Alachisoft®

NCache™
Distributed Cache for .NET

AppFabric Migration Guide for NCache

June 1, 2015

Contents

1 Preface

1.1 What is NCache?

NCache is a clustered caching solution in .NET that makes sharing and managing data in a cluster as simple as on a single server. It accomplishes this by coordinating updates to the data using cluster wide concurrency control and replicating and distributing data modifications across the cluster using the highest performing clustered protocol available. The primary purpose of NCache is to improve the performance of .NET & ASP.NET applications that would otherwise make expensive trips to database systems, web services, mainframes or other systems across the network.

1.2 Connecting with NCache

1.2.1 Accessing NCache Resources

You can subscribe to a variety of NCache resources including newsletters, webinars etc. Please visit Alachisoft for details.

1.2.2 Accessing NCache Documentation

You can access all NCache documentation at Online Help Documents on the Alachisoft website.

1.2.3 Contacting NCache Support

For comments, feedback and problems related to NCache or this manual, you can contact NCache Support. Simply send an email to support@alachisoft.com or visit the Alachisoft Technical Support page online.

2 AppFabric to NCache Migration

2.1 Configuration differences

AppFabric has been discontinued by Microsoft; hence, companies are looking for solutions that may be a substitute for, or an upgrade to AppFabric. NCache provides more control over the caching content than AppFabric and it is easy to migrate from AppFabric to NCache.

First of all NCache configuration and AppFabric configuration are completely different. AppFabric takes the cache configuration at runtime whereas NCache is configured before the application is run. The following table shows all the configuration classes that are used to configure AppFabric which are NOT required for NCache.

AppFabric Configuration class
`Microsoft.ApplciationServer.Caching.DataCacheServerEndpoint`
`Microsoft.ApplciationServer.Caching.DataCacheFactoryConfiguration`
`Microsoft.ApplciationServer.Caching.DataCacheLocalCacheProperties`
`Microsoft.ApplciationServer.Caching.DataCacheFactory`
`Microsoft.ApplciationServer.Caching.DataCache`
`Microsoft.ApplciationServer.Caching.DataCacheSecurity`
`Microsoft.ApplciationServer.Caching.DataCacheSecurityMode`
`Microsoft.ApplciationServer.Caching.DataCacheServiceAccountType`
`Microsoft.ApplciationServer.Caching.DataCacheLocalCacheInvalidationPolicy`
`Microsoft.ApplciationServer.Caching.DataCacheFactoryConfiguration`
`Microsoft.ApplciationServer.Caching.DataCacheTransportProperties`
`Microsoft.ApplciationServer.Caching.DataCacheProtectionLevel`

All these configurations and settings can be configured via NCache Manager or the command line tool (if using NCache OpenSource).

3 API Differences

AppFabric uses regions to define divisions in the caches. NCache uses groups to symbolize them. As regions reside inside a named cache and can have unique keys, NCache uses groups to reproduce this behavior.

For example you add "key1" with a string value in the default cache in AppFabric. Now you can add "key1" in a region created by you in the same cache in case of AppFabric, but for NCache you cannot add the same key to a different region in the same cache. You can read about Groups in NCache documentation.

3.1 Getting Cache instance

In AppFabric the user has to create a `DataCacheFactoryConfiguration` which needs to be supplied to a `DataCacheFactory` and from this factory object the user can get a cache instance. In NCache the configurations are taken care of by the Manager or the command line tool and the following command will return an instance of the cache if it is running and initialized.

```
NCache.Web.Caching.NCache.InitializeCache(cacheName);
```

There are many cache management functions available through this instance. We are going to compare some of the overloads of the APIs to get a brief idea of how things are different with NCache.

