datafolder can be just the name of a child data folder in the current data folder, a partial path (relative to the current data folder) and name or an absolute path (starting from root) and name.

Details

When you call it from the main thread, **ThreadGroupPutDF** removes *datafolder* from the main thread's data hierarchy and posts to the input queue of the thread group specified by *tgID*.

When you call it from a preemptive thread, use 0 for *tgID* and the data folder will be posted to the output queue of thread group to which thread belongs.

Input and output data folders may be retrieved from the queues by calling the string function **ThreadGroupGetDF** or **ThreadGroupGetDFR**.

Warning: Take care not to use any stale WAVE, NVAR, or SVAR variables that might contain references to objects in the data folder. Use **WAVEClear** on all WAVE reference variables that might contain references to waves that are in the data folder being posted before calling **ThreadGroupPutDF**. An error will occur if any waves in the data folder are in use or referenced in a WAVE variable.

Warning: Any DFREF variables that refer to the data folder (or any child thereof) must be cleared prior to executing this command. You can clear a DFREF using `dfr=$""`.

From the standpoint of the source thread, ThreadGroupPutDF is conceptually similar to KillDataFolder and, like KillDataFolder, if the current data folder is within *datafolder*, the current data folder is set to the parent of datafolder. You can not pass `root:` as *datafolder*.

See Also

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

ThreadGroupRelease

`ThreadGroupRelease(`*`tgID`*` [, `*`beGraceful`*`])`

The ThreadGroupRelease function releases the thread group referenced by *tgID*. *tgID* is the thread group ID returned by **ThreadGroupCreate**. Any data folders remaining in the group's input or output queues are discarded.

Specifying *tgID*=-2 kills all running threads in all thread groups.

Normally you call ThreadGroupRelease after all threads in the group have completed. In the event that threads are still running, they are aborted. An attempt is made to safely stop running threads but, if they continue to run, they are killed.

The *beGraceful* parameter, added in Igor Pro 9.00, affects thread worker functions that include catch blocks to handle aborts gracefully. If *beGraceful* is omitted or is 0, ThreadGroupRelease performs aborts in a way that interferes with such catch blocks. If *beGraceful* is 1, the catch blocks run normally. See **Aborting Threads** on page IV-337 for details.

ThreadGroupRelease returns zero if successful, -1 if an error occurred (probably invalid *tgID*), or -2 if a force quit was needed. In the latter case, you should restart Igor Pro.

See Also

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

ThreadGroupWait

`ThreadGroupWait(`*`tgID`*`, `*`waitms`*`)`

The ThreadGroupWait function returns index+1 of the first thread found still running after *waitms* milliseconds or returns zero if all are done.

tgID is the thread group ID returned by **ThreadGroupCreate** and *waitms* is milliseconds to wait.

If any of the threads of the group encountered a runtime error, the first such error will be reported now.