

# Namespace Func.Redis

## Classes

[ConnectionMultiplexerProvider](#)

[RedisSourcesProvider](#)

## Structs

[Error](#)

## Interfaces

[IConnectionMultiplexerProvider](#)

[IRedisService](#)

[ISourcesProvider](#)

# Class ConnectionMultiplexerProvider

Namespace: [Func.Redis](#)

Assembly: Func.Redis.dll

```
public class ConnectionMultiplexerProvider : IConnectionMultiplexerProvider
```

## Inheritance

[object](#) ← ConnectionMultiplexerProvider

## Implements

[IConnectionMultiplexerProvider](#)

## Inherited Members

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

## Constructors

### ConnectionMultiplexerProvider(RedisConfiguration)

```
public ConnectionMultiplexerProvider(RedisConfiguration config)
```

## Parameters

config [RedisConfiguration](#)

## Methods

### GetMultiplexer()

Retrieves the current instance of the connection multiplexer used to manage Redis connections.

```
public IConnectionMultiplexer GetMultiplexer()
```

## Returns

### `IConnectionMultiplexer`

An instance of `StackExchange.Redis.IConnectionMultiplexer` representing the connection multiplexer.







# Struct Error

Namespace: [Func.Redis](#)

Assembly: Func.Redis.dll

```
public readonly struct Error
```

## Inherited Members

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

## Properties

### Message

```
public string Message { get; }
```

Property Value

[string](#)

## Methods

### New(Exception)

```
[Pure]  
public static Error New(Exception ex)
```

Parameters

ex [Exception](#)

Returns

[Error](#)

## New(string)

[Pure]

```
public static Error New(string message)
```

### Parameters

message [string](#) 

### Returns

[Error](#)

# Interface IConnectionMultiplexerProvider

Namespace: [Func.Redis](#)

Assembly: Func.Redis.dll

```
public interface IConnectionMultiplexerProvider
```

## Methods

### GetMultiplexer()

Retrieves the current instance of the connection multiplexer used to manage Redis connections.

```
IConnectionMultiplexer GetMultiplexer()
```

## Returns

IConnectionMultiplexer

An instance of StackExchange.Redis.IConnectionMultiplexer representing the connection multiplexer.

# Interface IRedisService

Namespace: [Func.Redis](#)

Assembly: Func.Redis.dll

```
public interface IRedisService
```

## Methods

### ExecuteAsync<T>(Func<IDatabase, Task<T>>)

Executes the specified function within the context of a database operation and returns the result.

```
Task<Either<Error, T>> ExecuteAsync<T>(Func<IDatabase, Task<T>> exec)
```

## Parameters

**exec** [Func](#) [IDatabase](#), [Task](#) [T](#)

A function that takes an `StackExchange.Redis.IDatabase` instance and performs the desired operation.

## Returns

[Task](#) [Either](#) [Error](#), [T](#)

An `TinyFp.Either<L, R>` containing either the result of the operation or an error if the operation fails.

## Type Parameters

**T**

The type of the result produced by the function.

# ExecuteAsync<TIn, TOut>(Func<IDatabase, Task<TIn>>, Func<TIn, TOut>)

Executes a database operation and maps the result to a specified output type.

```
Task<Either<Error, TOut>> ExecuteAsync<TIn, TOut>(Func<IDatabase, Task<TIn>> exec,
Func<TIn, TOut> map)
```

## Parameters

**exec** [Func](#) <IDatabase, [Task](#) <TIn>>

A function that performs the database operation and returns a result of type **TIn**.

**map** [Func](#) <TIn, TOut>

A function that transforms the intermediate result of type **TIn** into the final result of type **TOut**.

## Returns

[Task](#) <Either<[Error](#), TOut>>

An `TinyFp.Either<L, R>` containing either an [Error](#) if the operation fails, or the mapped result of type **TOut** if the operation succeeds.

## Type Parameters

**TIn**

The type of the intermediate result produced by the database operation.

**TOut**

The type of the final result after mapping.

## Remarks

This method provides a mechanism to execute a database operation and process its result in a functional style. The **exec** function is responsible for interacting with the database, while the **map** function allows transformation of the intermediate result into the desired output.



# ExecuteUnsafeAsync<TIn, TOut>(Func<IDatabase, Task<TIn>>, Func<TIn, TOut>)

Executes an asynchronous operation on a database and maps the result to a specified output type.

```
Task<Either<Error, TOut>> ExecuteUnsafeAsync<TIn, TOut>(Func<IDatabase, Task<TIn>>  
exec, Func<TIn, TOut> map)
```

## Parameters

**exec** [Func](#) <IDatabase, [Task](#) <TIn>>

A function that performs the database operation and returns a task producing the intermediate result.

**map** [Func](#) <TIn, TOut>

A function that transforms the intermediate result into the final output type.

## Returns

[Task](#) <Either<[Error](#), TOut>>

A task that represents the asynchronous operation. The result is an `TinyFp.Either<L, R>` containing either an [Error](#) if the operation fails, or the mapped result of type **TOut** if successful.

## Type Parameters

**TIn**

The type of the intermediate result produced by the database operation.

**TOut**

The type of the final result after mapping.

## Execute<T>(Func<IDatabase, T>)

Executes the specified function within the context of a database operation and returns the result.

```
Either<Error, T> Execute<T>(Func<IDatabase, T> exec)
```

## Parameters

`exec` [Func](#) <IDatabase, T>

A function that takes an `StackExchange.Redis.IDatabase` instance and performs the desired operation.

## Returns

`Either<Error, T>`

An `TinyFp.Either<L, R>` containing either the result of the operation or an error if the operation fails.

## Type Parameters

`T`

The type of the result produced by the function.

## Execute<TIn, TOut>(Func<IDatabase, TIn>, Func<TIn, TOut>)

Executes a database operation and maps the result to a specified output type.

```
Either<Error, TOut> Execute<TIn, TOut>(Func<IDatabase, TIn> exec, Func<TIn, TOut> map)
```

## Parameters

`exec` [Func](#) <IDatabase, TIn>

A function that performs the database operation and returns a result of type `TIn`.

`map` [Func](#) <TIn, TOut>

A function that transforms the intermediate result of type **TIn** into the final result of type **TOut**.

## Returns

Either<[Error](#), TOut>

An `TinyFp.Either<L, R>` containing either an [Error](#) if the operation fails, or the mapped result of type **TOut** if the operation succeeds.

## Type Parameters

**TIn**

The type of the intermediate result produced by the database operation.

**TOut**

The type of the final result after mapping.

## Remarks

This method provides a mechanism to execute a database operation and process its result in a functional style. The `exec` function is responsible for interacting with the database, while the `map` function allows transformation of the intermediate result into the desired output.

# Interface ISourcesProvider

Namespace: [Func.Redis](#)

Assembly: Func.Redis.dll

```
public interface ISourcesProvider
```

## Methods

### GetDatabase()

Retrieves an instance of the database interface for performing operations.

```
IDatabase GetDatabase()
```

#### Returns

IDatabase

An object implementing the StackExchange.Redis.IDatabase interface, which provides methods for interacting with the database.

### GetServers()

Retrieves an array of servers currently available in the system.

```
IServer[] GetServers()
```

#### Returns

IServer[]

An array of StackExchange.Redis.IServer objects representing the available servers. The array will be empty if no servers are available.

# Class RedisSourcesProvider

Namespace: [Func.Redis](#)

Assembly: Func.Redis.dll

```
public class RedisSourcesProvider : ISourcesProvider
```








## Inheritance

[object](#)  ← RedisSourcesProvider

## Implements

[ISourcesProvider](#)

## Inherited Members

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

## Constructors

### RedisSourcesProvider(IConnectionMultiplexerProvider)

```
public RedisSourcesProvider(IConnectionMultiplexerProvider provider)
```

## Parameters

**provider** [IConnectionMultiplexerProvider](#)

## Methods

### GetDatabase()

Retrieves an instance of the database interface for performing operations.

```
public IDatabase GetDatabase()
```

## Returns

### IDatabase

An object implementing the StackExchange.Redis.IDatabase interface, which provides methods for interacting with the database.

## GetServers()

Retrieves an array of servers currently available in the system.

```
public IServer[] GetServers()
```

## Returns

### IServer[]

An array of StackExchange.Redis.IServer objects representing the available servers. The array will be empty if no servers are available.

# Namespace Func.Redis.Extensions

## Classes

[EitherLoggingExtensions](#)

[ServiceCollectionExtensions](#)

[WebApplicationExtensions](#)

## Enums

[RedisCapabilities](#)


# Class EitherLoggingExtensions

Namespace: [Func.Redis.Extensions](#)








Assembly: Func.Redis.dll

```
public static class EitherLoggingExtensions
```

## Inheritance

[object](#)  ← EitherLoggingExtensions

## Inherited Members

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



# Enum RedisCapabilities

Namespace: [Func.Redis.Extensions](#)

Assembly: Func.Redis.Extensions.dll

[Flags]

```
public enum RedisCapabilities
```

## Fields

Generic = 1

HashSet = 4

Key = 2

List = 32

Publish = 64

Set = 8

SortedSet = 16

Subscribe = 128

# Class ServiceCollectionExtensions

Namespace: [Func.Redis.Extensions](#)

Assembly: Func.Redis.Extensions.dll

```
public static class ServiceCollectionExtensions
```

## Inheritance

[object](#)  ← ServiceCollectionExtensions

## Inherited Members

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

## Methods

### AddRedisSerDes<T>(IServiceCollection)

Add a custom implementation of [IRedisSerDes](#)

```
public static IServiceCollection AddRedisSerDes<T>(this IServiceCollection services)  
where T : IRedisSerDes
```

## Parameters

**services** [IServiceCollection](#) 

## Returns

[IServiceCollection](#) 

## Type Parameters

**T**

# AddRedis<T>(IServiceCollection, IConfiguration, RedisCapabilities, bool, params Assembly[])

Add Redis services to the service collection

```
public static IServiceCollection AddRedis<T>(this IServiceCollection services,
    IConfiguration config, RedisCapabilities capabilities, bool addLogging = false,
    params Assembly[] assemblies) where T : IRedisSerDes
```

## Parameters

**services** [IServiceCollection](#)

**config** [IConfiguration](#)

Configuration provider to retrieve configuration parameters for [RedisConfiguration](#) and [RedisKeyConfiguration](#)

**capabilities** [RedisCapabilities](#)

Specify capabilities to be enabled

**addLogging** [bool](#)

Enable/disable logging

**assemblies** [Assembly](#)[]

Specify assemblies to be scanned for [IRedisSubscriber](#) implementations

## Returns

[IServiceCollection](#)

## Type Parameters

**T**

## Exceptions

[KeyNotFoundException](#)

# AddSystemJsonRedisSerDes(IServiceCollection)

Add [SystemJsonRedisSerDes](#) Redis serializer/deserializer

```
public static IServiceCollection AddSystemJsonRedisSerDes(this  
IServiceCollection services)
```

## Parameters

**services** [IServiceCollection](#) 

## Returns

[IServiceCollection](#) 

# Class WebApplicationExtensions

Namespace: [Func.Redis.Extensions](#)

Assembly: Func.Redis.Extensions.dll

```
public static class WebApplicationExtensions
```

## Inheritance

[object](#)  ← WebApplicationExtensions

## Inherited Members

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

## Methods

### UseRedisSubscribers(WebApplication)

```
public static WebApplication UseRedisSubscribers(this WebApplication app)
```

## Parameters

**app** [WebApplication](#) 

## Returns

[WebApplication](#) 

# Namespace Func.Redis.HashSet

## Interfaces

[IRedisHashSetService](#)

# Interface IRedisHashSetService

Namespace: [Func.Redis.HashSet](#)

Assembly: Func.Redis.dll

```
public interface IRedisHashSetService
```

## Methods

### Delete(string, string)

Deletes a specific field associated with the given key from the data store.

```
Either<Error, Unit> Delete(string key, string field)
```

#### Parameters

key [string](#) 

The key identifying the entity from which the field should be deleted. Cannot be null or empty.

field [string](#) 

The name of the field to delete. Cannot be null or empty.

#### Returns

Either<[Error](#), Unit>

An `TinyFp.Either<L, R>` indicating the result of the operation. Returns `Unit` if the deletion is successful, or an [Error](#) if the operation fails.

### Delete(string, params string[])

Deletes specified fields associated with the given key from the data store.

```
Either<Error, Unit> Delete(string key, params string[] fields)
```

## Parameters

**key** [string](#)

The unique identifier for the data entry to modify. Cannot be null or empty.

**fields** [string](#)[]

An array of field names to delete. If no fields are specified, the method deletes all fields associated with the key.

## Returns

Either<[Error](#), Unit>

An `TinyFp.Either<L, R>` indicating the result of the operation. Returns `Unit` if the operation succeeds, or an [Error](#) if the operation fails.

## Remarks

This method performs a partial or complete deletion of fields associated with the specified key. If the key does not exist, the operation fails and returns an [Error](#).

## DeleteAsync(string, string)

Deletes a specific field associated with the given key from the data store.

```
Task<Either<Error, Unit>> DeleteAsync(string key, string field)
```

## Parameters

**key** [string](#)

The key identifying the entity from which the field should be deleted. Cannot be null or empty.

**field** [string](#)

The name of the field to delete. Cannot be null or empty.



## Returns

[Task](#) [↗](#) <Either<[Error](#), Unit>>

An `TinyFp.Either<L, R>` indicating the result of the operation. Returns `Unit` if the deletion is successful, or an [Error](#) if the operation fails.

## DeleteAsync(string, params string[])

Deletes specified fields associated with the given key from the data store.

```
Task<Either<Error, Unit>> DeleteAsync(string key, params string[] fields)
```

## Parameters

**key** [string](#) [↗](#)

The unique identifier for the data entry to modify. Cannot be null or empty.

**fields** [string](#) [↗](#)[]

An array of field names to delete. If no fields are specified, the method deletes all fields associated with the key.

## Returns

[Task](#) [↗](#) <Either<[Error](#), Unit>>

An `TinyFp.Either<L, R>` indicating the result of the operation. Returns `Unit` if the operation succeeds, or an [Error](#) if the operation fails.

## Remarks

This method performs a partial or complete deletion of fields associated with the specified key. If the key does not exist, the operation fails and returns an [Error](#).