3.2 Add Operation

In AppFabric a basic Add operation looks like the following:

```
cacheInstance.Add(stringKey, stringValue);
```

In NCache the basic command is no different except when you are specifying a region in the command like:

```
cacheInstance.Add(stringKey, stringValue, regionName);
```

In this case to achieve the same functionality of a region, the following command must be used:

```
regionNameasCacheInstance.Add(stringKey, stringValue);
```

You have to register the region as a cache in NCache otherwise you will not be able to use the above command.

3.3 Bulk Get Operation

AppFabric has the following command in order to get multiple entries residing in the cache.

```
cacheInstance.BulkGet(listOfKeys, region);
```

This returns a List `KeyValuePair` in AppFabric whereas in NCache the Bulk command returns an `IDictionary` of <`string`, `object>` pair:

```
cacheInstance.GetBulk(arrayOfKeys);
```

The argument type is also different for both APIs as AppFabric requires a `List` whereas NCache required an `Array.`

3.4 Get Operation

AppFabric Get operation and NCache Get operation are the same in syntax.

3.5 GetAndLock Operation

AppFabric GetAndLock operation is synonymous to the overload of Get operation of NCache which takes a reference of LockHandle as an argument. For AppFabric a command looks like:

```
cacheInstance.GetAndLock(key, timeout, out lockHandle)
```

For NCache the same command is as follows:

```
cacheInstance.Get(key, timeout, ref this.lockHandle)
```

Both commands return an object as the value associated with the key.

3.6 GetCacheItem Operation

The GetCacheItem command for both AppFabric and NCache is the same with NCache having multiple overloads to filter the CacheItem returned.

3.7 GetIfNewerOperation

The GetIfNewer operation for AppFabric and NCache are exactly the same.

3.8 GetObjects Operation

3.8.1 GetObjectsByAllTags

AppFabric returns a list of `KeyValuePair<string, object>` and takes a list of `DataCacheTag` items to fetch the items that were added to the cache with the corresponding tags. NCache has the GetByAllTags command to substitute this functionality. For NCache a `Hashtable` is returned and the function takes an array of `Tag` as an argument.

AppFabric:

```
cacheInstance.GetObjectsByAllTags(listOfDataCacheTag, region)
```

NCache:

```
cacheInstance.GetByAllTags(arrayofTag)
```

3.8.2 GetObjectsByAnyTag Operation

Same as above, the GetObjectsByAnyTag returns a list in case of AppFabric and a hashtable in the case of Nache.

3.8.3 GetObjectsByTag

This operation takes a single `DataCacheTag` in case of AppFabric and a `Tag` in case of NCache.

3.8.4 GetObjectsInRegion

As mentioned earlier NCache uses caches to represent region therefore in order to get all the objects present in a region/cache NCache can return an enumerator which you can iterate over to get the key value pairs present in the cache.

3.9 Put Operation

AppFabric Put operation can be replaced with the NCache's Insert method. The simplest overload for both NCache and AppFabric is the same:

AppFabric:

```
cacheInstance.Put( key, value)
```

NCache:

```
cacheInstance.Insert( key, value)
```

In case of AppFabric a DataCacheItemVersion is returned whereas in case of NCache CacheItemVersion is returned which are essentially the same (See documentation).

3.10 PutAndUnlock Operation

AppFabric provides this functionality in one function where a locked item is unlocked and updated in the cache. The item can be updated with new `DataCacheTags` as well. In NCache the updating after unlocking the item is achieved by using an overload of the Insert function. A simple AppFabric command will look like the following in which the item is inserted with `DataCacheTags`.

```
cacheInstance.PutAndUnlock(key, value, lockHandle, dataCacheTags);
```

NCache takes `CacheItem` as argument which allows the user to add Tags directly to the item along with Group. Sub-Group, Expiration and Dependencies. The NCache overload is rich and provides more customizability.

```
cacheInstance.Insert(key, cacheItem, lockHandle, acquireLockBool)
```

Unlike AppFabric which takes an object as the new value for the key, NCache provides the richer `CacheItem` to have an item in the cache that holds more meta data that helps on forming organized and structured data groups.

