Unit Tests for Bosal

Legal Notices and Information

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
0.1	July 17, 2018	Initial separation of test cases from main bosal program.	GAM

Contents

Introduction

This document contains a set of unit tests for test harnesses generated by the `bosal` program. The tests presented here exercise the command interface to a test harness as generated by `bosal`. The test harness used in the tests is an integration of the automated lubrication and signal I/O domains from the book, *Models to Code*.

These two domains are integrated together and `bosal` is used to supply a minimal `main()` function required to create an executable program. The tests are accomplished using Tcl and the `tcltest` package. The test program starts the execution of the integrated domains and then connects a socket to the localhost port used by `bosal` test harnesses for communications. Commands sent to the test harness are simple ASCII records with minimal syntax. Responses are retrieved from the same socket. There are a number of Tcl commands defined in the test program that are used for the communicating to the test harness. These are discussed below.

Null Command

```
<<bosal harness tests>>=
test null-1.0 {
    Null command test
} -setup {
} -cleanup {
} -body {
    harnessCmdResp null
} -result {}
```

```
<<bosal harness tests>>=
test null-2.0 {
    Null command with wrong argument count
} -setup {
} -cleanup {
} -body {
    harnessCmdResp null foo
} -result {wrong # of arguments, 2: expected, null} -returnCodes error
```

Version Command

```
<<bosal harness tests>>=
test version-1.0 {
    Version command test
} -setup {
} -cleanup {
} -body {
    set version [harnessCmdResp version]
    log::info "testing harness created by bosal $version"
    return $version
} -result {1.2.2}
```

Query Domains

```
<<bosal harness tests>>=
test query-domains-1.0 {
    query domains successfully
} -setup {
} -cleanup {
```

```
} -body {
    harnessCmdResp query domains
} -result {lube sio}
```

```
<<bosal harness tests>>=
test query-domains-2.0 {
    query domains with too many arguments
} -setup {
} -cleanup {
} -body {
    harnessCmdResp query domains foo
} -result {wrong # of arguments, 3: expected, "query domains"} -returnCodes error
```

Query Domain Operation Parameters

```
<<bosal harness tests>>=
test query-doparams-1.0 {
    No parameters
} -setup {
} -cleanup {
} -body {
    harnessCmdResp query doparams lube init
} -result {}
```

```
<<bosal harness tests>>=
test query-doparams-2.0 {
    Some parameters
} -setup {
} -cleanup {
} -body {
    harnessCmdResp query doparams sio Read_point
} -result {pid SioPointID_t}
```

```
<<bosal harness tests>>=
test query-doparams-3.0 {
    Bad domain
} -setup {
} -cleanup {
} -body {
    harnessCmdResp query doparams foo Read_point
} -result {unknown domain, "foo"} -returnCodes error
```

```
<<bosal harness tests>>=
test query-doparams-3.1 {
    Bad domain operation name
} -setup {
} -cleanup {
} -body {
    harnessCmdResp query doparams sio foo
} -result {unknown operation, "foo"} -returnCodes error
```

Query Classes

```
<<bosal harness tests>>=
test query-classes-1.0 {
    query lube classes
} -setup {
} -cleanup {
} -body {
    harnessCmdResp query classes lube
} -result {Lubrication_Schedule Injector_Design Injector Autocycle_Session Machinery  ↩
    Reservoir}
```

```
<<bosal harness tests>>=
test query-classes-2.0 {
    query classes -- unknown domain
} -setup {
} -cleanup {
} -body {
    harnessCmdResp query classes foo
} -result {unknown domain, "foo"} -returnCodes error
```

```
<<bosal harness tests>>=
test query-classes-2.1 {
    query classes -- wrong number of arguments
} -setup {
} -cleanup {
} -body {
    harnessCmdResp query classes foo bar
} -result {wrong # of arguments, 4: expected, "query classes <domain>"}\
-returnCodes error
```