## Get(string, params (Type, string)[])

Retrieves an array of optional objects based on the specified key and type-field pairs.

```
Either<Error, Option<object>[]> Get(string key, params (Type, string)[] typeFields)
```

## Parameters

**key** [string](#)

The key used to identify the data to retrieve. Cannot be null or empty.

**typeFields** ([Type](#), [string](#))[]

A collection of type-field pairs, where each pair specifies the type and field name to query. The type represents the expected type of the data, and the field name identifies the specific field to retrieve.

## Returns

Either<[Error](#), Option<[object](#)>[]>

An `TinyFp.Either<L, R>` containing either an [Error](#) object if the operation fails, or an array of `TinyFp.Option<A>` objects representing the retrieved data. Each element in the array corresponds to a type-field pair, and will be [null](#) if the data for that pair is unavailable.

## GetAllAsync<T>(string)

Retrieves all values associated with the specified key.

```
Task<Either<Error, Option<(string, T)[]>>> GetAllAsync<T>(string key)
```

## Parameters

**key** [string](#)

The key used to look up the values. Cannot be null or empty.

## Returns

[Task](#) <Either<[Error](#), Option<([string](#), T)[]>>>

An `TinyFp.Either<L, R>` containing either an error or an optional array of key-value pairs. If successful, the result is an `TinyFp.Option<A>` containing an array of tuples, where each

tuple consists of a string key and a value of type **T**. If no values are found, the `TinyFp.Option<A>` will be empty.

## Type Parameters

**T**

The type of the values to retrieve.

## GetAll<T>(string)

Retrieves all values associated with the specified key.


```
Either<Error, Option<(string, T)[]>> GetAll<T>(string key)
```

## Parameters

key [string](#) 

The key used to look up the values. Cannot be null or empty.

## Returns

```
Either<Error, Option<(string , T)[]>>
```

An `TinyFp.Either<L, R>` containing either an error or an optional array of key-value pairs. If successful, the result is an `TinyFp.Option<A>` containing an array of tuples, where each tuple consists of a string key and a value of type **T**. If no values are found, the `TinyFp.Option<A>` will be empty.

## Type Parameters

**T**

The type of the values to retrieve.

## GetAsync(string, params (Type, string)[])

```
Task<Either<Error, Option<object>[]>> GetAsync(string key, params (Type, string)
[] typeFields)
```

## Parameters

key [string](#)

typeFields ([Type](#), [string](#))[]

## Returns

[Task](#) <Either<[Error](#), Option<[object](#)>[]>>

# GetAsync<T>(string, string)

Retrieves the value associated with the specified key and field.

```
Task<Either<Error, Option<T>>> GetAsync<T>(string key, string field)
```

## Parameters

key [string](#)

The key identifying the collection or object to search. Cannot be null or empty.

field [string](#)

The field within the collection or object to retrieve the value from. Cannot be null or empty.

## Returns

[Task](#) <Either<[Error](#), Option<T>>>

An `TinyFp.Either<L, R>` containing either an error or an optional value of type `T`. If the key or field does not exist, the result will contain an empty `TinyFp.Option<A>`.

## Type Parameters

`T`

The type of the value to retrieve.

## GetAsync<T>(string, params string[])

Retrieves an array of optional values associated with the specified key and fields.

```
Task<Either<Error, Option<T>[]>> GetAsync<T>(string key, params string[] fields)
```

### Parameters

**key** [string](#)

The key used to identify the values. Cannot be null or empty.

**fields** [string](#)[]

An optional array of field names to filter the values. If no fields are specified, all values associated with the key are retrieved.

### Returns

[Task](#) <Either<[Error](#), Option<T>[]>>

An `TinyFp.Either<L, R>` containing either an [Error](#) if the operation fails, or an array of `TinyFp.Option<A>` representing the retrieved values. The array will be empty if no values are found.

### Type Parameters

**T**

The type of the values to retrieve.

## GetFieldKeys(string)

Retrieves the field keys associated with the specified key.

```
Either<Error, Option<string[]>> GetFieldKeys(string key)
```

## Parameters

key [string](#)

The key for which field keys are to be retrieved. Cannot be null or empty.

## Returns

`Either<Error, Option<string[]>>`

An `TinyFp.Either<L, R>` containing either an [Error](#) if the operation fails, or an `TinyFp.Option<A>` representing the field keys as a string array. If no field keys are found, the `TinyFp.Option<A>` will be empty.

## GetFieldKeysAsync(string)

Retrieves the field keys associated with the specified key.

```
Task<Either<Error, Option<string[]>>> GetFieldKeysAsync(string key)
```

## Parameters

key [string](#)

The key for which field keys are to be retrieved. Cannot be null or empty.

## Returns

`Task<Either<Error, Option<string[]>>>`

An `TinyFp.Either<L, R>` containing either an [Error](#) if the operation fails, or an `TinyFp.Option<A>` representing the field keys as a string array. If no field keys are found, the `TinyFp.Option<A>` will be empty.

## GetValuesAsync<T>(string)

Retrieves the values associated with the specified key, if available.

```
Task<Either<Error, Option<T[]>>> GetValuesAsync<T>(string key)
```

## Parameters

key [string](#)

The key used to look up the values. Cannot be null or empty.

## Returns

[Task](#) <Either<[Error](#), Option<T[]>>>

An `TinyFp.Either<L, R>` containing either an error or an optional array of values. If the key is not found, the result will contain an empty option.

## Type Parameters

T

The type of the values to retrieve.

## GetValues<T>(string)

Retrieves the values associated with the specified key, if available.

```
Either<Error, Option<T[]>> GetValues<T>(string key)
```

## Parameters

key [string](#)

The key used to look up the values. Cannot be null or empty.

## Returns

Either<[Error](#), Option<T[]>>

An `TinyFp.Either<L, R>` containing either an error or an optional array of values. If the key is not found, the result will contain an empty option.

## Type Parameters

T

The type of the values to retrieve.

## Get<T>(string, string)

Retrieves the value associated with the specified key and field.

```
Either<Error, Option<T>> Get<T>(string key, string field)
```

### Parameters

key [string](#)

The key identifying the collection or object to search. Cannot be null or empty.

field [string](#)

The field within the collection or object to retrieve the value from. Cannot be null or empty.

### Returns

Either<[Error](#), Option<T>>

An `TinyFp.Either<L, R>` containing either an error or an optional value of type `T`. If the key or field does not exist, the result will contain an empty `TinyFp.Option<A>`.

### Type Parameters

`T`

The type of the value to retrieve.

## Get<T>(string, params string[])

Retrieves an array of optional values associated with the specified key and fields.

```
Either<Error, Option<T>[]> Get<T>(string key, params string[] fields)
```

### Parameters



key [string](#)

The key used to identify the values. Cannot be null or empty.

fields [string](#)[]

An optional array of field names to filter the values. If no fields are specified, all values associated with the key are retrieved.

## Returns

Either<[Error](#), Option<T>[]>

An `TinyFp.Either<L, R>` containing either an [Error](#) if the operation fails, or an array of `TinyFp.Option<A>` representing the retrieved values. The array will be empty if no values are found.

## Type Parameters

T

The type of the values to retrieve.

## SetAsync<T>(string, string, T)

Sets the specified value for a given key and field in the data store.

```
Task<Either<Error, Unit>> SetAsync<T>(string key, string field, T value)
```

## Parameters

key [string](#)

The key identifying the data store entry. Cannot be null or empty.

field [string](#)

The field within the entry to set the value for. Cannot be null or empty.

value T

The value to set for the specified field. Cannot be null.

## Returns

[Task](#)  <Either<[Error](#), Unit>>

An `TinyFp.Either<L, R>` indicating the result of the operation. Returns `Unit` if the operation succeeds, or an [Error](#) if it fails.

## Type Parameters

**T**

The type of the value to set.

## Remarks

This method updates the value for the specified field within the entry identified by the key. If the key or field does not exist, or if the operation fails, an [Error](#) is returned.

## SetAsync<T>(string, params (string, T)[])

Sets multiple key-value pairs in the specified storage, associating each pair with the given key.

```
Task<Either<Error, Unit>> SetAsync<T>(string key, params (string, T)[] pairs)
```

## Parameters

**key** [string](#) 

The key used to group the pairs in the storage. Cannot be null or empty.

**pairs** ([string](#) , T)[]

An array of tuples, where each tuple contains a string identifier and a value of type **T**. The identifiers must be unique within the specified key.

## Returns

[Task](#)  <Either<[Error](#), Unit>>

An `TinyFp.Either<L, R>` indicating the result of the operation. Returns `Unit` if the operation succeeds; otherwise, returns an [Error](#) describing the failure.

## Type Parameters

**T**

The type of the values to be stored.

## Set<T>(string, string, T)

Sets the specified value for a given key and field in the data store.

```
Either<Error, Unit> Set<T>(string key, string field, T value)
```

### Parameters

**key** [string](#)

The key identifying the data store entry. Cannot be null or empty.

**field** [string](#)

The field within the entry to set the value for. Cannot be null or empty.

**value** T

The value to set for the specified field. Cannot be null.

### Returns

Either<[Error](#), Unit>

An `TinyFp.Either<L, R>` indicating the result of the operation. Returns `Unit` if the operation succeeds, or an [Error](#) if it fails.

## Type Parameters

**T**

The type of the value to set.

### Remarks

This method updates the value for the specified field within the entry identified by the key. If the key or field does not exist, or if the operation fails, an [Error](#) is returned.

## Set<T>(string, params (string, T)[])

Sets multiple key-value pairs in the specified storage, associating each pair with the given key.

```
Either<Error, Unit> Set<T>(string key, params (string, T)[] pairs)
```

### Parameters

key [string](#)

The key used to group the pairs in the storage. Cannot be null or empty.

pairs ([string](#), T)[]

An array of tuples, where each tuple contains a string identifier and a value of type `T`. The identifiers must be unique within the specified key.

### Returns

Either<[Error](#), Unit>

An `TinyFp.Either<L, R>` indicating the result of the operation. Returns `Unit` if the operation succeeds; otherwise, returns an [Error](#) describing the failure.

### Type Parameters

`T`

The type of the values to be stored.

# Namespace Func.Redis.Key

## Interfaces

[IRedisKeyService](#)

# Interface IRedisKeyService

Namespace: [Func.Redis.Key](#)

Assembly: Func.Redis.dll

```
public interface IRedisKeyService
```

## Methods

### Delete(string)

Deletes the item associated with the specified key.

```
Either<Error, Unit> Delete(string key)
```

#### Parameters

key [string](#) 

The key identifying the item to delete. Cannot be null or empty.

#### Returns

Either<[Error](#), Unit>

An `TinyFp.Either<L, R>` indicating the result of the operation. Returns `TinyFp.Unit` if the deletion is successful, or an [Error](#) if the operation fails.

### Delete(string[])

Deletes the specified items identified by their keys.

```
Either<Error, Unit> Delete(string[] keys)
```

#### Parameters

keys [string](#)[]

An array of keys representing the items to delete. Each key must be a non-null, non-empty string.

## Returns

Either<[Error](#), Unit>

An `TinyFp.Either<L, R>` indicating the result of the operation. If the operation succeeds, the result will contain `Unit`. If the operation fails, the result will contain an [Error](#) describing the failure.

## Remarks

This method attempts to delete all items corresponding to the provided keys. If one or more keys cannot be deleted, the operation will fail and return an [Error](#).

## DeleteAsync(string)

Deletes the item associated with the specified key.

```
Task<Either<Error, Unit>> DeleteAsync(string key)
```

## Parameters

key [string](#)

The key identifying the item to delete. Cannot be null or empty.

## Returns

[Task](#) <Either<[Error](#), Unit>>

An `TinyFp.Either<L, R>` indicating the result of the operation. Returns `TinyFp.Unit` if the deletion is successful, or an [Error](#) if the operation fails.

## DeleteAsync(string[])

Deletes the specified items identified by their keys.

```
Task<Either<Error, Unit>> DeleteAsync(string[] keys)
```

## Parameters

keys [string](#)[]

An array of keys representing the items to delete. Each key must be a non-null, non-empty string.

## Returns

[Task](#) <Either<[Error](#), Unit>>

An `TinyFp.Either<L, R>` indicating the result of the operation. If the operation succeeds, the result will contain `Unit`. If the operation fails, the result will contain an [Error](#) describing the failure.

## Remarks

This method attempts to delete all items corresponding to the provided keys. If one or more keys cannot be deleted, the operation will fail and return an [Error](#).

## GetAsync<T>(string)

Retrieves the value associated with the specified key, if it exists.

```
Task<Either<Error, Option<T>>> GetAsync<T>(string key)
```

## Parameters

key [string](#)

The key used to locate the value. Cannot be null or empty.

## Returns

[Task](#) <Either<[Error](#), Option<T>>>

An `TinyFp.Either<L, R>` containing either an error if the operation fails, or an `TinyFp.Option<A>` representing the value if found. Returns `None` if the key does not exist.



## Type Parameters

**T**

The type of the value to retrieve.

## GetAsync<T>(string[])

Retrieves an array of optional values associated with the specified keys.

```
Task<Either<Error, Option<T>[]>> GetAsync<T>(string[] keys)
```

## Parameters

**keys** [string](#)[]

The keys used to locate the values. Each key must be a non-null, non-empty string.

## Returns

[Task](#) <Either<[Error](#), Option<T>[]>>

An `TinyFp.Either<L, R>` containing either an [Error](#) if the operation fails, or an array of `TinyFp.Option<A>` representing the values associated with the specified keys. If a key does not have an associated value, the corresponding element in the array will be `TinyFp.Option<A>.None()`.

## Type Parameters

**T**

The type of the values to retrieve.

## GetKeys(string)

Retrieves an array of keys that match the specified pattern.

```
Either<Error, string[]> GetKeys(string pattern)
```

## Parameters

**pattern** [string](#)

The pattern to match keys against. This can include wildcard characters or other pattern-matching syntax supported by the underlying implementation.

## Returns

`Either<Error, string[]>`

An `TinyFp.Either<L, R>` containing either an [Error](#) if the operation fails, or an array of strings representing the matching keys. If no keys match the pattern, the array will be empty.

## Remarks

The method uses pattern matching to filter keys. Ensure the pattern syntax is valid for the underlying implementation. The operation may fail due to connectivity issues or invalid patterns, in which case an [Error](#) will be returned.

