```
#!/bin/ash
# takes no args

#script_header_installed_dir="/usr/sbin"
script_header_installed_dir=/home/joe/myUtilities/dev-util/script_header_joetoo
# source script_header
header="${script_header_installed_dir%/}/script_header_joetoo"
if [ -f "$header" ]; then . "$header"; else printf '%s' "failed to source header; cannot
continue"; exit 1; fi

#-----[ variables ]-----------------------------------------------------

# note: all of these (including logFile and verbosity) get them anulled; need to explicit
ly reassign below
varlist="PN BUILD user logFile status_file config_dir verbosity"
varlist="${varlist} bool.INTERACTIVE bool.RESUME num_cmds starting_step stopping_step"

#-----[ functions ]-----------------------------------------------------

configure() {
  separator "$PN" "(configuring)"
  _c_FLAGGED=$FALSE

  # sets all variables named in varlist to null/$FALSE/No
  initialize_vars $varlist
  message "done initialization"

  # tier 0 - prerequisites (note: can't log until logging is set up below
  # initialize package name for this script
  message_n "assigning PN"
  PN=$(basename "$0")
  handle_result $? "$PN" '' || _c_FLAGGED=$TRUE

  # identify and assign user name
  message_n "assigning user"
  user=$(whoami)
  handle_result $? "$user" '' || _c_FLAGGED=$TRUE

  # assign and/or check logFile (may have been set by config
  # (do this after $user is assigned but before starting logging
  d_message "draft logFile name: /var/log/${PN}.log" 2
  message_n "assigninging logFile"
  if [ -z "$logFile" ]; then
    logFile="/var/log/${PN}.log"
    handle_result $? "now assigned: $logFile" "error assigning logFile" || _c_FLAGGED=$TR
UE
  else
    printf '%b' " (${BYon}already assigned: ${Gon}$logFile${Boff})"
    right_status 0
  fi
  # if it exists already, rotate it (with sudo, in a subshell)
  if [ -f "$logFile" ]; then
    # shift existing archives (delete the oldest, move others up)
    # use sudo sh -c "..." so we only need to call sudo once if needed for $user
    # (note: with " outside, the ' inside is just a literal single-quote
    #  char - interpreted AFTER expansion)
    message_n "rotating existing logFile(s)"
    sudo sh -c "
      [ -f '${logFile}.2' ] && mv -f '${logFile}.2' '${logFile}.3'
      [ -f '${logFile}.1' ] && mv -f '${logFile}.1' '${logFile}.2'
      mv -f '$logFile' '${logFile}.1'
      touch '$logFile'
      chown '${user}:${user}' '$logFile'
    " >/dev/null 2>&1
    handle_result $? "rotated" "error rotating logFile" || _c_FLAGGED=$TRUE
  else
    # create it and set permissoins
    message_n "creating logFile"
    sudo sh -c "
      touch '$logFile'
```

```
      chown '${user}:${user}' '$logFile'
    " >/dev/null 2>&1
    handle_result $? "created" "error rotating logFile" || _c_FLAGGED=$TRUE
  fi

  # tier 1 - global configuration
  # validate package config directory
  log_message_n "assigning config_dir"
  config_dir="/etc/${PN}"
  log_handle_result $? "$config_dir" "error assigning config_dir" || _c_FLAGGED=$TRUE
  validate_dir "$config_dir"

  # BUILD="string" should be located in config_dir
  log_message_n "sourcing BUILD variable assignment"
  . "${config_dir}/BUILD"
  log_handle_result $? "$BUILD" '' || _c_FLAGGED=$TRUE

  # initialize INTERACTIVE - can be overridden by sourced config below (and cmdline)
  log_message_n "assigning default INTERACTIVE"
  INTERACTIVE=$TRUE
  log_handle_result $? "$INTERACTIVE" '' || _c_FLAGGED=$TRUE

  # initialize RESUME - can be overridden by sourced config below (and cmdline)
  log_message_n "assigning default RESUME"
  RESUME=$FALSE
  log_handle_result $? "$RESUME" '' || _c_FLAGGED=$TRUE

  # if the package has a config file at /etc/$PN or in customization_root configure it he
re
  log_message "looking for config file(s)"
  validate_dir "/etc/${PN}"
  for _c_file in $(find "/etc/${PN}" -name "*.conf" -type f); do
    log_message_n "sourcing $_c_file"
    . "$_c_file"; log_handle_result $? || _c_FLAGGED=$TRUE
  done

  # tier 2 - local configuration
  # assign customization_root
  log_message_n "assigning customization_root variable"
  customization_root="${HOME}/.config/${PN}/"
  log_handle_result $? "$customization_root" "error assigning customization_root" || _c_F
LAGGED=$TRUE

  # assign and validate status_file and customization_root
  log_message_n "assigning status_file variable"