Query Attributes

```
<<bosal harness tests>>=
test query-attributes-1.0 {
    Query attributes of the Injector_Design class in the lube domain
} -setup {
} -cleanup {
} -body {
    harnessCmdResp query attributes lube Injector_Design
} -result {Delivery_window Good_injection_duration Max_dissipation_pressure  ↩
    Max_system_pressure Min_delivery_pressure Model}
```

```
<<bosal harness tests>>=
test query-attributes-2.0 {
    query attributes -- too many arguments
} -setup {
} -cleanup {
} -body {
    harnessCmdResp query attributes lube Injector_Design foo
} -result {wrong # of arguments, 5: expected, "query attributes <domain> <class>"}\
-returnCodes error
```

```
<<bosal harness tests>>=
test query-attributes-2.1 {
    query attributes -- too few arguments
} -setup {
} -cleanup {
} -body {
```

```
    harnessCmdResp query attributes lube
} -result {wrong # of arguments, 3: expected, "query attributes <domain> <class>"}\
-returnCodes error
```

```
<<bosal harness tests>>=
test query-attributes-2.2 {
    query attributes -- bad domain
} -setup {
} -cleanup {
} -body {
    harnessCmdResp query attributes foo bar
} -result {unknown domain, "foo"} -returnCodes error
```

```
<<bosal harness tests>>=
test query-attributes-2.3 {
    query attributes -- bad class
} -setup {
} -cleanup {
} -body {
    harnessCmdResp query attributes lube foo
} -result {unknown class, "foo"} -returnCodes error
```

Query Instances

```
<<bosal harness tests>>=
test query-instances-1.0 {
    query instances of Injector in lube domain
} -setup {
} -cleanup {
} -body {
    harnessCmdResp query instances lube Injector
} -result {total 3 named {in1 0 in2 1 in3 2}}
```

Query States

```
<<bosal harness tests>>=
test query-states-1.0 {
    query states of Reservoir class in lube domain
} -setup {
} -cleanup {
} -body {
    harnessCmdResp query states lube Reservoir
} -result {NORMAL LOW VERY_LOW EMPTY}
```

```
<<bosal harness tests>>=
test query-states-2.0 {
    query states of class with no state model
} -setup {
} -cleanup {
} -body {
    harnessCmdResp query states lube Injector_Design
} -result {}
```

Query Events

```
<<bosal harness tests>>=
test query-events-1.0 {
    query events of the Reservoir class in the lube domain
} -setup {
} -cleanup {
} -body {
    harnessCmdResp query events lube Reservoir
} -result {Low_injection_pressure Low_lube_level Normal_lube_level Too_many_low_lube_cycles ↩
    }
```

Query Event Parameters

```
<<bosal harness tests>>=
test query-evparams-1.0 {
    query event parameters of the Reservoir class in the lube domain
} -setup {
} -cleanup {
} -body {
    harnessCmdResp query evparams lube Reservoir Low_injection_pressure
} -result {}
```

```
<<bosal harness tests>>=
test query-evparams-2.0 {
    query event parameters of the Range_Limitation class in the sio domain
} -setup {
} -cleanup {
} -body {
    harnessCmdResp query evparams sio Range_Limitation New_point
} -result {pointValue SioPointValue_t}
```

Query Current State

```
<<bosal harness tests>>=
test query-current-1.0 {
    query current state of an Injector instance in the lube domain
} -setup {
} -cleanup {
} -body {
    harnessCmdResp query current lube Injector in1
} -result {SLEEPING}
```

```
<<bosal harness tests>>=
test query-current-2.0 {
    query current state for class with no state model
} -setup {
} -cleanup {
} -body {
    harnessCmdResp query current lube Injector_Design ihn4
} -result {Class does not have a state model}\
-returnCodes error
```

