LASER INTERFEROMETER GRAVITATIONAL WAVE OBSERVATORY
- LIGO -
CALIFORNIA INSTITUTE OF TECHNOLOGY
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Technical Note **LIGO-T1300126-v1** 2013/02/23

The Input Mode Cleaner Autolocker

A. Mullavey, V. Frolov, R. DeRosa, C. Mueller

California Institute of Technology
LIGO Project, MS 18-34
Pasadena, CA 91125
Phone (626) 395-2129
Fax (626) 304-9834
E-mail: info@ligo.caltech.edu

Massachusetts Institute of Technology
LIGO Project, Room NW22-295
Cambridge, MA 02139
Phone (617) 253-4824
Fax (617) 253-7014
E-mail: info@ligo.mit.edu

LIGO Hanford Observatory
Route 10, Mile Marker 2
Richland, WA 99352
Phone (509) 372-8106
Fax (509) 372-8137
E-mail: info@ligo.caltech.edu

LIGO Livingston Observatory
19100 LIGO Lane
Livingston, LA 70754
Phone (225) 686-3100
Fax (225) 686-7189
E-mail: info@ligo.caltech.edu

http://www.ligo.caltech.edu/

1 Introduction

The Input Mode Cleaner (IMC) auto-locker has been set-up at Livingston to assist with the commissioning effort. As the IMC will eventually be integrated with the full interferometer, the IMC auto-locker needs to exist within the Guardian framework [1]. As such it has been implemented using Guardian template scripts and the GuardTools and CaTools perl modules as described in the Guardian Developer's Guide [2].

The *runGuardian* script, which is the generic script used for each sub-system guardian, has been copied and converted into the *runIMCGuardian* script (see Appendix A). The *runIMCGuardian* is the script that oversees all in terms of the IMC guardian. It continuously runs in the background and calls on verify, transit, goto or recover scripts depending on the STATE and REQUEST epics string fields.

The *runIMCGuardian* calls on the *transit_QUIET_LOCKED* (Appendix B.1) script to transition the IMC from an unlocked to a locked state. Once in the locked state the runIMCGuardian calls on the *verify_LOCKED* script (Appendix B.2) to continuously check that the IMC is locked. When the IMC drops lock, the runIMCGuardian runs the *recover_LOCKED* script (Appendix B.3), which returns the IMC to the state where it can try to acquire lock again. This sequence is described in more detail in Section 3.

2 Installation and Running the IMC Guardian

The IMC auto-locker exists as part of the IMC manager guardian. As such the *runIMCGuardian* script needs to be running in the background. To run the IMC guardian, you will need to have the latest version of the GuardTools and CaTools perl modules installed. These perl modules live in the '/opt/rtcds/userapps/release/guardian', and by doing an svn update in this directory, you'll obtain the latest versions.

You will also need to install the IMC scripts directory. The most current version is on the svn under 'userapps/release/ioo/l1/scripts/imc'. The IMC scripts directory contains the *runIMCGuardian* script, as well as the verify, transit, goto and recover scripts needed.

You will also need the IMC Guardian medm screen (see Figure 1) to interface with the IMC guardian scripts. This is the file IMC_GUARDIAN.adl. Ensure that the channel names linked to this medm screen have the right IFO.

Before running the *runIMCGuardian* script, make sure that the STATE and REQUEST fields are set to 'QUIET'. This can be done by typing 'caput L1:IMC-GUARD_STATE QUIET' and 'caput L1:IMC-GUARD_REQUEST QUIET' on the command line.

To execute the *runIMCGuardian* script, log in as controls, navigate to the IMC scripts directory and type './runIMCGuardian' on the command line.

Before turning on the auto-locker (described in the next section), you'll also want to look-over the *transit_QUIET_LOCKED*, *mcdown* (see Appendix C.1) and *mcup* (see Appendix C.2) scripts, and make sure that the settings are relevant to the particular IMC you are working on.

