Open the Data Browser and then, on the command line, execute:

```
DemoPreemptiveBackgroundTask()
```

After the action stops, send more tasks to the background thread by executing

```
PostMoreFreqs()
```

While this is running, experiment with creating graphs, using dialogs, creating functions, etc. Note that both tasks run indefinitely.

To start over you need to stop the preemptive background task, stop the named background task, kill the graph, and delete the data. This function, which you can paste into the main procedure window, will do it.

```
Function StopDemo()
   NVAR threadGroupID = root:testdf:threadGroupID

   // Stop preemptive thread
   Variable status = ThreadGroupRelease(threadGroupID)

   // Stop named background task
   CtrlNamedBackground ThreadResultsTask, stop

   // Kill graph
   DoWindow /K ThreadResultsGraph

   // Kill data
   KillDataFolder root:testdf
End
```

Aborting Threads

If a procedure abort occurs, via a user abort or a programmed abort (see **Aborting Functions** on page IV-112), we would like all pre-emptive threads to stop running. In most cases, this is not something you need to worry about because the threads, by the nature of what they are doing, are guaranteed to stop in a relatively short period of time. It becomes an issue for threads that can run for arbitrarily long periods of time. In such cases, it is a good idea to handle the abort as explained in this section.

To signal running pre-emptive threads to stop, the thread-coordinating function, running in the main thread, must detect the abort and call ThreadGroupRelease. ThreadGroupRelease signals the pre-emptive threads to quit. For example:

```
// A thread worker function that may take a long time
ThreadSafe Function ThreadWorkerFunc(threadNumber, numWaves, numPoints)
   Variable threadNumber, numWaves, numPoints

   Printf "Thread worker %d starting\r", threadNumber

   try
      int i
      for(i=0; i<numWaves; i+=1)
         Make/FREE/N=(numPoints) junk=gnoise(1) // Busy work to take up time
      endfor
   catch
      // ThreadGroupRelease, called from the main thread,
      // causes this catch block to execute
      Printf "Thread worker %d aborted after %d iterations\r", threadNumber, i
      return -1
   endtry

   Printf "Thread worker %d quitting normally\r", threadNumber
   return 0
End

Function ThreadCoordinatingFunction(numThreads, numWaves, numPoints)
   int numThreads, numWaves, numPoints    // Runs in main thread
```