assay.trydemos - Function

Package	**utility**
Module	**misc**
Tool	**assay**

Tries the standard demonstrations

Synopsis
trydemos(demos)

Arguments

demos	Names of demos Allowed:String Default: All demos

Returns
Int - Number that failed

1.6.21 stopwatch - Tool

Package	**utility**
Module	**misc**

stopwatch style timer functions
include "stopwatch.g"

Constructors

stopwatch	Construct stopwatch tool

Functions

delete	destroy the tool
fmtvalue	formatted string value of timer with units
help	display help
reset	reset time value to zero
show	show value and run state of timer
start	start timer running

state	get run state of timer
stop	stop timer
timefunc	time a function
value	get numerical value of timer in seconds
zero	stop and reset timer

Description

The stopwatch class provides a set of timer functions with the functionality of a traditional stopwatch. Any number of stopwatches may be created, which can then individually be started, stopped, reset, displayed and interrogated. A function can be passed to a stopwatch, which is then run and timed. This is particularly useful for benchmarking purposes. Due to the distributed process nature of AIPS++, it is very difficult correctly to account for all the resources used by a given calculation. Thus only wall clock time is implemented in this class, as opposed to the "system, user and wall clock" accounting often found in simple single threaded applications. Multiple stopwatch timers will not interfere with each other.

Example

```
- include "stopwatch.g"              # no timer is created by default
T
- tim := stopwatch()
- tim.show()                         # timer created running
timer is RUNNING, value is 3.56s
F
- tim.stop()                         # stop and start
timer stopped at 9.31s
F
- tim.show()
timer is STOPPED, value is 9.31s
F
- tim.start()                        # restartable
timer restarted from 9.31s
F
-tim.reset()                         # reset doesn't change state
timer reset to zero
F
- tim.show()
timer is RUNNING, value is 12.35s
F
```

```
- tim.stop()                        # values can be retrieved
timer stopped at 1m 8.73s
F
- tim.value()
68.733093
- tim.fmtvalue()
1m 8.73s
- tim.state()
STOPPED
- tim.start(quiet=T)                # suppress messages
- split_time := tim.value()         # a value while running is a split time
- split_time
74.526214
```

See timefunc for examples of how to time functions. stopwatch tools also provide standard functions `ok()` and `delete()`.

stopwatch - Constructor

Package	**utility**
Module	**misc**
Tool	**stopwatch**

Construct stopwatch tool

Synopsis
stopwatch()

Description
Creates a stopwatch tool, initialized to zero and running. Any number of stopwatches may be created without interfering with each other.

Returns
stopwatch tool

---

stopwatch.start - Function

Package	**utility**
Module	**misc**
Tool	**stopwatch**

start timer running

Synopsis
start(quiet)

Description
In the running state, the timer advances at real time. The value of the timer is not reset to zero, and the timer may be restarted after it has been stopped.

Arguments

quiet	Suppress status output? Default: F

A timer is created zeroed and running, so you may not need to call `start()` explicitly. Similarly, there is no need to directly set the state of the timer when timing functions,

---

stopwatch.stop - Function

Package	**utility**
Module	**misc**
Tool	**stopwatch**

stop timer

Synopsis
stop(quiet)

Description
In the stopped state, the timer does not advance, though it can be interrogated for the value in different formats. It may be restarted.

Arguments

quiet	Suppress status output? Default: F

---

stopwatch.reset - Function

Package	**utility**
Module	**misc**
Tool	**stopwatch**

reset time value to zero

Synopsis
reset(quiet)

Description
The timer value is reset to zero, but the run state is not changed.

Arguments

quiet	Suppress status output? Default: F

---

stopwatch.zero - Function

Package	**utility**
Module	**misc**
Tool	**stopwatch**

stop and reset timer

Synopsis
zero(quiet)

Description
The timer is stopped and the value is reset to zero.

Arguments

quiet	Suppress status output? Default: F

---

stopwatch.value - Function

Package	**utility**
Module	**misc**
Tool	**stopwatch**

get numerical value of timer in seconds

Synopsis
value()

Returns
numerical value of timer in seconds

---

stopwatch.fmtvalue - Function

Package	**utility**
Module	**misc**
Tool	**stopwatch**

formatted string value of timer with units

Synopsis
fmtvalue(value)

Arguments

value	Time in seconds to be formatted Default: Current value()

Returns
formatted time string value with units

Description
If no explicit value is passed to the function, the current value of the stopwatch will be used. Leading fields with zero value are suppressed.

Example
An extreme example might be

```
- tim.fmtvalue()
27y 306d 20h 45m 54.44s
```

Or when passing in an explicit value

```
- tim.fmtvalue(90)
1m 30.00s
```

---

stopwatch.state - Function

Package	**utility**
Module	**misc**
Tool	**stopwatch**

get run state of timer

Synopsis
state()

Returns
either 'RUNNING' or 'STOPPED'

---

stopwatch.show - Function

Package	**utility**
Module	**misc**
Tool	**stopwatch**

show value and run state of timer

Synopsis
show()

Example

```
- tim.show()
timer is RUNNING, value is 1m 8.73s
F
```

---

stopwatch.timefunc - Function

Package	**utility**
Module	**misc**
Tool	**stopwatch**

time a function

Synopsis
timefunc(func, niter, name, quiet)

Description
Run a function, and display the runtime. The function to be timed should take no arguments, but constant arguments may be passed in by defining a function expression in the call. After the function is run, the numerical value of the runtime can be retrieved from the time with value.