Domain Operations

```
<<bosal harness tests>>=
test domainop-1.0 {
    Unknown domain
} -setup {
} -cleanup {
} -body {
    harnessCmdResp domainop foo init
} -result {unknown domain, "foo"} -returnCodes error
```

```
<<bosal harness tests>>=
test domainop-1.1 {
    No operation name
} -setup {
} -cleanup {
} -body {
    harnessCmdResp domainop sio
} -result {wrong # of arguments: 2: expected, domainop <domain> <operation> ?<arg1> <arg2>  ↩
    ...?}\
-returnCodes error
```

```
<<bosal harness tests>>=
test domainop-1.2 {
    Unknown operations
} -setup {
} -cleanup {
} -body {
    harnessCmdResp domainop sio foo
} -result {unknown operation, "foo"} -returnCodes error
```

```
<<bosal harness tests>>=
test domainop-1.3 {
    Missing operation argument
} -setup {
} -cleanup {
} -body {
    harnessCmdResp domainop lube Suspend_Autocycle_Session
} -result {wrong # arguments: got 0, expected 1} -returnCodes error
```

```
<<bosal harness tests>>=
test domainop-1.4 {
    Bad operation argument
} -setup {
} -cleanup {
} -body {
    harnessCmdResp domainop lube Suspend_Autocycle_Session foo
} -result {bad parameter: "foo"} -returnCodes error
```

```
<<bosal harness tests>>=
test domainop-2.0 {
    Initialize lube domain
} -setup {
} -cleanup {
} -body {
    harnessCmdResp eloop halt
    harnessCmdResp domainop lube init
} -result {}
```

```
<<bosal harness tests>>=
test domainop-2.1 {
    Initialize sio domain
} -setup {
} -cleanup {
} -body {
    harnessCmdResp domainop sio init
} -result {}
```

Read Attribute

```
<<bosal harness tests>>=
test read-1.0 {
    Read a single attribute
} -setup {
} -cleanup {
} -body {
    set result [harnessCmdResp read lube Injector_Design ihn4 Model]
    dict get $result Model
} -result {IHN4}
```

```
<<bosal harness tests>>=
test read-2.0 {
    Read all attributes
} -setup {
} -cleanup {
} -body {
    set result [harnessCmdResp read lube Injector_Design ihn4]
    set nattrs [dict size $result]
    set model [dict get $result Model]
    expr {$nattrs == 6 && $model eq "IHN4"}
} -result {1}
```

```
<<bosal harness tests>>=
test read-3.0 {
    Unknown attribute
} -setup {
} -cleanup {
} -body {
    harnessCmdResp read lube Injector_Design ihn4 foo
} -result {unknown attribute, "foo"} -returnCodes error
```

```
<<bosal harness tests>>=
test read-3.1 {
    Unknown attribute in a group
} -setup {
} -cleanup {
} -body {
    harnessCmdResp read lube Injector_Design ihn4 Model foo
} -result {unknown attribute, "foo"} -returnCodes error
```

Update Attribute

```
<<bosal harness tests>>=
test update-1.0 {
    Update a single attribute
} -setup {
    set oldwindow [dict get\
        [harnessCmdResp read lube Injector_Design ihn4 Delivery_window]\
        Delivery_window]
} -cleanup {
    harnessCmdResp update lube Injector_Design ihn4 Delivery_window $oldwindow
} -body {
    harnessCmdResp update lube Injector_Design ihn4 Delivery_window 42
    set result [harnessCmdResp read lube Injector_Design ihn4 Delivery_window]
    dict get $result Delivery_window
} -result {42}
```

State Machine Traces

```
<<bosal harness tests>>=
test trace-1.0 {
    Get trace status
} -setup {
} -cleanup {
} -body {
    harnessCmdResp trace
} -result {off}
```

```
<<bosal harness tests>>=
test trace-2.0 {
    Turn tracing on
} -setup {
} -cleanup {
    harnessCmdResp trace off
} -body {
    harnessCmdResp trace on
} -result {on}
```

```
<<bosal harness tests>>=
test trace-3.0 {
    Bad trace option
} -setup {
} -cleanup {
} -body {
    harnessCmdResp trace foo
} -result {unknown trace option, "foo": expected, "on | off"} -returnCodes error
```

```
<<bosal harness tests>>=
test trace-3.1 {
    Wrong number of trace arguments
} -setup {
} -cleanup {
} -body {
    harnessCmdResp trace on heavy
} -result {wrong # of arguments: 3: expected, "trace ?on | off?"}\
-returnCodes error
```

Run Event Loop

```
<<bosal harness tests>>=
test eloop-1.0 {
    Dispatch one event from the event loop
} -setup {
} -cleanup {
} -body {
    harnessCmdResp trace on
    for {set i 0} {$i < 10} {incr i} {
        harnessCmdResp eloop once
        set trace [waitForEventTrace type transition]
    }

    dict get $trace target
} -result {Reservoir.res2}
```