  status_file="${customization_root%/}/${PN}.status"
  log_handle_result $? "$status_file" '' || _c_FLAGGED=$TRUE
  validate_file "$status_file"   # note this also validates the directory

  # find (validates) *_example_local.cmdline_arguments; assign first match
  log_message_n "looking for local_process_cmdline_arguments"
  local_process_cmdline_arguments=$( \
    find "${customization_root%/}/" \
      -name "[[:alnum:]]*_example_local.cmdline_arguments" | \
    head -n 1 )
  log_handle_result $? "$local_process_cmdline_arguments" "error finding local_process_cm
dline_arguments" || _c_FLAGGED=$TRUE

  # set default verbosity if not set externally
  log_message_n "maybe assigning verbosity"
  if [ -z "$verbosity" ] ; then
    verbosity=3
    log_handle_result $? "now set: $verbosity" '' || _c_FLAGGED=$TRUE
  else
    log_handle_result $TRUE "already set: $verbosity" ''
  fi

  [ "$_c_FLAGGED" ] && ret=1
  unset -v _c_FLAGGED _c_file
  return $ret
```

```
}
# @usage initialize_variables || die "failed to initialize_variables"
# @rules PN is package name and should be assigned = $(basename "$0") to identify the cal
ling script
# @rules customization_root should contain BUILD
# @rules customization_root may contain XXXX_local.usage extension file
# @rules customization_root may contain XXXX_local.usage extension file
# @rules POSIX assignment 'BUILD="string"' should reside in "${customization_root%/}/BUIL
D"

initialize_command_sequence() # (POSIX) initialize command sequence for joetoo cli framew
ork
{ ret=0
  _ics_FLAGGED=$FALSE
  log_message_n "creating tempfile for command sequnce"
  cmd_seq_file=$(mktemp /tmp/${PN}.XXXXXX)
  log_handle_result $? "$cmd_seq_file" '' || _ics_FLAGGED=$TRUE

  log_message_n "setting trap to remove tempfile on EXIT"
  trap 'rm -f "$cmd_seq_file"' EXIT

  log_message_n "initializing command sequence"
  num_cmds=0   # global variable holding the length of the sequence
# heredoc needs to be on the left edge; no leading whitespace
# initialize/clear the command sequence file
while IFS= read -r _ics_entry; do
      [ -z "$_ics_entry" ] && continue
      # write directly to the file instead of using  ' set -- "$@" '
      printf '%s\n' "$_ics_entry" >> "$cmd_seq_file"
      num_cmds=$(( num_cmds + 1 ))
done <<EOF
printf '%s\n' "this is line 1"${US}print example line number one
printf '%s\n' "this is line 2"${US}print example line number two
printf '\n'${US}print a newline
printf '%s\n' "this is line 3 (after a blank line)"${US}print line number 3
test_function "${BGon}hello${Boff}"${US}run test_function
EOF
  log_handle_result $? "num_cmds: $num_cmds" '' || _ics_FLAGGED=$TRUE

  log_message_n "initailizing starting_step"
  starting_step=1
  log_handle_result $? "$starting_step" '' || _ics_FLAGGED=$TRUE

  log_message_n "initializing stopping step"
  stopping_step="$num_cmds"
  log_handle_result $? "$stopping_step" '' || _ics_FLAGGED=$TRUE

  [ "$_ics_FLAGGED" ] && ret=1
  unset -v _ics_entry
  return $ret
}
# @usage initialize_command_sequence || die "failed to initialize_command_sequence"
# @vars sets global num_cmds, cmd_seq_file
# @vars initializes global starting_step=1, stopping_step=$num_cmds
# @rule command sequency entry format 'command_string${US}description_string'
# @cont where: command_string is POSIX executable
# @cont and:   ${US} is teletype ero unit separator char \037 pre-cooked with printf
# @cont and:   description_string is a human-readable description of the command

usage()                # explain default usage; mod with local "usage module"
{
  N=$(( ${num_cmds} -1 ))
  separator "${PN}-${BUILD}" "(usage)"
  log_E_message "${BRon}Usage: ${BGon}${PN} [-[options]] ${Boff}"
  log_message "${BYon}valid commandline options --${Boff}"
  log_message "  -i | --interactive......: run supervised; confirm execution of each step
"
  log_message "  -n | --noninteractive...: run un-supervised; automatically do each step"
  log_message "  -s | --status....: return status (next step, step_number)"
  log_message "  -r | --resume....: resume at next step in status_file"
```

```
  log_message "  -v | --verbose...: increase verbosity"
  log_message "  -q | --quiet.....: decrease verbosity"
  log_message "  -[0-${N}]..........: save N to status file and resume at step N"
  log_message "  ${BYon}*${Boff} Single-character options may be combined."