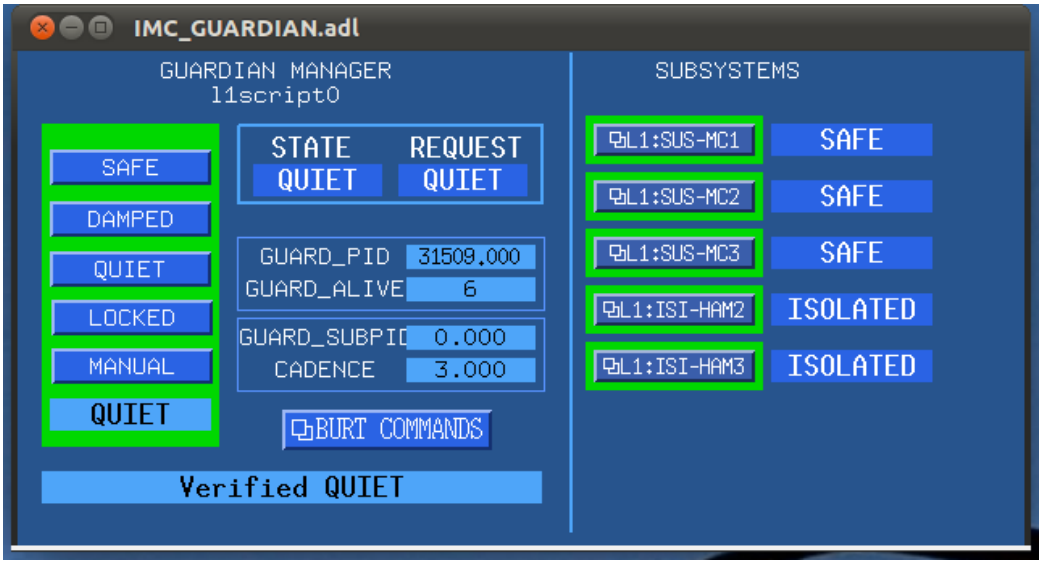

Figure 1: The IMC Guardian MEDM screen.

3 The Auto-locker

Here we provide a brief description of the IMC Guardian's auto-locker during lock acquisition and lock loss. Figure 2 shows the main functions of the *runIMCGuardian* script during this sequence and the STATE and REQUEST fields that it depends on.

We start at the QUIET state, which we define as being when the three MC suspensions are DAMPED and the HAM2 and HAM3 seismic isolation tables are ISOLATED. Each suspension and seismic isolation table has its own guardian, with which the IMC Guardian interacts with. Note: during commissioning this interaction between the IMC guardian and sub-system guardians has sometimes been inconvenient, and so with a few alterations to the scripts this dependency on the sub-system guardians can be removed.

To turn on the auto-locker, press the button labelled 'LOCKED' located on the left side of the IMC_GUARDIAN medm screen. This changes the REQUEST field from QUIET to LOCKED. The *runIMCGuardian* script sees that the STATE and REQUEST fields don't match and runs the *transit_QUIET_LOCKED* script, which starts off the locking sequence.

The *transit_QUIET_LOCKED* script first changes the STATE field to 'TRANSIT_QUIET_LOCKED' to inform the operator that it is in a state of transition. It then makes sure that the IMC servo board, SUS-MC2 lock filters and LSC to IMC path filters are in the correct state for locking the IMC, by running the *mcdown* script. In theory the system should already be in the correct state as the *mcdown* script is run when transitioning to the QUIET state and when recovering from a LOCKED state. However, we've added it here as a pre-caution against commissioner tom-foolery.

Before the IMC can be locked the PSL reference cavity needs to be locked. The PSL reference cavity has it's own auto locker and so the *transit_QUIET_LOCKED* script does a simple check of the reference cavity state, and doesn't proceed until it's locked.

Once the PSL reference cavity is locked the *transit_QUIET_LOCKED* script enables the fast feedback path from the IMC servo board to the PSL VCO, and the slow feedback path to the bottom stage of the MC2 suspension. It then waits for the IMC cavity to lock onto the carrier fringe. When