## GetKeysAsync(string)

Retrieves an array of keys that match the specified pattern.

```
Task<Either<Error, string[]>> GetKeysAsync(string pattern)
```

## Parameters

**pattern** [string](#)

The pattern to match keys against. This can include wildcard characters or other pattern-matching syntax supported by the underlying implementation.

## Returns

`Task<Either<Error, string[]>>`

An `TinyFp.Either<L, R>` containing either an [Error](#) if the operation fails, or an array of strings representing the matching keys. If no keys match the pattern, the array will be empty.

## Remarks

The method uses pattern matching to filter keys. Ensure the pattern syntax is valid for the underlying implementation. The operation may fail due to connectivity issues or invalid patterns, in which case an [Error](#) will be returned.

## Get<T>(string)

Retrieves the value associated with the specified key, if it exists.

```
Either<Error, Option<T>> Get<T>(string key)
```

## Parameters

key [string](#) 

The key used to locate the value. Cannot be null or empty.

## Returns

Either<[Error](#), Option<T>>

An `TinyFp.Either<L, R>` containing either an error if the operation fails, or an `TinyFp.Option<A>` representing the value if found. Returns `None` if the key does not exist.

## Type Parameters

T

The type of the value to retrieve.

## Get<T>(string[])

Retrieves an array of optional values associated with the specified keys.

```
Either<Error, Option<T>[]> Get<T>(string[] keys)
```

## Parameters

keys [string](#)[]

The keys used to locate the values. Each key must be a non-null, non-empty string.

## Returns

Either<[Error](#), Option<T>[]>

An `TinyFp.Either<L, R>` containing either an [Error](#) if the operation fails, or an array of `TinyFp.Option<A>` representing the values associated with the specified keys. If a key does not have an associated value, the corresponding element in the array will be `TinyFp.Option<A>.None()`.

## Type Parameters

T

The type of the values to retrieve.

## RenameKey(string, string)

Renames an existing key to a new key within the system.

```
Either<Error, Unit> RenameKey(string key, string newKey)
```

## Parameters

key [string](#)

The current name of the key to be renamed. Cannot be null or empty.

newKey [string](#)

The new name for the key. Cannot be null or empty.

## Returns

Either<[Error](#), Unit>

An `TinyFp.Either<L, R>` containing either an error describing the failure or a unit value indicating success.

## Remarks

If the operation succeeds, the key will be updated to the specified new key. If the operation fails, an error will be returned indicating the reason for failure.

## RenameKeyAsync(string, string)

Renames an existing key to a new key within the system.

```
Task<Either<Error, Unit>> RenameKeyAsync(string key, string newKey)
```

## Parameters

key [string](#) 

The current name of the key to be renamed. Cannot be null or empty.

newKey [string](#) 

The new name for the key. Cannot be null or empty.

## Returns

[Task](#)  <Either<[Error](#), Unit>>

An `TinyFp.Either<L, R>` containing either an error describing the failure or a unit value indicating success.

## Remarks

If the operation succeeds, the key will be updated to the specified new key. If the operation fails, an error will be returned indicating the reason for failure.

## SetAsync<T>(string, T)

Stores a value in the underlying data store associated with the specified key.

```
Task<Either<Error, Unit>> SetAsync<T>(string key, T value)
```

## Parameters

key [string](#)

The key used to identify the stored value. Cannot be null or empty.

value T

The value to store. Must be serializable and compatible with the data store.

## Returns

[Task](#) <Either<[Error](#), Unit>>

An `TinyFp.Either<L, R>` indicating the result of the operation. Returns `Unit` if the operation succeeds, or an [Error](#) if it fails.

## Type Parameters

T

The type of the value to store.

## Remarks

The method may fail if the key is invalid, the value cannot be serialized, or if there is an issue with the data store.

## SetAsync<T>(params (string, T)[])

```
Task<Either<Error, Unit>> SetAsync<T>(params (string, T)[] pairs)
```

## Parameters

pairs ([string](#), T)[]

## Returns

[Task](#) <Either<[Error](#), Unit>>

## Type Parameters

T

## Set<T>(string, T)

Stores a value in the underlying data store associated with the specified key.

```
Either<Error, Unit> Set<T>(string key, T value)
```

### Parameters

key [string](#) 

The key used to identify the stored value. Cannot be null or empty.

value T

The value to store. Must be serializable and compatible with the data store.

### Returns

Either<[Error](#), Unit>

An `TinyFp.Either<L, R>` indicating the result of the operation. Returns `Unit` if the operation succeeds, or an [Error](#) if it fails.

### Type Parameters

T

The type of the value to store.

### Remarks

The method may fail if the key is invalid, the value cannot be serialized, or if there is an issue with the data store.

## Set<T>(params (string, T)[])

Sets multiple key-value pairs in the underlying storage.

```
Either<Error, Unit> Set<T>(params (string, T)[] pairs)
```

## Parameters

`pairs` ([string](#), T)[]

An array of tuples, where each tuple contains a key as a string and a value of type `T`. The key must not be null or empty.

## Returns

`Either<Error, Unit>`

An `TinyFp.Either<L, R>` indicating the result of the operation. Returns `Unit` if the operation succeeds, or an [Error](#) if it fails.

## Type Parameters

`T`

The type of the values to be stored.

## Remarks

This method allows setting multiple key-value pairs in a single operation. If any pair fails to be set, the operation will return an [Error](#) describing the issue.



# Namespace Func.Redis.List

## Interfaces

[IRedisListService](#)

# Interface IRedisListService

Namespace: [Func.Redis.List](#)

Assembly: Func.Redis.dll

```
public interface IRedisListService
```

## Methods

### AppendAsync<T>(string, T)

Appends a value to the specified key in the underlying data store.

```
Task<Either<Error, Unit>> AppendAsync<T>(string key, T value)
```

## Parameters

key [string](#) 

The key to which the value will be appended. Cannot be null or empty.

value T

The value to append. Must be compatible with the type expected by the key.

## Returns

[Task](#)  <Either<[Error](#), Unit>>

An `TinyFp.Either<L, R>` indicating the result of the operation. Returns `Unit` if the operation succeeds, or an [Error](#) if the operation fails.

## Type Parameters

T

The type of the value to append.

## Remarks

The operation may fail if the key does not exist, the value is incompatible with the key, or if there is an issue with the underlying data store. Check the returned `TinyFp.Either<L, R>` to determine the outcome of the operation.

## AppendAsync<T>(string, params T[])

Appends the specified values to the collection associated with the given key.

```
Task<Either<Error, Unit>> AppendAsync<T>(string key, params T[] values)
```

## Parameters

**key** [string](#) 

The key identifying the collection to which the values will be appended. Cannot be null or empty.

**values** `T[]`

The values to append to the collection. Cannot be null.

## Returns

[Task](#)  `<Either<Error, Unit>>`

An `TinyFp.Either<L, R>` indicating the result of the operation. Returns `Unit` if the operation succeeds; otherwise, returns an [Error](#) describing the failure.

## Type Parameters

**T**

The type of the values to append.

## Append<T>(string, T)

Appends a value to the specified key in the underlying data store.

```
Either<Error, Unit> Append<T>(string key, T value)
```

## Parameters

key [string](#) 

The key to which the value will be appended. Cannot be null or empty.

value T

The value to append. Must be compatible with the type expected by the key.

## Returns

Either<[Error](#), Unit>

An `TinyFp.Either<L, R>` indicating the result of the operation. Returns `Unit` if the operation succeeds, or an [Error](#) if the operation fails.

## Type Parameters

T

The type of the value to append.

## Remarks

The operation may fail if the key does not exist, the value is incompatible with the key, or if there is an issue with the underlying data store. Check the returned `TinyFp.Either<L, R>` to determine the outcome of the operation.

## Append<T>(string, params T[])

Appends the specified values to the collection associated with the given key.

```
Either<Error, Unit> Append<T>(string key, params T[] values)
```

## Parameters

key [string](#) 

The key identifying the collection to which the values will be appended. Cannot be null or empty.

**values** T[]

The values to append to the collection. Cannot be null.

## Returns

Either<[Error](#), Unit>

An `TinyFp.Either<L, R>` indicating the result of the operation. Returns `Unit` if the operation succeeds; otherwise, returns an [Error](#) describing the failure.

## Type Parameters

**T**

The type of the values to append.

# GetAsync<T>(string, long)

Retrieves the value associated with the specified key and index.

```
Task<Either<Error, Option<T>>> GetAsync<T>(string key, long index)
```

## Parameters

**key** [string](#)

The key used to identify the value. Cannot be null or empty.

**index** [long](#)

The index used to locate the value. Must be a non-negative number.

## Returns

[Task](#) <Either<[Error](#), Option<T>>>>

An `TinyFp.Either<L, R>` containing either an error or an optional value. If the operation succeeds, the result will contain an `TinyFp.Option<A>` representing the value. If the

operation fails, the result will contain an [Error](#) describing the failure.

## Type Parameters

**T**

The type of the value to retrieve.

## GetAsync<T>(string, long, long)

Retrieves a range of values associated with the specified key.

```
Task<Either<Error, Option<T>[]>> GetAsync<T>(string key, long start, long stop)
```

## Parameters

**key** [string](#) 

The key identifying the collection of values. Cannot be null or empty.

**start** [long](#) 

The starting index of the range to retrieve. Must be greater than or equal to 0.

**stop** [long](#) 

The ending index of the range to retrieve. Must be greater than or equal to **start**.

## Returns

[Task](#)  <Either<[Error](#), Option<T>[]>>

An `TinyFp.Either<L, R>` containing either an [Error](#) if the operation fails, or an array of `TinyFp.Option<A>` representing the values in the specified range. The array will be empty if no values are found within the range.

## Type Parameters

**T**

The type of the values to retrieve.

## Get<T>(string, long)

Retrieves the value associated with the specified key and index.

```
Either<Error, Option<T>> Get<T>(string key, long index)
```

### Parameters

key [string](#)

The key used to identify the value. Cannot be null or empty.

index [long](#)

The index used to locate the value. Must be a non-negative number.

### Returns

Either<[Error](#), Option<T>>

An `TinyFp.Either<L, R>` containing either an error or an optional value. If the operation succeeds, the result will contain an `TinyFp.Option<A>` representing the value. If the operation fails, the result will contain an [Error](#) describing the failure.

### Type Parameters

T

The type of the value to retrieve.

## Get<T>(string, long, long)

Retrieves a range of values associated with the specified key.

```
Either<Error, Option<T>[]> Get<T>(string key, long start, long stop)
```

### Parameters

key [string](#)

The key identifying the collection of values. Cannot be null or empty.

start [long](#)

The starting index of the range to retrieve. Must be greater than or equal to 0.

stop [long](#)

The ending index of the range to retrieve. Must be greater than or equal to `start`.

## Returns

`Either<Error, Option<T>[]>`

An `TinyFp.Either<L, R>` containing either an [Error](#) if the operation fails, or an array of `TinyFp.Option<A>` representing the values in the specified range. The array will be empty if no values are found within the range.

## Type Parameters

**T**

The type of the values to retrieve.

## PopAsync<T>(string)

Removes and retrieves the value associated with the specified key from the underlying storage.

```
Task<Either<Error, Option<T>>> PopAsync<T>(string key)
```

## Parameters

key [string](#)

The key identifying the value to remove and retrieve. Cannot be null or empty.

## Returns

[Task](#) <Either<[Error](#), Option<T>>>>

An `TinyFp.Either<L, R>` containing an [Error](#) if the operation fails, or an `TinyFp.Option<A>` representing the retrieved value if successful. The `TinyFp.Option<A>` will be empty if the key does not exist.



## Type Parameters

**T**

The type of the value to retrieve.

## PopAsync<T>(string, long)

Retrieves and removes up to the specified number of items associated with the given key.

```
Task<Either<Error, Option<T>[]>> PopAsync<T>(string key, long count)
```

## Parameters

**key** [string](#)

The key identifying the collection of items to pop. Cannot be null or empty.

**count** [long](#)

The maximum number of items to retrieve. Must be greater than zero.

## Returns

[Task](#) <Either<[Error](#), Option<T>[]>>

An `TinyFp.Either<L, R>` containing either an [Error](#) if the operation fails, or an array of `TinyFp.Option<A>` representing the retrieved items. The array will be empty if no items are available.

## Type Parameters

**T**

The type of the items to retrieve.

## Pop<T>(string)

Removes and retrieves the value associated with the specified key from the underlying storage.

```
Either<Error, Option<T>> Pop<T>(string key)
```

## Parameters

key [string](#)

The key identifying the value to remove and retrieve. Cannot be null or empty.

## Returns

Either<[Error](#), Option<T>>

An `TinyFp.Either<L, R>` containing an [Error](#) if the operation fails, or an `TinyFp.Option<A>` representing the retrieved value if successful. The `TinyFp.Option<A>` will be empty if the key does not exist.

## Type Parameters

T

The type of the value to retrieve.

## Pop<T>(string, long)

Retrieves and removes up to the specified number of items associated with the given key.

```
Either<Error, Option<T>[]> Pop<T>(string key, long count)
```

## Parameters

key [string](#)

The key identifying the collection of items to pop. Cannot be null or empty.

count [long](#)

The maximum number of items to retrieve. Must be greater than zero.

## Returns

Either<[Error](#), Option<T>[]>

An `TinyFp.Either<L, R>` containing either an [Error](#) if the operation fails, or an array of `TinyFp.Option<A>` representing the retrieved items. The array will be empty if no items are available.

## Type Parameters

**T**

The type of the items to retrieve.

## PrependAsync<T>(string, T)

Adds the specified values to the beginning of the collection associated with the given key.

```
Task<Either<Error, Unit>> PrependAsync<T>(string key, T value)
```

## Parameters


**key** [string](#) 

The key identifying the collection to which the values will be prepended. Cannot be null or empty.

**value** T

## Returns

[Task](#)  <Either<[Error](#), Unit>>

An `TinyFp.Either<L, R>` indicating the result of the operation. Returns [true](#)  if the operation succeeds; otherwise, contains an [Error](#) describing the failure.

## Type Parameters

**T**

The type of the values to prepend.

## Remarks

This method modifies the collection associated with the specified key by adding the values to the beginning. If the key does not exist, a new collection may be created depending on the implementation.

