Use zero for *waitms* to just test or provide a large value to cause the main thread to sleep until the threads are finished. You can use INF to wait forever or until a user abort. If you know the maximum time the threads should take, you can use that value so you can print an error message or take other action if the threads don't return 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**.

ThreadGroupWait updates the internal state of all threads in the group.

Finding a Free Thread

If you pass -2 for *waitms*, ThreadGroupWait returns index+1 of the first free (not running) thread or 0 if all threads in the group are running.

This allows you to dispatch a thread anytime a free thread is available. See **Parallel Processing - Thread-at-a-Time Method** on page IV-332 for an example.

See Also

The **ThreadGroupCreate** function, **ThreadSafe Functions** on page IV-106, and **ThreadSafe Functions and Multitasking** on page IV-329.

ThreadProcessorCount

ThreadProcessorCount

The ThreadProcessorCount function returns the number of processors in your computer. For example, on a Macintosh Core Duo, it would return 2.

ThreadReturnValue

ThreadReturnValue(*tgID*, *index*)

The ThreadReturnValue function returns the value that the specified thread function returned when it exited. Returns NAN if thread is still running. *tgID* is the thread group ID returned by **ThreadGroupCreate** and *index* is the thread number.

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.

See Also

The **ThreadGroupCreate** function, **ThreadSafe Functions** on page IV-106, and **ThreadSafe Functions and Multitasking** on page IV-329.

ThreadSafe

ThreadSafe Function *funcName*()

The ThreadSafe keyword declaration specifies that a user function can be used for preemptive multitasking background tasks on multiprocessor computer systems.

A ThreadSafe function is one that can operate correctly during simultaneous execution by multiple threads. Such functions are generally limited to numeric or utility functions. Functions that access windows are not ThreadSafe. To determine if an operation is ThreadSafe, use the Command Help tab of the Help Browser and choose ThreadSafe from the pop-up menu.

ThreadSafe functions can call other ThreadSafe functions but may not call non-ThreadSafe functions. Non-ThreadSafe functions can call ThreadSafe functions.

See Also

ThreadSafe Functions on page IV-106 and **ThreadSafe Functions and Multitasking** on page IV-329.

ThreadStart

ThreadStart *tgID*, *index*, *WorkerFunc*(*param1*, *param2*,…)

The ThreadStart operation starts the specified function running in a preemptive thread.