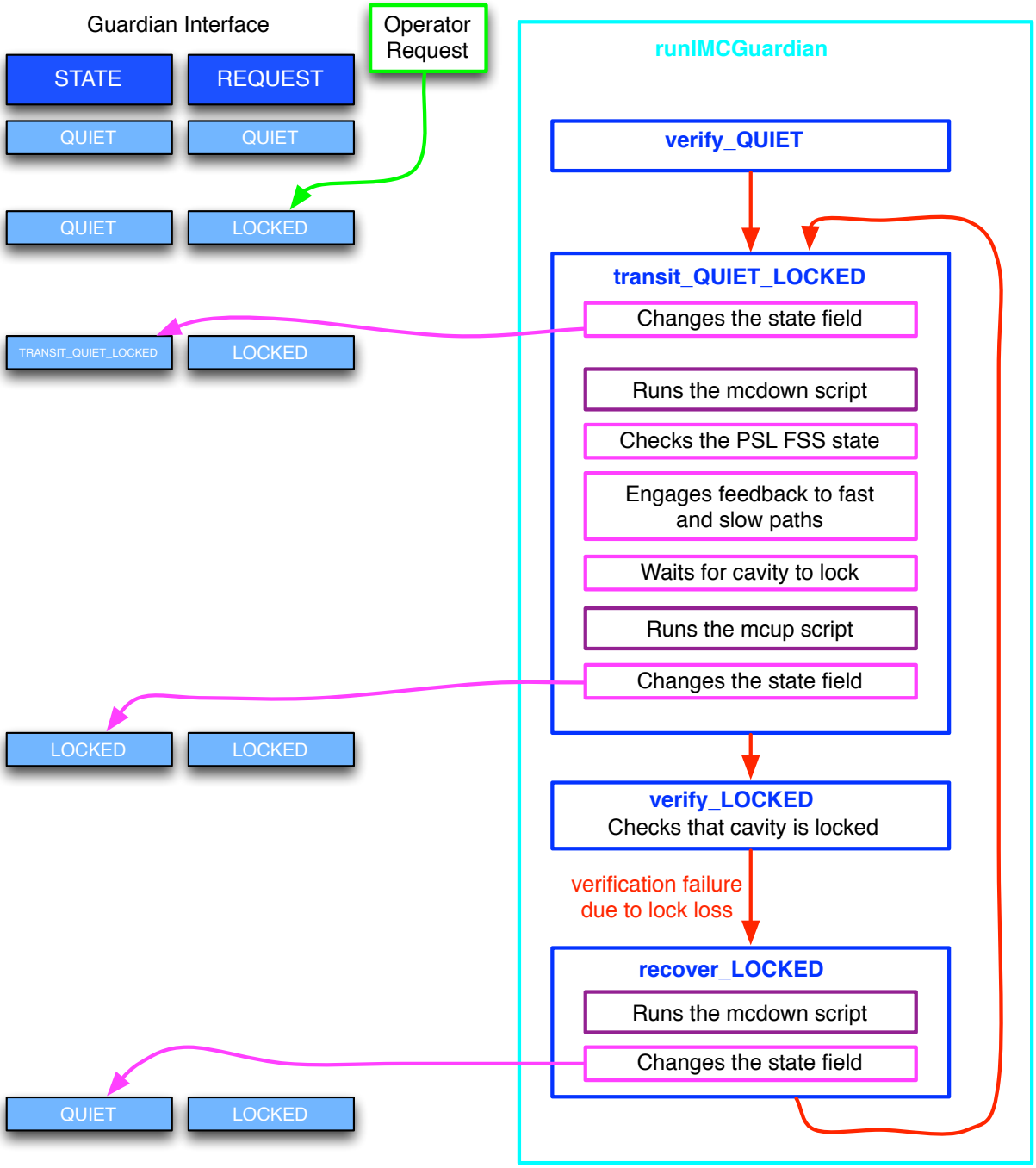

Figure 2: Flow chart showing the Autolocker sequence. The left shows the State and Request fields through the sequence. The right is a summary of workings of the continuously running *runIMCGuardian* script, including the guardian scripts that it calls on (blue boxes).

this happens the MC2-TRANS_PD channel becomes greater than the lock threshold (both set in the IMC_params.txt file).

This triggers the *transit_QUIET_LOCKED* script to then run the *mcup* script. The *mcup* script engages feedback to the middle stage of the MC2 suspension as part of the hierarchal control scheme. On the IMC servo board, it steps up the common gain, and engages the first boost filter. It also engages the boost filter in the MC2_M2_LOCK filter bank.

Once the *mcup* script is finished, the last act of the *transit_QUIET_LOCKED* script before exiting, is to change the STATE field to LOCKED.

The *runIMCGuardian* script sees that the STATE and REQUEST fields are both LOCKED and runs the *verify_LOCKED* script. The *verify_LOCKED* scripts only function is to check whether or not the MC-TRANS PD is above threshold. If it is above, the *verify_LOCKED* script reports to the runIMCGuardian that the LOCKED state is verified, in which case the runIMCGuardian loops around, continuously running the *verify_LOCKED* script.

If the MC-TRANS PD falls below threshold, the *verify_LOCKED* script reports a verification failure to the *runIMCGuardian* script, which then acts by runing the *recover_LOCKED* script. The *recover_LOCKED* script runs the *mcdown* script and then sets the STATE field back to QUIET.

The *runIMCGuardian* script sees that the STATE and REQUEST fields don't match and runs the *transit_QUIET_LOCKED* script again and the lock acquisition process starts again.

At any point in the sequence the auto-locker can be turned off by requesting a QUIET state. This runs the goto_QUIET script, which runs the *mcdown* script and returns the STATE field to QUIET. The *runIMCGuardian* sees that the STATE and REQUEST fields are both QUIET and starts continuously running the verify_QUIET script.

4 Future Improvements

Resonance flashes can trigger the *mcup* script, and currently there is no check during the running of the *mcup* script whether the cavity is still locked. The original guardian proposal was to fork off transit scripts and have the runGuardian script verify in the back ground. If we were to fork off the *transit_QUIET_LOCKED* script we could have the runIMCGuardian check if the cavity has actually caught lock, and if not terminate the *mcup* script and restart the sequence.

There is also a chance that the cavity might lock onto a bad mode. It would be good to have a check of when this happens, so that the runIMCGuardian could unlock from such modes.

