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

Document Type	**LIGO-T050166-00-Z**	2007/07/20

Operator Notification of Gamma-Ray Bursts and other Astrophysical Triggers

Rauha Rahkola

Distribution of this draft:

LIGO Scientific Collaboration

California Institute of Technology
LIGO Project - MS 51-33
Pasadena, CA 91125
Phone (626) 395-2129
Fax (626) 304-9834
E-mail: info@ligo.caltech.edu

Massachusetts Institute of Technology
LIGO Project - MS 20B-145
Cambridge, MA 01239
Phone (617) 253-4824
Fax (617) 253-7014
E-mail: info@ligo.mit.edu

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

Processed with LaTeX on 2007/07/20

1 Introduction

The External Triggers search team has designed and implemented a system to automatically notify staff at the detectors when an astrophysical event occurs. This is called the *GRB operator notification system*. With sufficient warning, interruptions such as breaking lock, entering the LVEA, and starting hardware injections can be moved to time intervals far from the gamma-ray burst trigger times. This is useful because it provides a reference of the state of the detectors around each gamma-ray burst trigger time.

The GRB operator notification system receives notices of gamma-ray burst events via email from the Gamma-ray burst Coordinates Network (GCN). The notices are parsed for information (most importantly the start time of the event) and discarded. The information is recorded in a *LIGO lightweight*-formatted file, which is an XML format used for storing data in the LIGO database. Additionally, if the notice arrives within an hour of the event start time, an alert is sent to the LHO and LLO control rooms, notifying operators of the event.

The system consists of two Perl scripts. The *parsing* script (currently running at Caltech) receives email notices and parses them for information. The *receiver* script (currently running at LHO, LLO, and GEO) listens on a TCP/IP port for alerts from the parsing script and passes them on to the control room via the instrument control software. (At LHO and LLO, the EPICS instrument control system is used, while at GEO it is the LabVIEW system.) Two-way socket communication is used by both scripts.

1.1 The role of operator notification

From the standpoint of an operator at the detector, every gamma-ray burst notice should be treated as a genuine event. In actuality, several of these notices are false alarms, as the satellites may trigger spuriously or subsequent instrumentation cannot verify the events. It is emphasized here that **these events are not used in the External Triggered searches**. The GCN produces two types of notices: (i) the GCN alerts, which are described in this document, and (ii) the GCN circulars. The GCN circulars are used to dispense information about validated gamma-ray burst events, and these are used by the External Triggers team to search for corresponding gravitational wave events. GCN circulars are normally sent at least several minutes after the gamma-ray burst event. This is undesirable from an operator standpoint because a decision to interrupt the LIGO detector data might occur within the intervening minutes between the event and the corresponding GCN circular.

The GCN alerts are produced automatically by the satellite trigger-processing software, and are normally received at the LIGO detector sites within five seconds of the trigger. Subsequent alerts concerning the same event are produced minutes after the trigger, and some alerts contain retraction statements for false triggers. These retraction statements are currently invisible to operators at the detector sites.

The events identified by the GCN alerts is not a superset of the events identified in the GCN circulars. Occasionally a gamma-ray burst may be identified by a satellite which does not produce GCN alerts. In particular, the Inter-Planetary Network identifies localized gamma-ray burst events through the triangulation of triggers from several distant satellites.

2 Parsing script: `parse_notices.pl`

The parsing script runs under the External Triggers username (*wk_sz*) at Caltech. The current environment is described here, although several configuration options are available on the command-line. Incoming email messages are distributed into individual files (suffix `.notice`) in the INBOX directory (`~/notices/InBox`). A corresponding OUTBOX directory tree is present as well (`~/notices/OutBox`), with the following contents:

`unparsed/` - a directory containing messages which are not currently parseable.

`parsed-GCN_alerts/` - a directory containing information (in individual files) parsed from GCN alert emails.

`parsed-GCN_circulars/` - a directory for information parsed from GCN circular emails (currently not used).

`parsed-SNEWS/` - a directory for information parsed from emails by the SuperNovae Early Warning System *SNEWS*.

`triggers.xml` - a file containing all trigger information in LIGO Lightweight format.

When the script is running, it generates a LOCKFILE which prevents other copies of the script from executing by accident (`~/.parse_notices.pl-pid`). The LOCKFILE merely contains the process ID of the original script. Diagnostic output from the parsing script is recorded in a LOGFILE (`parse_notices.log`). Any further output is directed to standard output and error.

The parsing script frequently checks the INBOX for new messages. If no messages are present, a *keep-alive* signal is sent to the receiver scripts, signifying that a socket connection can still be completed. This keep-alive signal is important, since the event rate is about once per day. If one or more messages are present, the parsing script reads the header information from each message and checks the sender against a list of accepted senders. Once the sender matches an entry in this list, the subject line is searched for recognized phrases (e.g. "GCN/HETE_POSITION") and a child process is started, passing the message on to the appropriate message handler. Notices from different astrophysical detectors have differing formats, so a message handler is created for each sype of notice. From the standpoint of the parsing script, the most relevant pieces of information from the notices are the timestamps. Time/Date information is stored in the notices either in a (seconds of day)/Julien date format or in one of the recognized ISO timestamp string formats. These are converted into GPS timestamps via the `tconvert` function of the LIGOTools library [1]. In this manner, the timestamp of the email message can be compared with that of the event contained within the email. If the differences between these two timestamps are below a threshold (the *stand-down interval*), the parsing script will open a connection with each of the receiver scripts in turn and send an alert signal. The receiver scripts pass the alerts along to the control room, notifying the operators and scientists that an astrophysical event has recently been observed. Staff are subsequently expected to maintain the detector in a good data-taking state until the stand-down interval has expired.

In addition to notifying operators in the control rooms, the parsing script sends email to the administrators and interested parties. Diagnostic messages, such as unrecognized emails or socket connection errors/warnings, are sent to the PARSING SCRIPT ADMINISTRATOR and the RECEIVER SCRIPT ADMINISTRATOR at each of the sites. A message is sent for each trigger acknowledged by one or more receiver scripts. Normal termination of the parsing script causes an email to be sent to all administrators. In the event of normal termination of the program, the LOCKFILE is removed. A cron job periodically checks for the existence of the LOCKFILE and restarts the program if it is

missing.

3 Receiver script: grb_server_MI.pl

Because the receiver script must interface with either the EPICS or LabVIEW control systems, this dissertation will not detail the interface with either system. Emphasis is placed here on the socket connection and major functions.

The receiver script runs on a machine which interfaces both with the Internet and with the control room network, which is usually behind a firewall. It listens on a TCP/IP port agreed on by the parsing and receiver administrators. Once it receives a connection from the parsing script, it forks a child process to handle communications while the parent process waits for another connection. The child process receives two kinds of signals from the parsing script: a keep-alive signal or an alert signal. Both signals are strings with the same format:

```
<sig_#> UTC-<YYYY>-<MM>-<DD>-<hh>:<mm>:<ss> UTC-<YYYY>-<MM>-<DD>-<hh>:<mm>:
```

where the two `UTC` timestamps mark the beginning and end of the stand-down interval, respectively. The signal flag (`<sig_#>`) indicates the type of event, which is one of the following:

1 A keep-alive signal. For this case, the first `UTC` timestamp is the current date and time, while the second timestamp is filled with `0`'s.

2 A HETE gamma-ray burst alert signal.

3 An INTEGRAL gamma-ray burst alert signal

4 A Swift gamma-ray burst alert signal

5-18 currently unassigned, but reserved for specific gamma-ray burst detectors

19 any other gamma-ray burst signal

20 A SNEWS supernova signal

21-38 currently unassigned, but reserved for specific types of supernova signals

39 any other supernova signal

40+ currently unassigned, but reserved for other astrophysical signals

Upon receiving a signal, the receiver script child process will send an acknowledgement back to the parsing script:

```
ACK <receiver_name> <original_signal>
```

where `<receiver_name>` is the name of the host machine that the receiver script is running on, and `<original_signal>` is the original signal sent by the parsing script. The receiver script then logs the signal in a file and communicates the information to the control room via the control system interface. In some cases, the signal is generated from a follow-up observation for the same GRB as an earlier signal. The receiver script recognizes when this happens and will refrain from re-alerting the control room.

4 Known issues and areas for improvement

Although the operator notification system is recognized to be in stable condition, there are some issues which have been known to interrupt normal communication. These are listed below in no particular order.

- Occasionally the machine running the parsing script (*alterf.ligo.caltech.edu*) is shut down without cleanly killing all processes. When the machine is restarted, the lock file is still present even though the parsing script is not running. One improvement in the crontab for *wk_sz* would be to check that the process ID listed in the lock file is actually running at the time. If not, the lock file should be removed, an entry made in the logfile, and the script be allowed to restart.
- Socket communication with the receivers is limited to 30 seconds, and each signal is sent through a different socket. Depending on network traffic at the time, communication with the GEO site can regularly exceed this time limit. Also, socket communication is discontinued with the GEO site several hours after the receiver script is restarted for unknown reasons.
- When the CIT machine goes down for maintenance, a backup system should be in place to continue handling of the GCN alerts.
- The parsing script should accurately keep track of child processes through each cycle of the loop.

A Script Walkthroughs

Key scripts for the execution of the operator notification system are included here.

A.1 A walkthrough of the parsing script

Below is a listing of the current parsing script running at Caltech. The script begins by declaring global variables, including the important `%trig_info` hash variable, which will eventually contain all the information obtained from the email alert.

```
1 #!/usr/bin/env perl
  #
  # parse_notices.pl parses emails from the Gamma-ray burst  ...
     Coordinates Network
  #     (GCN) and (soon) the SuperNovae Early Warning System.  It  ...
     extracts
  #     trigger information from the emails and optionally sends  ...
     notification
6 #     to control room operators at LHO, LLO, GEO, and CIT.
  #
  # Author:       R. Rahkola
  # Email:        rrahkola@ligo-wa.caltech.edu
  # Start Date:   April 25, 2003
```

```
# Date/Ver:     $Id: parse_notices.listing.tex,v 1.9 2006/12/06  ...
   19:05:13 rrahkola Exp $
########################################################################## ...

#########################
#####  Depends on the following Perl modules
use strict qw(refs subs);
use vars qw($LOGFILE $TCONVERT $TDELAY $email_to $email_event  ...
   $inbox $outbox $verbosity $debug
            %trig_info %det_info %obs_info %time_info %event_info % ...
               notice_info $am_child %IP_INFO
            $accepted_senders);
use Getopt::Std;
use Time::Local 'timegm_nocheck';
use Time::HiRes qw(gettimeofday tv_interval);
use IO::Socket;
#use File::Copy;
###:NOTE: RJR 2005.04.26 -- timeout field gets set to IP_TIMEOUT if ...
    there is an
###:NOTE:       error during the socket connection.  This value  ...
   gets decreased
###:NOTE:       every time the script loops through the $INBOX  ...
   directory looking
###:NOTE:       for notices until it reaches 0, then tries to send  ...
   a keep-alive
###:NOTE:       message.
###:NOTE: RJR 2005.11.14 -- timeout field gets set to  ...
   KEEPALIVE_TIMEOUT if the
###:NOTE:       alarm wakes up before the socket connection is  ...
   complete.
###:NOTE: RJR 2005.11.16 -- alarm is set to ALARM while socket  ...
   connection is opened.
###:NOTE:       It gets reset to 0 once the socket connection is  ...
   completed.
###:NOTE: RJR 2005.11.21 -- TIME_TCP is a flag used to determine  ...
   whether to output
###:NOTE:       the time it takes for a socket connection to finish ...
   .  Output is sent
###:NOTE:       to STDOUT.
use constant IP_TIMEOUT => 3600;
use constant KEEPALIVE_TIMEOUT => 20;
use constant ALARM => 120;
use constant TIME_TCP => 1;

my $children    =0;
$CENTURY        = 2000;
$EPOCH          = "J2000";
```

The %EVENT_TABLE hash variable customizes some actions the parsing script performs based on an event alert. Such actions include communicating with the receiver scripts and emailing interested parties that an event has been detected. The hash uses the name of the detector to customize the actions.

```
%EVENT_TABLE     = ( # format: det_name => (event_tag, "GRB"/"SN",  ...
   event_timeout)
        FREGATE  => [2,  "GRB", 3600],        # HETE
        IBIS     => [3,  "GRB", 3600],        # INTEGRAL
        BAT      => [4,  "GRB", 3600],        # Swift GRB
        XRT      => [4,  "GRB", 3600],        # Swift GRB follow-up ...
           /XRF
        UVOT     => [4,  "GRB", 3600],        # Swift GRB follow-up
        GRB      => [19, "GRB", 3600],        # Anonymous GRB
        SNEWS    => [20, "SN",  3600],        # SNEWS supernova
        SN       => [39, "SN",  3600],        # Anonymous supernova
        OTHER    => [40, "UNK", 3600]         # Anonymous event
);
```