## PrependAsync<T>(string, params T[])

Adds a key-value pair to the beginning of a collection or data structure.

```
Task<Either<Error, Unit>> PrependAsync<T>(string key, params T[] values)
```

### Parameters

**key** [string](#)

The key to associate with the value. Cannot be null or empty.

**values** T[]

### Returns

[Task](#) <Either<[Error](#), Unit>>

An `TinyFp.Either<L, R>` indicating the result of the operation. Returns `TinyFp.Unit` if the operation succeeds, or an [Error](#) if the operation fails.

### Type Parameters

**T**

The type of the value to associate with the specified key.

### Remarks

This method prepends the specified key-value pair, ensuring the key is unique within the collection. If the key already exists, the operation may fail and return an [Error](#).

## Prepend<T>(string, T)

Adds a key-value pair to the beginning of a collection or data structure.

```
Either<Error, Unit> Prepend<T>(string key, T value)
```

## Parameters

key [string](#)

The key to associate with the value. Cannot be null or empty.

value T

The value to associate with the key.

## Returns

Either<[Error](#), Unit>

An `TinyFp.Either<L, R>` indicating the result of the operation. Returns `TinyFp.Unit` if the operation succeeds, or an [Error](#) if the operation fails.

## Type Parameters

T

The type of the value to associate with the specified key.

## Remarks

This method prepends the specified key-value pair, ensuring the key is unique within the collection. If the key already exists, the operation may fail and return an [Error](#).

## Prepend<T>(string, params T[])

Adds the specified values to the beginning of the collection associated with the given key.

```
Either<Error, Unit> Prepend<T>(string key, params T[] values)
```

## Parameters

key [string](#)

The key identifying the collection to which the values will be prepended. Cannot be null or empty.

**values** T[]

The values to prepend to the collection. If no values are provided, the method performs no operation.

## Returns

Either<[Error](#), Unit>

An `TinyFp.Either<L, R>` indicating the result of the operation. Returns [true](#) if the operation succeeds; otherwise, contains an [Error](#) describing the failure.

## Type Parameters

**T**

The type of the values to prepend.

## Remarks

This method modifies the collection associated with the specified key by adding the values to the beginning. If the key does not exist, a new collection may be created depending on the implementation.

## ShiftAsync<T>(string)

Retrieves the value associated with the specified key and shifts it into an optional result.

```
Task<Either<Error, Option<T>>> ShiftAsync<T>(string key)
```

## Parameters

**key** [string](#)

The key used to locate the value. Cannot be null or empty.

## Returns

[Task](#) `<Either<Error, Option<T>>>`

An `TinyFp.Either<L, R>` containing either an error if the operation fails, or an optional value if the key is found.

## Type Parameters

**T**

The type of the value to retrieve.

## ShiftAsync<T>(string, long)

Retrieves and removes up to the specified number of items associated with the given key.

```
Task<Either<Error, Option<T>[]>> ShiftAsync<T>(string key, long count)
```

## Parameters

**key** [string](#)

The key identifying the collection of items to shift. Cannot be null or empty.

**count** [long](#)

The maximum number of items to retrieve and remove. Must be greater than zero.

## Returns

[Task](#) `<Either<Error, Option<T>[]>>`

An `TinyFp.Either<L, R>` containing either an [Error](#) if the operation fails, or an array of `TinyFp.Option<A>` representing the retrieved items. The array will be empty if no items are available.

## Type Parameters

**T**

The type of the items to retrieve.

## Shift<T>(string)

Retrieves the value associated with the specified key and shifts it into an optional result.

```
Either<Error, Option<T>> Shift<T>(string key)
```

### Parameters

key [string](#)

The key used to locate the value. Cannot be null or empty.

### Returns

Either<[Error](#), Option<T>>

An `TinyFp.Either<L, R>` containing either an error if the operation fails, or an optional value if the key is found.

### Type Parameters

T

The type of the value to retrieve.

## Shift<T>(string, long)

Retrieves and removes up to the specified number of items associated with the given key.

```
Either<Error, Option<T>[]> Shift<T>(string key, long count)
```

### Parameters

key [string](#)

The key identifying the collection of items to shift. Cannot be null or empty.

count [long](#)

The maximum number of items to retrieve and remove. Must be greater than zero.



## Returns

`Either<Error, Option<T>[]>`

An `TinyFp.Either<L, R>` containing either an [Error](#) if the operation fails, or an array of `TinyFp.Option<A>` representing the retrieved items. The array will be empty if no items are available.

## Type Parameters

**T**

The type of the items to retrieve.

## Size(string)

Retrieves the size, in bytes, of the data associated with the specified key.

```
Either<Error, long> Size(string key)
```

## Parameters

**key** [string](#)<sup>↗</sup>

The key identifying the data whose size is to be retrieved. Cannot be null or empty.

## Returns

`Either<Error, long↗>`

An `TinyFp.Either<L, R>` containing either an [Error](#) if the operation fails, or a [long](#)<sup>↗</sup> representing the size of the data in bytes if successful.

## SizeAsync(string)

Retrieves the size, in bytes, of the data associated with the specified key.

```
Task<Either<Error, long>> SizeAsync(string key)
```

## Parameters

key [string](#)

The key identifying the data whose size is to be retrieved. Cannot be null or empty.

## Returns

[Task](#) <Either<[Error](#), [long](#)>>

An `TinyFp.Either<L, R>` containing either an [Error](#) if the operation fails, or a [long](#) representing the size of the data in bytes if successful.

# Namespace Func.Redis.Models

## Classes

[RedisConfiguration](#)

[RedisKeyConfiguration](#)


# Class RedisConfiguration

Namespace: [Func.Redis.Models](#)

Assembly: Func.Redis.dll

```
public record RedisConfiguration : IEquatable<RedisConfiguration>
```








## Inheritance

[object](#)  ← RedisConfiguration

## Implements

[IEquatable](#)  <[RedisConfiguration](#)>

## Inherited Members

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

## Properties

### ConnectionString

The connection string to the Redis server

```
public string ConnectionString { get; init; }
```

### Property Value

[string](#) 

# Class RedisKeyConfiguration

Namespace: [Func.Redis.Models](#)

Assembly: Func.Redis.dll

```
public record RedisKeyConfiguration : IEquatable<RedisKeyConfiguration>
```








## Inheritance

[object](#)  ← RedisKeyConfiguration

## Implements

[IEquatable](#)  <[RedisKeyConfiguration](#)>

## Inherited Members

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

## Properties

### KeyPrefix

The prefix to be added to the key

```
public string KeyPrefix { get; init; }
```

### Property Value

[string](#) 

# Namespace Func.Redis.Publisher

## Interfaces

[IRedisPublisherService](#)

# Interface IRedisPublisherService

Namespace: [Func.Redis.Publisher](#)

Assembly: Func.Redis.dll

```
public interface IRedisPublisherService
```

## Methods

### Publish(string, object)

Publishes a message to the specified channel.

```
Either<Error, Unit> Publish(string channel, object message)
```

## Parameters

**channel** [string](#) 

The name of the channel to which the message will be published. Cannot be null or empty.

**message** [object](#) 

The message to be published. Typically an object representing the data to send. Cannot be null.

## Returns

Either<[Error](#), Unit>

An `TinyFp.Either<L, R>` indicating the result of the operation. Returns `Unit` if the message is successfully published, or an [Error](#) if the operation fails.

## Remarks

This method allows sending messages to a specific channel, which can be used for communication or event propagation. Ensure that the channel name and message are valid and conform to the expected format for the underlying system.

# PublishAsync(string, object)

Publishes a message to the specified channel.

```
Task<Either<Error, Unit>> PublishAsync(string channel, object message)
```

## Parameters

[channel](#) [string](#)<sup>↗</sup>

The name of the channel to which the message will be published. Cannot be null or empty.

[message](#) [object](#)<sup>↗</sup>

The message to be published. Typically an object representing the data to send. Cannot be null.

## Returns

[Task](#)<sup>↗</sup> <Either<[Error](#), Unit>>

An `TinyFp.Either<L, R>` indicating the result of the operation. Returns `Unit` if the message is successfully published, or an [Error](#) if the operation fails.

## Remarks

This method allows sending messages to a specific channel, which can be used for communication or event propagation. Ensure that the channel name and message are valid and conform to the expected format for the underlying system.



# Namespace Func.Redis.SerDes

## Interfaces

[IRedisSerDes](#)

# Interface IRedisSerDes

Namespace: [Func.Redis.SerDes](#)

Assembly: Func.Redis.dll

```
public interface IRedisSerDes
```

## Methods

### Deserialize(RedisValue, Type)

Deserializes the specified Redis value into an object of the given type.

```
Option<object> Deserialize(RedisValue value, Type type)
```

#### Parameters

**value** RedisValue

The Redis value to deserialize. Must not be null or empty.

**type** [Type](#) 

The target type to deserialize the value into. Must not be null.

#### Returns

Option<[object](#) >

An `TinyFp.Option<A>` containing the deserialized object if successful; otherwise, an empty `TinyFp.Option<A>` if deserialization fails.

#### Remarks

This method attempts to convert the Redis value into the specified type. If the conversion is not possible, the method returns an empty `TinyFp.Option<A>` rather than throwing an exception.

## Deserialize<T>(HashEntry[])

Deserializes an array of hash entries into an array of key-value pairs, where the key is a string and the value is of the specified type.

```
Option<(string, T)[]> Deserialize<T>(HashEntry[] entries)
```

### Parameters

**entries** HashEntry[]

An array of hash entries to be deserialized. Each entry represents a key-value pair.

### Returns

Option<(string, T)[]>

An array of key-value pairs, where the key is a string and the value is of type **T**. Returns an empty array if no entries are provided.

### Type Parameters

**T**

The type of the value to deserialize each hash entry into.

## Deserialize<T>(RedisValue)

Deserializes the specified Redis value into an instance of the specified type.

```
Option<T> Deserialize<T>(RedisValue value)
```

### Parameters

**value** RedisValue

The Redis value to deserialize. Must not be null or empty.

### Returns

## Option<T>

An `TinyFp.Option<A>` containing the deserialized object if successful; otherwise, an empty `TinyFp.Option<A>` if the value cannot be deserialized.

## Type Parameters

**T**

The type to deserialize the Redis value into.

## Remarks

This method attempts to convert the Redis value into the specified type **T**. If the conversion fails, the method returns an empty `TinyFp.Option<A>` instead of throwing an exception.

## Deserialize<T>(RedisValue[])

Deserializes an array of `StackExchange.Redis.RedisValue` objects into an array of the specified type.

```
Option<T[]> Deserialize<T>(RedisValue[] values)
```

## Parameters

**values** `RedisValue[]`

An array of `StackExchange.Redis.RedisValue` objects to deserialize. Cannot be null.

## Returns

`Option<T[]>`

An `TinyFp.Option<A>` containing the deserialized array of type **T**. If deserialization fails, the `TinyFp.Option<A>` will represent an empty or error state.

## Type Parameters

**T**

The type to which each `StackExchange.Redis.RedisValue` will be deserialized.

## Remarks

This method attempts to deserialize each `StackExchange.Redis.RedisValue` in the input array into the specified type. Ensure that the type `T` is compatible with the data contained in the `StackExchange.Redis.RedisValue` objects.

## Serialize<T>(T)

Serializes the specified value into a Redis-compatible format.

```
RedisValue Serialize<T>(T value)
```

## Parameters

`value` `T`

The value to serialize. Cannot be null.

## Returns

`RedisValue`

A `StackExchange.Redis.RedisValue` representing the serialized form of the input value.

## Type Parameters

`T`

The type of the value to serialize.

## Remarks

This method converts the input value into a format suitable for storage in Redis. Ensure that the type `T` is supported by the serialization mechanism.

# Namespace Func.Redis.SerDes.Json

## Classes

[SpanJsonRedisSerDes](#)

[SystemJsonRedisSerDes](#)

# Class SpanJsonRedisSerDes

Namespace: [Func.Redis.SerDes.Json](#)

Assembly: Func.Redis.dll

```
public class SpanJsonRedisSerDes : IRedisSerDes
```








## Inheritance

[object](#)  ← SpanJsonRedisSerDes

## Implements

[IRedisSerDes](#)

## Inherited Members

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

## Methods

### Deserialize(RedisValue, Type)

Deserializes the specified Redis value into an object of the given type.

```
public Option<object> Deserialize(RedisValue value, Type type)
```

## Parameters


**value** RedisValue

The Redis value to deserialize. Must not be null or empty.

**type** [Type](#) 

The target type to deserialize the value into. Must not be null.

## Returns

Option<[object](#) >

An `TinyFp.Option<A>` containing the deserialized object if successful; otherwise, an empty `TinyFp.Option<A>` if deserialization fails.

## Remarks

This method attempts to convert the Redis value into the specified type. If the conversion is not possible, the method returns an empty `TinyFp.Option<A>` rather than throwing an exception.

## Deserialize<T>(HashEntry[])

Deserializes an array of hash entries into an array of key-value pairs, where the key is a string and the value is of the specified type.

```
public Option<(string, T)[]> Deserialize<T>(HashEntry[] entries)
```

## Parameters

**entries** `HashEntry[]`

An array of hash entries to be deserialized. Each entry represents a key-value pair.

## Returns

`Option<(string, T)[]>`

An array of key-value pairs, where the key is a string and the value is of type `T`. Returns an empty array if no entries are provided.

## Type Parameters

**T**

The type of the value to deserialize each hash entry into.

## Deserialize<T>(RedisValue)

Deserializes the specified Redis value into an instance of the specified type.



```
public Option<T> Deserialize<T>(RedisValue value)
```

## Parameters

**value** RedisValue

The Redis value to deserialize. Must not be null or empty.

## Returns

Option<T>

An TinyFp.Option<A> containing the deserialized object if successful; otherwise, an empty TinyFp.Option<A> if the value cannot be deserialized.

## Type Parameters

**T**

The type to deserialize the Redis value into.

## Remarks

This method attempts to convert the Redis value into the specified type **T**. If the conversion fails, the method returns an empty TinyFp.Option<A> instead of throwing an exception.

## Deserialize<T>(RedisValue[])

Deserializes an array of StackExchange.Redis.RedisValue objects into an array of the specified type.

```
public Option<T[]> Deserialize<T>(RedisValue[] values)
```

## Parameters

**values** RedisValue[]

An array of StackExchange.Redis.RedisValue objects to deserialize. Cannot be null.