A runIMCGuardian Script

```
#!/usr/bin/perl -w -I /ligo/cdscfg
#
#  runGuardian
#

use strict;
use stdenv;
use CaTools;
use GuardTools;
use Sys::Hostname;
use Getopt::Std;
use POSIX qw ( _exit :sys_wait_h);
INIT_ENV($ENV{IFO});
use IO::Handle;


my $imclogfile = "IMCGuardian_log.txt";

## open statelog.txt file, and append to.
#open LOG, ">>$imclogfile";

## write standard error and standard output to logfile
#STDOUT->fdopen(\*LOG, 'w');
#STDERR->fdopen(\*LOG, 'w');

print "$$ Starting guardian script\n";

our $opt_v;
getopts('v');


################################ Initialization
#my $SubSys = uc($ARGV[0]);
my $SubSys = "${IFO}:IMC";
my $separator = "_";
if (index($SubSys, "-")<=0){
                $separator = "-";
}

my $ScriptDir = "/opt/rtcds/userapps/release/ioo/${ifo}/scripts/imc";

#$SIG{__WARN__} = sub {&guardIMCLog($SubSys,@_)};
#$SIG{__DIE__} = sub {&guardIMCLog($SubSys,@_)};

#AM 120726 - On initial startup get the subsys dcu id or die if the subsys
# is not running. For restartup, carry over the subsys dcu id from last program run.
my $DCU_ID;
if (@ARGV < 1) {
    eval { ($DCU_ID) = caGet("${SubSys}${separator}DCU_ID") };
    die "Error: subsystem $SubSys is not running." unless ($DCU_ID ne '');
}
else {$DCU_ID = $ARGV[0]};


# Register process with first part of hostname
# First check we're the only guardian running
my ($guard_pid) = caGet("${SubSys}${separator}GUARD_PID");
if($guard_pid != 0 && $guard_pid != $$ &&(kill 0, $guard_pid)){
  die "Error: guardian already running for $SubSys.";
```

```
}
caPut("${SubSys}${separator}GUARD_PID", $$);

my $hostname = hostname;
if (length($hostname) > 20) {$hostname = substr($hostname, 0, 10);}
caPut("${SubSys}${separator}GUARD_HOST", $hostname);


########## MAIN LOOP ################
guardComment($SubSys, "Starting Guardian loop");
# Due to memory issues with perl, we will restart the
# script every 100 cycles
my $repeat = 0;
my $alive;
while ($repeat < 1000) {
  $repeat++;
#while (1) {

  # If the system has died, we want to wait and then skip the loop until it returns
  eval{ ($alive) = caGet("${SubSys}${separator}GUARD_ALIVE_COUNTEROUT") };
  if ($alive eq '') { print "boo."; sleep(5); next; }

  # The execution of this loop is controlled by the contents of the
  # STATE and REQUEST fields. There are four possibilities:
  #
  #   1. STATE unrecognized     Something bad has happened. Goto UNKNOWN
  #   2. STATE == REQUEST       Run the verify_STATE script
  #   3. STATE != REQUEST       Run a transit or goto script
  #   4. REQUEST unrecognized   Make a message and stay in STATE

  # Get a current list of valid states.  We do this
  # in each loop so states can be added without restarting
  # the Guardian.

#my @ValidStates = guardListStates( $SubSys );

  # Check whether the sub-process is still running
  # kill 0 sends a signal to see if the process is there
  # If it's not there, register a sub-pid of 0
  my ($subpid) = caGet("${SubSys}${separator}GUARD_SUBPID");
  if( $subpid != 0 && !(kill 0, $subpid) ) {
    caPut("${SubSys}${separator}GUARD_SUBPID", 0);
  }

  # clean up zombies from forked processes
  my $kid;
  do{ $kid = waitpid(-1, WNOHANG); } while $kid > 0;

  # Reset the watchdog timer
  caPut("${SubSys}${separator}GUARD_ALIVE_RESET", 1);

  # Sleep for the user-defined cadence
  my ($cadence) = caGet("${SubSys}${separator}GUARD_CADENCE");
  sleep ($cadence);

  # Get the current and requested state AFTER the sleep
  my ($CurrentState, $RequestState) =
  caGet(("${SubSys}${separator}GUARD_STATE", "${SubSys}${separator}GUARD_REQUEST")
      );
  $RequestState = uc($RequestState);
```

```
120
  ################## STATE == REQUEST:

  # Runs the verify script if the Current and Request State match and also if
  # the system is in a transiton state
125
  #if ($CurrentState eq $RequestState or $CurrentState =~ m/^TRANSIT/) {
  if ($CurrentState =~ m/$RequestState$/) {
    guardComment($SubSys,"Veryfying ${CurrentState}");
        #guardIMCLog($SubSys,"Verifying ${CurrentState}");
130     if (guardRunScript($SubSys,"verify_${CurrentState}")){
        #print "Verification Fail\n";
        guardComment($SubSys,"Verification Failed");
    #guardIMCLog($SubSys,"Verification Failed");
        guardStatus($SubSys,1);
135     guardRunScript($SubSys,"recover_${CurrentState}");
        } else{
    guardStatus($SubSys,0);
    guardComment($SubSys, "Verified_${CurrentState}");
    next;
140     }
  }