Arguments

func	Function to be timed
niter	Number of iterations Default: 1
name	Output label Default: '' or func
quiet	Suppress status output? Default: F

Example

```
                                    # a function that might take a while
- function iter_sin (iter=10,len=10000) {
+ v:=[1:len]; while (iter>0) { iter:=iter-1; v := sin(v); } v; }

- tim.timefunc(iter_sin)             # time it
Timing function...
return value is [0.462957899 0.471936887 0.137069347 -0.448762012 ... ]
Function took 0.15s  to run
F
-                                    # can supply args this way
+ tim.timefunc(function () { iter_sin(1000) })
Timing function...
```

```
return value is [0.0545929715 0.0546066474 0.0510562464 -0.0545696198 ... ]
Function took 9.31s  to run
F
-                                    # or this way
+ function tmpfunc() { iter_sin(1000) }
tim.timefunc(tmpfunc,quiet=T)
F
- tim.show()
timer is STOPPED, value is 9.21s      # timings are reasonably consistent
F
-                                    # can average multiple runs
+ t.timefunc(function() { iter_sin(1000)}, 3, 'big_demo')
Timing function big_demo 3 times...
return value is [0.0545929715 0.0546066474 0.0510562464 -0.0545696198 ... ]
Function big_demo took 9.20s to run
return value is [0.0545929715 0.0546066474 0.0510562464 -0.0545696198 ... ]
Function big_demo took 9.26s to run
return value is [0.0545929715 0.0546066474 0.0510562464 -0.0545696198 ... ]
Function big_demo took 9.31s to run

Function big_demo took an average of 9.26s to run, over 3 iterations
T
```

The func argument may also be be a string containing the name of the function to be timed. In this case, the name argument will default to the function name.

---

stopwatch.help - Function

Package	**utility**
Module	**misc**
Tool	**stopwatch**

display help

Synopsis
help()

Example

```
- tim.help()
Stopwatch style timing.  Available member functions:
```

```
tim := stopwatch()  # timer is created running and zeroed

tim.start()        # timer returns mutable values
tim.stop()         # timer returns fixed values
tim.reset()        # zeros time w/o state change
tim.zero()         # stop followed by reset
tim.state()        # returns 'RUNNING' or 'STOPPED'
tim.value()        # return current value in seconds
tim.fmtvalue(v)    # return string formatted value, with units
tim.show()         # print state and value of timer
tim.timefunc(func,niter,label)
                   # time function f.  Should take no parameters
tim.delete()       # destroy the tool

Most functions take an optional quiet argument to suppress informational
messages.  Notice that tim.timefunc(function () {foo(5)}) is a valid call
if you wish to avoid defining a new function to provide arguments.  After
timing a function, the numerical interval can be retrieved with tim.value()
F
```

---

stopwatch.delete - Function

Package	**utility**
Module	**misc**
Tool	**stopwatch**

destroy the tool

Synopsis
delete()

1.6.22 edit - Function

Package	**utility**
Module	**misc**

Edit and optionally execute a file

Synopsis
edit(file, execute, editor)
include "edit.g"

Synopsis
edit(file, execute=T, editor='')(file, execute, editor)

Description
Edit 'file'. If 'editor' is set, use it, otherwise use the editor specified by the 'EDITOR' environment variable if set, otherwise use 'vi'. If 'execute' is T, execute (i.e. include), the file.

Arguments

file	File to edit Allowed:string
execute	Read the file into glish? Allowed:T or F Default: T
editor	Editor to use, if unset use EDITOR Allowed:string Default: ''

Returns
T if OK, fail or F on failure

Example

```
ok := edit('myfile.g');
ok := edit('myfile.g', execute=F);    # Edit, but don't execute
ok := edit('myfile.g', editor='vi');  # Edit with vi
```

1.6.23 note - Function
Package utility
Module misc

Send a message to the logger or terminal

Synopsis
note(. . . , priority, origin, time, postcli)
include "note.g"

Synopsis
edit(..., priority='NORMAL', origin='Glish', time='')(. . . , priority, origin, time, postcli)

Description
Use `note` to send a message to the logger if AIPS++is running, otherwise it sends it to the screen. This is useful if you are writing Glish functions that should function in AIPS++and non-AIPS++environments. In particular, sending a message doesn't start AIPS++, but if AIPS++is already started then the message is logged.
The priority defaults to `NORMAL` and the origin to `Glish`, however these can be overridden if desired.

Arguments

. . .	in	components of the message Allowed:string
priority	in	Message priority Allowed:DEBUGGING,NORMAL,WARN,SEVERE Default: NORMAL
origin	in	source of the message Allowed:string Default: 'Glish'
time	in	time as a string Allowed:string Default: Current time
postcli	in	Show message in CLI window too? Allowed:Bool Default: F

Example

```
- note('The value of x is', x);
```

See Also

1. The AIPS++logger
2. throw

1.6.24 throw - Function

Package utility
Module misc

Send a SEVERE message, then fail

Synopsis
throw(. . . , origin, postcli)
include "note.g"

Synopsis
throw(..., origin='Glish')(. . . , origin, postcli)

Description
It is often useful to be able to both print an error message (to the logger if it is defined, otherwise to the terminal), and then return a fail. This function implements that idiom.
The priority is always set to `SEVERE`.

Arguments

. . .	in	components of the message Allowed:string
origin	in	source of the message Allowed:string Default: 'Glish'
postcli	in	Show message in CLI window too? Allowed:Bool Default: F

Returns
fail with the message in it

Example