## Returns

## Option<T[]>

An `TinyFp.Option<A>` containing the deserialized array of type `T`. If deserialization fails, the `TinyFp.Option<A>` will represent an empty or error state.

## Type Parameters

`T`

The type to which each `StackExchange.Redis.RedisValue` will be deserialized.

## Remarks

This method attempts to deserialize each `StackExchange.Redis.RedisValue` in the input array into the specified type. Ensure that the type `T` is compatible with the data contained in the `StackExchange.Redis.RedisValue` objects.

## Serialize<T>(T)

Serializes the specified value into a Redis-compatible format.

```
public RedisValue Serialize<T>(T value)
```

## Parameters

`value T`

The value to serialize. Cannot be null.

## Returns

`RedisValue`

A `StackExchange.Redis.RedisValue` representing the serialized form of the input value.

## Type Parameters

`T`

The type of the value to serialize.

## Remarks

This method converts the input value into a format suitable for storage in Redis. Ensure that the type **T** is supported by the serialization mechanism.


# Class SystemJsonRedisSerDes

Namespace: [Func.Redis.SerDes.Json](#)

Assembly: Func.Redis.dll

```
public class SystemJsonRedisSerDes : IRedisSerDes
```








## Inheritance

[object](#)  ← SystemJsonRedisSerDes

## Implements

[IRedisSerDes](#)

## Inherited Members

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

## Methods

### Deserialize(RedisValue, Type)

Deserializes the specified Redis value into an object of the given type.

```
public Option<object> Deserialize(RedisValue value, Type type)
```

## Parameters


**value** RedisValue

The Redis value to deserialize. Must not be null or empty.

**type** [Type](#) 

The target type to deserialize the value into. Must not be null.

## Returns

Option<[object](#) >

An `TinyFp.Option<A>` containing the deserialized object if successful; otherwise, an empty `TinyFp.Option<A>` if deserialization fails.

## Remarks

This method attempts to convert the Redis value into the specified type. If the conversion is not possible, the method returns an empty `TinyFp.Option<A>` rather than throwing an exception.

## Deserialize<T>(HashEntry[])

Deserializes an array of hash entries into an array of key-value pairs, where the key is a string and the value is of the specified type.

```
public Option<(string, T)[]> Deserialize<T>(HashEntry[] entries)
```

## Parameters

**entries** `HashEntry[]`

An array of hash entries to be deserialized. Each entry represents a key-value pair.

## Returns

`Option<(string, T)[]>`

An array of key-value pairs, where the key is a string and the value is of type `T`. Returns an empty array if no entries are provided.

## Type Parameters

**T**

The type of the value to deserialize each hash entry into.

## Deserialize<T>(RedisValue)

Deserializes the specified Redis value into an instance of the specified type.

```
public Option<T> Deserialize<T>(RedisValue value)
```

## Parameters

**value** RedisValue

The Redis value to deserialize. Must not be null or empty.

## Returns

Option<T>

An TinyFp.Option<A> containing the deserialized object if successful; otherwise, an empty TinyFp.Option<A> if the value cannot be deserialized.

## Type Parameters

**T**

The type to deserialize the Redis value into.

## Remarks

This method attempts to convert the Redis value into the specified type **T**. If the conversion fails, the method returns an empty TinyFp.Option<A> instead of throwing an exception.

## Deserialize<T>(RedisValue[])

Deserializes an array of StackExchange.Redis.RedisValue objects into an array of the specified type.

```
public Option<T[]> Deserialize<T>(RedisValue[] values)
```

## Parameters

**values** RedisValue[]

An array of StackExchange.Redis.RedisValue objects to deserialize. Cannot be null.

## Returns

## Option<T[]>

An `TinyFp.Option<A>` containing the deserialized array of type `T`. If deserialization fails, the `TinyFp.Option<A>` will represent an empty or error state.

## Type Parameters

`T`

The type to which each `StackExchange.Redis.RedisValue` will be deserialized.

## Remarks

This method attempts to deserialize each `StackExchange.Redis.RedisValue` in the input array into the specified type. Ensure that the type `T` is compatible with the data contained in the `StackExchange.Redis.RedisValue` objects.

## Serialize<T>(T)

Serializes the specified value into a Redis-compatible format.

```
public RedisValue Serialize<T>(T value)
```

## Parameters

`value T`

The value to serialize. Cannot be null.

## Returns

`RedisValue`

A `StackExchange.Redis.RedisValue` representing the serialized form of the input value.

## Type Parameters

`T`

The type of the value to serialize.

## Remarks

This method converts the input value into a format suitable for storage in Redis. Ensure that the type `T` is supported by the serialization mechanism.



# Namespace Func.Redis.Set

## Interfaces

[IRedisSetService](#)

# Interface IRedisSetService

Namespace: [Func.Redis.Set](#)

Assembly: Func.Redis.dll

```
public interface IRedisSetService
```

## Methods

### AddAsync<T>(string, T)

Adds a value to the collection with the specified key.

```
Task<Either<Error, Unit>> AddAsync<T>(string key, T value)
```

## Parameters

key [string](#) 

The key associated with the value. Cannot be null or empty.

value T

The value to add. Cannot be null.

## Returns

[Task](#)  <Either<[Error](#), Unit>>

An `TinyFp.Either<L, R>` indicating the result of the operation. Returns `Unit` if the addition is successful, or an [Error](#) if the operation fails.

## Type Parameters

T

The type of the value to add.

## Remarks

The method ensures that the key-value pair is added to the collection. If the key already exists, the operation may fail depending on the implementation, and an [Error](#) will be returned.

## Add<T>(string, T)

Adds a value to the collection with the specified key.

```
Either<Error, Unit> Add<T>(string key, T value)
```

## Parameters

key [string](#) 

The key associated with the value. Cannot be null or empty.

value T

The value to add. Cannot be null.

## Returns

Either<[Error](#), Unit>

An `TinyFp.Either<L, R>` indicating the result of the operation. Returns `Unit` if the addition is successful, or an [Error](#) if the operation fails.

## Type Parameters

T

The type of the value to add.

## Remarks

The method ensures that the key-value pair is added to the collection. If the key already exists, the operation may fail depending on the implementation, and an [Error](#) will be returned.

# DeleteAsync<T>(string, T)

Deletes an item identified by the specified key and value from the underlying data store.

```
Task<Either<Error, Unit>> DeleteAsync<T>(string key, T value)
```

## Parameters

**key** [string](#)

The unique key identifying the item to delete. Cannot be null or empty.

**value** T

The value associated with the key, used for validation or additional context. Cannot be null.

## Returns

[Task](#) <Either<[Error](#), Unit>>

An `TinyFp.Either<L, R>` indicating the result of the operation. Returns `Unit` if the deletion is successful, or an [Error](#) if the operation fails.

## Type Parameters

**T**

The type of the value associated with the key.

## Remarks

The method performs a deletion operation and returns a result encapsulated in an `TinyFp.Either<L, R>`. If the operation fails, the [Error](#) provides details about the failure, such as invalid input or a missing item.

# DeleteAsync<T>(string, params T[])

Deletes the specified values associated with the given key.

```
Task<Either<Error, Unit>> DeleteAsync<T>(string key, params T[] values)
```

## Parameters

**key** [string](#)

The key identifying the values to delete. Cannot be null or empty.

**values** `T[]`

The values to delete. If no values are provided, the method performs no operation.

## Returns

[Task](#) `<Either<Error, Unit>>`

An `TinyFp.Either<L, R>` indicating the result of the operation. Returns [true](#) if the deletion is successful; otherwise, returns an [Error](#) describing the failure.

## Type Parameters

**T**

The type of the values to delete.

## Remarks

This method does not guarantee the existence of the specified values prior to deletion. Ensure the key and values are valid and consistent with the underlying data store.

## Delete<T>(string, T)

Deletes an item identified by the specified key and value from the underlying data store.

```
Either<Error, Unit> Delete<T>(string key, T value)
```

## Parameters

**key** [string](#)

The unique key identifying the item to delete. Cannot be null or empty.

**value** `T`

The value associated with the key, used for validation or additional context. Cannot be null.

## Returns

Either<[Error](#), Unit>

An `TinyFp.Either<L, R>` indicating the result of the operation. Returns `Unit` if the deletion is successful, or an [Error](#) if the operation fails.

## Type Parameters

**T**

The type of the value associated with the key.

## Remarks

The method performs a deletion operation and returns a result encapsulated in an `TinyFp.Either<L, R>`. If the operation fails, the [Error](#) provides details about the failure, such as invalid input or a missing item.

## Delete<T>(string, params T[])

Deletes the specified values associated with the given key.

```
Either<Error, Unit> Delete<T>(string key, params T[] values)
```

## Parameters

**key** [string](#) 

The key identifying the values to delete. Cannot be null or empty.

**values** T[]

The values to delete. If no values are provided, the method performs no operation.

## Returns

Either<[Error](#), Unit>

An `TinyFp.Either<L, R>` indicating the result of the operation. Returns [true](#) if the deletion is successful; otherwise, returns an [Error](#) describing the failure.

## Type Parameters

**T**

The type of the values to delete.

## Remarks

This method does not guarantee the existence of the specified values prior to deletion. Ensure the key and values are valid and consistent with the underlying data store.

## DifferenceAsync<T>(string, string)

Computes the difference between two sets of data identified by the specified keys.

```
Task<Either<Error, T[]>> DifferenceAsync<T>(string key1, string key2)
```

## Parameters

**key1** [string](#)

The key identifying the first set of data. Cannot be null or empty.

**key2** [string](#)

The key identifying the second set of data. Cannot be null or empty.

## Returns

[Task](#) <Either<[Error](#), T[]>>

An `TinyFp.Either<L, R>` containing either an error if the operation fails, or an array of elements representing the difference between the two sets.

## Type Parameters

**T**

The type of elements in the sets.

## Remarks

The difference is calculated as the elements present in the first set but not in the second set. If either key does not correspond to a valid set, an error is returned.

## Difference<T>(string, string)

Computes the difference between two sets of data identified by the specified keys.

```
Either<Error, T[]> Difference<T>(string key1, string key2)
```

## Parameters

key1 [string](#)

The key identifying the first set of data. Cannot be null or empty.

key2 [string](#)

The key identifying the second set of data. Cannot be null or empty.

## Returns

Either<[Error](#), T[]>

An `TinyFp.Either<L, R>` containing either an error if the operation fails, or an array of elements representing the difference between the two sets.

## Type Parameters

T

The type of elements in the sets.

## Remarks

The difference is calculated as the elements present in the first set but not in the second set. If either key does not correspond to a valid set, an error is returned.

## GetAllAsync<T>(string)



Retrieves all values associated with the specified key.

```
Task<Either<Error, Option<T>[]>> GetAllAsync<T>(string key)
```

## Parameters

key [string](#)

The key used to look up the values. Cannot be null or empty.

## Returns

[Task](#) <Either<[Error](#), Option<T>[]>>

An `TinyFp.Either<L, R>` containing either an error or an array of optional values. If the key is not found, the array will be empty.

## Type Parameters

T

The type of the values to retrieve.

## GetAll<T>(string)

Retrieves all values associated with the specified key.

```
Either<Error, Option<T>[]> GetAll<T>(string key)
```

## Parameters

key [string](#)

The key used to look up the values. Cannot be null or empty.

## Returns

Either<[Error](#), Option<T>[]>

An `TinyFp.Either<L, R>` containing either an error or an array of optional values. If the key is not found, the array will be empty.

## Type Parameters

**T**

The type of the values to retrieve.

## IntersectAsync<T>(string, string)

Computes the intersection of two sets stored under the specified keys.

```
Task<Either<Error, T[]>> IntersectAsync<T>(string key1, string key2)
```

## Parameters

**key1** [string](#)

The key identifying the first set. Cannot be null or empty.

**key2** [string](#)

The key identifying the second set. Cannot be null or empty.

## Returns

[Task](#) <Either<[Error](#), T[]>>

An `TinyFp.Either<L, R>` containing either an error if the operation fails, or an array of elements representing the intersection of the two sets. If either key does not exist or the sets are empty, the result will be an empty array.

## Type Parameters

**T**

The type of elements in the sets.

## Intersect<T>(string, string)

Computes the intersection of two sets stored under the specified keys.

```
Either<Error, T[]> Intersect<T>(string key1, string key2)
```

## Parameters

**key1** [string](#)

The key identifying the first set. Cannot be null or empty.

**key2** [string](#)

The key identifying the second set. Cannot be null or empty.

## Returns

Either<[Error](#), T[]>

An `TinyFp.Either<L, R>` containing either an error if the operation fails, or an array of elements representing the intersection of the two sets. If either key does not exist or the sets are empty, the result will be an empty array.

## Type Parameters

**T**

The type of elements in the sets.

## PopAsync<T>(string)

```
Task<Either<Error, Option<T>>> PopAsync<T>(string key)
```

## Parameters

**key** [string](#)

## Returns

[Task](#) <Either<[Error](#), Option<T>>>

## Type Parameters

T

## Pop<T>(string)

Either<Error, Option<T>> Pop<T>(string key)

## Parameters

key [string](#)

## Returns

Either<[Error](#), Option<T>>

## Type Parameters

T

## Size(string)

Retrieves the size, in bytes, of the object associated with the specified key.

Either<Error, long> Size(string key)

## Parameters

key [string](#)

The unique identifier of the object whose size is to be retrieved. Cannot be null or empty.

## Returns

Either<[Error](#), [long](#)>

An `TinyFp.Either<L, R>` containing either an [Error](#) if the operation fails, or a [long](#) representing the size of the object in bytes if successful.

## SizeAsync(string)

Retrieves the size, in bytes, of the object associated with the specified key.

```
Task<Either<Error, long>> SizeAsync(string key)
```

### Parameters

key [string](#)

The unique identifier of the object whose size is to be retrieved. Cannot be null or empty.

### Returns

[Task](#) <Either<[Error](#), [long](#)>>

An `TinyFp.Either<L, R>` containing either an [Error](#) if the operation fails, or a [long](#) representing the size of the object in bytes if successful.

## UnionAsync<T>(string, string)

Combines the values associated with two specified keys into a single array.

```
Task<Either<Error, T[]>> UnionAsync<T>(string key1, string key2)
```

### Parameters

key1 [string](#)

The first key whose associated values will be included in the union.

key2 [string](#)

The second key whose associated values will be included in the union.