# Notes: if verification fails, run recover_${CurrentState} script. This script
    could check why
145# the verification failed and respond to the fail mode. It should also kill any
    running transit scripts,
# and restore to some state. Alternatively, it could do nothing.


  ######### ########## STATE != REQUEST:
150
  # Runs a transit (or goto) script if the two states don't match, except for when
      the
  # current state is a transition state that ends in the request state (confusing).

  #if ( grep(/$RequestState/, @ValidStates) &&
155  # grep(/$CurrentState/, @ValidStates) &&
  # $RequestState ne "")
  #if ($RequestState ne $CurrentState and $CurrentState !~ m/$RequestState$/) {
  if ($CurrentState !~ m/$RequestState$/) {
    #print "REQUESTED STATE $RequestState DIFFERENT FROM CURRENT STATE
        $CurrentState.\n" if $opt_v;
160   guardComment($SubSys, "From_${CurrentState}_to_${RequestState}");
    # Try to run a transit_INITIAL_FINAL script
    # if (!guardForkScript($SubSys, "transit_${CurrentState}_${RequestState}")) {
    #   print "Running transit_${CurrentState}_${RequestState}.\n" if $opt_v;
    #   guardComment($SubSys, "Running transit_${CurrentState}_${RequestState}");
165 # }
    if (!guardRunScript($SubSys, "transit_${CurrentState}_${RequestState}")) {
    print "Running_transit_${CurrentState}_${RequestState}.\n" if $opt_v;
    guardComment($SubSys, "Running_transit_${CurrentState}_${RequestState}");
    }
170 # Try to run a goto_STATE script
    elsif (guardRunScript($SubSys, "goto_${RequestState}") != -1) {
      print "Ran_goto_${RequestState}.\n" if $opt_v;
      guardComment($SubSys, "Did_GOTO_${RequestState}");
    }
175 else {
      # Couldn't transition, post a message.
```

```
      print "Error: couldn't go to $RequestState.\n" if $opt_v;
      guardComment($SubSys,  "Error! Couldn't get to $RequestState");
      caPut("${SubSys}${separator}GUARD_REQUEST", $CurrentState);
    }

    next;
  }


} # End of While Loop

#close LOG;

print "$$ At end of while loop, start new guardian\n";

# Restarts runGuardian after while loop ends. Also carries over $DCU_ID, in case
    the front end has died.
#if ($opt_v) {
# exec("$ScriptDir/runIMCGuardian -v $SubSys $DCU_ID");
#} else {
# exec("$ScriptDir/runIMCGuardian $SubSys $DCU_ID");
#}
if ($opt_v) {
  exec("$ScriptDir/runIMCGuardian_new -v $DCU_ID");
} else {
  exec("$ScriptDir/runIMCGuardian_new $DCU_ID");
}

############################
############################

################################ guardForkScript
sub guardForkScript {
  my ($SubSys, $script, $args) = @_;
  my $arguments = ($SubSys);
  if(ref $args eq 'ARRAY') {
    $arguments .= join(@$args, ' ');
  }

  my $prog;
  if(-e "${ScriptDir}/${script}"){
    $prog = "${ScriptDir}/$script";
  } else{
#   guardLog($SubSys, "Couldn't find ${dir}/${script}");
#   guardLog($SubSys, "Couldn't find ${InheritDir}/${script}");
    return 1;
  }
  CA->context_destroy();

  my $child_pid = fork;
  if($child_pid) {
    caPut("${SubSys}${separator}GUARD_SUBPID", $child_pid);
    return 0;
  } else{
    setpgrp; # set child process as group leader for all
    # sub-processes
    system "$prog $arguments";
    _exit 0;
  }

}
```

```
################################## guardRunScript
sub guardRunScript{
  my ($SubSys, $script, $args) = @_;
  my @arguments = ($SubSys);
  if(ref $args eq 'ARRAY') {
    push @arguments, @$args;
  }

  my $prog;
  if( -e "${ScriptDir}/${script}" ) {
    $prog = "${ScriptDir}/${script}";
  }
       else {
    # Couldn't find the script
  # guardLog($SubSys, "Couldn't find ${dir}/${script}");
  # guardLog($SubSys, "Couldn't find ${InheritDir}/${script}");
    return -1;
  }
# guardLog($SubSys, "Running $prog") if $opt_v;
  system $prog, @arguments;
  my $exit_status = $? >> 8;
  print "Exit status is $exit_status.\n" if $opt_v;
  return $? >> 8; #bitshift by 8 to get real exit status
}

################################ guardIMCLOG
sub guardIMCLog {
  my ($SubSys, $comment) = @_;
       my $separator = "_";
       if (index($SubSys, "-")<=0){
                    $separator = "-";
       }
       my $logdir = "$ENV{USERAPPS_DIR}/ioo/${ifo}/scripts/imc/";
       my $nowloc = localtime;
       my $nowgps = time;
       my $logfile = $logdir . "IMCguard_log.txt";
       open(LOG,">>${logfile}") or return "Couldn't open ${logfile} for appending
           data\n";
       print LOG "[${nowloc} - ${nowgps}] ${comment}\n";
       close LOG;
       return $comment;
}
```

