IServiceCollection

`assemblyName` string

Type Parameters

`TDbContext`

AddUnitOfWork<TDbContext>(IServiceCollection)

The AddUnitOfWork extension method for the IServiceCollection, which adds the UnitOfWork as a Scoped service.

```
public static void AddUnitOfWork<TDbContext>(this IServiceCollection services) where
TDbContext : DbContext
```

Parameters

`services` IServiceCollection

Type Parameters

`TDbContext`

ApplyDbConfigurations(ModelBuilder, Assembly)

This extension method to the ModelBuilder class allows the application to apply database configurations from a specified assembly.

```
public static void ApplyDbConfigurations(this ModelBuilder modelBuilder,
Assembly assembly)
```

Parameters

`modelBuilder` ModelBuilder

The ModelBuilder instance on which to apply the assembly configurations.

`assembly` Assembly

The assembly from which configurations should be applied.

Remarks

The method loads the specified assembly and applies its configurations to the dbContext via the ModelBuilder ApplyConfigurationsFromAssembly method.

ApplyDbConfigurations(ModelBuilder, string)

This extension method to the ModelBuilder class allows the application to apply database configurations from a specified assembly.

```
public static void ApplyDbConfigurations(this ModelBuilder modelBuilder,
string assemblyName)
```

Parameters

`modelBuilder` ModelBuilder

The ModelBuilder instance on which to apply the assembly configurations.

`assemblyName` string

The name of the assembly from which configurations should be applied.

Remarks

The method loads the specified assembly and applies its configurations to the dbContext via the ModelBuilder ApplyConfigurationsFromAssembly method.

ScanRepositoriesIn<TDbContext>(IServiceCollection, Assembly)

The AddRepositories extension method for the IServiceCollection, which scans all assemblies in the current domain for types marked with the RepositoryAttribute.

```
public static void ScanRepositoriesIn<TDbContext>(this IServiceCollection services,
Assembly assembly) where TDbContext : DbContext
```

Parameters

`services` IServiceCollection

The IServiceCollection instance to which the types are added.

`assembly` Assembly

The assembly to scan

Type Parameters

`TDbContext`

Remarks

The located type(s) that have the RepositoryAttribute are added to the IServiceCollection as a Scope. If no such types are found, it throws a CoreInjectionsException.

Exceptions

CoreException

Thrown when no types with the RepositoryAttribute are found.

ScanRepositoriesIn<TDbContext>(IServiceCollection, string)

The AddRepositories extension method for the IServiceCollection, which scans all assemblies in the current domain for types marked with the RepositoryAttribute.

```
public static void ScanRepositoriesIn<TDbContext>(this IServiceCollection services,
string assemblyName) where TDbContext : DbContext
```

Parameters

`services` IServiceCollection

The IServiceCollection instance to which the types are added.

`assemblyName` string

The name of the assembly to scan

Type Parameters

`TDbContext`

Remarks

The located type(s) that have the RepositoryAttribute are added to the IServiceCollection as a Scope. If no such types are found, it throws a CoreInjectionsException.

Exceptions

CoreException

Thrown when no types with the RepositoryAttribute are found.

Namespace Solstice.Repository.Specifications

Classes

CoreSpecificationEvaluator<T>

Specification<T>

The `coreSpecifications<T>` class implements the `IcoreSpecifications<T>` interface for providing generic ways to define specifications for querying the database.

Interfaces

ICoreSpecifications<T>

Class CoreSpecificationEvaluator<T>

Namespace: Solstice.Repository.Specifications

Assembly: Solstice.Repository.dll

```
public class CoreSpecificationEvaluator<T> where T : class
```

Type Parameters

T

Inheritance

object ← CoreSpecificationEvaluator<T>

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

CoreSpecificationEvaluator()

```
protected CoreSpecificationEvaluator()
```

Methods

GetQuery(IQueryable<T>, ICoreSpecifications<T>)

```
public static IQueryable<T> GetQuery(IQueryable<T> query, ICoreSpecifications<T>
specifications)
```

Parameters

query IQueryable<T>

`specifications` ICoreSpecifications<T>

Returns

IQueryable<T>

Interface ICoreSpecifications<T>

Namespace: Solstice.Repository.Specifications

Assembly: Solstice.Repository.dll

```
public interface ICoreSpecifications<T>
```

Type Parameters

T

Properties

ComplexIncludes

```
Collection<Func<IQueryable<T>, IIncludableQueryable<T, object>>> ComplexIncludes {
get; }
```

Property Value

Collection<Func<IQueryable<T>, IIncludableQueryable<T, object>>>

Distincts

```
bool Distincts { get; }
```

Property Value

bool

FilterCondition

```
Expression<Func<T, bool>> FilterCondition { get; }
```

Property Value

Expression<Func<T, bool>>

GroupBys

```
Expression<Func<T, object>> GroupBys { get; }
```

Property Value

Expression<Func<T, object>>

OrderByDescendings

```
Collection<Expression<Func<T, object>>> OrderByDescendings { get; }
```

Property Value

Collection<Expression<Func<T, object>>>

OrderBys

```
Collection<Expression<Func<T, object>>> OrderBys { get; }
```

Property Value

Collection<Expression<Func<T, object>>>

SimpleIncludes

```
Collection<Expression<Func<T, object>>> SimpleIncludes { get; }
```

Property Value

Collection<Expression<Func<T, object>>>

Class Specification<T>

Namespace: Solstice.Repository.Specifications

Assembly: Solstice.Repository.dll

The `coreSpecifications<T>` class implements the `IcoreSpecifications<T>` interface for providing generic ways to define specifications for querying the database.

```
public class Specification<T> : ICoreSpecifications<T>
```

Type Parameters

`T`

The type of the object which specifications are applied to. Something like an EF Core model type.

