Caching for Non-Web Applications – Part 1

A great feature of ASP.NET applications is the Cache class which allows you to store values that are commonly used. Caching data can avoid round-trips to database servers, which can save a lot of time. Until 2010, there was no good way in a Windows Service, Windows Form or WPF application to cache data except by writing your own class. Enter the MemoryCache class, part of the System.Runtime.Caching namespace. This class allows you to add data to a cache and set a time-out so that data can be removed from memory when it is no longer used. This blog post will show you the basics of using this class.

Add Key/Value Pairs

The MemoryCache class has a Default property that refers to a single instance of a MemoryCache. For most applications, you only require one cache object. Of course, you may always create a new instance of a MemoryCache object if you need additional cache objects for your application. In this blog post you are going to just use the default instance.

To add a new value to the cache, you supply three items; a unique key, a value to insert, and how long you want the value to stay in the cache. You are not allowed to insert a null value into the cache, but any other value is allowed. The following code snippet shows the basics of adding a new value to the default MemoryCache instance.

```
MemoryCache.Default.Add("Key1", "Value 1",
   DateTimeOffset.Now.AddSeconds(5));
```

The first parameter you pass to the Add method is a unique key value that you use to retrieve the value later. The second parameter is the value to add. In this case, I just added a simple string, however, this can be any data type. The last parameter is a DateTimeOffset type to specify how much time the item should stay in the cache before it is automatically removed.

Check to See if Value was Inserted

Instead of just calling the Add method blindly as I did in the previous code snippet, you should check to see if the value was added. If you attempt to add the same key value to the same cache, the call will fail and the value is not updated. You can check the return value from the Add method to determine if the value was added.

```
bool ret;

ret = MemoryCache.Default.Add("Key1", "Value 1",
           DateTimeOffset.Now.AddSeconds(5));
if (ret) {
  MessageBox.Show("Key did NOT exist");
}
else {
  MessageBox.Show("The Key did already exist");
}
```

Add Using a Cache Policy

If you wish to set other meta-data information about the item in your cache, create a CacheItemPolicy object and use that to add your item. One of the things you can do with a CacheItemPolicy is set a SlidingExpiration property. The previous code set an absolute time to expire your cache object. The SlidingExpiration property is a TimeSpan to keep the item in your cache, but that time span renews each time the item in the cache is accessed.

```
bool ret;

CacheItemPolicy pol = new CacheItemPolicy();
pol.SlidingExpiration = new TimeSpan(0, 0, 5);
ret = MemoryCache.Default.Add("Key1", "Value 1", pol);
if (ret) {
  MessageBox.Show("Key did NOT exist");
}
else {
  MessageBox.Show("The Key did already exist");
}
```

In the previous code the SlidingExpiration property of a new CacheItemPolicy object is set to 5 seconds. If you access the item in the cache at 4 seconds, then the item will stay in the cache for another 5 seconds. If you access it again after 3 seconds, then another 5 seconds is added on. However, if you do not access the item, then that item is removed from the cache after 5 seconds has elapsed.

Add Using the Default Indexer

If you wish to add an item without setting any expiration time, you can use the default indexer. Pass the key name within square brackets to either add a new, or update an existing key, within the MemoryCache object.

```
MemoryCache.Default["Key1"] = "Value 1a";
```

Access the Data in the Cache

After you have entered data in the cache, you need to retrieve it, check to see if it is still in there, count it, etc. The MemoryCache object, of course, supplies you with the appropriate methods to perform these operations. To retrieve a value, use the Get method. You should check to see that the key exists before you retrieve it.

```
if (MemoryCache.Default.Get("Key1") != null) {
  MessageBox.Show(
     MemoryCache.Default.Get("Key1").ToString());
}
else {
  MessageBox.Show("Cache Item Does Not Exist");
}
```

If you attempt to Get a key that does not exist within the cache a null value is returned. If it does exist, the Get method returns an object type to you. You will need to cast the data to the appropriate data type prior to using it.

Use the Default Indexer

Instead of using the Get method you may also use the indexer property to retrieve a value from the cache. The indexer to the MemoryCache object is just like any other .NET indexer. You pass the key name within square brackets to the MemoryCache object and it will return the value if one exists. The code below provides the exact same functionality as the previous sample.

```
if (MemoryCache.Default[THE_KEY] != null) {
  MessageBox.Show(MemoryCache.Default[THE_KEY].ToString());
}
else {
  MessageBox.Show("Cache Item Does Not Exist");
}
```

Using the Contains Method

Instead of using the Get method and having it return a null if the key does not exist, use the Contains method to just check to see if a key exists in the cache. The code below shows an example of using the Contains method.

```
if (MemoryCache.Default.Contains("Key1")) {
  MessageBox.Show("Cache Item Exists");
}
else {
  MessageBox.Show("Cache Item Does NOT Exist");
}
```

Count the Items in the Cache

Call the GetCount method to determine how many items are currently cached in the MemoryCache object. This method returns an integer value.

```
MessageBox.Show(MemoryCache.Default.GetCount().ToString());
```

Remove an Item from the Cache

At some point, you might with to manually remove an item from the cache, instead of just letting the item expire. To accomplish this, call the Remove method passing in the key value.

```
MemoryCache.Default.Remove("Key1");
```

Summary

In this blog post you were introduced to the MemoryCache object. This object allows you to cache data in any type of application, even a non-web application. This class is like the ASP.NET Cache object in that it allows you to time-out values you put into the cache. You can add values to the cache using the Add or default indexer. Retrieve values from the cache using the Get method or the default indexer. You may also count, check for key existence, and remove items from the cache.

Sample Code

You can download the code for this sample at www.pdsa.com/downloads. Choose the category “PDSA Blogs”, then locate the sample **Caching for Non-Web Applications – Part 1**.