### Returns

[Task](#) <Either<[Error](#), `T[]`>>

An `TinyFp.Either<L, R>` containing either an error if the operation fails, or an array of values of type `T` representing the union of the values associated with `key1` and `key2`.

## Type Parameters

**T**

The type of the values associated with the keys.

## Remarks

If either key does not exist or an error occurs during the operation, the result will contain an [Error](#) describing the issue. Otherwise, the result will contain the combined values from both keys, with duplicates removed if applicable.

## Union<T>(string, string)

Combines the values associated with two specified keys into a single array.

```
Either<Error, T[]> Union<T>(string key1, string key2)
```

## Parameters

**key1** [string](#) 

The first key whose associated values will be included in the union.

**key2** [string](#) 

The second key whose associated values will be included in the union.

## Returns

Either<[Error](#), T[]>

An `TinyFp.Either<L, R>` containing either an error if the operation fails, or an array of values of type **T** representing the union of the values associated with **key1** and **key2**.

## Type Parameters

**T**

The type of the values associated with the keys.

## Remarks

If either key does not exist or an error occurs during the operation, the result will contain an [Error](#) describing the issue. Otherwise, the result will contain the combined values from both keys, with duplicates removed if applicable.

# Namespace Func.Redis.SortedSet

## Classes

[RedisSortedSetService](#)

## Interfaces

[IRedisSortedSetService](#)



# Interface IRedisSortedSetService

Namespace: [Func.Redis.SortedSet](#)

Assembly: Func.Redis.dll

```
public interface IRedisSortedSetService
```

## Methods

AddAsync<T>(string, IEnumerable<(T Value, double Score)>)

```
Task<Either<Error, Unit>> AddAsync<T>(string key, IEnumerable<(T Value, double Score)> values)
```

## Parameters

key [string](#)

values [IEnumerable](#) <(T [Value](#), [double](#) [Score](#))>

## Returns

[Task](#) <Either<[Error](#), Unit>>

## Type Parameters

T

AddAsync<T>(string, T, double)

```
Task<Either<Error, Unit>> AddAsync<T>(string key, T value, double score)
```

## Parameters

key [string](#)

value T

score [double](#)

Returns

[Task](#) <Either<[Error](#), Unit>>

Type Parameters

T

## Add<T>(string, IEnumerable<(T Value, double Score)>)

Add values with scores to a sorted set key.

```
Either<Error, Unit> Add<T>(string key, IEnumerable<(T Value, double Score)> values)
```

Parameters

key [string](#)

values [IEnumerable](#) <(T [Value](#), [double](#) [Score](#))>

Returns

Either<[Error](#), Unit>

TinyFp.Unit or [Error](#)

Type Parameters

T

## Add<T>(string, T, double)

Add a value with score to a sorted set key.

```
Either<Error, Unit> Add<T>(string key, T value, double score)
```

## Parameters

key [string](#)

value T

score [double](#)

## Returns

Either<[Error](#), Unit>

TinyFp.Unit or [Error](#)

## Type Parameters

T

# DecrementAsync<T>(string, T, double)

Decrement the score of a value in a sorted set key.

```
Task<Either<Error, Unit>> DecrementAsync<T>(string key, T value, double score)
```

## Parameters

key [string](#)

value T

score [double](#)

## Returns

[Task](#) <Either<[Error](#), Unit>>

## Type Parameters

T

## Decrement<T>(string, T, double)

Decrement the score of a value in a sorted set key.

```
Either<Error, Unit> Decrement<T>(string key, T value, double score)
```

### Parameters

key [string](#)

value T

score [double](#)

### Returns

Either<[Error](#), Unit>

### Type Parameters

T

## IncrementAsync<T>(string, T, double)

Retrieves the intersection of values associated with the specified keys.

```
Task<Either<Error, Unit>> IncrementAsync<T>(string key, T value, double score)
```

### Parameters

key [string](#)

value T

score [double](#)

### Returns

[Task](#) `<Either<Error, Unit>>`

An `TinyFp.Either<L, R>` containing either an error if the operation fails, or an array of values of type `T` that represent the intersection of the values associated with the provided keys.

## Type Parameters

`T`

The type of the values to be intersected.

## Increment<T>(string, T, double)

Increments the score associated with the specified key by the given value.

```
Either<Error, Unit> Increment<T>(string key, T value, double score)
```

## Parameters

`key` [string](#)

The key identifying the item whose score is to be incremented. Cannot be null or empty.

`value` `T`

The value to increment the score by. Must be compatible with the type parameter `T`.

`score` [double](#)

The amount by which the score is incremented. Must be a valid double value.

## Returns

`Either<Error, Unit>`

An `TinyFp.Either<L, R>` indicating the result of the operation. Returns `Unit` if the operation succeeds, or an [Error](#) if the operation fails.

## Type Parameters

`T`

The type of the value to be incremented. Typically a numeric type.

## Remarks

This method is typically used in scenarios where scores are tracked and updated dynamically, such as leaderboards or ranking systems. Ensure that the key exists and is valid before calling this method to avoid errors.

## IntersectAsync<T>(string[])

Retrieves the intersection of values associated with the specified keys.

```
Task<Either<Error, T[]>> IntersectAsync<T>(string[] keys)
```

## Parameters

**keys** [string](#)[]

An array of keys used to identify the values to intersect. Cannot be null or empty.

## Returns

[Task](#) <Either<[Error](#), T[]>>

An `TinyFp.Either<L, R>` containing either an error if the operation fails, or an array of values of type `T` that represent the intersection of the values associated with the provided keys.

## Type Parameters

**T**

The type of the values to be intersected.

## Intersect<T>(string[])

Retrieves the intersection of values associated with the specified keys.

```
Either<Error, T[]> Intersect<T>(string[] keys)
```

## Parameters

**keys** [string](#)[]

An array of keys used to identify the values to intersect. Cannot be null or empty.

## Returns

Either<[Error](#), T[]>

An `TinyFp.Either<L, R>` containing either an error if the operation fails, or an array of values of type `T` that represent the intersection of the values associated with the provided keys.

## Type Parameters

**T**

The type of the values to be intersected.

## Length(string)

Retrieves the length of the value associated with the specified key.

```
Either<Error, long> Length(string key)
```

## Parameters

**key** [string](#)

The key identifying the value whose length is to be retrieved. Cannot be null or empty.

## Returns

Either<[Error](#), [long](#)>

An `TinyFp.Either<L, R>` containing either an [Error](#) if the operation fails, or a [long](#) representing the length of the value if the operation succeeds.

## LengthAsync(string)

Retrieves the length of the value associated with the specified key.

```
Task<Either<Error, long>> LengthAsync(string key)
```

## Parameters

key [string](#)

The key identifying the value whose length is to be retrieved. Cannot be null or empty.

## Returns

[Task](#) <Either<[Error](#), [long](#)>>

An `TinyFp.Either<L, R>` containing either an [Error](#) if the operation fails, or a [long](#) representing the length of the value if the operation succeeds.

## LengthByScore(string, double, double)

Retrieves the count of elements in a sorted set stored at the specified key, where the elements' scores fall within the given range.

```
Either<Error, long> LengthByScore(string key, double min, double max)
```

## Parameters

key [string](#)

The key identifying the sorted set. Cannot be null or empty.

min [double](#)

The minimum score of the range. Elements with scores greater than or equal to this value are included.

max [double](#)

The maximum score of the range. Elements with scores less than or equal to this value are included.



## Returns

Either<[Error](#), [long](#)>

An `TinyFp.Either<L, R>` containing either an error if the operation fails, or the count of elements within the specified score range.

## Remarks

The score range is inclusive of both `min` and `max`.

## LengthByScoreAsync(string, double, double)

Retrieves the count of elements in a sorted set stored at the specified key, where the elements' scores fall within the given range.

```
Task<Either<Error, long>> LengthByScoreAsync(string key, double min, double max)
```

## Parameters

`key` [string](#)

The key identifying the sorted set. Cannot be null or empty.

`min` [double](#)

The minimum score of the range. Elements with scores greater than or equal to this value are included.

`max` [double](#)

The maximum score of the range. Elements with scores less than or equal to this value are included.

## Returns

[Task](#) <Either<[Error](#), [long](#)>>

An `TinyFp.Either<L, R>` containing either an error if the operation fails, or the count of elements within the specified score range.

## Remarks

The score range is inclusive of both `min` and `max`.

## LengthByValueAsync<T>(string, T, T)

Retrieves the length of a collection stored under the specified key, filtered by a range of values.

```
Task<Either<Error, long>> LengthByValueAsync<T>(string key, T min, T max)
```

### Parameters

**key** [string](#)

The key identifying the collection in the data store. Cannot be null or empty.

**min** T

The minimum value of the range used to filter the collection.

**max** T

The maximum value of the range used to filter the collection.

### Returns

[Task](#) <Either<[Error](#), [long](#)>>

An `TinyFp.Either<L, R>` containing either an [Error](#) if the operation fails, or a [long](#) representing the count of items in the collection that fall within the specified range.

### Type Parameters

**T**

The type of the values used to filter the collection.

### Remarks

The method filters the collection based on the provided range [`min`, `max`]. Ensure that `min` is less than or equal to `max` to avoid unexpected results.

# LengthByValue<T>(string, T, T)

Retrieves the length of a collection stored under the specified key, filtered by a range of values.

```
Either<Error, long> LengthByValue<T>(string key, T min, T max)
```

## Parameters

key [string](#)

The key identifying the collection in the data store. Cannot be null or empty.

min T

The minimum value of the range used to filter the collection.

max T

The maximum value of the range used to filter the collection.

## Returns

Either<[Error](#), [long](#)>

An `TinyFp.Either<L, R>` containing either an [Error](#) if the operation fails, or a [long](#) representing the count of items in the collection that fall within the specified range.

## Type Parameters

T

The type of the values used to filter the collection.

## Remarks

The method filters the collection based on the provided range [`min`, `max`]. Ensure that `min` is less than or equal to `max` to avoid unexpected results.

# RangeByScoreAsync<T>(string, double, double)

Retrieves a range of elements from a sorted collection based on their score values.

```
Task<Either<Error, T[]>> RangeByScoreAsync<T>(string key, double min, double max)
```

## Parameters

**key** [string](#) 

The key identifying the sorted collection. Cannot be null or empty.

**min** [double](#) 

The minimum score value for the range. Elements with scores less than this value are excluded.

**max** [double](#) 

The maximum score value for the range. Elements with scores greater than this value are excluded.

## Returns

[Task](#)  <Either<[Error](#), T[]>>

An `TinyFp.Either<L, R>` containing either an [Error](#) if the operation fails, or an array of elements of type `T` that fall within the specified score range. If no elements match the criteria, an empty array is returned.

## Type Parameters

**T**

The type of elements in the collection.

## RangeByScore<T>(string, double, double)

Retrieves a range of elements from a sorted collection based on their score values.

```
Either<Error, T[]> RangeByScore<T>(string key, double min, double max)
```

## Parameters

key [string](#)

The key identifying the sorted collection. Cannot be null or empty.

min [double](#)

The minimum score value for the range. Elements with scores less than this value are excluded.

max [double](#)

The maximum score value for the range. Elements with scores greater than this value are excluded.

## Returns

Either<[Error](#), T[]>

An `TinyFp.Either<L, R>` containing either an [Error](#) if the operation fails, or an array of elements of type `T` that fall within the specified score range. If no elements match the criteria, an empty array is returned.

## Type Parameters

`T`

The type of elements in the collection.

## RankAsync<T>(string, T)

Retrieves the rank of a specified value within a collection associated with the given key.

```
Task<Either<Error, Option<long>>> RankAsync<T>(string key, T value)
```

## Parameters

key [string](#)

The key identifying the collection in which the rank is calculated. Cannot be null or empty.

value `T`

The value whose rank is to be determined. Must exist within the collection.

## Returns

[Task](#) `<Either<Error, Option<long>>>`

An `TinyFp.Either<L, R>` containing either an [Error](#) if the operation fails, or an `TinyFp.Option<A>` representing the rank of the value as a zero-based index. Returns `None` if the value is not found in the collection.

## Type Parameters

**T**

The type of the value to rank. Must be comparable within the collection.

## Remarks

The rank is determined based on the ordering of the collection associated with the specified key. If the key does not exist or the value is not found, the method returns `None` within the result.

## Rank<T>(string, T)

Retrieves the rank of a specified value within a collection associated with the given key.

```
Either<Error, Option<long>> Rank<T>(string key, T value)
```

## Parameters

**key** [string](#)

The key identifying the collection in which the rank is calculated. Cannot be null or empty.

**value** **T**

The value whose rank is to be determined. Must exist within the collection.

## Returns

`Either<Error, Option<long>>`

An `TinyFp.Either<L, R>` containing either an [Error](#) if the operation fails, or an `TinyFp.Option<A>` representing the rank of the value as a zero-based index. Returns `None` if the value is not found in the collection.

## Type Parameters

**T**

The type of the value to rank. Must be comparable within the collection.

## Remarks

The rank is determined based on the ordering of the collection associated with the specified key. If the key does not exist or the value is not found, the method returns `None` within the result.

## RemoveAsync<T>(string, IEnumerable<T>)

Removes the specified values associated with the given key from the collection.

```
Task<Either<Error, Unit>> RemoveAsync<T>(string key, IEnumerable<T> values)
```

## Parameters


**key** [string](#) 

The key identifying the collection from which the values will be removed. Cannot be null or empty.

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

The values to be removed from the collection. Cannot be null.

## Returns

[Task](#)  <[Either](#) <[Error](#), `Unit`>>

An `TinyFp.Either<L, R>` indicating the result of the operation. Returns `Unit` if the removal is successful, or an [Error](#) if the operation fails.

## Type Parameters

T

The type of the values to be removed.

## RemoveAsync<T>(string, T)

Removes the specified value associated with the given key from the collection.

```
Task<Either<Error, Unit>> RemoveAsync<T>(string key, T value)
```

### Parameters

key [string](#)

The key identifying the value to be removed. Cannot be null or empty.

value T

The value to be removed. Must match the value associated with the specified key.

