Garbage Collection Notification

.NET 4.0 introduced the new concept of Garbage Collection Notification; this allows you to receive notifications just before the system collects garbage, as well as when the collection is completed successfully. Why you might want to do this may not be immediately obvious, but this *does* open up some useful options.

You may want to suspend a resource-intensive operation while the garbage is being collected. This may speed up garbage collection, especially if this is a background process that won't affect the UI.

You may want to flag the load balancer to skip a particular server during garbage collection. This will also speed up garbage collection, while forcing users to the server that will be more responsive.

You may want to explicitly remove data from Session when it becomes obvious that resources are being stressed. Depending on your specific application, it may be possible to delay eventual garbage collection, or streamline the process by explicitly removing objects from Session or Web Cache.

However you use it, this is a new feature and, without a doubt, we will see exciting new applications as we get more familiar with it.

So how do we make this magic work? This actually takes several steps to implement, but they are not difficult steps. Garbage Collection Notification sounds like an event, and you might expect it to follow the normal event wire-up mechanism, but it doesn't. At first, this may sound counter-intuitive, but there is actually a good reason for creating a new notification workflow. You may recall from earlier that poorly controlled event handlers are a common source of memory leaks. That means that, if notifications were implemented as a traditional event off of the GC, you would be guaranteed a memory leak. The code handling the notification would never be flagged as being eligible for collection because the GC, which never goes out of scope, would have a reference to it until the event handler was explicitly removed. The notification process developed for Garbage Collection Notification makes this much more explicit.

Instead, we start by calling `GC.RegisterForFullGCNotification`, and, when we are no longer interested in being notified, we explicitly call `GC.CancelFullGCNotification.` In the middle, we setup a thread to continuously call `GC.WaitForFullGCApproach` and `GC.WaitForFullGCComplete;` and these two methods will return a value from the `GCNotificationStatus` enumeration, which has the potential values shown in Table 6.1.

`Succeeded`	The notification is ready to be raised.
`Failed`	The notification failed for some reason.
`Canceled`	The notification was canceled by calling `CancelFullGCNotification.`
`Timeout`	The timeout specified as a parameter to the `wait` methods has passed.
`NotApplicable`	This result will generally mean that notification was not registered or Concurrent GC is enabled.

Table 6.1: The possible values for the `GCNotificationStatus.`

We set up monitoring with code like this (Listing 6.2).

```
GC.RegisterForFullGCNotification(30, 30);
var notificationThread = new Thread(WaitForFullGcProc);
notificationThread.Start();
// Application logic that needs to be
// notified about garbage collections
GC.CancelFullGCNotification();
```

Listing 6.2: Registering for GC notification.

You may be wondering about the magic constants in the call to `RegisterForFullGC Notification.` This

method takes two parameters, `maxGenerationThreshold` and `largeObjectHeapThreshold`. Both parameters can be integers between 1 and 99. Larger values for these parameters will cause the notification to be raised earlier, and smaller values will cause the notification to be raised closer to when the event actually takes place.

The first parameter allows you to specify that you want to be notified based on the number of objects that have survived to Generation 2, and the second parameter specifies that you want to be notified based on the size of the Large Object Heap. However, neither parameter specifies an absolute value, so passing in 30 for the `maxGenerationThreshold` does not imply triggering a Notification when there are 30 objects in Generation 2; it simply means that you want to be notified earlier than if you had passed in a value of 10. Unless you are specifically more interested in one trigger over the other, you may want to pass in the same value for each parameter, as this will help ensure that you are notified at the same stage, regardless of the trigger.

A larger value will give you more time to deal with memory pressure, but you just need to be careful not to set it too high. The higher you set the thresholds, the quicker you get notified but the longer you have to wait on the GC. Play with these parameters in your application to see what works best for your load and the types of objects consuming your memory.

In its simplest form, the `WaitForFullGcProc` may look as in Listing 6.3.

```
public static void WaitForFullGcProc()
{
  while (true)
  {
     var status = GC.WaitForFullGCApproach();
     InterpretNotificationStatus(status,
         “Full notifications”);
     // Check for a notification of a completed collection.
     status = GC.WaitForFullGCComplete();
     InterpretNotificationStatus(status,
         “Full notifications complete”);
     Thread.Sleep(500);
  }
}
```

Listing 6.3: Simple implementation of a `WaitForFullGC`.

And Listing 6.4 shows how `InterpretNotificationStatus` might look.

```
private static void InterpretNotificationStatus
  (GCNotificationStatus status, string whichOne)
{
 switch (status)
 {
  case GCNotificationStatus.Succeeded:
   {
      Console.WriteLine(whichOne + “raised.”);
      break;
   }
  case GCNotificationStatus.Canceled:
   {
      Console.WriteLine(whichOne + “canceled.”);
      break;
   }
  case GCNotificationStatus.Timeout:
   {
      Console.WriteLine(whichOne
```

```
            + “notification timed out.”);
        break;
    }
  case GCNotificationStatus.NotApplicable:
    {
        Console.WriteLine(“not applicable.”);
        break;
    }
  }
}
```

Listing 6.4: A simple implementation showing how to interpret the notification status.

Here, we are simply writing out to the console what notification we are receiving. You can take whatever actions are appropriate in your situation in the individual case statements.