The %IP_INFO hash variable contains protocol information for each of the receiver scripts running at the sites. Beside the IP address/port, and contact information, %IP_INFO keeps track of the number of consecutive successful connections with the sites (field success) and maintains a countdown field (timeout) which gets set if an error occurs during communication with a site. Another attempt at communication will be made when the countdown returns to zero. Contact information is collected into a variable $remote_contacts for diagnostic messages which affect all the sites.

```
%IP_INFO        = (
        LHO => { ip => '198.129.208.138', port => 7070, timeout =>  ...
           0, success => 0,
                 contact => 'barker_d@ligo-wa.caltech.edu' },
        GEO => { ip => '130.75.117.151',  port => 7070, timeout =>  ...
           0, success => 0,
                 contact => 'joshua.smith@aei.mpg.de,siohen@aei.mpg ...
                    .de' },
        LLO => { ip => '130.39.245.3',    port => 7070, timeout =>  ...
           0, success => 0,
                 contact => 'lisab@ligo-la.caltech.edu' }

);
#       CIT => { ip => 0, port => 34512, timeout => 0 }
my $remote_contacts = join ',', map( $IP_INFO{$_}{contact}, sort  ...
   keys %IP_INFO );
```

The array @accepted_senders is a list of senders which are officially recognized by the parsing script. Thus the script will not try to parse spam email or messages from mailer daemons.

```
###:NOTE: RJR 2004.07.23 -- replaced $my_email with  ...
   $accepted_senders to only recognize
###:NOTE:       senders of External Trigger emails
```

```
#$my_email    = 'sn@ligo.caltech.edu';
@accepted_senders = ('vxw@capella.gsfc.nasa.gov', 'aavso@aavso.org ...
   ');
```

Next, the parsing script imports the handler modules in `parse_GCN.pl` and declares all internal subroutines.

```
70 ##########################
   #####  Declare various subroutines here
   ### RJR 2004.08.27 put parsing subroutines in parse_XXX.pl
   require "parse_GCN.pl";
   #sub handle_NOPARSE;
75 #sub handle_HETE;
   #sub handle_IPN;
   #sub handle_INTEGRAL;
   #sub handle_RXTE;
   #sub handle_SWIFT;
80 #sub handle_GCN_circular;
   sub get_timestamp;
   sub check_times;
   sub notify_ops;
   sub send_trigger;
85 sub spawn;
   sub REAPER;
   sub fatal;
   sub logmsg {
       open LOGFILE or &fatal("Can't open log file.", 1);
90     print LOGFILE "[", scalar gmtime, " GMT]: @_\n";
       close LOGFILE;
   }
```

Various environment-dependent variables are set. Most importantly, the location of the executable `tconvert`, which converts between GPS and UTC timestamps, is defined.

```
93 ##########################
   #####  Important environment variables (for tconvert, etc.)
95 # CIT:
   $ENV{LIGOTOOLS}      ="/ligoapps/ligotools";
   # backgammon: $ENV{LIGOTOOLS}      ="/opt/ligotools";
   # UofO: $ENV{LIGOTOOLS}      ="/home/cern/ligo/ligotools";
   $ENV{PATH}      = "/ldcg/bin:/lal/bin:$ENV{LIGOTOOLS}/bin:$ENV{PATH ...
      }";
100 $ENV{LD_LIBRARY_PATH} = "/ldcg/lib:/lal/lib:$ENV{LIGOTOOLS}/lib";
   #$ENV{LD_LIBRARY_PATH} = "/ldcg/lib:/lal/lib:$ENV{LIGOTOOLS}/lib: ...
      $ENV{LD_LIBRARY_PATH}";
   # location of the notice files (raw notices, parsed notices, log  ...
      file)
   #$OUTPUT_DIR    = "/home/sn/public_html/triggers/post-s2";
   #$RAW_PREFIX    = "raw_GRB-";
105 #$PARSED_PREFIX = "parsed_GRB-";
```

```
#$SUMMARY_FILE  = "GRB_summary";
# location of 'tconvert'-- converts UTC to GPS
$TCONVERT       = "$ENV{LIGOTOOLS}/bin/tconvert";
# location of 'tdelay'-- converts RA & DEC to a time delay from LHO ...
    to LLO
$TDELAY         = "time_delay";
# location of trigger file.  This can also be a product of this  ...
   script.
$TRIGFILE       = "triggers.xml";
```

Various command-line options are available, and are dscribed in the following section of code. The parsing script will use default values if they are not set on the command-line.

```
#########################
#####  Top-level user variables
getopts('hvd:i:o:l:e:g:s:p:nt');
# Verbosity, Debug levels
$verbosity      = ($opt_v) ? $opt_v : 0;
$debug          = ($opt_d) ? $opt_d : 0;
# File/Directory variables
$inbox          = ($opt_i) ? $opt_i : "$ENV{HOME}/notices/InBox/";
$outbox         = ($opt_o) ? $opt_o : "$ENV{HOME}/notices/OutBox/";
$email_to       = ($opt_e) ? $opt_e : 'rrahkola@ligo-wa.caltech.edu ...
   ';
$email_event    = ($opt_g) ? $opt_g : 0;
$email_event    .= ($opt_s) ? ",$opt_s" : "";
$LOGFILE        = ($opt_l) ? $opt_l : "$ENV{HOME}/notices/ ...
   parse_notices.log";
$LOGFILE        = ">>$LOGFILE";
$PRODUCT        = ($opt_p) ? $opt_p : 0;
$NOTIFY_OPS     = ($opt_n) ? (1 - $opt_n) : 1;
$SEND_TRIGGER   = ($opt_t) ? (1 - $opt_t) : 1;
# Help message
if ($opt_h) {
    print <<USAGE;
Usage: $0 [-v] [-d <debug_level>] [-i <inbox_dir>] [-o < ...
   outbox_parent_dir>]
       [-e <email_addresses>] [-g <email_addresses>] [-s < ...
          email_addresses>]
       [-l <logfile>] [-p <product>] [-n] [-t]

where: inbox_dir      = the directory of the incoming emails to be  ...
   parsed
   outbox_parent_dir  = the parent directory of the parsed emails ( ...
      must contain
                         subdirectories 'unparsed', 'parsed- ...
                            GCN_alerts',
                         'parsed-GCN_circulars', 'parsed-snews')
```

```
    email_addresses    = a comma-separated list of emails addresses  ...
       (NO SPACES!)
    logfile            = the full pathname of the file to send  ...
       messages to
    product            = one of 'template',

Use option: -h  to get this help message
            -e  to specify email addresses for all notices ( ...
               including diagnostics)
            -g  to specify email addresses for GRB notices
            -s  to specify email addresses for Supernovae notices
            -n  to preclude sending operator notification
            -t  to preclude sending trigger information to  ...
               metadatabase
            -i  to specify the location of incoming messages
            -o  to specify the location of read messages
            -l  to specify the location of diagnostic output
            -v  to receive messages on standard out
            -d  to set the debug level
                1 - email messages to standard out

USAGE
    exit ($opt_h - 1);
}
```

The parsing script next initiates a lock file, a normally-hidden file in the user's home directory. The filename is based on the name of the executable— (in this case `.parse_notices.pl-pid`)— and prints the current process ID into the file. A crontab entry for the External Triggers username at Caltech checks every ten minutes that this lock file exists. If it doesn't exist, the script will be restarted. Currently the crontab will not verify that the process contained in the lock file is actually running.

```
#########################
#####  Initialize Files / Variables
### Start PID file
if (-e "$ENV{HOME}/.$0-pid" and !$PRODUCT) {
    print "Unable to start $0.  Please remove $ENV{HOME}/.$0-pid\n ...
       ";
    &fatal("Existing process ID for $0", -1);
}
open(OPID, ">$ENV{HOME}/.$0-pid");
print OPID $$;
close OPID;
###  Begin logfile session
&logmsg("vvvv===============  Beginning $0");
###  Change LHO's IP address to a local one if we're in debugging  ...
   mode
$IP_INFO{LHO}{ip} = 'localhost' if $debug;
```

```
175 ###  Reduce the timeout for a receiver if we're in debugging mode
    #$IP_TIMEOUT = 10 if $debug;
```

In addition to parsing alerts, the script will initialize a LIGO Lightweight trigger file if requested with the `-p template` option. This is useful because the `%trig_info` hash variable determines the order of fields in the LIGO Lightweight table. Thus a field can be added to `%trig_info` and a new trigger file can be generated automatically. The code below describes the `%trig_info` hash variable and, if requested, initialize a LIGO Lightweight trigger file. It also declares subarrays, or "cuts" of the `%trig_info` hash.

```
177 ##########################
    #####  Output trigger file template: header info
    if ($PRODUCT eq "template") {
180     &logmsg("Creating trigger file template in $TRIGFILE.");
        open(TRIGGER,">${outbox}/$TRIGFILE")
            or &fatal("Can't open OutBox/$TRIGFILE for writing.", 1);

        print TRIGGER <<XML_HEAD;
185 <?xml version="1.0" encoding="utf-8"?>
    <!DOCTYPE LIGO_LW SYSTEM "http://www.cacr.caltech.edu/projects/ ...
       ligo_lw.dtd">
    <LIGO_LW Name="ligo:ldas:file">
       <Table Name="external_trigger:table">
    XML_HEAD
190 }


    ##########################
    #####  External-Trigger database hash; use as initialization
195 # NOTE: Each trigger MUST HAVE three timestamps: one for the event, ...
        one for
    # the notice (as sent to GCN), and one for the email distributed by ...
        the GCN
    # (or other trigger distribution program).
    %trig_info = ( event_timestamp => "",   event_timefmt => "",
                   notice_timestamp => "",  notice_timefmt => "",
200                email_timestamp => "",   email_timefmt => "",
    # NOTE: Each trigger MUST HAVE an event type (e.g. GRB, XRF, etc.)  ...
       and an
    # notice type (e.g. "HETE S/C_Alert", "HETE Gnd Analysis", etc.)
                   event_type => "",        notice_type => "",
    # NOTE: The 'notice_id' is only unique for each originator.  Other  ...
       originators
205 # may eventually have the same notice_id.  The 'notice_sequence' is ...
        for
    # updates to the same original notice (e.g. possibly containing  ...
       direction
    # information)
```

```
                notice_id => "",         notice_sequence => "",
    # NOTE: These are the coordinates of the event.  'Z' is the  ...
       redshift of the
210 # event, 'RA' is the right ascension, 'DEC' is the declination, and ...
       'EPOCH' is
    # the epoch for which these coordinates are valid.
    # !!!WARNING!!!  Currently the only valid epoch is J2000.  This may ...
       be the
    # !!!WARNING!!!  assumed epoch for subsequent scripts, and if no  ...
       epoch is
    # !!!WARNING!!!  given in the notice.
215 # NOTE: If a direction is obtained for the event, it is possible to ...
       calculate
    # several variables, including attenuation due to the antenna  ...
       pattern for the
    # various GW detectors, time delays, altitude and azimuth angles  ...
       relative to
    # each IFO, a multi-detector antenna pattern attenuation factor.   ...
       These will
    # be generated using an external script.
220             event_z => "",           event_z_err => "",
                event_ra => "",          event_ra_err => "",
                event_dec => "",         event_dec_err => "",
                event_epoch => "",       event_err_type => "",
                event_status => "",      event_number_grb => "",
225             event_number_gcn => "",
    # NOTE: The detector name is a string (e.g. "WXM", "SXC", etc.)  ...
       identifying
    # the main detector used to obtain coordinates for the event.   ...
       Other detectors
    # which observed the event are listed in the variable 'det_alts'.   ...
       Detectors
    # for gamma-ray bursts usually have a frequency band ('det_band')  ...
       associated
230 # with the trigger.  Most detectors should have a signal-to-noise  ...
       ratio
    # ('det_snr'), a peak flux ('det_peak'), and a time-integrated  ...
       fluence
    # ('det_fluence') as well.  The integration times for both the peak ...
       flux
    # ('det_peak_int') and fluence ('det_fluence_int') are also given.
    # 'det_fluence_int' is equivalent to the duration, even if no  ...
       fluence is
235 # given.  Peak flux and fluence may have attached units.  No peak  ...
       flux or
    # fluence will be given if they do not match the frequency band (if ...
       any) of
```

```
    # the trigger.
                  det_name => "",          det_alts => "",
                  det_band => "",          det_snr  => "",
240               det_peak => "",          det_peak_int => "",
                  det_fluence => "",       det_fluence_int => "",
    # NOTE: The ligo fields correspond to values calculated using the  ...
       LALApps
    # modules.  These include the delay from LHO to LLO (+ means the  ...
       GRB was seen
    # by LHO first.), and the average antenna factor for LHO and LLO
245               ligo_fave_lho => "",     ligo_fave_llo => "",
                  ligo_delay => "",
    # NOTE: The observer location and pointing is important mainly for  ...
       questions
    # regarding the field of view.  If for instance, the field of view  ...
       included
    # the sun or a known source, the trigger may need to be ignored.   ...
       Observer
250 # location is given in fixed-Earth coordinates; direction of  ...
       pointing is given
    # in right ascension and declination coordinates corresponding to  ...
       the same
    # epoch as the event.
                  obs_fov_ra => "",        obs_fov_ra_width => "",
                  obs_fov_dec => "",       obs_fov_dec_width => "",
255               obs_loc_lat => "",       obs_loc_long => "",
                  obs_loc_ele => "",
    # NOTE: 'notice_url' is the location at which this notice may be  ...
       retrieved.
    # 'notice_comments' may be shortened to fit within the allowed  ...
       database
    # entry.
260               notice_url => "",        notice_comments => "" );


    ##########################
    #####  Temporary/partial database hashes.  Copy these to store  ...
       temporary
265 #####  values for comparison with %trig_info without writing over  ...
       existing
    #####  %trig_info entries.
    foreach $tmpkey (sort keys %trig_info) {
        print "\%trig_info{$tmpkey}\t:",$trig_info{$tmpkey},"\n"
            if ($verbosity and $debug > 1);
270     $det_info{$tmpkey}    = "" if $tmpkey =~ /^det/;
        $obs_info{$tmpkey}    = "" if $tmpkey =~ /^obs/;
        $time_info{$tmpkey}   = "" if $tmpkey =~ /time(stamp|fmt)/;
```

```
        $event_info{$tmpkey}  = "" if $tmpkey =~ /^event/;
        $notice_info{$tmpkey} = "" if $tmpkey =~ /^notice/;

        ### Output key to trigger file if desired
        if ($PRODUCT eq "template") {
            my $colname = $tmpkey;  # defaults to $tmpkey
            my $coltype = "lstring";
            # Special key for event_timestamp -> start_time,  ...
               start_time_ns
            if ($tmpkey =~ /event_timestamp/) {
                print TRIGGER <<EVENT_TIMESTAMP;
          <Column Name="external_trigger:start_time" Type="int_4s" />
          <Column Name="external_trigger:start_time_ns" Type="int_4s"  ...
             />
EVENT_TIMESTAMP
                next;
            # Change "*_timestamp" keys to "*_time", except as noted  ...
               above
            } elsif ($tmpkey =~ /(\w+)_timestamp/) {
                $colname = "$1_time";
                $coltype = "int_4s";
            # Ignore "*_timefmt" keys
            } elsif ($tmpkey =~ /timefmt/) {
                next;
            # Set $coltype to real_4 for any location values
            } elsif ($tmpkey =~ /_(dec|ra|z|ele|lat|long)/) {
                $coltype = "real_4";
            }
            print TRIGGER <<COLUMN;
          <Column Name="external_trigger:$colname" Type="$coltype" />
COLUMN
        }
    }


    ##########################
    #####  Output trigger file template: stream element
    #####  :NOTE: exit when done, as this script was called with the -p ...
        argument
    if ($PRODUCT eq "template") {
        print TRIGGER <<STREAM;
          <Stream Name="external_trigger:table" Type="Local" Delimiter ...
             =",">
    STREAM

        close TRIGGER or &fatal("Can't close OutBox/$TRIGFILE.", 1);
        &fatal("Finished writing trigger template to $TRIGFILE", 0);
```

```
}
```