Instrumentation Traces

```
<<bosal harness tests>>=
test instr-1.0 {
    Instrumentation traces
} -setup {
    harnessCmdResp instr on
} -cleanup {
    harnessCmdResp instr off
} -body {
    harnessCmdResp domainop sio Read_point 0
    set match [waitForInstrTrace message *Continuous_Point_readPoint*]
    return [dict get $match message]
} -result {sio: Continuous_Point_readPoint:*} -match glob
```

Fatal Error Traces

```
<<bosal harness tests>>=
test fatal-1.0 {
    Fatal error traces
} -setup {
} -cleanup {
} -body {
    harnessCmdResp signal lube Injector in3 Good_injection
    catch {harnessCmdResp eloop once}
    set match [waitForFatalTrace message *]
    return [dict get $match message]
} -result {*CH*} -match glob
```

Signal Event

```
<<bosal harness tests>>=
test signal-1.0 {
    signal an event
} -setup {
    harnessCmdResp trace on
```

```
    set insts [harnessCmdResp query instances lube Reservoir]
    set res [lindex [dict get $insts named] 0]
    set res_state [harnessCmdResp query current lube Reservoir $res]
    log::debug "reservoir state = $res_state"
} -cleanup {
    harnessCmdResp trace off
} -body {
    harnessCmdResp signal lube Reservoir $res Low_lube_level
    harnessCmdResp eloop once
    set trace [waitForEventTrace type transition event Low_lube_level]
    dict get $trace newstate
} -result {LOW}
```

```
<<bosal harness tests>>=
test signal-2.0 {
    signal an event, unknown instance
} -setup {
} -cleanup {
} -body {
    harnessCmdResp signal lube Reservoir foo Low_lube_level
} -result {unknown instance, "foo"} -returnCodes error
```

```
<<bosal harness tests>>=
test signal-2.1 {
    signal an event, unknown event
} -setup {
} -cleanup {
} -body {
    harnessCmdResp signal lube Reservoir res1 foo
} -result {unknown event, "foo"} -returnCodes error
```

```
<<bosal harness tests>>=
test signal-2.2 {
    signal an event, bad parameter count
} -setup {
} -cleanup {
} -body {
    harnessCmdResp signal lube Reservoir res1 Low_lube_level 100
} -result {wrong # arguments: got 1, expected 0} -returnCodes error
```

Delayed Signal

```
<<bosal harness tests>>=
test delaysignal-1.0 {
    delayed signaling
} -setup {
    harnessCmdResp trace on
} -cleanup {
    harnessCmdResp trace off
} -body {
    harnessCmdResp delaysignal lube Reservoir res2 1000 Low_lube_level
    set start [clock milliseconds]
    harnessCmdResp eloop toc wait

    set trace [waitForEventTrace type transition event Low_lube_level]
    set end [clock milliseconds]

    set wait [expr {$end - $start}]
```

```
    log::debug "measured $wait delay"
    expr {$wait >= 1000 && $wait < 1100}
} -result {1}
```

Cancel Delayed Signal

```
<<bosal harness tests>>=
test cancel-1.0 {
    cancel a delayed signal
} -setup {
    harnessCmdResp trace on
} -cleanup {
    harnessCmdResp trace off
} -body {
    harnessCmdResp delaysignal lube Reservoir res2 1000 Normal_lube_level
    after 200
    harnessCmdResp cancel lube Reservoir res2 Normal_lube_level
    waitForEventTrace type transition event Normal_lube_level
} -result {timed out on receiving event traces} -returnCodes error
```

Time Remaining for a Delayed Signal

```
<<bosal harness tests>>=
test remaining-1.0 {
} -setup {
} -cleanup {
    harnessCmdResp cancel lube Reservoir res2 Normal_lube_level
} -body {
    harnessCmdResp delaysignal lube Reservoir res2 1000 Normal_lube_level
    after 200
    set remaining [harnessCmdResp remaining lube Reservoir res2\
            Normal_lube_level]
    log::debug "remaining time = $remaining ms"
    expr {$remaining > 600 && $remaining <= 800}
} -result {1}
```

