WoW Threads API

## A library of asynchronous, non-preemptive threads.

<Introduction to the API>

# Create Thread

**thread\_h = thread:create( yieldTicks, threadFunction, ... )**

**Description**: creates a table of thread attributes called a thread handle. The library and the WoW Client can call this service.

**Context**: This method can be called by other threads and the WoW client.

**Params**:

- yieldTicks (number): The number of ticks for the thread.

- threadFunction (function): The function to be executed by the thread.

- ... (any): Additional arguments to be passed to the thread function.

**Returns**: thread handle (table):

## Usage

This snippet creates a thread, main\_h, whose function, sayHello, is a while loop in which the thread uses *thread:yield()* to suspend itself every second (60 ticks) and then, if it receives a SIG\_ALERT, prints a greeting in the Chat Frame.

local function sayHello()

local DONE = false

*while not DONE do*

*thread:yield()*

*local signal, sender\_h, data = thread:getSignal()*

*if signal == SIG\_ALERT then*

*print(“Hello everyone!”)*

*elseif* signal == SIG\_TERMINATE then

DONE = true

end

end

***local main\_h = thread:create( 60, sayHello )***

# Yield the Processor

**thread:yield()**

**Description**: yields the processor to the next thread in the Thread Control Table (TCB).

**Context**: Can only be called by other threads.

**Params**: None

**Returns**: None

## Usage

This snippet creates a thread, main\_h, whose function, sayHello, is a while loop in which the thread uses *thread:yield()* to suspend itself every second (60 ticks) and then, if it receives a SIG\_ALERT, prints a greeting in the Chat Frame.

local function sayHello()

local DONE = false

*while not DONE do*

***thread:yield()***

*local signal, sender\_h, data = thread:getSignal()*

*if signal == SIG\_ALERT then*

*print(“Hello everyone!”)*

*elseif* signal == SIG\_TERMINATE then

DONE = true

end

end

local main\_h = thread:create( 60, sayHello )

# Delay a Thread

**thread:delay( delayTicks )**

**Description**: delays a thread by the specified number of ticks. Thread context required.

**Context**: Can only be called by other threads.

**Params**:

- delayTicks (number): the number of ticks to delay

**Returns** None

## Usage

# Obtain a Thread’s Unique Identifier

**threadId = thread:getId( thread\_h )**

**Description**: **Returns** the thread's numerical Id. Thread context required.

**Context**: Can only be called by other threads.

**Params**:

- thread\_h (handle)

**Returns** (number) threadId

## Usage

# Get the Currently Running Thread

**self\_h = thread:getSelf()**

**Description**: **Returns** the handle of the calling thread. Thread context required.

**Context**: Can only be called by other threads.

**Params**: None

**Returns** (handle) thread\_h, (number) threadId

## Usage

# Determine If Two Threads are Identical

**areEqual = thread:areEqual( H1, H2 )**

**Description**: determines whether two thread handles are identical. Thread context required.

**Context**: Can only be called by other threads.

**Params**

- thread1\_h (handle)

- thread2\_h (handle)

**Returns** (boolean) true if equal

## Usage

# Get the Thread’s Parent (creator) Thread

**parent\_h = thread:getParent( thread\_h )**

**Description**: gets the specified thread's parent. Thread context required.

**Context**: Can only be called by other threads.

**Params**:

- thread\_h (handle)

**Returns** nil if thread has no parent.

## Usage

# Get a Table of a Thread’s Children

**childThreads = thread:getChildThreads( thread\_h )**

**Description**: gets a table of the specified thread's children. Thread context required.

**Context**: Can only be called by other threads.

**Params**:

thread\_h (handle)

**Returns** childThreads (table)

## Usage

# Send A Thread A Signal

**sendSignal( target\_h, signal, ... )**

**Description**: sends a signal to the specified thread.

**Context**: Can be called by both the WoW client and another thread.

**Params**:

- target\_h (handle) the thread to which the signal is to be sent.

- signal (number)

- ... (varargs). data to be passed to the receiving thread

**Returns** None

## Usage

# Get A Signal

**signal, sender\_h, data = thread:getSignal()**

**Description**: retrieves a signal sent to the calling thread. Thread context required.

**Context**: Can only be called by other threads.

**Params**:

**Returns** signal (number), sender\_h (handle), data (...)

## Usage

# Get A Signal’s Name

**signalName = thread:getSignalName( signal )**

**Description**: gets the string name of the specified signal. Thread context NOT required.

**Context**: Can only be called by other threads.

**Params**:

- signal(number

**Returns**: signalName (string)

## Usage

# Get A Thread’s Execution State

**thread\_h = thread:getState( thread\_h )**

**Description**: gets the thread's execution state.

**Context**: Can only be called by other threads.

**Params**:

- thread\_h (handle)

**Returns**: state (string) = ("dead", "running", or "suspended")

## Usage