Inheritance

object ← Specification<T>

Implements

ICoreSpecifications<T>

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Examples

This sample shows how to create a new instance of `coreSpecifications<T>`.

```
var specs = new coreSpecifications<MyModel>();
specs.SetFilterCondition(x => x.Property > 0);
specs.ApplyOrderBy(x => x.AnotherProperty);
```

Remarks

Properties and methods in this class allow for including related data, ordering, filtering, and grouping the result of queries against a DBSet of the specified type `T`. Note: The 'Include',

'OrderBy', 'OrderByDescending', 'FilterCondition', and 'GroupBy' are Expressions and something like LINQ queries.

Constructors

Specification()

```
public Specification()
```

Specification(Expression<Func<T, bool>>?)

```
public Specification(Expression<Func<T, bool>>? filterCondition)
```

Parameters

`filterCondition` Expression<Func<T, bool>>

Properties

ComplexIncludes

```
public Collection<Func<IQueryable<T>, IIncludableQueryable<T, object>>>
ComplexIncludes { get; }
```

Property Value

Collection<Func<IQueryable<T>, IIncludableQueryable<T, object>>>

Distincts

```
public bool Distincts { get; }
```

Property Value

bool

FilterCondition

```
public Expression<Func<T, bool>> FilterCondition { get; }
```

Property Value

Expression<Func<T, bool>>

GroupBys

```
public Expression<Func<T, object>> GroupBys { get; }
```

Property Value

Expression<Func<T, object>>

OrderByDescendings

```
public Collection<Expression<Func<T, object>>> OrderByDescendings { get; }
```

Property Value

Collection<Expression<Func<T, object>>>

OrderBys

```
public Collection<Expression<Func<T, object>>> OrderBys { get; }
```

Property Value

Collection<Expression<Func<T, object>>>

SimpleIncludes

```
public Collection<Expression<Func<T, object>>> SimpleIncludes { get; }
```

Property Value

Collection<Expression<Func<T, object>>>

Methods

Distinct()

Permet d'ajouter un Distinct à la requête

```
public Specification<T> Distinct()
```

Returns

Specification<T>

GroupBy(Expression<Func<T, object>>)

Permet d'ajouter un GroupBy à la requête

```
public Specification<T> GroupBy(Expression<Func<T, object>> groupByExpression)
```

Parameters

`groupByExpression` Expression<Func<T, object>>

Returns

Specification<T>

Include(Func<IQueryable<T>, IIncludableQueryable<T, object>>)

Permet d'effectuer un .Include sur une entité

```
public Specification<T> Include(Func<IQueryable<T>, IIncludableQueryable<T,
object>> includeExpression)
```

Parameters

`includeExpression` Func<IQueryable<T>, IIncludableQueryable<T, object>>

Returns

Specification<T>

Remarks

Exemple: new Specification().Include(item => item.Include(item => item.User).ThenInclude(item => item.Address))

Include(Expression<Func<T, object>>)

Permet d'effectuer un .Include sur une entité

```
public Specification<T> Include(Expression<Func<T, object>> includeExpression)
```

Parameters

`includeExpression` Expression<Func<T, object>>

Returns

Specification<T>

Remarks

Exemple: new Specification().Include(item => item.User)

OrderBy(Expression<Func<T, object>>)

Permet d'ajouter un OrderBy à la requête
Si on en ajoute plusieurs, ils seront exécutés dans l'ordre d'ajout (OrderBy, ThenBy, ThenBy, ...)

```
public Specification<T> OrderBy(Expression<Func<T, object>> orderByExpression)
```

Parameters

`orderByExpression` Expression<Func<T, object>>

Returns

Specification<T>

OrderByDescending(Expression<Func<T, object>>)

Permet d'ajouter un OrderByDescending à la requête
Si on en ajoute plusieurs, ils seront exécutés dans l'ordre d'ajout (OrderByDescending, ThenByDescending, ThenByDescending, ...)

```
public Specification<T> OrderByDescending(Expression<Func<T,
object>> orderByDescendingExpression)
```

Parameters

`orderByDescendingExpression` Expression<Func<T, object>>

Returns

Specification<T>

SetFilterCondition(Expression<Func<T, bool>>)

```
public void SetFilterCondition(Expression<Func<T, bool>> filterExpression)
```

Parameters

`filterExpression` Expression<Func<T, bool>>