Create an Instance

```
<<bosal harness tests>>=
test create-1.0 {
    create an instance of Reservoir
} -setup {
} -cleanup {
    harnessCmdResp delete sio Conversion $inst
} -body {
    set inst [harnessCmdResp create sio Conversion]
    log::debug "created instance, \"$inst\""
    return $inst
} -result {[0-9]} -match regexp
```

Create an Instance Asynchronously

```
<<bosal harness tests>>=
test createasync-1.0 {
    create instance asynchronously
} -setup {
    harnessCmdResp trace on
} -cleanup {
    harnessCmdResp trace off
} -body {
    harnessCmdResp signal lube Autocycle_Session acs3 Change_schedule Test2
    harnessCmdResp eloop toc
    set trace [waitForEventTrace type creation]
    waitForEventTrace type transition event Created
    return [dict get $trace event]
} -result {New_session}
```

Code Organization

Test Utility Procedures

```
<<test utility procs>>=
proc startHarnessExec {logfile} {
    exec ./ls_harness > $logfile &
    after 500
}
```

```
<<test utility procs>>=
proc setupHarnessComm {{port 3906}} {
    log::info "connecting to localhost:$port"
    variable hchan [socket localhost $port]
    chan configure $hchan -blocking true -buffering line
    chan event $hchan readable\
        [list [namespace current]::handleHarnessInput $hchan]
}
```

```
<<test utility procs>>=
proc cleanupHarnessComm {} {
    variable hchan
    catch {chan close $hchan}
}
```

```
<<test utility procs>>=
proc putsToHarness {cmd} {
    variable hchan
    puts $hchan $cmd
}
```

```
<<test utility procs>>=
proc handleHarnessInput {chanId} {
    set llen [chan gets $chanId line]
    if {$llen == -1} {
        if {[chan eof $chanId]} {
            log::warn "EOF on harness input -- closing \"$chanId\""
            chan close $chanId
        } elseif {[chan blocked $chanId]} {
```

```
            log::notice "partial line received"
        }
        return
    }

    if {$llen != 0} {
        log::debug "harness response: \"$line\""

        lassign $line resp_type resp_value

        switch -exact -- $resp_type {
            cmd {
                handleCmdResponse $resp_value
            }
            trace {
                handleTraceResponse $resp_value
            }
            fatal {
                handleFatalResponse $resp_value
            }
            instr {
                handleInstrResponse $resp_value
            }
            default {
                error "unknown response type, \"$resp_value\""
            }
        }
    }
}
```

```
<<test utility variables>>=
variable cmdTimeout 3000
variable cmdPattern
variable cmdSyncVar {}
```

```
<<test utility procs>>=
proc harnessCmdResp {args} {
    variable cmdPattern
    variable cmdTimeout
    set cmdPattern [dict create\
        name [lindex $args 0]\
        timer [after $cmdTimeout [namespace code cmdTimeout]]
    ]
    putsToHarness $args

    set response [waitForCmdResponse]
    set result [dict get $response result]
    if {[dict get $response status] eq "error"} {
        error $result
    }
    return $result
}
```

```
<<test utility procs>>=
proc handleCmdResponse {respValue} {
    variable cmdPattern

    set expectedName [dict get $cmdPattern name]
    set recvdName [dict get $respValue name]

    if {[string match $expectedName $recvdName]} {
```

```
        signalCmdResponse $respValue
    } else {
        log::error "expected response to $expectedName command,\
                got response to $recvdName command"
        signalCmdResponse ERROR
    }
}
```

```
<<test utility procs>>=
proc cmdTimeout {} {
    set [namespace current]::cmdSyncVar TIMEOUT
}
```

```
<<test utility procs>>=
proc signalCmdResponse {value} {
    set [namespace current]::cmdSyncVar $value
}
```

```
<<test utility procs>>=
proc waitForCmdResponse {} {
    vwait [namespace current]::cmdSyncVar
    variable cmdPattern
    variable cmdSyncVar

    if {$cmdSyncVar eq "TIMEOUT"} {
        error "timeout for command, \"[dict get $cmdPattern name]\""
    } elseif {$cmdSyncVar eq "FATAL"} {
        error "fatal error while executing, \"[dict get $cmdPattern name]\""
    } else {
        after cancel [dict get $cmdPattern timer]
    }

    if {[dict get $cmdSyncVar name] ne [dict get $cmdPattern name]} {
        error "expected response for command, \"[dict get $cmdPattern name]\",\
                got, \"[dict get $cmdSyncVar name]\""
    }
    return $cmdSyncVar
}
```

```
<<test utility variables>>=
variable traceSyncVar {}
variable tracesReceived [::struct::queue]
variable traceTimeout 3000
```

```
<<test utility procs>>=