Next the parsing script defines actions for various inter-process communication signals. The usual action is to send an email to the administrators and exit. Some subroutines may not be executed, depending on the operating system. The `USR1` signal is unique in that it writes a diagnostic messaage in the LOGFILE. The diagnostic message currently contains the tally of successful communications with each receiver or the number of cycles remaining until another communication attempt will be made.

```
##########################
#####  Handle various signals using fatal subroutine
# This one may not work.
$SIG{KILL} = sub {&fatal("Received kill signal.  Shutting down...", ...
    0, $remote_contacts)};
$SIG{TERM} = sub {&fatal("Received term signal.  Shutting down...", ...
    0, $remote_contacts)};
$SIG{INT}  = sub {&fatal("Received interrupt signal.  Shutting down ...
   ...", 0, $remote_contacts)};
$SIG{ABRT} = sub {&fatal("Received abort signal.  Shutting down ...
   ...", 0, $remote_contacts)};
$SIG{HUP}  = sub {&fatal("Received hangup signal.  Shutting down ...
   ...", 0, $remote_contacts)};
$SIG{QUIT} = sub {&fatal("Received quit signal.  Shutting down...", ...
    0, $remote_contacts)};
###:NOTE: 2005.11.15 (RJR) Tried creating a diagnostic tool which  ...
   can be used
###:NOTE:       to get variable information without killing the  ...
   process.  Add
###:NOTE:       your variable to the list of output below.  Then  ...
   restart the
###:NOTE:       client, send it a USR1 kill signal, and check the  ...
   logfile.
$SIG{USR1} = sub {
    my $diagnostics;
    $diagnostics .= "\n\tNumber of child processes: $children.";
    foreach $receiver (sort keys %IP_INFO) {
        $diagnostics .= "\n\t$receiver: timeout $IP_INFO{$receiver ...
           }{timeout}";
        $diagnostics .= ",\tconnect_count $IP_INFO{$receiver}{ ...
           success}";
    }
    &logmsg("Received usr1 signal. Diagnostic info:", $diagnostics) ...
       ;
};
# This one signifies the child process is done; set to REAPER only  ...
   upon debug
# because it exits erroneously if several files are evaluated at  ...
   once.
```

```
340 if ($debug > 1) {
        $SIG{CHLD} = \&REAPER;
    } elsif ($debug) {
        &logmsg("Ignoring \$SIG{CHLD} signals.");
        $SIG{CHLD} = 'IGNORE';
345 #} else { $SIG{CHLD} = sub { $children -= 1; } }
    } else { $SIG{CHLD} = 'IGNORE' }
```

This is the start of the main loop, which will cycle until the process is terminated. The loop cycles after a keep-alive signal is transmitted to each receiver. The loop begins by resetting the number of recognized children processes to zero and checking the INBOX directory for messages.

```
347 ########################
    #####  Cycle every second, looking for incoming mail
    while ( sleep 1 or sleep 2 ) {
350     ####################
        #####  Re-cycle as long as we have children processes (try not  ...
           to clobber
        #####  the socket communication)
    #    next if $children;
        ### Start with baseline-- 0 children (get incremented handling  ...
           each file)
355     $children=0;
        ####################
        #####  Read contents of inbox directory
        opendir INBOX, $inbox
            or &fatal("Could not read directory $inbox: $!", 1);
360     my @allfiles = readdir INBOX;
        close INBOX;
        print "Files in $inbox: @allfiles\n" if ($verbosity and $debug) ...
           ;
```

If any files exist in the INBOX directory, a sub-loop is started which cycles through all the message files. Any hidden files are ignored, while files which do not end in "notice" generate an email message, terminating the process. Only files ending in "notice" are considered as potential alerts and are copied to the OUTBOX/`unparsed` directory. The "From:" line is compared with the list of accepted senders. If the sender does not match any of the accpeted senders, the file is deleted from both the INBOX and the OUTBOX. In particular, if the message file is a self-addressed message or an error from the sendmail daemon, an entry is made in the LOGFILE before the file is deleted.

```
364     ####################
365     #####  If there are any notices, start a handler script
        foreach $infile (@allfiles) {
            my $success=1;
            ###  Ignore any hidden files/directories
            if ($infile =~ /^\.+/) {
370             &logmsg("Encountering hidden file: $infile") if ($debug ...
                    > 1);
```

```
            next; #  foreach $infile (@allfiles) ...
        ###  Handle any recognized files
        } elsif ($infile =~ /notice$/) {
            ### Copy (link) the new file to the OutBox
375         eval { link "${inbox}/${infile}","${outbox}/unparsed/${ ...
               infile}"; }
                or &fatal("Could not link file $inbox/$infile: $!", ...
                    1);

            ### Read the file into an array; check for originator &
            ### subject information; this info assigns the handler
380         open(NOTICE, "${outbox}/unparsed/${infile}");
            my @info = <NOTICE>;
            close NOTICE;
            my %notice = map /^(Date|From|Subject): (.+)/, @info;
#            print "Notice: @notice\n"
385         print "Notice from $notice{'From'} re: $notice{'Subject ...
               '} at $notice{'Date'}\n"
                if $verbosity;

            ### Catch all emails not originating from an accepted  ...
               sender address
            unless (scalar(grep { $notice{'From'} =~ /$_/ }  ...
               @accepted_senders) ) {
390         # Also, just delete messages from MAILER-DAEMON or  ...
               myself, which indicate email problems
                ($notice{'From'} =~ /(MAILER-DAEMON|wk_sz\@ligo\. ...
                   caltech\.edu)/) ?
                    &logmsg("Deleting returned/self-addressed  ...
                       message: $infile") :
                    &fatal("Received unauthorized email; Subject:  ...
                       $notice{'Subject'}", -2);
                unlink "${outbox}/unparsed/${infile}", "${inbox}/${ ...
                   infile}"
395                 or &fatal("Could not delete $infile: $!.", 1);
                next; #  foreach $infile (@allfiles) ...
            }
```

If the message is from an accepted sender, the "Subject:" line is checked against several regular expressions. Once a match is found, the process is forked, and the child process runs the appropriate handler module on the confirmed alert (cf. A.2). If no match is found, an email is sent to the PARSING SCRIPT ADMINISTRATOR, containing the subject line of the message, indicating the message is unparseable. Whether or not a match is found, the parent process deletes the message file in the INBOX. The handler modules are responsible for deleting successfully parsed messages from the OUTBOX.

```
398         ### Spawn an appropriate handler
            $_ = $notice{'Subject'};
```

```
400             SWITCH: {
    #:NOTE: 2005.04.12 RJR-- I've expanded the list of emails which get ...
        deleted automatically,
    #:NOTE: including Swift notices of pointing changes or lightcurve  ...
       information.
                    (/^BACODINE_POSITION/ or
                     /^GCN\/SWIFT_\w{3,4}_(LIGHTCURVE|IMAGE|SRC)/ or
405                  /^GCN\/SWIFT_(POINTING_DIR|SC_SLEW|FOM_OBSERVE)/)  ...
                        and do {
                        $success &= &spawn(\&handle_NOPARSE, $infile,\% ...
                           notice);
                        $children +=1;
                        last SWITCH;
                    };
410                 /^GCN\/HETE_POSITION/ and do {
                        $success &= &spawn(\&handle_HETE, $infile, \% ...
                           notice);
                        $children +=1;
                        last SWITCH;
                    };
415                 /^GCN\/IPN_POSITION/ and do {
                        $success &= &spawn(\&handle_IPN, $infile, \% ...
                           notice);
                        $children +=1;
                        last SWITCH;
                    };
420                 /^GCN\/INTEGRAL_POSITION/ and do {
                        $success &= &spawn(\&handle_INTEGRAL, $infile,  ...
                           \%notice);
                        $children +=1;
                        last SWITCH;
                    };
425                 /^GCN\/SWIFT_(BAT|XRT|UVOT)_(ALERT|POSITION)/ and  ...
                       do {
                        $success &= &spawn(\&handle_SWIFT, $infile, \% ...
                           notice);
                        $children +=1;
                        last SWITCH;
                    };
430                 &logmsg("Unable to identify file OutBox/unparsed/ ...
                       $infile");
                    &fatal("Can't parse email-- Subject: $notice{' ...
                       Subject'}", -2);
                    $success &= 1;
                    last SWITCH;
                }
435
```

```
            ###  For unrecognized files, produce a fatal error
            } else {
                &fatal("Improper filename ($infile) in $inbox", 1);
            }
            print "$infile spawned successfully? $success\n" if $debug;

        ####################
        ###  Delete the file if parsed successfully; otherwise log the  ...
           error
            if ($success) {
                &logmsg("Deleting file from InBox: $infile");
                unlink "${inbox}/${infile}"
                    or &fatal("Could not delete $infile: $!.", 1);
                next; # foreach $infile (@allfiles) {
            }
            else { &logmsg("Could not parse file: $infile") }
        } # end of foreach $infile (@allfiles) {
```