Scripts/runIMCGuardian

B Guardian Scripts

B.1 transit_QUIET_LOCKED

```
#!/usr/bin/perl -w -I /ligo/cdscfg
#
#  transit_TEMPLATE
#
use strict;
use stdenv;
use GuardTools;
use CaTools;
INIT_ENV($ENV{IFO});

# Do an argument check
my $SubSys = shift;
my $separator = "_";
if (index($SubSys, "-")<=0){
                $separator = "-";
}

my $guard_alive;
eval { ($guard_alive) = caGet("${SubSys}${separator}GUARD_ALIVE_COUNTEROUT") };
die "Bad subsystem '$SubSys'\n" unless ($guard_alive ne '');

# Define the current, requested, and transition states
my $STATE = "QUIET";
my $REQUEST = "LOCKED";
my $TRANSIT_STATE = "TRANSIT_${STATE}_${REQUEST}";

# Tell the guardian we're transitioning
guardState($SubSys, $TRANSIT_STATE);

my $scripts_dir = "/opt/rtcds/userapps/release/ioo/${ifo}/scripts/imc";
my $mc_up_script = "$scripts_dir/mcup";
my $mc_down_script = "$scripts_dir/mcdown";

# Set the alarms for the transition state
#system("burtwb -f $burtdir/$TRANSIT_STATE.guardsnap");

##############################################################################
##############################################################################
# Run any code necessary to transition the state from STATE to REQUEST:
##############################################################################
##############################################################################

# Fringe counter check to see if cavity is moving too much or not enough

##############################################################################
# print sprintf("%04d-%02d-%02d %02d:%02d:%02d ",
system($mc_down_script);

##############################################################################
### Read parameters from file

my $params_file = 'imc_params.txt';
open my $params_fh, '<', $params_file or die "Can't open $params_file: $!";

###store into a hash
```

```
my %params;
while(<$params_fh>){
    chomp;
    my ($key,$crap,$val) = split /\s/;
    $params{$key} = ($val);
}

close $params_fh;

my $mon_chan = $params{mon_chan};
my $lock_threshold = $params{lock_threshold};


################################################################################

my $refcav_chan = "${IFO}:PSL-FSS_AUTOLOCK_STATE";
my $refcav_state = 0;

#my $mon_chan = "${IFO}:IMC-TRANS_OUTPUT";
#my $mon_chan = "${IFO}:IMC-MC2_TRANS_SUM_INMON";

#my $lock_threshold = 5000; # number for IMC TRANS 1 W input
#my $lock_threshold = 2000; # number for MC2 TRANS 1 W input
#my $lock_threshold = 9000; # number for MC2 TRANS 3 W input
#my $lock_threshold = 300; # number for MC2 TRANS 100 m W input

my $mon_val;
my $request;

################################################################################
# Make sure that the reference cavity is locked

while($refcav_state != 4){
    caPut("${SubSys}${separator}GUARD_ALIVE_RESET", 1);
    ($refcav_state) = caGet($refcav_chan);
    print "Waiting for refcav to be resonant \n";
    ($request) = caGet("${SubSys}${separator}GUARD_REQUEST");
    if($request eq $STATE){goto STOP}
    sleep (1);
}

# while(1){
#     ($refcav_state) = caGet($refcav_chan);
#     if($refcav_state == 4){goto REFCAVLOCKED}
#     print "Waiting for refcav to be resonant \n";
#     sleep (1);
# }

# REFCAVLOCKED:

print "Refcav is resonant, turn on feedback.\n";

################################################################################
# Start feeding back to VCO (Value of 1 means open switch, 0 means closed)
#caPut("${IFO}:PSL-FSS_VCO_WIDE_ON", 0);

caPut("${IFO}:IMC-REFL_SERVO_FASTEN", 1);

# Feedback IMC_L to MC2
caSwitch(("${IFO}:IMC-L"),("OUTPUT_ON"));
```

```
caSwitch(("${IFO}:SUS-MC2_M2_LOCK_L"),("OUTPUT_ON")); # make sure MC2 drive is
    enabled
caSwitch(("${IFO}:SUS-MC2_M3_LOCK_L"),("OUTPUT_ON")); # make sure MC3 drive is
    enabled
caPut("${IFO}:IMC-L_GAIN",1);

print "Turned feedback on, now wait to lock.\n";

######################################################################
# Wait for cavity to lock

while(1){
    caPut("${SubSys}${separator}GUARD_ALIVE_RESET", 1);
    ($mon_val) = caGet($mon_chan);
    ($request) = caGet("${SubSys}${separator}GUARD_REQUEST");
    if($request eq $REQUEST && $mon_val>$lock_threshold){goto LOCKED}
    if($request eq $STATE){goto STOP}
    #print "Not Locked Yet!!!";
    sleep(0.5);
}

STOP:
print "Stopped trying to lock, go to $request state.\n";

caPut("${IFO}:IMC-L_GAIN",0);
#caPut("${IFO}:PSL-FSS_VCO_WIDE_ON", 1);
caPut("${IFO}:IMC-REFL_SERVO_FASTEN", 0);


# Should also return the servo board settings

exit 0;

LOCKED:
print "Locked. Running Up Script\n";

system($mc_up_script);

print "Finished Up Script\n";

######################################################################
######################################################################
######################################################################

# Tell the guardian we're done transitioning
guardState($SubSys,$REQUEST);

exit 0;
```