proc handleTraceResponse {respValue} {
    variable tracesReceived

    $tracesReceived put $respValue

    set [namespace current]::traceSyncVar TRACE
}
```

```
<<test utility procs>>=
proc waitForEventTrace {args} {
    variable tracesReceived
    variable traceSyncVar
    variable traceTimeout
```

```
    set expect $args
    while {true} {
        while {[$tracesReceived size] != 0} {
            set actual [$tracesReceived get]

            set traceMatched 0
            dict for {key value} $expect {
                if {[dict exists $actual $key] &&\
                        [string match $value [dict get $actual $key]]} {
                    incr traceMatched
                }
            }
            if {$traceMatched == [dict size $expect]} {
                return $actual
            } else {
                log::notice "discarding trace, \"$actual\":\
                        failed to match, \"$expect\""
            }
        }

        set timer [after $traceTimeout [namespace code traceTimeout]]
        vwait [namespace current]::traceSyncVar
        if {$traceSyncVar eq "TIMEOUT"} {
            error "timed out on receiving event traces"
        } else {
            after cancel $timer
        }
    }
}
```

```
<<test utility procs>>=
proc traceTimeout {} {
    set [namespace current]::traceSyncVar TIMEOUT
}
```

```
<<test utility variables>>=
variable instrSyncVar {}
variable instrReceived [::struct::queue]
variable instrTimeout 3000
```

```
<<test utility procs>>=
proc handleInstrResponse {respValue} {
    variable instrReceived

    $instrReceived put $respValue

    set [namespace current]::instrSyncVar INSTR
}
```

```
<<test utility procs>>=
proc waitForInstrTrace {args} {
    variable instrReceived
    variable instrSyncVar
    variable instrTimeout

    set expect $args
    while {true} {
        while {[$instrReceived size] != 0} {
            set actual [$instrReceived get]
```

```
            set instrMatched 0
            dict for {key value} $expect {
                if {[dict exists $actual $key] &&\
                        [string match $value [dict get $actual $key]]} {
                    incr instrMatched
                }
            }
            if {$instrMatched == [dict size $expect]} {
                return $actual
            } else {
                log::notice "discarding instr, \"$actual\":\
                        failed to match, \"$expect\""
            }
        }

        set timer [after $instrTimeout [namespace code instrTimeout]]
        vwait [namespace current]::instrSyncVar
        if {$instrSyncVar eq "TIMEOUT"} {
            error "timed out on receiving instrumentation traces"
        } elseif {$instrSyncVar eq "FATAL"} {
            error "fatal error while waiting for instrumentation trace"
        } else {
            after cancel $timer
        }
    }
}
```

```
<<test utility procs>>=
proc instrTimeout {} {
    set [namespace current]::instrSyncVar TIMEOUT
}
```

```
<<test utility variables>>=
variable fatalSyncVar {}
variable fatalReceived [::struct::queue]
variable fatalTimeout 3000
```

```
<<test utility procs>>=
proc handleFatalResponse {respValue} {
    variable fatalReceived

    $fatalReceived put $respValue

    set [namespace current]::fatalSyncVar FATAL
    set [namespace current]::cmdSyncVar FATAL
    set [namespace current]::instrSyncVar FATAL
}
```

```
<<test utility procs>>=
proc waitForFatalTrace {args} {
    variable fatalReceived
    variable fatalSyncVar
    variable fatalTimeout

    set expect $args
    while {true} {
        while {[$fatalReceived size] != 0} {
            set actual [$fatalReceived get]

            set fatalMatched 0
```

```
            dict for {key value} $expect {
                if {[dict exists $actual $key] &&\
                        [string match $value [dict get $actual $key]]} {
                    incr fatalMatched
                }
            }
            if {$fatalMatched == [dict size $expect]} {
                return $actual
            } else {
                log::notice "discarding fatal, \"$actual\":\
                        failed to match, \"$expect\""
            }
        }

        set timer [after $fatalTimeout [namespace code fatalTimeout]]
        vwait [namespace current]::fatalSyncVar
        if {$fatalSyncVar eq "TIMEOUT"} {
            error "timed out on receiving fatal error response"
        } else {
            after cancel $timer
        }
    }
}
```

```
<<test utility procs>>=
proc fatalTimeout {} {
    set [namespace current]::fatalSyncVar TIMEOUT
}
```