### Returns

[Task](#) <Either<[Error](#), Unit>>

An `TinyFp.Either<L, R>` indicating the result of the operation. Returns `Unit` if the removal is successful, or an [Error](#) if the operation fails.

### Type Parameters

T

The type of the value to be removed.

### Remarks

The method will fail if the specified key does not exist in the collection or if the value does not match the one associated with the key.

## RemoveRangeByScore(string, double, double)



Removes all elements in the sorted set stored at the specified key with scores within the given range.

```
Either<Error, Unit> RemoveRangeByScore(string key, double start, double stop)
```

## Parameters

key [string](#) 

The key identifying the sorted set. Cannot be null or empty.

start [double](#) 

The inclusive lower bound of the score range.

stop [double](#) 

The inclusive upper bound of the score range.

## Returns

Either<[Error](#), Unit>

An `TinyFp.Either<L, R>` indicating the result of the operation. Returns `Unit` if the operation succeeds, or an [Error](#) if it fails.

## Remarks

This method operates on a sorted set and removes elements whose scores fall within the specified range. The range is inclusive of both `start` and `stop`.

## RemoveRangeByScoreAsync(string, double, double)

Removes all elements in the sorted set stored at the specified key with scores within the given range.

```
Task<Either<Error, Unit>> RemoveRangeByScoreAsync(string key, double start, double stop)
```

## Parameters

key [string](#)

The key identifying the sorted set. Cannot be null or empty.

start [double](#)

The inclusive lower bound of the score range.

stop [double](#)

The inclusive upper bound of the score range.

## Returns

[Task](#) <Either<[Error](#), Unit>>

An `TinyFp.Either<L, R>` indicating the result of the operation. Returns `Unit` if the operation succeeds, or an [Error](#) if it fails.

## Remarks

This method operates on a sorted set and removes elements whose scores fall within the specified range. The range is inclusive of both `start` and `stop`.

# RemoveRangeByValueAsync<T>(string, T, T)

Removes a range of values from a data store based on the specified key and value range.

```
Task<Either<Error, Unit>> RemoveRangeByValueAsync<T>(string key, T min, T max)
```

## Parameters

key [string](#)

The key identifying the data store or collection from which values will be removed. Cannot be null or empty.

min T

The minimum value of the range to remove. Values less than this will not be affected.

max T

The maximum value of the range to remove. Values greater than this will not be affected.

## Returns

[Task](#)  <Either<[Error](#), Unit>>

An `TinyFp.Either<L, R>` indicating the result of the operation. Returns `Unit` if the operation succeeds, or an [Error](#) if the operation fails.

## Type Parameters

**T**

The type of the values to compare. Must be comparable.

## Remarks

The method removes all values within the specified range [`min`, `max`] associated with the given `key`. The range is inclusive of both bounds.

# RemoveRangeByValue<T>(string, T, T)

Removes a range of values from a data store based on the specified key and value range.

```
Either<Error, Unit> RemoveRangeByValue<T>(string key, T min, T max)
```

## Parameters

`key` [string](#) 

The key identifying the data store or collection from which values will be removed. Cannot be null or empty.

`min` T

The minimum value of the range to remove. Values less than this will not be affected.

`max` T

The maximum value of the range to remove. Values greater than this will not be affected.

## Returns

Either<[Error](#), Unit>

An TinyFp.Either<L, R> indicating the result of the operation. Returns `Unit` if the operation succeeds, or an [Error](#) if the operation fails.

## Type Parameters

**T**

The type of the values to compare. Must be comparable.

## Remarks

The method removes all values within the specified range [`min`, `max`] associated with the given `key`. The range is inclusive of both bounds.

## Remove<T>(string, IEnumerable<T>)

Removes the specified values associated with the given key from the collection.

```
Either<Error, Unit> Remove<T>(string key, IEnumerable<T> values)
```

## Parameters

`key` [string](#)

The key identifying the collection from which the values will be removed. Cannot be null or empty.

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

The values to be removed from the collection. Cannot be null.

## Returns

Either<[Error](#), Unit>

An TinyFp.Either<L, R> indicating the result of the operation. Returns `Unit` if the removal is successful, or an [Error](#) if the operation fails.

## Type Parameters

T

The type of the values to be removed.

## Remove<T>(string, T)

Removes the specified value associated with the given key from the collection.

```
Either<Error, Unit> Remove<T>(string key, T value)
```

### Parameters

key [string](#)<sup>↗</sup>

The key identifying the value to be removed. Cannot be null or empty.

value T

The value to be removed. Must match the value associated with the specified key.

### Returns

Either<[Error](#), Unit>

An `TinyFp.Either<L, R>` indicating the result of the operation. Returns `Unit` if the removal is successful, or an [Error](#) if the operation fails.

### Type Parameters

T

The type of the value to be removed.

### Remarks

The method will fail if the specified key does not exist in the collection or if the value does not match the one associated with the key.

## ScoreAsync<T>(string, T)

Calculates a score based on the provided key and value, returning either an error or an optional score.

```
Task<Either<Error, Option<double>>> ScoreAsync<T>(string key, T value)
```

## Parameters

key [string](#)

A string representing the key used to identify the scoring context. Cannot be null or empty.

value T

The value associated with the key, which influences the scoring calculation.

## Returns

[Task](#) <Either<[Error](#), Option<[double](#)>>>

An `TinyFp.Either<L, R>` containing either an [Error](#) if the operation fails, or an `TinyFp.Option<A>` representing the calculated score. The score may be absent if no valid result is produced.

## Type Parameters

T

The type of the value used in the scoring operation.

## Score<T>(string, T)

Calculates a score based on the provided key and value, returning either an error or an optional score.

```
Either<Error, Option<double>> Score<T>(string key, T value)
```

## Parameters

key [string](#)

A string representing the key used to identify the scoring context. Cannot be null or empty.

**value** T

The value associated with the key, which influences the scoring calculation.

## Returns

`Either<Error, Option<double>>`

An `TinyFp.Either<L, R>` containing either an [Error](#) if the operation fails, or an `TinyFp.Option<A>` representing the calculated score. The score may be absent if no valid result is produced.

## Type Parameters

**T**

The type of the value used in the scoring operation.

# UnionAsync<T>(string[])

Combines the values associated with the specified keys into a single array.

```
Task<Either<Error, T[]>> UnionAsync<T>(string[] keys)
```

## Parameters

**keys** [string](#)[]

An array of keys whose associated values will be combined. Cannot be null or empty.

## Returns

[Task](#)<Either<[Error](#), T[]>>

An `TinyFp.Either<L, R>` containing either an error if the operation fails, or an array of values of type **T** if the operation succeeds.

## Type Parameters

**T**

The type of the values to be combined.

## Union<T>(string[])

Combines the values associated with the specified keys into a single array.

```
Either<Error, T[]> Union<T>(string[] keys)
```

### Parameters

**keys** [string](#)[]

An array of keys whose associated values will be combined. Cannot be null or empty.

### Returns

Either<[Error](#), T[]>

An `TinyFp.Either<L, R>` containing either an error if the operation fails, or an array of values of type **T** if the operation succeeds.

### Type Parameters

**T**

The type of the values to be combined.



# Class RedisSortedSetService

Namespace: [Func.Redis.SortedSet](#)

Assembly: Func.Redis.dll

```
public class RedisSortedSetService : IRedisSortedSetService
```








## Inheritance

[object](#)  ← RedisSortedSetService

## Implements

[IRedisSortedSetService](#)

## Inherited Members

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

## Constructors

### RedisSortedSetService(ISourcesProvider, IRedisSerDes)

```
public RedisSortedSetService(ISourcesProvider sourcesProvider, IRedisSerDes serDes)
```

## Parameters

**sourcesProvider** [ISourcesProvider](#)

**serDes** [IRedisSerDes](#)

## Methods

### AddAsync<T>(string, IEnumerable<(T Value, double Score)>)

inheritdoc cref="Add{T}(string, IEnumerable{(T Value, double Score)})"/>

```
public Task<Either<Error, Unit>> AddAsync<T>(string key, IEnumerable<(T Value,  
double Score)> values)
```

## Parameters

key [string](#)

values [IEnumerable](#) <(T [Value](#), [double](#) [Score](#))>

## Returns

[Task](#) <Either<[Error](#), Unit>>

## Type Parameters

T

# AddAsync<T>(string, T, double)

inheritdoc cref="Add{T}(string, T, double)"/>

```
public Task<Either<Error, Unit>> AddAsync<T>(string key, T value, double score)
```

## Parameters

key [string](#)

value T

score [double](#)

## Returns

[Task](#) <Either<[Error](#), Unit>>

## Type Parameters

T

# Add<T>(string, IEnumerable<(T Value, double Score)>)

Add values with scores to a sorted set key.

```
public Either<Error, Unit> Add<T>(string key, IEnumerable<(T Value, double Score)> values)
```

## Parameters

key [string](#)

values [IEnumerable](#) <(T [Value](#), [double](#) [Score](#))>

## Returns

Either<[Error](#), Unit>

TinyFp.Unit or [Error](#)

## Type Parameters

T

# Add<T>(string, T, double)

Add a value with score to a sorted set key.

```
public Either<Error, Unit> Add<T>(string key, T value, double score)
```

## Parameters

key [string](#)

value T

score [double](#)

## Returns

Either<[Error](#), Unit>

TinyFp.Unit or [Error](#)

## Type Parameters

T

## DecrementAsync<T>(string, T, double)

Decrement the score of a value in a sorted set key.

```
public Task<Either<Error, Unit>> DecrementAsync<T>(string key, T value, double score)
```

## Parameters

key [string](#)<sup>↗</sup>

value T

score [double](#)<sup>↗</sup>

## Returns

[Task](#)<sup>↗</sup> <Either<[Error](#), Unit>>

## Type Parameters

T

## Decrement<T>(string, T, double)

Decrement the score of a value in a sorted set key.

```
public Either<Error, Unit> Decrement<T>(string key, T value, double score)
```

## Parameters

key [string](#)<sup>↗</sup>

value T

score [double](#)

Returns

Either<[Error](#), Unit>

Type Parameters

T

## IncrementAsync<T>(string, T, double)

Retrieves the intersection of values associated with the specified keys.

```
public Task<Either<Error, Unit>> IncrementAsync<T>(string key, T value,  
double score)
```

Parameters

key [string](#)

value T

score [double](#)

Returns

[Task](#) <Either<[Error](#), Unit>>

An `TinyFp.Either<L, R>` containing either an error if the operation fails, or an array of values of type T that represent the intersection of the values associated with the provided keys.

Type Parameters

T

The type of the values to be intersected.

# Increment<T>(string, T, double)

Increments the score associated with the specified key by the given value.

```
public Either<Error, Unit> Increment<T>(string key, T value, double score)
```

## Parameters

key [string](#) 

The key identifying the item whose score is to be incremented. Cannot be null or empty.

value T

The value to increment the score by. Must be compatible with the type parameter T.

score [double](#) 

The amount by which the score is incremented. Must be a valid double value.

## Returns

Either<[Error](#), Unit>

An `TinyFp.Either<L, R>` indicating the result of the operation. Returns `Unit` if the operation succeeds, or an [Error](#) if the operation fails.

## Type Parameters

T

The type of the value to be incremented. Typically a numeric type.

## Remarks

This method is typically used in scenarios where scores are tracked and updated dynamically, such as leaderboards or ranking systems. Ensure that the key exists and is valid before calling this method to avoid errors.

# IntersectAsync<T>(string[])

Retrieves the intersection of values associated with the specified keys.

```
public Task<Either<Error, T[]>> IntersectAsync<T>(string[] keys)
```

## Parameters

keys [string](#)[]

An array of keys used to identify the values to intersect. Cannot be null or empty.

## Returns

[Task](#) <Either<[Error](#), T[]>>

An `TinyFp.Either<L, R>` containing either an error if the operation fails, or an array of values of type `T` that represent the intersection of the values associated with the provided keys.

## Type Parameters

`T`

The type of the values to be intersected.

## Intersect<T>(string[])

Retrieves the intersection of values associated with the specified keys.

```
public Either<Error, T[]> Intersect<T>(string[] keys)
```

## Parameters

keys [string](#)[]

An array of keys used to identify the values to intersect. Cannot be null or empty.

## Returns

Either<[Error](#), T[]>

An `TinyFp.Either<L, R>` containing either an error if the operation fails, or an array of values of type `T` that represent the intersection of the values associated with the provided

keys.

## Type Parameters

T

The type of the values to be intersected.

## Length(string)

Retrieves the length of the value associated with the specified key.

```
public Either<Error, long> Length(string key)
```


### Parameters

key [string](#) 

The key identifying the value whose length is to be retrieved. Cannot be null or empty.

### Returns

`Either<Error, long>`

An `TinyFp.Either<L, R>` containing either an [Error](#) if the operation fails, or a [long](#)  representing the length of the value if the operation succeeds.

## LengthAsync(string)

Retrieves the length of the value associated with the specified key.

```
public Task<Either<Error, long>> LengthAsync(string key)
```

### Parameters

key [string](#) 

The key identifying the value whose length is to be retrieved. Cannot be null or empty.



## Returns

[Task](#) [↗](#) <Either<[Error](#), [long](#)[↗](#)>>

An `TinyFp.Either<L, R>` containing either an [Error](#) if the operation fails, or a [long](#)[↗](#) representing the length of the value if the operation succeeds.

## LengthByScore(string, double, double)

Retrieves the count of elements in a sorted set stored at the specified key, where the elements' scores fall within the given range.

```
public Either<Error, long> LengthByScore(string key, double min, double max)
```

## Parameters

**key** [string](#)[↗](#)

The key identifying the sorted set. Cannot be null or empty.

**min** [double](#)[↗](#)

The minimum score of the range. Elements with scores greater than or equal to this value are included.

**max** [double](#)[↗](#)

The maximum score of the range. Elements with scores less than or equal to this value are included.

## Returns

Either<[Error](#), [long](#)[↗](#)>

An `TinyFp.Either<L, R>` containing either an error if the operation fails, or the count of elements within the specified score range.

## Remarks

The score range is inclusive of both `min` and `max`.

# LengthByScoreAsync(string, double, double)

Retrieves the count of elements in a sorted set stored at the specified key, where the elements' scores fall within the given range.

```
public Task<Either<Error, long>> LengthByScoreAsync(string key, double min, double max)
```

## Parameters

key [string](#)

The key identifying the sorted set. Cannot be null or empty.

min [double](#)

The minimum score of the range. Elements with scores greater than or equal to this value are included.

max [double](#)

The maximum score of the range. Elements with scores less than or equal to this value are included.