Scripts/transit_QUIET_LOCKED

B.2 verify_LOCKED

```
#!/usr/bin/perl -w -I /ligo/cdscfg
#
#  verify_TEMPLATE
#
use stdenv;
INIT_ENV($ENV{IFO});
use CaTools;
use GuardTools;
use strict;

# Do an argument check
my $SubSys = shift;
my $separator = "_";
if (index($SubSys, "-")<=0){
                $separator = "-";
}

my $guard_alive;
eval { ($guard_alive) = caGet("${SubSys}${separator}GUARD_ALIVE_COUNTEROUT") };
die "Bad subsystem '$SubSys'\n" unless ($guard_alive ne '');

# The name of the state we want to verify and a verification flag
my $VerifyState = "LOCKED";
my $verified = 0;

################################################################
# Mainly here we want to verify that the IMC is still locked (i.e. mon>threshold)
# We also want to make sure suspensions haven't tripped

# my $mcsus_pre = "${IFO}:SUS-";
# my @mcsus = qw(MC1 MC2 MC3);      #add suspension here
# my $mc_length = @mcsus;
# #my @sus_chan_pre = map {$sus_pre . $_} @sus;
# my @mcsus_state_chans = map {$mcsus_pre . $_ . "_GUARD_STATE"} @mcsus;
# my @mcsus_state_vals = caGet(@mcsus_state_chans);
# my @damped = ("DAMPED","LOCKED","DAMPED");

#########################################################################
### Read parameters from file

my $params_file = 'imc_params.txt';
open my $params_fh, '<', $params_file or die "Can't open $params_file: $!";

###store into a hash

my %params;
while(<$params_fh>){
    chomp;
    my ($key,$crap,$val) = split /\s/;
    $params{$key} = ($val);
}

close $params_fh;

my $mon_chan = $params{mon_chan};
my $lock_threshold = $params{lock_threshold};

#########################################################################
# Check to make sure IMC_TRANS power is greater than some value
```

```
#my $mon_chan = "${IFO}:IMC-TRANS_OUTPUT";
#my $mon_chan = "${IFO}:IMC-MC2_TRANS_SUM_INMON";
#my $lock_threshold = 5000; # number for IMC TRANS 1 W input
#my $lock_threshold = 3000; # number for MC2 TRANS 1 W input
#my $lock_threshold = 9000; # number for MC2 TRANS 3 W input
#my $lock_threshold = 300; # number for MC2 TRANS 100 mW input
my ($mon_val) = caGet($mon_chan);

#print "${mon_val}\n";
#print "${lock_threshold}\n";

# use this condition if triggering off reflected power:
if($mon_val<$lock_threshold){
    print "IMC Dropped Locked!\n";
    $verified = 0;
}
# elsif("@mcsus_state_vals" ne "@damped"){
#     print "Check MC SUS states!\n"
#   $verified = 0;
#}
else{$verified = 1;}

##################################################################

# Do any other work needed to verify the state
#$verified = guardVerifyState($SubSys, $VerifyState);

# if verified, will exit with a value of 0,
#if not verified will exit with a value of 1
if($verified){
  # take any action on successful verification
  exit 0;
}
# take any action on unsuccessful verification
exit 1;
```

Scripts/verify_LOCKED

B.3 recover_LOCKED

```
#!/usr/bin/perl -w -I /ligo/cdscfg
#
#  transit_TEMPLATE
#
use strict;
use stdenv;
use GuardTools;
use CaTools;
INIT_ENV($ENV{IFO});


#my $SubSys = shift;
my $SubSys = "${IFO}:IMC";

########
# if lock loss and watchdogs trip, request ready state?
guardComment($SubSys,"Recover");
print "Return to QUIET state\n";
########
# Run the down script. This might be unnecessary, as the transit_QUIET_LOCKED also
    runs the down script.

my $scripts_dir = "/opt/rtcds/userapps/release/ioo/${ifo}/scripts/imc";
my $mc_down_script = "$scripts_dir/mcdown";

print "Running Down Script\n";

system($mc_down_script);

#######

print "Back in QUIET state\n";
guardComment($SubSys,"Returned to QUIET");

guardState($SubSys,"QUIET");
```