Test Script

```
<<bosal-test.test>>=
#!/usr/bin/env tclsh
# DO NOT EDIT THIS FILE!
# THIS FILE IS AUTOMATICALLY GENERATED FROM A LITERATE PROGRAM SOURCE FILE.
#
<<copyright info>>

package require Tcl 8.6
package require cmdline
package require logger
package require struct::queue

set optlist {
    {level.arg warn {Logging level}}
    {nostart {Do not start harness executable}}
    {port.arg 3906 {TCP port for harness communications}}
    {log.arg {test.log} {Log file for harness output}}
}
array set options [::cmdline::getKnownOptions argv $optlist]

logger::setlevel $options(level)

package require tcltest
eval tcltest::configure $argv

namespace eval ::bosal::test {
    <<test utility variables>>
    <<test utility procs>>
}
```

```
namespace eval ::bosal::test {
    namespace import ::tcltest::*
    ::logger::initNamespace [namespace current] $::options(level)

    if {!$::options(nostart)} {
        startHarnessExec $::options(log)
    }
    setupHarnessComm $::options(port)

    <<bosal harness tests>>

    cleanupHarnessComm
    cleanupTests
}
```

Copyright Information

```
<<copyright info>>=
# This software is copyrighted 2017 - 2018 by G. Andrew Mangogna.
# The following terms apply to all files associated with the software unless
# explicitly disclaimed in individual files.
#
# The authors hereby grant permission to use, copy, modify, distribute,
# and license this software and its documentation for any purpose, provided
# that existing copyright notices are retained in all copies and that this
# notice is included verbatim in any distributions. No written agreement,
# license, or royalty fee is required for any of the authorized uses.
# Modifications to this software may be copyrighted by their authors and
# need not follow the licensing terms described here, provided that the
# new terms are clearly indicated on the first page of each file where
# they apply.
#
# IN NO EVENT SHALL THE AUTHORS OR DISTRIBUTORS BE LIABLE TO ANY PARTY FOR
# DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING
# OUT OF THE USE OF THIS SOFTWARE, ITS DOCUMENTATION, OR ANY DERIVATIVES
# THEREOF, EVEN IF THE AUTHORS HAVE BEEN ADVISED OF THE POSSIBILITY OF
# SUCH DAMAGE.
#
# THE AUTHORS AND DISTRIBUTORS SPECIFICALLY DISCLAIM ANY WARRANTIES,
# INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY,
# FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT.  THIS SOFTWARE
# IS PROVIDED ON AN "AS IS" BASIS, AND THE AUTHORS AND DISTRIBUTORS HAVE
# NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS,
# OR MODIFICATIONS.
#
# GOVERNMENT USE: If you are acquiring this software on behalf of the
# U.S. government, the Government shall have only "Restricted Rights"
# in the software and related documentation as defined in the Federal
# Acquisition Regulations (FARs) in Clause 52.227.19 (c) (2).  If you
# are acquiring the software on behalf of the Department of Defense,
# the software shall be classified as "Commercial Computer Software"
# and the Government shall have only "Restricted Rights" as defined in
# Clause 252.227-7013 (c) (1) of DFARs.  Notwithstanding the foregoing,
# the authors grant the U.S. Government and others acting in its behalf
# permission to use and distribute the software in accordance with the
# terms specified in this license.
```

Index