  log_message "    e.g. ${BGon}${PN} --verbose -nqr8${Boff} would resume non-interactivel
y"
  log_message "    (automatic, unsupervised) at step 8 with normal verbosity"
  log_message "${BMon}Caveat:${Boff}"
  log_message "   -i (interactive/supervised) is on by default"
  log_message "   -r (resume) sets starting_step to # in [ $status_file ]"
  log_message "   -[0-${N}] sets starting_step (default 0 otherwise)"
  # source user-script specific usage-module which should be built in the same format
  log_message "${BMon}additional ${customization_root} - commandline options:${Boff}"
  [ -f ${customization_root%/}/local.usage ] && source ${customization_root%/}/local.usag
e
  printf '\n'
  log_message "${BYon}${PN} workflow sequence (steps):${Boff}"
  for _u_step in $(seq 1 $num_cmds); do
    _u_entry=$(sed -n "${_u_step}p" "$cmd_seq_file") # extract the specific line correspo
nding to the current step counter
    _u_desc="${_u_entry#*${US}}"                    # human readable description
    printf '    %b[%b%02d%b]%b: ' "${BBon}" "${Mon}" "$_u_step" "${BBon}" "${Boff}"
    printf '%b%s%b\n' "${Con}" "${_u_desc}" "${Boff}"
  done

  unset -v _u_step _u_entry _u_desc
  die "dying; process [$$]"
}

validate_dir()  # validate or create a directory $1
{   ret=0
    [ $# -ne 1 ] && { log_E_message "Error: must specify dir_to_validate" ; return 1 ; }
    _vd_dir_to_validate=$1
    log_message_n "validating dir"
    if [ -d "${_vd_dir_to_validate}" ] ; then
        printf '%b%s%b' "${BGon}" " (valid) ... " "${Boff}"
        log_right_status $TRUE
    else
        printf '%b%s%b' "${BYon}" " (creating) ... " "${Boff}"
        # use sudo in case current user is not root
        sudo mkdir -p "${_vd_dir_to_validate}"
        log_handle_result $? "created" "failed to create $_vd_dir_to_validate" || return
1
    fi
    [ -d "${_vd_dir_to_validate}" ] ; ret=$?     # final validation
    unset -v _vd_dir_to_validate
    return $ret
}

validate_file()  # validate or create command sequence status file $1
{   ret=0
    [ $# -ne 1 ] && { log_E_message "Error: must specify file_to_validate" ; return 1 ; }
    _vf_file_to_validate=$1
    _vf_dir_to_validate=$(dirname "${_vf_file_to_validate}")
    validate_dir "$_vf_dir_to_validate"
    log_message_n "validating file"
    if [ -f "${_vf_file_to_validate}" ] ; then
        printf '%b%s%b' "${BGon}" " (valid)" "${Boff}"
        log_right_status $TRUE
    else
        # use sudo in case current user is not root
        sudo sh -c "
            printf '%b%s%b' '${BYon}' ' (creating)' '${Boff}'
            printf '%s' '1' > '${_vf_file_to_validate}'  # 1-indexed command sequence
            chown '${user}:${user}' '${_vf_file_to_validate}'
        " >/dev/null 2>&1
        log_handle_result $? "created" "failed to create ${_vf_file_to_validate}" || retu
rn 1
    fi
    # final validation
```

```
    [ -f "${_vf_file_to_validate}" ] ; ret=$?
