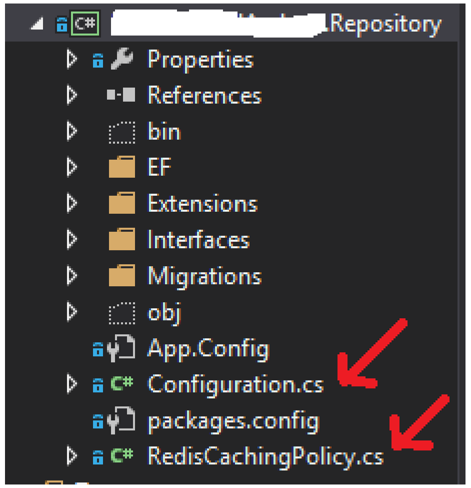

```
[redacted]Repository
  Properties
  References
  bin
  EF
  Extensions
  Interfaces
  Migrations
  obj
  App.Config
  Configuration.cs
  packages.config
  RedisCachingPolicy.cs
```

In the repository project, or the project you have your Entity Framework Code-First, Model-First, Database-First classes you should have these two classes:

1. Configuration.cs
2. RedisCachingPolicy.cs

The first one, Entity Framework looks for by convention when it is initialised, and calls the default constructor.

The second one controls sliding and absolute expiration parameters for the cache to allow for stale item eviction etc.

```
using EFCache;
using EFCache.Redis;
using System.Configuration;
using System.Data.Entity;
using System.Data.Entity.Core.Common;

namespace YourProjectNameSpace.Repository
{
    public class Configuration : DbConfiguration
    {
        public Configuration()
        {
            var useRedis = false;
            bool.TryParse(ConfigurationManager.AppSettings["UseRedisCache"], out useRedis);

            if (!useRedis) return;

            var cache = new RedisCache(ConfigurationManager.AppSettings["RedisConnectionString"]);

            var transactionHandler = new CacheTransactionHandler(cache);

            AddInterceptor(transactionHandler);

            Loaded +=
                (sender, args) => args.ReplaceService<DbProviderServices>(
                    (s, _) => new CachingProviderServices(s, transactionHandler, new RedisCachingPolicy()));
        }
    }
}
```

My web.config file, in my MVC project – the start-up project, the user interface has a couple of application settings:

1. `UseRedisCache`
2. `RedisConnectionString`

The first setting is a bool which indicates whether we want to use redis cache or not.

The second setting is a valid Redis connection string.

```
    <add key="UseRedisCache" value="true" />
    <add key="RedisConnectionString" value="localhost:6379,allowAdmin=true" />
  </appSettings>
```

I have “allowAdmin=true” in my connection string as I have methods in the library that allows clients to clear the cache if desired. I have not tested this with an Azure Redis connection string, and at the moment have never seen a valid Azure Redis connection string – yours might look completely different. As you can see, mine points to a localhost version for testing.

The RedisCachingPolicy class inherits CachingPolicy from EFCache and overrides the expiration timeouts so you can control sliding and absolute expiration times.

```
using EFCache;
using System;
using System.Collections.ObjectModel;
using System.Data.Entity.Core.Metadata.Edm;

namespace YourProjectNameSpace.Repository
{
    public class RedisCachingPolicy : CachingPolicy
    {
        protected override void GetExpirationTimeout(ReadOnlyCollection<EntitySetBase> affectedEntitySets, out TimeSpan slidin
gExpiration,
            out DateTimeOffset absoluteExpiration)
        {
            slidingExpiration = TimeSpan.FromMinutes(5);
            absoluteExpiration = DateTimeOffset.Now.AddMinutes(30);
        }
    }
}
```

I have a sliding expiration of 5 minutes and absolute expiration of 30 minutes. You should tune these values so you get good cache hit rates, but don’t rely on stale data for too long. The rate of queries and calls to your db will play a part in your decision.

In another project that uses the Redis caching package, I have a different setup in the configuration.cs class.

I check the application setting as I do in the examples above for “CacheType” instead and I choose between Redis or InMemory. In the example above it simply just “returns out” without adding caching to Entity Framework.

```
using System.Configuration;
using EFCache;
using System;
using System.Data.Entity;
using System.Data.Entity.Core.Common;

namespace YourProjectNameSpace.Repository
{
    public class Configuration : DbConfiguration
    {
        private readonly static Lazy<ICache> Instance = new Lazy<ICache>(() =>
        {
            if (string.IsNullOrEmpty(ConfigurationManager.AppSettings["CacheType"])) return new InMemoryCache();

            return ConfigurationManager.AppSettings["CacheType"].Equals("Redis")
                ? (ICache) new RedisCache("localhost:6379,allowAdmin=true")
                : new InMemoryCache();
        });

        public Configuration()
        {
            var transactionHandler = new CacheTransactionHandler(Instance.Value);

            AddInterceptor(transactionHandler);

            Loaded +=
                (sender, args) => args.ReplaceService<DbProviderServices>(
                    (s, _) => new CachingProviderServices(s, transactionHandler));

        }
    }
}
```

You can pass a “new RedisCachingPolicy” in the last line new CachingProviderServices(s, transactionHandler, **new RedisCachingPolicy()**)); if you desire.