3.11 Remove Operation

AppFabric and NCache have similar Remove operations except for the return type. In case of AppFabric the return type of a remove function is a Boolean which signifies whether the item is successfully removed or not. In NCache the return type is object which is to say the value of the object associated with the said removed key. This functionality of NCache is similar to Getting the item and Deleting it from cache. If the client simply wants to delete the item without having get the item as well NCache provides the Delete functionality which takes in almost identical arguments (See documentation).

```
cacheInstance.Remove(key, dataCacheItemVersion)
```

In NCache you can use either Remove or Delete:

```
cacheInstance.Remove(key, cacheItemVersion)

cacheInstance.Delete(key, cacheItemVersion)
```

3.12 ResetObjectTimeout Operation

AppFabric provides the functionality of resetting the expiration time of the item present in cache by using the following function:

```
cacheInstance.ResetObjectTimeout( key, newTimeout)
```

In NCache the client can accomplish the same behavior by inserting the same item with the same value with a new absolute expiration:

```
cacheInstance.Insert( key, value, absoluteExpiration, null,
CacheItemPriority.Default)
```

This may seem more tedious as to maintain the value associated with the key at the client end would require more resources. NCache provides a better alternative. The client can simply add the item with Sliding expiration. This is just like the expiration used in AppFabric but when an item is updated or fetched from the cache the timeout is automatically reset for the key. Therefore, the need for managing the timeouts of the objects is handled by NCache making the client's job less hectic.

3.13 Unlock Operation

AppFabric and NCache share the same functionality when it comes to unlocking an item present in the cache. Both use a key and a lock handle object to unlock the respective item.

AppFabric:

```
cacheInstance.Unlock( key, dataCacheLockHadle)
```

NCache:

```
cacheInstance.Unlock( key, lockHandle)
```

3.14 CallBack Registration

AppFabric allows the client to register event notification on cache and item level. NCache provides a similar behavior. The difference comes in the layout of the delegate used for either.

For AppFabric:

```
public delegate void DataCacheNotificationCallback(string cacheName, string
regionName, string key, DataCacheItemVersion version, DataCacheOperations
cacheOperation, DataCacheNotificationDescriptor nd);
```

For NCache:

```
public delegate void CacheDataNotificationCallback(string key,
CacheEventArg cacheEventArgs);
```

As you can see the AppFabric delegate is verbose whereas NCache is concise and to the point. The `CacheEventArg` class contains all the required values such as the value associated with the key, the meta-data and any other information for which the client has registered the event for.

Registering a cache level callback is defined for each as follows:

AppFabric:

```
cacheInstance.AddCacheLevelCallback( dataCacheOperations,
dataCacheNotificationCallback)
```

For NCache:

```
cacheInstance.RegisterCacheDataNotification( cacheDataNotificationCallback,
eventType, EventDataFilter.DataWithMetaData)
```

As you can see NCache provides a more comprehensive mechanism to register a notification callback. Also, note here that AppFabric allows one callback to be registered for only ONE kind of event meaning you can have one callback for Item added and another for Item updated. NCache allows the client to add a single callback with multiple types of events like the following modified command from above:

```
cacheInstance.RegisterCacheDataNotification( cacheDataNotificationCallback,
EventType.ItemAdded | EventType.ItemUpdated,
EventDataFilter.DataWithMetaData)
```

NCache also provides multiple callbacks to be registered for single item.

NCache, however, does not provide alternatives to certain events supported by AppFabric such as failure notification callback in a failed operation triggers an event and user delegate's code is executed.

However, the client can find ways to mimic the same behavior by using the extensive functionality provided by NCache.

4 Conclusion

NCache provides alternatives to all the functionality provided by AppFabric and it provides it splendidly. NCache has a lot more to offer to the clients as there are many APIs that have not been touched in this guide. For more information on NCache and Alachisoft visit www.alachisoft.com