Once all the message files from the INBOX have been deleted, a keep-alive signal is sent to the receiver scripts running at the detector sites. The keep-alive signal is in the same format as a trigger (cf. 3). If any sites are currently in timeout, the corresponding `timeout` field of `%IP_INFO` is decremented and no keep-alive signal is sent. Otherwise, a timed subroutine is run which sends the keep-alive signal to the site (using `&notify_ops`. If the timer expires before the keep-alive signal is sent, the subroutine exits abnormally with a timeout error. The timeout error is recorded in the LOGFILE. The duration of the IP communication is printed on standard out if indicated by the `TIME_TCP` constant declared at the beginning of the script.

```
        ####################
        ###  Send keep-alive signal when finished
        # wait for children processes to start up and handle socket  ...
           communication
#    sleep $children if $children;
        my @ops_msg  = (1);
        my @ta       = gmtime();
        push @ops_msg, sprintf("UTC-%4d-%02d-%02d-%02d:%02d:%02d",  ...
           (1900 + $ta[5]),
                               (1 + $ta[4]), $ta[3], $ta[2], $ta[1],  ...
                                  $ta[0]);
        push(@ops_msg, "UTC-0000-00-00-00:00:00");

        ### Loop over each receiver (e.g. LHO, LLO, GEO, CIT, etc.)
    ###:NOTE: RJR 2005.11.21 -- trying to see if LLO is dropped out  ...
       because it's last in the IP list.
    #    foreach $receiver (sort keys %IP_INFO) {
        foreach $receiver ('LLO','LHO','GEO') {
            # check for a hold on a particular receiver before  ...
               proceeding
            if ($IP_INFO{$receiver}{timeout}) {
```

```
            --$IP_INFO{$receiver}{timeout};
        } else {
            eval {
                local $SIG{ALRM} = sub {
                    $IP_INFO{$receiver}{timeout} =  ...
                       KEEPALIVE_TIMEOUT;
                    die "Timeout over $receiver socket $IP_INFO{ ...
                       $receiver}{ip}:" .
                        "$IP_INFO{$receiver}{port}";
                };
                ### Send a keep-alive signal to receiver
                alarm ALARM;
                my $start_time = [gettimeofday];
                $IP_INFO{$receiver}{timeout} = IP_TIMEOUT,
                $IP_INFO{$receiver}{success} = -1 if
                    &notify_ops( $IP_INFO{$receiver}{ip},
                                 $IP_INFO{$receiver}{port},
                                 $IP_INFO{$receiver}{contact}, \ ...
                                    @ops_msg );
                print "Time to send keep-alive to $IP_INFO{ ...
                   $receiver}{ip}: ",
                    &tv_interval($start_time), "\n" if TIME_TCP;
                alarm 0;
            };                  # end of eval {

            # Log timeout errors; exit on any other error from eval ...
               {} statement
            if ($@) {
                chomp $@;
                ($@ =~ /Timeout over \w+ socket/) ?
                    &logmsg("$@ while sending keepalive, after",
                        $IP_INFO{$receiver}{success}, "successes;  ...
                           counting down from",
                        $IP_INFO{$receiver}{timeout}) :
                    &fatal("Syntax error over socket while sending  ...
                       keepalive\n$@",
                           4, $IP_INFO{$receiver}{contact});
                $IP_INFO{$receiver}{success} = 0;
            } else {
                # Tally the number of consecutive successful  ...
                   connections.
                $IP_INFO{$receiver}{success} = $IP_INFO{$receiver}{ ...
                   success}+1;
                &fatal("Sent keep-alive : $ops_msg[0] $ops_msg[1]  ...
                   $ops_msg[2]",
                    -2, $IP_INFO{$receiver}{contact}) if $debug  ...
                       > 1;
```

```
            }                       # end of if ($@) {
505       }                         # end of if ($IP_INFO{$receiver}{ ...
             timeout}) {
    }                               # end of foreach $receiver (sort  ...
       keys %IP_INFO) {
} # end of while ( sleep 1 or sleep 2 ) {
```

Control of the parsing script should never go outside the `while()` loop. The following guarantees that the script terminates with some debug information passed to standard out.

```
508 &fatal("Passed through while loop. Some important variables: \$\@: ...
      $@, \$!:$!, \$?:$?, \$\$:$$ ...",
          0, $remote_contacts);
510 exit 0;
```

The `&get_timestamp()` subroutine follows. It takes as input a timestamp in one of the recognized formats and returns the equivalent GPS timestamp. It first fills a Perl time array variable (`@ctime`) with time fields obtained from the input timestamp. Next it writes a standard format timestring (`$tmptime`) and passes it to the LIGOTools `tconvert` function.

```
511 ##############################################################
    ####  get_timestamp(timestring) subroutine
    ##############################################################
    sub get_timestamp {
515     my ($timezone, $partial_seconds, $time_out, $time_fmt);
        my @ctime = ("undef") x 9;
        my $timestamp = shift;
        print "Parsing timestamp: $timestamp\n" if ($verbosity and  ...
           $debug);
        ### Most times are in hh:mm:ss(.ss) format
520     if ($timestamp =~ m"(\d{1,2}):(\d{1,2}):(\d{1,2})(\.\d+|)") {
            # arranging array according to localtime() output:
            # ($sec,$min,$hour,$mday,$mon,$year,$wday,$yday,$isdst)
            ### :NOTE: RJR 2003.12.05 We'll use $partial_seconds at the ...
                end
            ### because gmtime() deals only with integer seconds.
525         $partial_seconds = ($4) ? $4 : 0;
            $ctime[0] = $3 + $partial_seconds;
            $ctime[1] = $2;
            $ctime[2] = $1;
        }
530     ### Recognized timezones are [ECMP][SD]T, UTC?,  ...
           [+\-][012][0-9]00
        ### (We're going to shift the time to UTC after executing  ...
           timegm())
        $timezone = $1 if ($timestamp =~ m"\b([ECMP][SD]T)\b" or
                           $timestamp =~ m"\b(UTC?)\b" or
                           $timestamp =~ m"\d+:\d+:\d+\s ...
                              +\+?(\-?[012][0-9]00)\b");
535     ### Recognized dates are:
```

```
        ###   (DAY,?|) DD MONTH YYYY
        ###   (YY)?YY/MM/DD
        ###   MM/DD/YY(YY)?
        ###   YYYY.MM.DD
        # Parse (DAY,?|) DD MONTH (YY)YY date string
        if ($timestamp =~ m"(\w{2,9}|),? (\d+) (\w{3,9}) (\d{2}|\d{4})  ...
           ") {
            my %test = ("month" => $3, "day_of_week" => $1);
            $ctime[3] = $2;
            $ctime[4] = 0;
            my @test = qw(Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov  ...
               Dec);
            foreach (@test) { ($test{month} =~ /$_/) ? last : $ctime ...
               [4]++ }
            $ctime[5] = ($4 < 1900) ? ($CENTURY - 1900 + $4) : $4 -  ...
               1900;
            ### :NOTE: RJR 2003.12.05 Actually, the following effort  ...
               gets reversed
            ### when elements [6..8] of @ctime are set to 0.
            $ctime[6] = 0;
            @test = qw(Sun Mon Tue Wed Thu Fri Sat);
            foreach (@test) { ($test{day_of_week} =~ /$_/) ? last :  ...
               $ctime[6]++ }
        }
        # Only determines YY/MM/DD if "TJD;" is found (GRB_DATE string)
        elsif ($timestamp =~ /TJD;/ and $timestamp =~ m"(\d+)/(\d+)/(\d ...
           +)") {
            $ctime[3] = $3;
            $ctime[4] = $2 - 1;
            $ctime[5] = ($1 < 100) ? ($CENTURY - 1900 + $1) : $1 -  ...
               1900;
        }
        # Parse MM/DD/YY(YY)? date string
        elsif ($timestamp =~ m"(\d+)/(\d+)/(\d+)") {
            $ctime[3] = $2;
            $ctime[4] = $1 - 1;
            $ctime[5] = ($3 < 100) ? ($CENTURY - 1900 + $3) : $3 -  ...
               1900;
        }
        else {
            &logmsg("get_timestamp: Can't determine time format for:  ...
               $timestamp");
        }

        ####################
        ### Check that the relavant fields in @ctime are filled before  ...
           continuing
```

```
    if (grep /undef/, @ctime[0..5]) {
        &logmsg("get_timestamp: Can't parse timestamp ($timestamp)  ...
           ").
            "into ctime array (" .(join ' ',@ctime) .")";
575     $time_out = 0;
        $time_fmt = "unknown";
        return ($time_out, $time_fmt);
    } else {
        # In order to run timegm, we need to clear the rest of the  ...
           fields
580     @ctime[6..8] = (0,0,0);

        ######################
        ### Convert the time to UTC using $timezone
        #print "\$timezone: $timezone\n" if ($verbosity and $debug) ...
           ;
585     if ($timezone =~ /^\-?\d{4}$/) {
            $ctime[2] -= int($timezone / 100);
        } elsif ($timezone =~ /^[ECMP][SD]T$/) {
            $ctime[2] += grep /$timezone/,
                         (0,0,0,0,"EDT","EST|CDT","CST|MDT","MST| ...
                            PDT","PST");
590     } elsif ($timezone =~ /^UTC?$/) {
            # It's okay, we don't have to do anything here
        } else {
            # It's not okay-- what could the timezone be?
            &logmsg("get_timestamp: Can't understand timezone  ...
               $timezone");
595         $time_out = 0;
            $time_fmt = "unknown";
            return ($time_out, $time_fmt);
        }
    }
600

    ######################
    ### Parse the timestamp into (scalar localtime()) format
    print "\@ctime yields (", (join ',', @ctime),")\n"        if ( ...
       $verbosity and $debug);
605 ### :NOTE: RJR 2003.12.05 The following method doesn't quite  ...
       work because
    ### we don't get the fractions of a second afterward.  We'll  ...
       add the
    ### fraction after checking the GPS time.
    my $tmptime = scalar gmtime(timegm_nocheck(@ctime));
    print "Parsed timestamp: $tmptime\n" if ($verbosity and $debug) ...
       ;
```

```
610
        #    print "Initial value of \$!: $!\n";
        ### We should probably change the response to $SIG{CHLD} here  ...
           before
        ### forking a system call process
        eval {
615 #        local $SIG{CHLD} = 'IGNORE';
            local $SIG{CHLD} = 'DEFAULT';
            $tmptime   = `$TCONVERT $tmptime`;
        };
        &logmsg("get_timestamp: eval {tconvert \$tmptime}; returned $@ ...
           ") if $@;
620
        # :NOTE: RJR 2003.12.05 Unfortunately, I can't find a way to  ...
           successfully
        # check the return code for TCONVERT.  It looks like currently  ...
           the
        # backticks are returning -1 : "No child processes".
        if ($tmptime and $tmptime =~ /(\d{9,10})/) {
625         # :NOTE: RJR 2003.12.05 Remember to add partial_seconds on, ...
                as
            # gmtime() deals only with integer seconds.
            $time_out = $1 + $partial_seconds;        $time_fmt = "GPS ...
               ";
        } else { $time_out = timegm_nocheck(@ctime); $time_fmt = " ...
           unknown"; }

630     # Send a fake time; we're just testing the code here
        $time_out = 729000000, $time_fmt = "GPS" if $debug > 1;
        return ($time_out, $time_fmt);
    }
```

