ThreadGroupGetDF (deprecated)	**ThreadGroupGetDFR**
ThreadGroupWait	**ThreadReturnValue**
ThreadGroupRelease	

To run a threadsafe function preemptively, you first create a thread group using **ThreadGroupCreate** and then call **ThreadStart** to start your worker function. Usually you will use the same function for each thread of a group although they can be different.

The worker function must be defined as threadsafe and must return a real or complex numeric result. The return value can be obtained after the function finishes by calling **ThreadReturnValue**.

The worker function can take variable and wave parameters. It can not take pass-by-reference parameters or data folder reference parameters.

Any waves you pass to the worker are accessible to both the main thread and to your preemptive thread. Such waves are marked as being in use by a thread and Igor will refuse to perform any operations that could change the size of the wave.

You can determine if any threads of a group are still running by calling **ThreadGroupWait**. Use zero for the "milliseconds to wait" parameter to just test if all threads are finished. Use a larger value to cause the main thread to sleep until all threads are finished. If you know the maximum time the threads should take, you can use that value and print an error message or take other action if the threads don't finish in time.

When ThreadGroupWait is called, Igor updates certain internal variables including variables that track whether a thread has finished and what result it returned. Therefore you must call ThreadGroupWait before calling ThreadReturnValue.

Once you are finished with a given thread group, call **ThreadGroupRelease**.

If your threads can run for a long time, you should detect aborts and make your threads quit. See **Aborting Threads** on page IV-337 for details.

The Igor debugger can not be used with threadsafe functions. See **Debugging ThreadSafe Code** on page IV-225 for details.

The hard part of using multithreading is devising a scheme for partitioning your data processing algorithms into threads.

Thread Data Environment

When a thread is started, Igor creates a root data folder for that thread. This root data folder and any data objects that the thread creates in it are private to the thread. This constitutes a separate data hierarchy for each thread.

Data is transferred, when you request it, from the main thread to a preemptive thread and vice-versa using input and output queues. The "currency" of these queues is the data folder, which provides considerable flexibility for passing data to threads and for retrieving results. Each thread group has an input queue to which the main thread may post data and an output queue from which the main thread may retrieve results.

The terms "input" and "output" are relative to the preemptive thread. The main thread posts a data folder to the input queue to send input to the preemptive thread. The preemptive thread retrieves the data folder from the input queue. After processing, the preemptive thread may post a data folder to the output queue. The main thread reads output from the preemptive thread by retrieving the data folder from the output queue.

Use **ThreadGroupPutDF** to post data folders and **ThreadGroupGetDFR** to retrieve them. These are called from both the main thread and from preemptive threads.