    unset -v _vf_file_to_validate _vf_dir_to_validate
    return $ret
}


process_cmdline() {
    separator "$PN" "(process_cmdline)"

    # local helper (defined only inside this scope) to get status
    _pc_emit_status() {
        if [ -f "$status_file" ]; then
            read _pc_val < "$status_file"
            log_message "${BWon}Status: Step $(( _pc_val - 1 )) complete; next is [ ${BMo
n}${_pc_val:-1}${BWon} ]${Boff}"
        else
            log_message "${BWon}Status: No status file; sequence not yet started${Boff}"
        fi
        exit 0
    }

    # baseline optstring (standard flags only) (see getopts --help)
    _pc_optstring="insrvq"
    d_log_message "_pc_optstring: $_pc_optstring" 5
    d_log_message "\$@: $@" 5
    while [ $# -gt 0 ]; do
        d_log_message "\$1: $1" 5
        case "$1" in

            # tier 1 - standard long options
            --status)  # POSIX math: report step-1 as completed
                if [ -f "$status_file" ]; then read _pc_val < "$status_file"
                    log_message "${BWon}Status: Step $(( _pc_val - 1 )) complete; next is
 [ ${BMon}${_pc_val}${BWon} ]${Boff}"
                else log_message "${BWon}Status: No status file; sequence not yet started
${Boff}"; fi; exit 0 ;;
            --resume) RESUME=$TRUE; if isint "$2"; then starting_step="$2"; shift 2
                else  read starting_step < "$status_file" 2>/dev/null || starting_step=1;
 shift; fi
                d_log_message "set RESUME = $(status_color $RESUME)$(TrueFalse RESUME)${B
off}; starting_step: [${Mon}starting_step${Boff}]" 3 ;;
            --interactive)    INTERACTIVE=$TRUE; shift
                d_log_message "set INTERACTIVE = $(status_color $INTERACTIVE)$(TrueFalse
$INTERACTIVE)${Boff};" 3 ;;
            --noninteractive) INTERACTIVE=$FALSE; shift
                d_log_message "set INTERACTIVE = $(status_color $INTERACTIVE)$(TrueFalse
$INTERACTIVE)${Boff};" 3 ;;
            --verbose)        verbosity=$(( verbosity + 1 )); shift
                d_log_message "set verbosity: [${Mon}$verbosity${Boff}]" 3 ;;
            --quiet)          verbosity=$(( verbosity - 1 )); shift
                d_log_message "set verbosity: [${Mon}$verbosity${Boff}]" 3 ;;

            # tier 2 - clustered shorts & embedded numbers (e.g. -iv12)
            -*) # remove '-', and "peel" digits from the cluster (e.g., "iv24" -> "24")
                # ( tr: -d deletes; -c '0-9' specifies the complement of numbers )
                _pc_cluster=${1#-}; _pc_num=$(printf '%s' "$_pc_cluster" | tr -d -c '0-9'
)
                if [ -n "$_pc_num" ]; then starting_step="$_pc_num"; RESUME=$TRUE; fi
                # use getopts to process the letters in the cluster (see getopts --help)
                # each iteration of "getopts $_pc_optstring _pc_opt" assigns the next
                # letter in whatever positional parameter OPTIND is pointing at
                # when it runs out of letters, it increments OPTIND
                # (but this loop shifts and resets OPTIND to 1)
                while getopts "$_pc_optstring" _pc_opt; do
                    d_log_message "_pc_opt: $_pc_opt" 5
                    case "$_pc_opt" in
                        i) INTERACTIVE=$TRUE
                            d_log_message "set INTERACTIVE = $(status_color $INTERACTIVE)
$(TrueFalse $INTERACTIVE)${Boff};" 3 ;;
                        n) INTERACTIVE=$FALSE
```

```
                            d_log_message "set INTERACTIVE = $(status_color $INTERACTIVE)
$(TrueFalse $INTERACTIVE)${Boff};" 3 ;;
                        v) [ "$verbosity" -lt 6 ] && verbosity=$(( verbosity + 1 ))
                            d_log_message "set verbosity: [${Mon}$verbosity${Boff}]" 3 ;;
                        q) [ "$verbosity" -gt 0 ] && verbosity=$(( verbosity - 1 ))
                            d_log_message "set verbosity: [${Mon}$verbosity${Boff}]" 3 ;;
                        r) RESUME=$TRUE
                            # read file only if no number was found in this cluster
                            if [ -z "$_pc_num" ]; then read starting_step < "$status_file
" 2>/dev/null || starting_step=1; fi
                            d_log_message "set RESUME = $(status_color $RESUME)$(TrueFals
e RESUME)${Boff}  starting_step: [${Mon}starting_step${Boff}]" 3 ;;
                        s) # re-use the status logic above (manually)
                           read _pc_val < "$status_file" 2>/dev/null || _pc_val=1
                           log_message "Next step: $_pc_val"; exit 0 ;;
                        *) # check local hook extension if it exists
                            if [ "$(command -v $local_process_cmdline_arguments)" ]; then
                                "$local_process_cmdline_arguments" "$_pc_opt" "$OPTARG"
                                _rs_result=$?