The `&check_times()` subroutine follows. It finds the difference between the event timestamp and the date of the email and compares this value to the stand-down interval corresponding to the type of event (found in `%EVENT_TABLE`). If the email was sent during the defined stand-down interval, an alert signal is sent to each of the sites, in the same manner that the keep-alive signal was sent above. The response from each site is recorded in the LOGFILE.

```
634 ############################################################
635 ####  check_times() subroutine
    ####  Compares email_ and event_ timestamps (if possible) to see if ...
        the email
    ####  has been delayed too long to be noteworthy.
    ####  :NOTE: It's difficult to use the current (computer) time,  ...
       because the
    ####  :NOTE: computer time may have been incorrectly set.
640 ############################################################
    sub check_times {
```

```
        ####################
        ###  Check whether timestrings are compareable
645     if ($time_info{email_timefmt} ne "GPS" or
            $time_info{event_timefmt} ne "GPS")   {
            &logmsg("check_times: unknown time format -- cannot compare ...
                times.");
            return 3;
        }
650     my $time_diff = int( $trig_info{email_timestamp} -
                             $trig_info{event_timestamp}  );
        ####################
        ###  Look for the detector information in %EVENT_TABLE
        my $event_ref;
655     if (exists $EVENT_TABLE{$det_info{det_name}}) {
                $event_ref = $EVENT_TABLE{$det_info{det_name}};
        } elsif ($trig_info{event_type} =~ /GRB/) {
                $event_ref = $EVENT_TABLE{GRB};
        } elsif ($trig_info{event_type} =~ /SN/) {
660             $event_ref = $EVENT_TABLE{SN};
        } else { $event_ref = $EVENT_TABLE{OTHER}; }
        print "@{$event_ref}\n" if $verbosity;

        ####################
665     ###  Check if email timestamp is sufficiently close to event  ...
           timestamp
        if ($time_diff < $$event_ref[2]) {
            &logmsg("check_times: Notifying operators of $$event_ref[1] ...
                event at GPS " .
                    "$trig_info{event_timestamp}.");

670     ### Put together the TCP/IP message
            my $tmp_time  = int( $trig_info{event_timestamp} );
            my @ops_msg   = ($$event_ref[0]);
            eval {local $SIG{CHLD} = 'IGNORE'; $tmp_time = `$TCONVERT  ...
               $tmp_time`};
            &logmsg("check_times: eval (tconvert \$tmptime); returned  ...
               $@") if $@;
675
            if ($tmp_time =~ /^(\w+) (\d+) (\d+) ([\d:]+) (UTC)/) {
                my $mo =
                    int((index "JanFebMarAprMayJunJulAugSepOctNovDec",  ...
                       $1)/3 +1);
                push @ops_msg, sprintf("%s-%d-%02d-%02d-%s", $5, $3,  ...
                   $mo, $2, $4)
680         } else { &logmsg("check_times: unable to parse $tmp_time")  ...
               }
```

```
$tmp_time     = int ($trig_info{event_timestamp} +  ...
   $$event_ref[2]);
eval {local $SIG{CHLD} = 'IGNORE'; $tmp_time = `$TCONVERT  ...
   $tmp_time`};
&logmsg("check_times: eval (tconvert \$tmptime); returned  ...
   $@") if $@;

if ($tmp_time =~ /^(\w+) (\d+) (\d+) ([\d:]+) (UTC)/) {
    my $mo =
        int((index "JanFebMarAprMayJunJulAugSepOctNovDec",  ...
           $1)/3 +1);
    push @ops_msg, sprintf("%s-%d-%02d-%02d-%s", $5, $3,  ...
       $mo, $2, $4)
} else { &logmsg("check_times: unable to parse $tmp_time")  ...
   }
print "Ops message: ", (join ' ', @ops_msg), "\n" if  ...
   $verbosity;

# $confirmation-- the list of servers which have  ...
   acknowledged the event.
my $confirmation = "";
### Loop over each receiver (e.g. LHO, LLO, GEO, CIT, etc.)
foreach $receiver (sort keys %IP_INFO) {
    # check for a hold on a particular receiver before  ...
       proceeding
    if (!$IP_INFO{$receiver}{timeout}) {
        my $start_time = [gettimeofday];
        eval {
            local $SIG{ALRM} = sub {
                die "Timeout over $receiver socket ",  ...
                   $IP_INFO{$receiver}{ip}, ":",
                    $IP_INFO{$receiver}{port};
            };
            ### Notify receiver of event; set timeout on  ...
               error
            alarm ALARM;
            $IP_INFO{$receiver}{timeout} = IP_TIMEOUT,
            $IP_INFO{$receiver}{success} = -1 if
                &notify_ops( $IP_INFO{$receiver}{ip},
                             $IP_INFO{$receiver}{port},
                             $IP_INFO{$receiver}{contact},  ...
                                \@ops_msg );
            print "Time to send event_$ops_msg[0] to  ...
               $IP_INFO{$receiver}{ip}: ",
                &tv_interval($start_time), "\n" if TIME_TCP ...
                   ;
            alarm 0;
```

```
715                 };                  # end of eval {
                    # Tally the number of consecutive successful  ...
                       connections.
                    $IP_INFO{$receiver}{success} = ($@) ? 0 : $IP_INFO{ ...
                       $receiver}{success}+1;
                    # Exit on any error from eval{} statement, since  ...
                       this is a child process
                    if ($@) {
720                     chomp $@;
                        ($@ =~ /Timeout over \w+ socket/) ?
                            &fatal("$@ while sending $$event_ref[1]  ...
                               trigger",
                                   -3, $IP_INFO{$receiver}{contact}) :
                            &fatal("Syntax error over socket while  ...
                               sending $$event_ref[1] trigger\n$@",
725                                4, $IP_INFO{$receiver}{contact});
                    } else {
                        $confirmation .= "\n\tTime to notify ${receiver ...
                           }: " . &tv_interval($start_time);
                    }                   # end of if ($@) {
                # Report whether a receiver was in timeout during an  ...
                   event
730             } else {
                    &logmsg("check_times: receiver",$IP_INFO{$receiver ...
                       }{ip},
                            "was in timeout during event.  Timeout  ...
                               count:",
                            $IP_INFO{$receiver}{timeout});
                }                       # end of if (!$IP_INFO{$receiver}{ ...
                   timeout}) {
735         }                           # end of foreach $receiver (sort  ...
               keys %IP_INFO) {
            if ($confirmation) {
                &fatal("Sent $trig_info{notice_type} $$event_ref[1]  ...
                   notice:\n$ops_msg[0] $ops_msg[1] " .
                       "$ops_msg[2]\n$trig_info{notice_comments}\n\ ...
                          nServers responding:$confirmation",
                       -4, $email_event)
740                 unless $debug;
            } else {
                &fatal("Could not send $trig_info{notice_type}  ...
                   $$event_ref[1] notice:\n$ops_msg[0] " .
                       "$ops_msg[1] $ops_msg[2]\n$trig_info{ ...
                          notice_comments}\n\nNo servers responded.",
                       -4, $email_event)
745                 unless $debug;
            }    # end of if ($confirmation) {
```

```
        # Email delay was too long
        } else {
750         &logmsg("check_times: Will not notify operator of event  ...
               $trig_info{event_type}: " .
                    "email sent $time_diff seconds after event.");
        }        # end of if ($time_diff < $$event_ref[2]) {


755     return 0;
    }
```

The `&notify_ops()` subroutine follows. It is responsible for managing the IP communication. It first opens a socket connection with the given IP and port. Then it spawns a child process which is responsible for sending the TCP/IP message. The parent process listens on the connection for an acknowledgement and returns this to the calling function. The child process normally exits on its own. Any errors are sent to both the PARSING SCRIPT ADMINISTRATOR and the relevant RECEIVER SCRIPT ADMINISTRATOR.

```
757 ############################################################
    ####  notify_ops(\@tokens) subroutine
    ####  Exit codes:
760 ####   0: Normal termination
    ####   1: Function bypassed because $NOTIFY_OPS is unset or zero
    ####   2: Function was unable to initiate/complete socket  ...
       communication
    ############################################################
    sub notify_ops {
765     my $remote_ipaddr  = shift;
        my $remote_port    = shift;
        my $remote_contact = shift;
        my $ref_tokens     = shift;
        my ($sock_time, $kidpid, $read_data);
770
        &logmsg("notify_ops: Will not send message @$ref_tokens to  ...
           $remote_ipaddr"),
        return 1
            unless ($NOTIFY_OPS);

775     ######################
        ###  Open socket to server
        &logmsg("notify_ops: Opening socket to $remote_ipaddr") if  ...
           $debug;
        $sock_time = scalar times;
        my $sock = new IO::Socket::INET ( PeerAddr => $remote_ipaddr,
780                                       PeerPort => $remote_port,
                                          Proto    => 'tcp',
                                          )
```

```
            or &logmsg("Could not create socket to $remote_ipaddr"),
                &fatal("Could not create socket to $remote_ipaddr", -3, ...
                    $remote_contact),
785             return 2;

        $sock->autoflush(1);

        ####################
        ###  Fork socket for read/write
790     &fatal("Can't fork socket comm. $!", 3, $remote_contact) unless ...
            defined($kidpid = fork());

        ### Parent is in charge of reading data from server
        if ($kidpid) {
795         # do domething with server messages
            while (defined($read_data = <$sock>)) {
                chomp $read_data;
                &logmsg("notify_ops: $remote_ipaddr: $read_data")
                    if ($$ref_tokens[0] != 1 or $debug > 1);
800         }
            kill("TERM", $kidpid);

            $sock_time = scalar times - $sock_time;

805     ### Child is in charge of sending data to server
        } else {
            $am_child = 1;
            &logmsg(sprintf("notify_ops: sending %s\n", (join ' ', ...
               @$ref_tokens)))
                if $debug;
810         print $sock (join ' ',@$ref_tokens), "\n";
            close($sock);
            exit 0;
        }

815     close($sock);
        &logmsg("notify_ops: Closed socket connection") if $debug;
        return;
    }
```

The `&send_trigger()` subroutine follows. This subroutine is called from the handler modules to make an entry in the LIGO Lightweight trigger file. The entry is essentially a dump of the `%trig_info` hash variable contents into a file. All fields are written as quoted strings with the following exceptions. The timestamps are verified to be in GPS format and written in *second* and *nanosecond* integer format. Positions are written as signed ten-character decimal numbers. Comments from the email alerts are truncated to forty characters in the trigger file (however, they are still printed in full in the parsed notice file).

```
819 ###############################################################
820 ####  send_trigger() subroutine
    ####  Sends the trigger information to the metadatabase.
    ####  Currently unused
    ###############################################################
    sub send_trigger {
825     my $success   = 1;

        &logmsg("send_trigger: will not send $infile trigger to  ...
           $TRIGFILE"),
        return 1
            unless $SEND_TRIGGER;
830
        &logmsg("send_trigger: writing $infile trigger to $TRIGFILE");
        open(TRIGGER, ">>${outbox}/$TRIGFILE")
            or &fatal("Can't open OutBox/$TRIGFILE for appending.", 1);

835     ####################
        ### print values for each key in %trig_info
        foreach $tmpkey ( sort keys %trig_info ) {
            # print event_timestamp as <start_time>,<stop_time>  ...
               integers
            if ( $tmpkey =~ /^event_timestamp$/ and
840              $trig_info{event_timefmt} eq "GPS" ) {
                my @tmpval = (int($trig_info{$tmpkey}));
                push @tmpval,
                int(($trig_info{$tmpkey} - $tmpval[0]) * 1000000000);
                printf TRIGGER '%d,%d,', @tmpval;
845             next;
            # print timestamps as integers
            } elsif ( $tmpkey =~ /(\w+)_timestamp/ ) {
                ($trig_info{"$1_timefmt"} eq "GPS") ?
                    printf TRIGGER '%d,', $trig_info{$tmpkey} :
850                 printf TRIGGER '%d,', 0;
                next;
            # ignore time formats
            } elsif ( $tmpkey =~ /(\w+)_timefmt/ ) {
                next;
855         # print location information as real numbers
            } elsif ( $tmpkey =~ /_(dec|ra|z|ele|lat|long)/ ) {
                ($trig_info{$tmpkey}) ?
                    printf TRIGGER '%10.5e,', $trig_info{$tmpkey} :
                    printf TRIGGER '%10.5e,', 0.0;
860             next;
            # limit string lengths for comments and urls
            } elsif ( $tmpkey =~ /_(comments|url)/ ) {
```

```
                $trig_info{$tmpkey} =~ s/,/\\,/;        ### LIGO-LW  ...
                   requires commas to be escaped
                (printf TRIGGER '"%-40s",', substr($trig_info{$tmpkey ...
                   },0,40)), next
865                 unless length($trig_info{$tmpkey}) < 40;
            }
            # print everything else as strings
            printf TRIGGER '"%s",', $trig_info{$tmpkey};
        }
870     # terminate the trigger line
        print TRIGGER "\n";

        close TRIGGER
            or &fatal("Can't close OutBox/$TRIGFILE.", 1);
875     return $success;
    }
```

The `&spawn()` and `&REAPER()` subroutines follows. The `&spawn()` subroutine starts a child process and passes it a block of code to execute. The child process exits when it finishes processing the code block, sending a `CHLD` signal back to the parent process. This signal causes the parent process to call the `&REAPER()` subroutine, which currently only determines the child's process ID.

```
877 ##############################################################
    ####  spawn(coderef, infile, hashref) subroutine
    ####  Spawns a child process which handles the email
880 ##############################################################
    sub spawn {
        ###############
        ###  Check calling arguments
        print "@_\n" if ($verbosity and $debug > 1);
885     my $coderef = shift;
        my @argref  = @_;
        unless ($coderef && ref($coderef) eq 'CODE') {
            &fatal("Spawn process called with incorrect arguments:  ...
               $coderef, @_", 3);
        }
890
        ###############
        ###  Fork the process
        my $pid;
        if (!defined($pid = fork)) {
895         &fatal("Cannot fork handler script", 3);
            return 0;
        } elsif ($pid) {
            &logmsg("Spawned child PID $pid") if $debug;
            return 1;
900     } else { $am_child = 1 }
```

```
        ###############
        ###  Only the child reaches this point
        exit &$coderef(@argref);
905 }

    sub REAPER {
        $waitedpid = wait;
        #$SIG{CHLD} = \&REAPER; #  if you don't have sigaction(2)
910     &logmsg("Reaped  child PID $waitedpid with exit $?") if $debug;
    }
```