## Returns

[Task](#) <Either<[Error](#), [long](#)>>

An `TinyFp.Either<L, R>` containing either an error if the operation fails, or the count of elements within the specified score range.

## Remarks

The score range is inclusive of both `min` and `max`.

# LengthByValueAsync<T>(string, T, T)

Retrieves the length of a collection stored under the specified key, filtered by a range of values.

```
public Task<Either<Error, long>> LengthByValueAsync<T>(string key, T min, T max)
```

## Parameters

**key** [string](#)

The key identifying the collection in the data store. Cannot be null or empty.

**min** T

The minimum value of the range used to filter the collection.

**max** T

The maximum value of the range used to filter the collection.

## Returns

[Task](#) <Either<[Error](#), [long](#)>>>

An `TinyFp.Either<L, R>` containing either an [Error](#) if the operation fails, or a [long](#) representing the count of items in the collection that fall within the specified range.

## Type Parameters

**T**

The type of the values used to filter the collection.

## Remarks

The method filters the collection based on the provided range [`min`, `max`]. Ensure that `min` is less than or equal to `max` to avoid unexpected results.

## LengthByValue<T>(string, T, T)

Retrieves the length of a collection stored under the specified key, filtered by a range of values.

```
public Either<Error, long> LengthByValue<T>(string key, T min, T max)
```

## Parameters

**key** [string](#)

The key identifying the collection in the data store. Cannot be null or empty.

**min** `T`

The minimum value of the range used to filter the collection.

**max** `T`

The maximum value of the range used to filter the collection.

## Returns

Either<[Error](#), [long](#)>

An `TinyFp.Either<L, R>` containing either an [Error](#) if the operation fails, or a [long](#) representing the count of items in the collection that fall within the specified range.

## Type Parameters

**T**

The type of the values used to filter the collection.

## Remarks

The method filters the collection based on the provided range [`min`, `max`]. Ensure that `min` is less than or equal to `max` to avoid unexpected results.

# RangeByScoreAsync<T>(string, double, double)

Retrieves a range of elements from a sorted collection based on their score values.

```
public Task<Either<Error, T[]>> RangeByScoreAsync<T>(string key, double min, double max)
```

## Parameters

**key** [string](#)

The key identifying the sorted collection. Cannot be null or empty.

**min** [double](#)

The minimum score value for the range. Elements with scores less than this value are excluded.

max [double](#)

The maximum score value for the range. Elements with scores greater than this value are excluded.

## Returns

[Task](#) `<Either<Error, T[]>>`

An `TinyFp.Either<L, R>` containing either an [Error](#) if the operation fails, or an array of elements of type `T` that fall within the specified score range. If no elements match the criteria, an empty array is returned.

## Type Parameters

`T`

The type of elements in the collection.

# RangeByScore<T>(string, double, double)

Retrieves a range of elements from a sorted collection based on their score values.

```
public Either<Error, T[]> RangeByScore<T>(string key, double min, double max)
```

## Parameters

key [string](#)

The key identifying the sorted collection. Cannot be null or empty.

min [double](#)

The minimum score value for the range. Elements with scores less than this value are excluded.

max [double](#)

The maximum score value for the range. Elements with scores greater than this value are excluded.

## Returns

`Either<Error, T[]>`

An `TinyFp.Either<L, R>` containing either an [Error](#) if the operation fails, or an array of elements of type `T` that fall within the specified score range. If no elements match the criteria, an empty array is returned.

## Type Parameters

`T`

The type of elements in the collection.

## RankAsync<T>(string, T)

Retrieves the rank of a specified value within a collection associated with the given key.

```
public Task<Either<Error, Option<long>>> RankAsync<T>(string key, T value)
```

## Parameters

`key` [string](#) 

The key identifying the collection in which the rank is calculated. Cannot be null or empty.

`value` `T`

The value whose rank is to be determined. Must exist within the collection.

## Returns

[Task](#)  `<Either<Error, Option<long >>>`

An `TinyFp.Either<L, R>` containing either an [Error](#) if the operation fails, or an `TinyFp.Option<A>` representing the rank of the value as a zero-based index. Returns `None` if the value is not found in the collection.

## Type Parameters

**T**

The type of the value to rank. Must be comparable within the collection.

## Remarks

The rank is determined based on the ordering of the collection associated with the specified key. If the key does not exist or the value is not found, the method returns **None** within the result.

## Rank<T>(string, T)

Retrieves the rank of a specified value within a collection associated with the given key.

```
public Either<Error, Option<long>> Rank<T>(string key, T value)
```

## Parameters

**key** [string](#) 

The key identifying the collection in which the rank is calculated. Cannot be null or empty.

**value** **T**

The value whose rank is to be determined. Must exist within the collection.

## Returns

[Either](#)<[Error](#), [Option](#)<[long](#)  >>

An `TinyFp.Either<L, R>` containing either an [Error](#) if the operation fails, or an `TinyFp.Option<A>` representing the rank of the value as a zero-based index. Returns **None** if the value is not found in the collection.

## Type Parameters

**T**

The type of the value to rank. Must be comparable within the collection.

## Remarks

The rank is determined based on the ordering of the collection associated with the specified key. If the key does not exist or the value is not found, the method returns **None** within the result.

## RemoveAsync<T>(string, IEnumerable<T>)

Removes the specified values associated with the given key from the collection.

```
public Task<Either<Error, Unit>> RemoveAsync<T>(string key, IEnumerable<T> values)
```

## Parameters

key [string](#) 

The key identifying the collection from which the values will be removed. Cannot be null or empty.

values [IEnumerable](#)  <T>

The values to be removed from the collection. Cannot be null.

## Returns

[Task](#)  <Either<[Error](#), Unit>>

An `TinyFp.Either<L, R>` indicating the result of the operation. Returns **Unit** if the removal is successful, or an [Error](#) if the operation fails.

## Type Parameters

**T**

The type of the values to be removed.

## RemoveAsync<T>(string, T)

Removes the specified value associated with the given key from the collection.



```
public Task<Either<Error, Unit>> RemoveAsync<T>(string key, T value)
```

## Parameters

key [string](#)

The key identifying the value to be removed. Cannot be null or empty.

value T

The value to be removed. Must match the value associated with the specified key.

## Returns

[Task](#) <Either<[Error](#), Unit>>

An `TinyFp.Either<L, R>` indicating the result of the operation. Returns `Unit` if the removal is successful, or an [Error](#) if the operation fails.

## Type Parameters

T

The type of the value to be removed.

## Remarks

The method will fail if the specified key does not exist in the collection or if the value does not match the one associated with the key.

## RemoveRangeByScore(string, double, double)

Removes all elements in the sorted set stored at the specified key with scores within the given range.

```
public Either<Error, Unit> RemoveRangeByScore(string key, double start, double stop)
```

## Parameters

key [string](#)

The key identifying the sorted set. Cannot be null or empty.

**start** [double](#)

The inclusive lower bound of the score range.

**stop** [double](#)

The inclusive upper bound of the score range.

## Returns

Either<[Error](#), Unit>

An `TinyFp.Either<L, R>` indicating the result of the operation. Returns `Unit` if the operation succeeds, or an [Error](#) if it fails.

## Remarks

This method operates on a sorted set and removes elements whose scores fall within the specified range. The range is inclusive of both `start` and `stop`.

## RemoveRangeByScoreAsync(string, double, double)

Removes all elements in the sorted set stored at the specified key with scores within the given range.

```
public Task<Either<Error, Unit>> RemoveRangeByScoreAsync(string key, double start, double stop)
```

## Parameters

**key** [string](#)

The key identifying the sorted set. Cannot be null or empty.

**start** [double](#)

The inclusive lower bound of the score range.

**stop** [double](#)

The inclusive upper bound of the score range.

## Returns

[Task](#)  <Either<[Error](#), Unit>>

An `TinyFp.Either<L, R>` indicating the result of the operation. Returns `Unit` if the operation succeeds, or an [Error](#) if it fails.

## Remarks

This method operates on a sorted set and removes elements whose scores fall within the specified range. The range is inclusive of both `start` and `stop`.

## RemoveRangeByValueAsync<T>(string, T, T)

Removes a range of values from a data store based on the specified key and value range.

```
public Task<Either<Error, Unit>> RemoveRangeByValueAsync<T>(string key, T min,
T max)
```

## Parameters

key [string](#) 

The key identifying the data store or collection from which values will be removed. Cannot be null or empty.

min T

The minimum value of the range to remove. Values less than this will not be affected.

max T

The maximum value of the range to remove. Values greater than this will not be affected.

## Returns

[Task](#)  <Either<[Error](#), Unit>>

An `TinyFp.Either<L, R>` indicating the result of the operation. Returns `Unit` if the operation succeeds, or an [Error](#) if the operation fails.

## Type Parameters

T

The type of the values to compare. Must be comparable.

## Remarks

The method removes all values within the specified range [[min](#), [max](#)] associated with the given [key](#). The range is inclusive of both bounds.

## RemoveRangeByValue<T>(string, T, T)

Removes a range of values from a data store based on the specified key and value range.

```
public Either<Error, Unit> RemoveRangeByValue<T>(string key, T min, T max)
```

## Parameters

key [string](#) 

The key identifying the data store or collection from which values will be removed.  
Cannot be null or empty.

[min](#) T

The minimum value of the range to remove. Values less than this will not be affected.

[max](#) T

The maximum value of the range to remove. Values greater than this will not be affected.

## Returns

[Either](#)<[Error](#), Unit>

An [TinyFp.Either](#)<L, R> indicating the result of the operation. Returns [Unit](#) if the operation succeeds, or an [Error](#) if the operation fails.

## Type Parameters

T

The type of the values to compare. Must be comparable.

## Remarks

The method removes all values within the specified range [[min](#), [max](#)] associated with the given [key](#). The range is inclusive of both bounds.

## Remove<T>(string, IEnumerable<T>)

Removes the specified values associated with the given key from the collection.

```
public Either<Error, Unit> Remove<T>(string key, IEnumerable<T> values)
```

## Parameters

[key](#) [string](#) 

The key identifying the collection from which the values will be removed. Cannot be null or empty.

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

The values to be removed from the collection. Cannot be null.

## Returns

[Either](#)<[Error](#), Unit>

An [TinyFp.Either](#)<L, R> indicating the result of the operation. Returns [Unit](#) if the removal is successful, or an [Error](#) if the operation fails.

## Type Parameters

[T](#)

The type of the values to be removed.

## Remove<T>(string, T)

Removes the specified value associated with the given key from the collection.

```
public Either<Error, Unit> Remove<T>(string key, T value)
```

## Parameters

**key** [string](#)

The key identifying the value to be removed. Cannot be null or empty.

**value** **T**

The value to be removed. Must match the value associated with the specified key.

## Returns

**Either**<[Error](#), Unit>

An `TinyFp.Either<L, R>` indicating the result of the operation. Returns `Unit` if the removal is successful, or an [Error](#) if the operation fails.

## Type Parameters

**T**

The type of the value to be removed.

## Remarks

The method will fail if the specified key does not exist in the collection or if the value does not match the one associated with the key.

## ScoreAsync<T>(string, T)

Calculates a score based on the provided key and value, returning either an error or an optional score.

```
public Task<Either<Error, Option<double>>> ScoreAsync<T>(string key, T value)
```

## Parameters

**key** [string](#)

A string representing the key used to identify the scoring context. Cannot be null or empty.

value T

The value associated with the key, which influences the scoring calculation.

## Returns

[Task](#) [↗](#) <Either<[Error](#), Option<[double](#) [↗](#)>>>

An `TinyFp.Either<L, R>` containing either an [Error](#) if the operation fails, or an `TinyFp.Option<A>` representing the calculated score. The score may be absent if no valid result is produced.

## Type Parameters

T

The type of the value used in the scoring operation.

## Score<T>(string, T)

Calculates a score based on the provided key and value, returning either an error or an optional score.

```
public Either<Error, Option<double>> Score<T>(string key, T value)
```

## Parameters

key [string](#) [↗](#)

A string representing the key used to identify the scoring context. Cannot be null or empty.

value T

The value associated with the key, which influences the scoring calculation.

## Returns

Either<[Error](#), Option<[double](#) [↗](#)>>

An `TinyFp.Either<L, R>` containing either an [Error](#) if the operation fails, or an `TinyFp.Option<A>` representing the calculated score. The score may be absent if no valid

result is produced.

## Type Parameters

**T**

The type of the value used in the scoring operation.

## UnionAsync<T>(string[])

Combines the values associated with the specified keys into a single array.

```
public Task<Either<Error, T[]>> UnionAsync<T>(string[] keys)
```

## Parameters

keys [string](#)[]

An array of keys whose associated values will be combined. Cannot be null or empty.

## Returns

[Task](#)<Either<[Error](#), T[]>>

An `TinyFp.Either<L, R>` containing either an error if the operation fails, or an array of values of type **T** if the operation succeeds.

## Type Parameters

**T**

The type of the values to be combined.

## Union<T>(string[])

Combines the values associated with the specified keys into a single array.

```
public Either<Error, T[]> Union<T>(string[] keys)
```



## Parameters

**keys** [string](#)[]

An array of keys whose associated values will be combined. Cannot be null or empty.

## Returns

Either<[Error](#), T[]>

An `TinyFp.Either<L, R>` containing either an error if the operation fails, or an array of values of type `T` if the operation succeeds.

## Type Parameters

**T**

The type of the values to be combined.

# Namespace Func.Redis.Subscriber

## Interfaces

[IRedisSubscriber](#)

# Interface IRedisSubscriber

Namespace: [Func.Redis.Subscriber](#)

Assembly: Func.Redis.dll

```
public interface IRedisSubscriber
```

## Methods

### GetSubscriptionHandler()

Retrieves the subscription handler for processing Redis messages.

```
(string, Action<RedisChannel, RedisValue>) GetSubscriptionHandler()
```

## Returns

```
(string↗, Action↗<RedisChannel, RedisValue>)
```

A tuple containing the subscription channel name as a [string](#)<sup>↗</sup> and the callback [Action<T1, T2>](#)<sup>↗</sup> to handle incoming messages.