                            else
                                log_E_message "Invalid cmdline argument [${Mon}$1${Boff}]
"
                                usage
                            fi
                            # got here w/o getting shunted to usage - check return ffom l
ocal extension
                            # (code 0: success; 6: used an operand (need extra shift); ot
her: fail
                            [ "$_rs_result" -eq 6 ] && shift
                            ret="$_rs_result"
                            ;;
                    esac
                    d_log_message "just finished an inner loop iteration" 1
                    d_log_message "_pc_opt: $_pc_opt" 1
                    d_log_message "OPTIND: $OPTIND" 1
                    d_log_message "OPTARG: $OPTARG" 1
                    d_log_message "\$1: $1" 1
                    d_log_message "\$@: $@" 1
                done  # current positional parameter's letters have all been checked
                d_log_message "just exited inner loop with \$1 = $1" 1
                shift $(( OPTIND - 1 )) # shift to next word/arg
                OPTIND=1 ;;             # point at first word/arg again

            # tier 3 - positional numbers (e.g. just '12') ---
            [0-9]*) # if it is not a pure number (like 24j); treat as operand or error
                if isint "$1"; then starting_step="$1"; RESUME=$TRUE; shift
                else log_E_message "Invalid operand: $1"; usage; fi ;;
            *) shift ;;  # ignore all else
        esac
        d_log_message "just finished an outer loop iteration" 1
        d_log_message "_pc_opt: $_pc_opt" 1
        d_log_message "OPTIND: $OPTIND" 1
        d_log_message "OPTARG: $OPTARG" 1
        d_log_message "\$1: $1" 1
        d_log_message "\$@: $@" 1
    done  # done with $1 - each case above shifted at least 1, so drive on

    unset -v _pc_optstring _pc_cluster _pc_num _pc_val _pc_opt _pc_total_steps
    unset -f _pc_emit_status
    return $ret
}
# usage process_cmdline [args]
# args $@: command line arguments to be parsed
# vars INTERACTIVE, RESUME, starting_step, verbosity, status_file
# ret 0: success; exit 1: invalid operand or out of range
# deps isint, log_message, d_log_message, log_E_message, usage, tr, printf, read, cat
# rule  if $local_process_cmdline_arguments file exists,
# @cont it extends functionality local to the calling script
# rule starting_step is validated against global num_cmds
# rule last-instruction-wins precedence is enforced via linear while-loop;
```

```
# ex process_cmdline -iv24; process_cmdline --resume 5; process_cmdline -s

run_sequence() # (POSIX) run command sequence; $1=status_file, $2=start, $3=stop
{ ret=0
  # guard 1 - minimum arguments check (status, start, stop, + at least 1 cmd)
  [ $# -lt 3 ] && { log_E_message "usage: run_sequence <status_file> <start> <stop>"; ret
urn 1; }
  _rs_status_file="$1"           # assign the status file
  _rs_start="${2:-1}"            # guard 2 - start step (default to zero but ensure it is
 an integer in range)
  ! isint "$_rs_start" && { log_E_message "run_sequence: start_step '$2' must be an integ
er 1 - ${num_cmds}"; return 1; }
  if [ "$_rs_start" -lt 1 ] || [ "$_rs_start" -gt "$num_cmds" ]; then
     log_E_message "run_sequence: start_step '$2' must be an integer 1 - ${num_cmds}"; re
turn 1; fi
  _rs_stop="${3:-$num_cmds}" # guard 3 - stop step (default to num cmds but ensure it is
an integer in range)
  ! isint "$_rs_stop" && { log_E_message "run_sequence: stop_step '$3' must be '' or an i
nteger 1 - ${num_cmds}"; return 1; }
  if [ "$_rs_stop" -lt 1 ] || [ "$_rs_stop" -gt "$num_cmds" ]; then
    log_E_message "run_sequence: stop_step '$3' must be '' or an integer 1 - ${num_cmds}"
; return 1; fi
  _rs_step=1
  while [ "$_rs_step" -le "$_rs_stop" ]; do        # enforces upper bound