Scripts/recover_LOCKED

C Extra Scripts

C.1 mcdown

```
#!/usr/bin/perl -w -I /ligo/cdscfg
use stdenv;
INIT_ENV($ENV{IFO});
use GuardTools;
use CaTools;
use strict;

#########################################################################
### Read parameters from file

my $params_file = 'imc_params.txt';
open my $params_fh, '<', $params_file or die "Can't open $params_file: $!";

###store into a hash

my %params;
while(<$params_fh>){
    chomp;
    my ($key,$crap,$val) = split /\s/;
    $params{$key} = ($val);
}

close $params_fh;

my $in1gain = $params{in1gain};
#########################################################################
# ramp power to 3 W
#caPut("H1:PSL-RS_TARGETPOSITIONDEGREES",380); # phi = 90 for HPO, phi = 380 for no
    HPO
#caPut("H1:PSL-RS_EXECUTE","Fall");
#sleep(1);
#caPut("H1:PSL-RS_EXECUTE","Rise");

# turn off feedback
caPut("${IFO}:IMC-L_GAIN",0);
caPut("${IFO}:IMC-REFL_SERVO_FASTEN", 0);

# reduce fss gain
caPut("${IFO}:PSL-FSS_COMMON_GAIN",35);

# disable mc2:m2 output and boost and m1 output
caPut("${IFO}:SUS-MC2_M1_LOCK_L_GAIN",0);
caPut("${IFO}:SUS-MC2_M2_LOCK_L_GAIN",0);
caSwitch("${IFO}:SUS-MC2_M2_LOCK_L","FM3_OFF");
caPut("${IFO}:SUS-MC2_M1_LOCK_L_RSET",2);

# disable imc wfs and clear the control histories
caPut("${IFO}:IMC-WFS_SWTCH","OFF");
caPut("${IFO}:IMC-WFS_A_P_RSET",2);
caPut("${IFO}:IMC-WFS_B_P_RSET",2);
caPut("${IFO}:IMC-MC2_DISP_V_RSET",2);
caPut("${IFO}:IMC-IM4_T_P_RSET",2);

caPut("${IFO}:IMC-WFS_A_Y_RSET",2);
caPut("${IFO}:IMC-WFS_B_Y_RSET",2);
caPut("${IFO}:IMC-MC2_DISP_H_RSET",2);
```

```
caPut("${IFO}:IMC-IM4_T_Y_RSET",2);

# imc servo settings
my $servo_prefix = "${IFO}:IMC-REFL_SERVO_";

# engage the necessary filters
my @servofilt = qw(COMCOMP COMFILTER COMBOOST);
my @servofilt_chans = map { $servo_prefix . $_ } @servofilt;
my @servofilt_vals = ("On", "Off", 0);
caPut(@servofilt_chans, @servofilt_vals);

# polarities(0 = +ve, 1 = -ve)
my @polarities = qw(IN1POL FASTPOL);
my @polarities_chans = map { $servo_prefix . $_ } @polarities;
my @polarities_vals = (0, 0);
caPut(@polarities_chans,@polarities_vals);

# gains
my @amplifiers = qw(IN1GAIN FASTGAIN);
my @amplifier_chans = map { $servo_prefix . $_ } @amplifiers;
#my @amplifier_gains = (-26, 0); # 10W
#my @amplifier_gains = (-30, 0); #3W
#my @amplifier_gains = (-20, 0); #1W
#my @amplifier_gains = (-4, 0); #100 mW
my @amplifier_gains = ($in1gain, 0);
caPut(@amplifier_chans,@amplifier_gains);

# switches
my @servosw = qw(IN1EN IN2EN SLOWBYPASS);
my @servosw_chans = map { $servo_prefix . $_ } @servosw;
my @servosw_vals = ("On", "Off", "On");
caPut(@servosw_chans, @servosw_vals);

# open refl shutter
#my $scripts_dir = "/opt/rtcds/userapps/release/ioo/${ifo}/scripts/imc";
#my $open_shutter_script = "$scripts_dir/openShutter.bash";
#system($open_shutter_script);
```

Scripts/mcdown

C.2 mcup

```
#!/usr/bin/perl -w -I /ligo/cdscfg
use strict;
use stdenv;
use GuardTools;
use CaTools;
INIT_ENV($ENV{IFO});

# turn on mc2-m2 lock output and boost
sleep(3);
caPut("${IFO}:SUS-MC2_M2_LOCK_L_GAIN",0.1);
caSwitch(("${IFO}:SUS-MC2_M2_LOCK_L"),("FM3_ON"));


# step the psl fss gain from 10dB to 17dB
#caStep("${IFO}:PSL-FSS_COMMON_GAIN","+1,7",{delay=>1});

sleep(1);

# step the imc gain up 30 db
caStep("${IFO}:IMC-REFL_SERVO_IN1GAIN","+7,4",{delay=>1});

# common boost filters (values 0,1,2,3 for 0x 1x 2x or 3x boosts)
caPut("${IFO}:IMC-REFL_SERVO_COMBOOST", 1);

# engage the asc
caPut("${IFO}:IMC-WFS_SWTCH","ON");

caSwitch(("${IFO}:SUS-MC2_M1_LOCK_L"),("OUTPUT_ON"));
caPut("${IFO}:SUS-MC2_M1_LOCK_L_TRAMP",5);
caPut("${IFO}:SUS-MC2_M1_LOCK_L_GAIN",1);
```

Scripts/mcup

References

[1] M. Evans and S. Waldman. System guardian. Technical Report LIGO-T1000131-05, LIGO Laboratory, April 2010.

[2] C. Kucharczyk, B. Lantz, and S. Waldman. Guardian developer's guide. Technical Report LIGO-T1100608-v4, LIGO Laboratory, July 2012.