*TeamDev C# Redis Client*

TeamDev Redis Client is a c# Client for REDIS Redis is an advanced key-value store that support many features and that focus on performance. The c# client helps you developing fast applications based in redis database, focusing your development on your objects and not on persistance problems.

**Redis Client**

TeamDev Redis Client is a c# Client for [Redis](http://redis.io/).  
Redis is an advanced key-value store that supports many features and that focuses on performance.  
  
Redis Client is based on [redis-sharp](https://github.com/migueldeicaza/redis-sharp) for the basic communication functions, but it offers some differences,  
a better abstraction from the Redis Protocol, and an usability more close to .Net standard objects in managing Hashes, Sets, Lists and keys.  
Unlike redis-sharp it also supports all commands of Redis up to version 2.2.5  
  
TeamDev Redis Client is not just another provider for Redist :  
it introduces a **DocumentStore** to simplify data management in a **mixed-environment** where you have some data stored in a relational database and some data stored in Redis.  
This will help you to integrate two different ways of managing data without changing your data structure.  
  
With this important feature you can :

* Store an Entity in a relational DataBase AND in Redis (ex: for caching purpose) without too much code.
* Move an Entity from the relational DataBase to Redis and Vice-Versa.
* Store a part of an entity in the relational DataBase and a part of the same entity in Redis.
* Use Redis to store complex objects

It's compatible with all kinds of Entities including **LinqToSql**, **EntityFramework**, **NHibernate** and all kind of Object Relational Mappers (ORM) that use a Class to manage data.  
  
The communication layer has been rewritten according to the new Unified Protocol introduced in version 1.2.

**And now some examples:**

**Base usage**

var redis = new RedisDataAccessProvider();

redis.SendCommand(RedisCommand.SET, "mykey", "myvalue");

redis.WaitComplete();

In RedisCommand enum you will find all supported Redis' commands.  
The provider helps you with some basic Helpers to access all data structures supported in redis.

**List example:**

// Clear the list

redis.List["test"].Clear();

// Append a value

redis.List["test"].Append(i.ToString());

// Count items

var cc = redis.List["test"].Count;

// Left Pop

var r = redis.List["test"].LeftPop();

// Get All Values in the list.

var range = redis.List["test"].Values;

**Hash example:**

// Two way for Setting a value to an hash.

// First way

redis.Hash["myhash"].Set("myfield", "value");

// Second way

redis.Hash["myhash"]["myfield"] = "value";

// reading the hash

var result = redis.Hash["myhash"]["myfield"];

result = redis.Hash["myhash"].Get("myfield");

// All fields in the hash

redis.Hash["myhash"].Keys;

// All values in the hash

redis.Hash["myhash"].Values;

// All key-value pairs in the hash

redis.Hash["myhash"].Items;

redis.Hash["myhash"].ContainsKey("myfield");

You don't have to remember all syntax items of the server but you can use it in a more readable way.  
Helpers will let you access data structures in a way more close to .Net Objects.  
So if you want to set a key in the store, you can do it more pratically just like setting a key-value structure or you can use the redis command directly.

redis.Keys["key"] = "value";

or

redis.SendCommand(RedisCommand.SET, "key", "value");

redis.WaitComplete();

The client supports

* Hashes
* Sets
* SortedSets
* Keys
* Lists

You can access structure's helpers via the properties in the provider.

**Document Store**

The Document Store is the new feature introduced to help integrations between relational datas and non relational datas.  
  
Usually, in modern data access design, you have a model of classes to manage data, and you will use some kind of ORM to access DataBase and to  
persist and retrieve the objects from database.  
ORM usually helps you to write query and filters using some kind of languages.  
  
In some scenarios it's not convenient to store data in a RDBMS, but some data can be easly persisted in a key-value store.  
If this values are part of an entity you will have to manage data access manually and to persist and retrieve data manually.  
**Documet Store** does this job for you.  
It will help you to store data in a key-value way without give to you the work of accessing the store.

**How does it work:**

First of all we need a key to store data ( the store is a KEY-VALUE store, so we need a key)  
Usually you already have the key, and it's stored on your entity as property.  
  
All you have to do is to decorate the property that will contain the key value with the ***DocumentStoreKeyAttribute***

public class myEntity

{

[DocumentStoreKey]

public string mykeyproperty { get; set;}

}

After that you can use the DocumentStore on the "myEntity" to store the entire Entity on Redis.

var store = DocumentStore.New;

store.Set(theentity);

or you can store list of entities in one command

var store = DocumentStore.New;

List<myEntity> entities = new List<myEntity>();

... omissis ...

store.Set(entities);

To retrieve datas you havo only to do the same , but a different command

var store = DocumentStore.New;

store.Get<myEntity>( "100"); // where "100" is the value contained in the property key when we have saved the entity.

If you have to save the entity on the RDBMS and only a part of the entity in Redis,  
you have to decorate the property that you have to save in Redis with the attribute ***DocumentValueAttribute***  
and you have to use the command ***PartialSet*** and ***PartialGet*** to manage the data.

In the following picture you can see the base classes of the provider.  
  