    if [ "$_rs_step" -lt "$_rs_start" ]; then
      _rs_step=$(( _rs_step + 1 )); continue; fi   # enforces lower bound (shift into ran
ge)
    # assign commadn and description from input, delimitted by unit separator ($US)
    _rs_entry=$(sed -n "${_rs_step}p" "$cmd_seq_file") # extract the specific line corres
ponding to the current step counter
    _rs_cmd="${_rs_entry%%${US}*}"  # the actual command
    _rs_desc="${_rs_entry#*${US}}"  # human readable description
    _rs_clean_entry=$(printf -- '%b' "$_rs_entry" | strip_ansi | _translate_escapes)
    _rs_clean_cmd=$(printf -- '%b' "$_rs_cmd" | strip_ansi | _translate_escapes)

    d_log_message "_rs_start: $_rs_start" 5
    d_log_message "_rs_stop: $_rs_stop" 5
    d_log_message "_rs_step: $_rs_step" 5
    d_log_message "_rs_clean_entry: $_rs_clean_entry" 5
    d_log_message "_rs_clean_cmd: $_rs_clean_cmd" 5
    d_log_message "_rs_desc: $_rs_desc" 5
    d_log_message "termwidth: $(termwidth)" 5

    # if INTERACTIVE, confirm before executing the step
    if [ "$INTERACTIVE" ]; then
      _rs_msg="${BYon}Are you ready to (${_data_color}${_rs_step}${BYon})"
      _rs_msg="${_rs_msg} ${_op_color}${_rs_desc}${BYon}? ${Boff}"
      yns_prompt "$_rs_msg"
      _rs_response="$response"
    else
      _rs_response="yes"
    fi  # INTERACTIVE?
    # execute according to response
    case "$_rs_response" in
      [Yy]* )
        # start with a separator for every step
        d_log_separator "$PN" "(step [$_rs_step]: $_rs_desc)" 1
        d_log_message "${LBon}Executing: ${BYon}${_rs_cmd}${Boff}" 1
        d_log_message "executing step [$_rs_step]: $_rs_clean_cmd" 2
        # eval expands variables inside the string
        eval "$_rs_cmd"; _rs_result=$?
        ### don't die - let the calling script decide what to do based on return
        ### d_log_handle_result $_rs_result 1 || die "failed to execute step [$_rs_step]:
 $_rs_clean_cmd"
        # display/log result and set bit in ret status binary bitmask
        d_log_handle_result $_rs_result '' "failed with exit code [$_rs_result]" 1 || \
            ret=$(( ret | 1 << _rs_step )) ;;
      [Ss]* )
        _rs_msg="${BYon}Skipping step [${_data_color}${_rs_step}${BYon}]:"
        _rs_msg="${_rs_msg} ${_op_color}${_rs_desc}${BYon} as instructed${Boff}"
```

```
        d_log_message "$_rs_msg" 1 ;;
      * )
        d_log_message "${BRon}Aborting sequence, as instructed${Boff}" 1
        ret=1; break ;;
    esac
    _rs_step=$(( _rs_step + 1 ))
  done
  unset -v _rs_status_file _rs_step _rs_cmd _rs_result _rs_entry _rs_desc
  unset -v _rs_response _rs_clean_cmd _rs_start _rs_stop _rs_msg
  return $ret
}
# @usage: run_sequence <status_file> <start> <stop>
# @args: $1 (string) path to sequence status file
# @args: $2 (int) starting step (1-indexed)
# @rule: $2 start must be '' or integer 1(default) - num_cmds
# @args: $3 (int) stopping step (1-indexed)
# @rule: $3 stop must be '' or integer 1 - num_cmds(default)
# @vars: INTERACTIVE, logFile, PN, US, _data_color, _op_color
# @ret: 0 on success, 1 on abort, or bitmask of failed steps
# @deps: isint, yns_prompt, d_log_separator, d_log_message, d_log_handle_result, strip_an
si, _translate_escapes

test_function() {
  ret=0; _tf_in="$1"
  printf '%b\n' "this was printed by the test function: $_tf_in" || ret=1
  unset -v _tf_in
  return $ret
}
# @usage test_function <message>
#-----[ main script ]-------------------------------------------------

configure || die "failed to configure"

log_show_config
log_message "(BEFORE process_cmdline)"

process_cmdline "$@" || die "failed to process_cmdline"

log_show_config
log_message "(AFTER process_cmdline)"

run_sequence "$status_file" '' ''
ret=$?
log_message_n "run_sequence completed with "
log_handle_result $ret '' ''
exit $ret
```