The `&fatal()` subroutine follows. This subroutine is responsible for sending all diagnostic and event confirmation emails, as well as possibly terminating the script. The subroutine is given a message to include in the email, a code identifying the subject of the email, and an optional list of blind carbon copy recipients. The codes are described in the listing below. Most importantly, for codes below -1, control will return back to the calling function; and codes at or above -1 will cause the script to terminate. `&fatal()` depends on the existence of a mail process handler, in this case `/usr/lib/sendmail`.

```
912 ############################################################
    ####  fatal(msg, err) subroutine
    ####  Sends email to appropriate users regarding the nature of
915 ####  the failure.
    ####  Some exit codes:
    ####   -4 - Not fatal; GRB message sent to control rooms
    ####   -3 - Not fatal; receiving socket is not available
    ####   -2 - Not fatal; diagnostic/debug messages
920 ####   -1 - Existing process id
    ####    0 - Normal or user-prompted termination
    ####    1 - File or directory I/O error
    ####    2 - Parse error
    ####    3 - Fork error
925 ####    4 - Socket syntax error
    ############################################################
    sub fatal {
        local $SIG{CHLD} = 'DEFAULT';
        my $msg  = shift;
930     my $code = shift;
        my $bcc  = (scalar @_) ? shift : "";
        $bcc     = ($bcc) ? "\nBcc: $bcc" : "";
        my $prefix    = "[$0 ". (scalar gmtime) ." GMT]:";
        my $ch_pid    = ($am_child) ? "(child process $$)" : "(process  ...
           $$)";
935     my %err_codes = ( -4 => "GRB confirmation message",
                          -3 => "TCP/IP warning",
                          -2 => "Diagnostic message",
                          -1 => "Existing PID",
                           0 => "Normal termination",
```

```
940                         1 => "File or directory I/O error",
                            2 => "Parse error",
                            3 => "Fork/Child error",
                            4 => "TCP/IP error"
                          );
945     $msg     = ($code < -1) ? $msg : "Process terminated.\n$msg";
        ( ($debug) ? open(SENDMAIL, ">&STDOUT") :
          open(SENDMAIL, "|/usr/lib/sendmail -t") )
            or die "Can't fork for sendmail: $!\n";
        print SENDMAIL <<"EOF";
950 From: External Trigger Subgroup <wk_sz\@acrux.ligo.caltech.edu>
    To: $email_to$bcc
    Subject: $err_codes{$code} in $0
    $prefix $msg $ch_pid
    EOF
955     close(SENDMAIL) or warn "sendmail didn't close nicely; exit  ...
           status $? (error $!)";

        ### Don't need to exit if sending a diagnostic message.
        if ($code < -1) { return 0 }
        ### Remove PID file (if not terminated with "Existing PID"  ...
           error)
960     unlink "$ENV{HOME}/.$0-pid" unless ($am_child or $code < 0);
        ### Finish logfile
        &logmsg("^^^^===============  Ending $0") unless $am_child;
        ### Exit with appropriate code
        exit $code;
965 }
```

A.2 A walkthrough of the parsing script's handlers

Below is a listing of the handler modules for the parsing script. It currently consists of five subroutines. The first subroutine (`handle_NOPARSE`) deletes GCN emails which are either sent for testing purposes or have no use in alerting the control room to detected events. These emails are identified by their subject line.

```
1 ############################################################
  ####  handle_NOPARSE(infile, hashref) subroutine
  ####  This handles messages from GCN which are not meant to be  ...
     parsed.  It just
  ####  deletes the file from OutBox and returns.
  ############################################################
6 sub handle_NOPARSE {
      my $success = 1;
      my $infile  = shift;
      my $hashref = shift;
      if ($$hashref{Subject} =~ /BACODINE_POSITION/
```

```
        or $$hashref{Subject} =~
         /SWIFT_(\w{3,4}_LIGHTCURVE|\w{3,4}_IMAGE|\w{3,4}_SRC| ...
            POINTING_DIR|SC_SLEW|FOM_OBSERVE)/) {
        &logmsg("Deleting file from OutBox/unparsed: ${infile}  ...
           Subject: $$hashref{Subject}")
         if $debug;
        unlink "${outbox}/unparsed/${infile}"
            or &fatal("Can't unlink ${outbox}/unparsed/${infile ...
               }",1);
    }
    return $success;
}
```

The next handler parses the HETE alerts. There are four kinds of HETE alerts which are parsed by this module, but they are not distinguishable from the subject line. The module starts by loading variables and opening the HETE alert file.

```
############################################################
####  handle_HETE(infile, hashref) subroutine
####  This handles HETE position notices.  It records various
####  information from the notice and puts it into a hash array
####  (%trig_info) for dumping to the OutBox/parsed-GCN_alerts/
####  subdirectory.
####
####  It also checks the event time to see if operators should
####  be notified.
############################################################
sub handle_HETE {
    my (@notice, $i, $infile, $hashref, $success);
    $success    = 1;
    $infile     = shift;
    $hashref    = shift;
    &logmsg("Parsing file from OutBox/unparsed: ${infile}") if  ...
       $debug;
    open(NOTICE, "${outbox}/unparsed/${infile}")
        or &fatal("Can't open OutBox/unparsed/$infile for reading", ...
            1);
    @notice = <NOTICE>;
    $i = 0;
    close NOTICE
        or &fatal("Can't close OutBox/unparsed/$infile.", 1);
    print "@notice" if ($verbosity and $debug);
    foreach $tmpkey (sort keys %trig_info) {
        print "\%trig_info{$tmpkey}\t: $trig_info{$tmpkey}\n"
            if ($verbosity and $debug > 2);
    }
```

The module loops through the lines of the email, passing through five different sections. The loop consists of a series of tests, trying to match the current line with a regular expression. Once

the line is matched, the data is passed to a hash variable, and control returns to the top of the loop with the next line of the email.

The first section deals with timestamps from the `Date:`, `NOTICE_DATE:` and `GRB_DATE:` and `GRB_TIME:` entries. Times are entered into the LIGO Lightweight table in GPS format, so these entries must be converted into that format, using the `get_timestamp()` subroutine. The `GRB_DATE:` and `GRB_TIME:` entries get combined before being passed to `get_timestamp()`.

```
47      while (($i < $#notice) and ($_ = $notice[++$i])) {
            chomp;
            ### For extra debugging, record the email.
50          &logmsg("  ${infile} | $_") if ($debug > 1);

            ######################
            ###  Handle timestamp entries
            # email_timestamp, email_timefmt
55          ($trig_info{email_timestamp}, $trig_info{email_timefmt}) =
                &get_timestamp($1), next if /^Date:\s+(.+)$/;
            # notice_timestamp, notice_timefmt
            ($trig_info{notice_timestamp}, $trig_info{notice_timefmt})  ...
               =
                &get_timestamp($1), next if /^NOTICE_DATE:\s+(.+)$/;
60          # event_timestamp, event_timefmt
            if (/^GRB_(DATE|TIME):\s+(.+)$/) {
                $tmpevent{"GRB_$1"} = $2;
                ($trig_info{event_timestamp}, $trig_info{event_timefmt ...
                   }) =
                    &get_timestamp("$tmpevent{GRB_DATE} $tmpevent{ ...
                       GRB_TIME}")
65                  if (defined $tmpevent{'GRB_DATE'} and
                        defined $tmpevent{'GRB_TIME'} );
                next;
            }
```

Next, the handler looks for entries identifying the type of notice (one of "HETE S/C_Alert", "HETE S/C_Update", "HETE S/C_Last", or "HETE Ground Analysis"), and other information pertaining to the email notice itself. A guess is made as to the URL of the notice on the GCN website.

```
69          ######################
70          ###  Handle notice entries
            # notice_type
            $trig_info{notice_type} = $1, next  if /^NOTICE_TYPE:\s ...
               +(.*)$/;
            # notice_id, notice_sequence
            $trig_info{notice_id} = $1,
75          $trig_info{notice_sequence} = $2, next
                if /^TRIGGER_NUM:\s+(\d+),\s+Seq_Num: (\d+)$/;
            # notice_comments
            if (/^COMMENTS:\s+(.*)$/) {
```

```
            my $tmp_comments = $1;
            while ( (++$i < $#notice) and
                    ($notice[$i] =~ /^(COMMENTS:|)\s+(.*)$/) ) {
                $tmp_comments .= " $2";
            }
            --$i;
            $trig_info{notice_comments} = $tmp_comments;
            next;
        }
        # Take a guess at notice_url
        $trig_info{notice_url} =
            "http://gcn.gsfc.nasa.gov/other/$trig_info{notice_id}. ...
               hete"
            if ($trig_info{notice_id} and !$trig_info{notice_url});
```

The handler next looks for matches on the spacecraft (a.k.a. the "observer"). HETE includes information about where the spacecraft is pointing, as well as the longitudinal position of the spacecraft itself. Since HETE's orbit is roughly about the equator, the latitude position is assumed to be 0. The module also has hardcoded the spacecraft's field of view. The altitude of the spacecraft, necessary for any calculations determining when the gamma-ray burst would arrive at the LIGO detectors, is not given.

```
        ######################
        ###  Handle observer entries
        # obs_fov_ra
        $trig_info{obs_fov_ra} = $1, next
            if /^SC_-Z_RA:\s+([\d\-\.]+)(\s+\[deg\]|)/;
        # obs_fov_dec
        $trig_info{obs_fov_dec} = $1, next
            if /^SC_-Z_DEC:\s+([\d\-\.]+)(\s+\[deg\]|)/;
        # HETE fov is essentially pi/2; we'll hardcode this in for  ...
           now
        $trig_info{obs_fov_ra_width} = 180; #degrees
        $trig_info{obs_fov_dec_width} = 90; #degrees
        # HETE is usually around the equater, so obs_loc_lat is 0.
        $trig_info{obs_loc_lat} = "0";
        # obs_loc_long (positive if east, negative if west)
        $trig_info{obs_loc_long} = $1, next
            if /^SC_LONG:\s+(\d+)\s+\[deg East\]/;
        $trig_info{obs_loc_long} = -1 * $1, next
            if /^SC_LONG:\s+(\d+)\s+\[deg West\]/;
        # No obs_loc_ele (yet)
```

Next, the handler looks at the detectors listed in the notice. Usually, the first detector to report a trigger is the FREnch GAmma-ray TElescope (FREGATE), although occasionally, the Wide-field X-ray Monitor (WXM) may trigger on an exceptionally loud X-ray flash. It is important to determine which detectors observed the burst, so the module goes to some effort to list all the detectors involved in observing the event. If a burst is only observed by FREGATE, there is a good probability that the event is not really a gamma-ray burst at all, but some observed source or some

noise. Also, the Soft X-ray Camera (SXC) has finer resolution than the WXM, and so can produce a smaller error circle for the position of the burst. The module accumulates the SNR of the burst in the detector, as well as the photon fluence and the energy-level for the trigger.

```
111         ######################
            ###  Handle detector entries; for HETE, this is WXM, SXC,  ...
               and FREGATE
            if (/^TRIGGER_SOURCE/) {
            # det_name, det_band
115             $trig_info{det_name} = "FREGATE";
            # det_band
                $trig_info{det_band} = $1 if /([\d\-]+ .?eV)/;
                next;
            }
120         # det_alts (preliminary)
            if (/^(WXM|SXC)/) {
                my $tmpdet = $1;
                push @dets, $tmpdet if (!@dets or grep !/$tmpdet/,  ...
                   @dets);
            }
125         # det_snr
            $trig_info{det_snr} = $1, next if (/SIG\/NOISE:\s+([\d\.]+) ...
               /);
            # det_peak, det_peak_int unavailable (yet)
            # det_fluence, det_fluence_int
            if (/^GAMMA_RATE/) {
130             $trig_info{det_fluence} = $1 if m"([\d\.]+\s+\[cnts ...
                   [^\]]*\])";
                $trig_info{det_fluence_int} = $1 if m"([\d\.]+\s\[sec ...
                   .*\])";
                next;
            }
        }
135

    # det_alts (final)
    $trig_info{det_alts} = join ' ', @dets;
```

The last section the module looks for deals with the event itself, namely it's location in the sky. The SXC and WXM cameras have essentially the same format for distinguishing the location of the event. The SXC defines an error box by its center and corners. The WXM defines an error circle by its center and the corners of its circumscribed square. Coordinates are repeated using three epochs of equatorial coordinates. Only those coordinates matching the `$EPOCH` variable (currently defined in `parse_notices.pl` as "J2000") are retained.

```
138     ######################
        ###  Handle event entries
140     ###  NOTE: In general, SXC is more accurate than WXM (is more  ...
           accurate
        ###        than FREGATE)
```

```
    # event_type (always GRB for HETE)
    $trig_info{event_type} = "GRB";
    # no event_z or event_z_err (yet)

    ### Handling coordinates
    my ($coords_type, $is_in_coords, @ra_error, @dec_error);
    # SXC camera
    if (grep /^SXC_CNTR/, @notice) {
        my @sxc_entries = map {
            $coords_type = $1 if /^(SXC_CNTR_\w+:)/;
            $is_in_coords=
                (/^SXC/ or ($is_in_coords and s/^\s+/${coords_type} ...
                    /)) ? 1 : 0;
            $is_in_coords ? $_ : ();
        } @notice;
        foreach (@sxc_entries) {
        # event_ra
            $trig_info{event_ra} = $1, next
                if (/^SXC_CNTR_RA/ and /$EPOCH/ and m"([\-\d\.]+)d ...
                   ");
        # event_dec
            $trig_info{event_dec} = $1, next
                if (/^SXC_CNTR_DEC/ and /$EPOCH/ and m"([\-\d\.]+)d ...
                   ");
        # event_ra_err, event_dec_err (preliminary)
            if (m"^SXC_CORNER.?:\s+([\-\d\.]+)\s+([\-\d\.]+)\s+\[ ...
               deg\]") {
                push @ra_error, $1;
                push @dec_error, $2;
                next;
            }
        }
        # event_err_type (Always 'error box' for SXC, 'error disk'  ...
           for WXM)
        $trig_info{event_err_type} = "error box";
        # event_epoch (Always 'J2000' (or whatever $EPOCH is) for  ...
           SXC, WXM)
        $trig_info{event_epoch}    = $EPOCH;
        # event_ra_err (final)
        @ra_error = sort {$a <=> $b} @ra_error;
        $trig_info{event_ra_err}   =
            (int (($ra_error[$#ra_error] - $ra_error[1]) * 5000)) / ...
                10000
            if $trig_info{event_ra};
        # event_dec_err (final)
        @dec_error = sort {$a <=> $b} @dec_error;
        $trig_info{event_dec_err}  =
```

```
            (int (($dec_error[$#dec_error] - $dec_error[1]) * 5000) ...
               ) / 10000
            if $trig_info{event_dec};
        # WXM camera
185     } elsif (grep /^WXM_CNTR/, @notice) {
          my @wxm_entries = map {
            $is_in_coords = (/^WXM/ || ($is_in_coords and /^\s+/))  ...
               ? 1 : 0;
            $_ if $is_in_coords;
          } @notice;
190       foreach (@wxm_entries) {
          # event_ra
            $trig_info{event_ra} = $1
              if (/^WXM_CNTR_RA/ and /$EPOCH/ and m"([\-\d\.]+)d ...
                 ");
          # event_dec
195         $trig_info{event_dec} = $1
              if (/^WXM_CNTR_DEC/ and /$EPOCH/ and m"([\-\d\.]+)d ...
                 ");
          # event_ra_err, event_dec_err (preliminary)
            if (m"^WXM_CORNER.*:\s+([\-\d\.]+)\s+([\-\d\.]+)\s+\[ ...
               deg\]") {
              push @ra_error, $1;
200           push @dec_error, $2;
            }
          }
          # event_err_type (Always 'error box' for SXC, 'error disk'  ...
             for WXM)
          $trig_info{event_err_type} = "error disk";
205       # event_epoch (Always 'J2000' (or whatever $EPOCH is) for  ...
             SXC, WXM)
          $trig_info{event_epoch}    = $EPOCH;
          # event_ra_err (final)
          @ra_error = sort {$a <=> $b} @ra_error;
          $trig_info{event_ra_err}   =
210         (int (($ra_error[$#ra_error] - $ra_error[1]) * 5000)) / ...
              10000
            if ($trig_info{event_ra} and scalar(@ra_error));
          # event_dec_err (final)
          @dec_error = sort {$a <=> $b} @dec_error;
          $trig_info{event_dec_err}  =
215         (int (($dec_error[$#dec_error] - $dec_error[1]) * 5000) ...
               ) / 10000
            if ($trig_info{event_dec} and scalar(@dec_error));
        }
```

Finally, the handler writes the `%trig_info` hash variable to a file and creates subdivision hash variables. The event and email timestamps are compared via the `check_times()`

subroutine, and an alert is sent to the control rooms if the email was sent soon enough after the event. Regardless, the `%trig_info` hash is saved in LIGO Lightweight format via the `send_trigger()` subroutine.

```
        ######################
        ###  Done Handling entries; fill %time_keys
    open(PARSED, ">${outbox}/parsed-GCN_alerts/${infile}")
        or &fatal("Can't open OutBox/parsed-GCN_alerts/${infile}  ...
           for writing.",1);
    foreach (sort keys %trig_info) {
        print "  $_\t: $trig_info{$_}\n" if ($verbosity and $debug) ...
           ;
        print PARSED "  $_\t: $trig_info{$_}\n";
        $time_info{$_}   = $trig_info{$_} if /time(stamp|fmt)/;
        $det_info{$_}    = $trig_info{$_} if /^det/;
        $obs_info{$_}    = $trig_info{$_} if /^obs/;
        $event_info{$_}  = $trig_info{$_} if /^event/;
        $notice_info{$_} = $trig_info{$_} if /^notice/;
    }
    close PARSED
        or &fatal("Can't close OutBox/parsed-GCN_alerts/${infile ...
           }.",1);
    &logmsg("Deleting file from OutBox/unparsed: ${infile}") if  ...
       $debug;
    unlink "${outbox}/unparsed/${infile}"
        or &fatal("Can't unlink OutBox/unparsed/${infile}",1);

    ######################
    ### Send notification to control room operators if necessary
    $success &= &check_times() and &logmsg("$tmpreturn");

    ######################
    ### Send event to metadatabase
    $success &= &send_trigger() and &logmsg("$tmpreturn");

    return $success;
}
```

The next handler module is a placeholder for alerts from the InterPlanetary Network project. These alerts are extremely rare, as they depend on triangulation calculations from multiple gamma-ray detectors, and data from these detectors is not available in real-time. If any alerts come, they will be saved in the OUTBOX.

```
############################################################
####  handle_IPN(infile, hashref) subroutine
############################################################
sub handle_IPN {
    my $success=1;
    my $infile  = shift;
```

```
    my $hashref = shift;
    if ($$hashref{Subject} =~ /GCN\/IPN_POSITION/) {
        &logmsg("Will not delete file from OutBox/unparsed: ${ ...
           infile}") if $debug;
#        unlink "${outbox}/unparsed/${infile}"
#            or &fatal("Can't unlink ${outbox}/unparsed/${infile ...
   }",1);
    }

    return $success;
}
```

The following handler parses alerts from the INTErnational Gamma-Ray Laboratory (INTEGRAL). The module is based on code from the HETE handler, and only minor differences exist between INTEGRAL and HETE alerts. Only these differences will be noted in this appendix.

```
############################################################
####  handle_INTEGRAL(infile, hashref) subroutine
####  This handles INTEGRAL position notices.  It records various
####  information from the notice and puts it into a hash array
####  (%trig_info) for dumping to the OutBox/parsed-GCN_alerts/
####  subdirectory.
####
####  It also checks the event time to see if operators should
####  be notified.
############################################################
sub handle_INTEGRAL {
#    return 0;
    my (@notice, $i, $infile, $hashref, $success);
    $success    = 1;
    $infile     = shift;
    $hashref    = shift;
    &logmsg("Parsing file from OutBox/unparsed: ${infile}") if  ...
       $debug;
    open(NOTICE, "${outbox}/unparsed/${infile}")
        or &fatal("Can't open OutBox/unparsed/$infile for reading", ...
            1);
    @notice = <NOTICE>;
    $i = 0;
    close NOTICE
        or &fatal("Can't close OutBox/unparsed/$infile.", 1);
    print "@notice" if ($verbosity and $debug);
    foreach $tmpkey (sort keys %trig_info) {
        print "\%trig_info{$tmpkey}\t: $trig_info{$tmpkey}\n"
            if ($verbosity and $debug > 2);
    }

    while (($i < $#notice) and ($_ = $notice[++$i])) {
        chomp;
```

```
        ### For extra debugging, record the email.
        &logmsg(" ${infile} | $_") if ($debug > 1);
295
        ######################
        ###  Handle timestamp entries
        # email_timestamp, email_timefmt
        ($trig_info{email_timestamp}, $trig_info{email_timefmt}) =
300         &get_timestamp($1), next if /^Date:\s+(.+)$/;
        # notice_timestamp, notice_timefmt
        ($trig_info{notice_timestamp}, $trig_info{notice_timefmt})  ...
           =
            &get_timestamp($1), next if /^NOTICE_DATE:\s+(.+)$/;
        # event_timestamp, event_timefmt
305     if (/^GRB_(DATE|TIME):\s+(.+)$/) {
            $tmpevent{"GRB_$1"} = $2;
            ($trig_info{event_timestamp}, $trig_info{event_timefmt ...
               }) =
                &get_timestamp("$tmpevent{GRB_DATE} $tmpevent{ ...
                   GRB_TIME}")
                if (defined $tmpevent{'GRB_DATE'} and
310                 defined $tmpevent{'GRB_TIME'} );
            next;
        }

        ######################
315     ###  Handle notice entries
        # notice_type
        $trig_info{notice_type} = $1, next if /^NOTICE_TYPE:\s+(.*) ...
           $/;
        # notice_id, notice_sequence
        $trig_info{notice_id} = $1,
320     $trig_info{notice_sequence} = $2, next
            if /^TRIGGER_NUM:\s+(\d+),\s+Sub_Num: (\d+)$/;
        # notice_comments
        if (/^COMMENTS:\s+(.*)$/) {
            my $tmp_comments = $1;
325         while ( (++$i < $#notice) and
                    ($notice[$i] =~ /^(COMMENTS:|)\s+(.*)$/) ) {
                $tmp_comments .= " $2";
            }
            --$i;
330         $trig_info{notice_comments} = $tmp_comments;
            next;
        }
        # Take a guess at notice_url
        $trig_info{notice_url} =
```

```
        "http://gcn.gsfc.nasa.gov/other/$trig_info{notice_id}. ...
           integral"
        if ($trig_info{notice_id} and !$trig_info{notice_url});
```

The field of view for INTEGRAL's burst detector is much smaller than HETE's, and is marked accordingly. Also, INTEGRAL's orbit is highly elliptical and spacecraft position information is not present in the alerts.

```
    ######################
    ###  Handle observer entries
    # obs_fov_ra
    $trig_info{obs_fov_ra} = $1, next
        if (/$EPOCH/ and m"^SC_RA:\s+([\d\-\.]+)(\s+\[deg\]|)") ...
           ;
    # obs_fov_dec
    $trig_info{obs_fov_dec} = $1, next
        if (/$EPOCH/ and m"^SC_DEC:\s+([\d\-\.]+)(\s+\[deg\]|) ...
           ");
    # INTEGRAL's IBIS fov is 9-by-9 degrees; we'll hardcode  ...
       this for now
    # obs_fov_ra_width
    # obs_fov_dec_width
    $trig_info{obs_fov_ra_width}    = 9; #degrees
    $trig_info{obs_fov_dec_width}   = 9; #degrees
    # INTEGRAL's orbit is highly elliptical, ranging from 9E3  ...
       to 155E3 km
    # obs_loc_lat (positive if north, negative is south)
    # obs_loc_long (positive if east, negative if west)
    # obs_loc_ele
```

Only one detector appears in INTEGRAL alerts. Additionally, very little detector information is present in the alerts. Namely, fluence and peak flux is not present.

```
    ######################
    ###  Handle detector entries; for INTEGRAL, it's IBIS, SPI, ...
        JEM-X, OMC
    # det_name, det_band
    $trig_info{det_name} = "IBIS";
    # det_band
    # det_alts
    # det_snr
    $trig_info{det_snr} = $1, next
        if (/^GRB_INTEN:/ and m"([\d\-\.]+)(\s+\[sigma\]|)");
    # det_peak, det_peak_int
    # det_fluence, det_fluence_int

    ######################
    ###  Handle event entries
    # event_type (always GRB for INTEGRAL)
    $trig_info{event_type} = "GRB";
```

```
370         # no event_z or event_z_err (yet)

            ### Handling coordinates
    #        my ($coords_type, $is_in_coords, @ra_error, @dec_error);
            # event_ra
375         $trig_info{event_ra} = $1, next
                if (/^GRB_RA/ and/$EPOCH/ and /([\d\-\.]+)d/);
            # event_dec
            $trig_info{event_dec} = $1, next
                if (/^GRB_DEC/ and/$EPOCH/ and /([\d\-\.]+)d/);
380         # event_ra_err, event_dec_err
            $trig_info{event_ra_err} = $trig_info{event_dec_err} = $1,  ...
               next
                if (/^GRB_ERROR:/ and m"([\d\-\.]+)(\s+\[arcmin|)");
            # event_err_type (Always 'error box' for SXC, 'error disk'  ...
               for WXM)
            $trig_info{event_err_type} = "error disk";
385         # event_epoch (Always 'J2000' (or whatever $EPOCH is) for  ...
               SXC, WXM)
            $trig_info{event_epoch} = $EPOCH;
        }


390     ######################
        ###  Done Handling entries; fill %time_keys
        open(PARSED, ">${outbox}/parsed-GCN_alerts/${infile}")
            or &fatal("Can't open OutBox/parsed-GCN_alerts/${infile}  ...
               for writing.",1);
        foreach (sort keys %trig_info) {
395         print "  $_\t: $trig_info{$_}\n" if ($verbosity and $debug) ...
               ;
            print PARSED "  $_\t: $trig_info{$_}\n";
            $time_info{$_}   = $trig_info{$_} if /time(stamp|fmt)/;
            $det_info{$_}    = $trig_info{$_} if /^det/;
            $obs_info{$_}    = $trig_info{$_} if /^obs/;
400         $event_info{$_}  = $trig_info{$_} if /^event/;
            $notice_info{$_} = $trig_info{$_} if /^notice/;
        }
        close PARSED
            or &fatal("Can't close OutBox/parsed-GCN_alerts/${infile ...
               }.",1);
405     &logmsg("Deleting file from OutBox/unparsed: ${infile}") if  ...
           $debug;
        unlink "${outbox}/unparsed/${infile}"
            or &fatal("Can't unlink OutBox/unparsed/${infile}",1);

        ####################
```

```
    ### Send notification to control room operators if necessary
    $success &= &check_times() and &logmsg("$tmpreturn");

    ######################
    ### Send event to metadatabase
    $success &= &send_trigger() and &logmsg("$tmpreturn");

    return $success;
}
```

The next handler module parses alerts from the Swift project. Swift generates over twenty different alerts, but only a handful are suitable for automatically obtaining information about an event. The first alert generated by Swift following an event is usually the "SWIFT_BAT_POSITION", but sometimes the X-Ray Telescope (XRT) may generate the initial alert.

Again, this code is based on the HETE handler module, so only differences will be explained below.

```
############################################################
####  handle_SWIFT(infile, hashref) subroutine
####  This handles SWIFT position notices.  It records various
####  information from the notice and puts it into a hash array
####  (%trig_info) for dumping to the OutBox/parsed-GCN_alerts/
####  subdirectory.
####
####  It also checks the event time to see if operators should
####  be notified.
############################################################
sub handle_SWIFT {
#    return 0;
    my (@notice, $i, $infile, $hashref, $success);
    $success    = 1;
    $infile     = shift;
    $hashref    = shift;
    &logmsg("Parsing file from OutBox/unparsed: ${infile}") if  ...
       $debug;
    open(NOTICE, "${outbox}/unparsed/${infile}")
        or &fatal("Can't open OutBox/unparsed/$infile for reading", ...
            1);
    @notice = <NOTICE>;
    $i = 0;
    close NOTICE
        or &fatal("Can't close OutBox/unparsed/$infile.", 1);
    print "@notice" if ($verbosity and $debug);
    foreach $tmpkey (sort keys %trig_info) {
        print "\%trig_info{$tmpkey}\t: $trig_info{$tmpkey}\n"
            if ($verbosity and $debug > 2);
    }
```

```
    while (($i < $#notice) and ($_ = $notice[++$i])) {
        chomp;
        ### For extra debugging, record the email.
        &logmsg(" ${infile} | $_") if ($debug > 1);

        ######################
        ###  Handle timestamp entries
        # email_timestamp, email_timefmt
        ($trig_info{email_timestamp}, $trig_info{email_timefmt}) =
            &get_timestamp($1), next if /^Date:\s+(.+)$/;
        # notice_timestamp, notice_timefmt
        ($trig_info{notice_timestamp}, $trig_info{notice_timefmt})  ...
           =
            &get_timestamp($1), next if /^NOTICE_DATE:\s+(.+)$/;
        # event_timestamp, event_timefmt
        if (/^(GRB|IMG_START)_(DATE|TIME):\s+(.+)$/) {
            $tmpevent{"GRB_$2"} = $3;
            ($trig_info{event_timestamp}, $trig_info{event_timefmt ...
               }) =
                &get_timestamp("$tmpevent{GRB_DATE} $tmpevent{ ...
                   GRB_TIME}")
                if (defined $tmpevent{'GRB_DATE'} and
                    defined $tmpevent{'GRB_TIME'} );
            next;
        }
```

Swift describes their notices using the project name and detector name (one of "BAT", "XRF" or "UVOT"), e.g. "Swift-BAT Position". HETE and INTEGRAL just use the project name, e.g. "HETE S/C_Alert".

```
        ######################
        ###  Handle notice entries
        # notice_type
        $trig_info{notice_type} = "Swift-$1 $2",
            $trig_info{det_name} = $1, next if /^NOTICE_TYPE:\s+ ...
               Swift-(\w+) ([\w\s\-]*)$/;
        # notice_id, notice_sequence
        $trig_info{notice_id} = $1,
        $trig_info{notice_sequence} = $2, next
            if /^TRIGGER_NUM:\s+(\d+),\s+Seg_Num: (\d+)$/;
        # notice_comments
        if (/^COMMENTS:\s+(.*)$/) {
            my $tmp_comments = $1;
            while ( (++$i < $#notice) and
                    ($notice[$i] =~ /^(COMMENTS:|)\s+(.*)$/) ) {
                $tmp_comments .= " $2";
            }
            --$i;
            $trig_info{notice_comments} = $tmp_comments;
```

```
            next;
        }
        # Take a guess at notice_url
        $trig_info{notice_url} =
            "http://gcn.gsfc.nasa.gov/other/$trig_info{notice_id}. ...
               swift"
            if ($trig_info{notice_id} and !$trig_info{notice_url});
```

Swift gamma-ray burst alerts do not include information about the spacecraft. This information is instead put into a separate alert, which is currently ignored by the parsing script. Unlike INTEGRAL, Swift's orbit is nearly circular, with an elevation about 600 km from the earth's surface.

```
#         ######################
#         ###  Handle observer entries
#         # obs_fov_ra
#         $trig_info{obs_fov_ra} = $1, next
#             if (/$EPOCH/ and m"^SC_RA:\s+([\d\-\.]+)(\s+\[deg\]|) ...
   ");
#         # obs_fov_dec
#         $trig_info{obs_fov_dec} = $1, next
#             if (/$EPOCH/ and m"^SC_DEC:\s+([\d\-\.]+)(\s+\[deg\]|) ...
   ");
        # SWIFT's BAT fov is 100x60 degrees (1.4sr); we'll hardcode ...
            this for now
        # obs_fov_ra_width
        # obs_fov_dec_width
        $trig_info{obs_fov_ra_width}    = 100; #degrees
        $trig_info{obs_fov_dec_width}   = 60; #degrees
        # SWIFT's orbit is nearly circular, at around 600km
        # obs_loc_lat (positive if north, negative is south)
        # obs_loc_long (positive if east, negative if west)
        # obs_loc_ele
        $trig_info{obs_loc_ele}         = 600; #km
```

Like HETE, Swift uses several detectors to observe gamma-ray bursts. Unlike HETE, Swift puts information from other detectors into separate alerts.

```
        ######################
        ###  Handle detector entries; for SWIFT, it's BAT, XRT, or  ...
           UVOT
        # det_name, det_band
        #:NOTE: det_name is given in notice_type
        # det_band
###:NOTE: RJR- Stopped here 2005.02.01 ###
        # det_alts
        # det_snr
        $trig_info{det_snr} = $1, next
            if (/^GRB_INTEN:/ and m"([\d\-\.]+)(\s+\[sigma\]|)");
        # det_peak, det_peak_int
        # det_fluence, det_fluence_int
```

Swift can assign a style to a GRB, called a "TRIGGER_INDEX". This index gets included into the `event_type` field of the `%trig_info` hash variable if present.

```
524         ######################
525         ###  Handle event entries
            # event_type (always GRB for SWIFT)
            $trig_info{event_type} = "GRB" if !$trig_info{event_type};
            $trig_info{event_type} = "$trig_info{event_type} $1", next
                if /^TRIGGER_INDEX:\s+(.*)$/;
530         # no event_z or event_z_err (yet)

            ### Handling coordinates
            ## :NOTE: RJR 2005.02.01 -- Trying to handle a multi-line  ...
               coordinate
            ## :NOTE: listing correctly.  We may not always be using  ...
               J2000 for the
535         ## :NOTE: epoch, or GCN may not always list J2000 as the  ...
               first epoch.
            # event_ra
            if (/^GRB_RA/) { $i--; # reset our iterator to read the  ...
               GRB_RA line again
                # Keep reading the next line until we reach the current ...
                    $EPOCH
                while ($_ = $notice[++$i]) {
540                 chomp;
                    # Quit if we've left the GRB_RA coordinate lines
                    # :NOTE: Our iterator will be moved forward again  ...
                       in the
                    # :NOTE: larger loop.
                    $i--, last if ($_ !~ /^(GRB_RA|\s+)/);
545                 # This is the line we're looking for-- parse out  ...
                       the degrees
                    $trig_info{event_ra} = $1, next
                        if (/$EPOCH/ and /([\d\-\.]+)d/);
                }
                next; # iterate through the larger loop
550         }
            # event_dec
            if (/^GRB_DEC/) { $i--; # reset our iterator to read the  ...
               GRB_DEC line again
                # Keep reading the next line until we reach the current ...
                    $EPOCH
                while ($_ = $notice[++$i]) {
555                 chomp;
                    # Quit if we've left the GRB_RA coordinate lines
                    # :NOTE: Our iterator will be moved forward again  ...
                       in the
                    # :NOTE: larger loop.
```

```
                $i--, last if ($_ !~ /^(GRB_DEC|\s+)/);
                # This is the line we're looking for-- parse out  ...
                   the degrees
                $trig_info{event_dec} = $1, next
                    if (/$EPOCH/ and /([\d\-\.]+)d/);
            }
            next; # iterate through the larger loop
        }
        # event_ra_err, event_dec_err
        $trig_info{event_ra_err} = $trig_info{event_dec_err} = $1,  ...
           next
            if (/^GRB_ERROR:/ and m"([\d\-\.]+)(\s+\[arcmin|)");
        # event_err_type (Always 'error disk' for SWIFT
        $trig_info{event_err_type} = "error disk";
        # event_epoch (Always 'J2000' (or whatever $EPOCH is) for  ...
           SWIFT)
        $trig_info{event_epoch} = $EPOCH;
    }

    ######################
    ###  Done Handling entries; fill %time_keys
    open(PARSED, ">${outbox}/parsed-GCN_alerts/${infile}")
        or &fatal("Can't open OutBox/parsed-GCN_alerts/${infile}  ...
           for writing.",1);
    foreach (sort keys %trig_info) {
        print "  $_\t: $trig_info{$_}\n" if ($verbosity and $debug) ...
           ;
        print PARSED "  $_\t: $trig_info{$_}\n";
        $time_info{$_}   = $trig_info{$_} if /time(stamp|fmt)/;
        $det_info{$_}    = $trig_info{$_} if /^det/;
        $obs_info{$_}    = $trig_info{$_} if /^obs/;
        $event_info{$_}  = $trig_info{$_} if /^event/;
        $notice_info{$_} = $trig_info{$_} if /^notice/;
    }
    close PARSED
        or &fatal("Can't close OutBox/parsed-GCN_alerts/${infile ...
           }.",1);
    &logmsg("Deleting file from OutBox/unparsed: ${infile}") if  ...
       $debug;
    unlink "${outbox}/unparsed/${infile}"
        or &fatal("Can't unlink OutBox/unparsed/${infile}",1);

    ####################
    ### Send notification to control room operators if necessary
    $success &= &check_times() and &logmsg("$tmpreturn");
```

```
    ####################
    ### Send event to metadatabase
    $success &= &send_trigger() and &logmsg("$tmpreturn");

    return $success;
}
```

A.3 A walkthrough of the receiver script

Below is a listing of a test receiver script, based on the current version of the script at LHO [2]. For clarity of reading, the EPICS control interface commands have been replaced with pseudocode.

The receiver script starts by including some standard Perl header information: some comments about the script's function, a list of Perl modules to use, any necessary environment variables, and hard-coded internal variables.

```
#!/usr/bin/env perl
#
# Gamma Ray Burst Notification Server Test Code.
#
# Parses received string for the syntax:
# <TYPE> <EVENT TIME> <STANDDOWN TIME>
# Where:
#      <TYPE> is one of
#                      1 = KEEP ALIVE
#                      2-19 GRB (2=HETE,3=INTEGRAL,4=SWIFT)
#                      20-39 SN (20=SNEWS)
#
#      <TIME> syntax is UTC-2003-11-03-03:08:44 for 03:08:04 on nov ...
    3rd 2003
#
use strict;
BEGIN { $ENV{PATH} = '/usr/ucb:/bin';
      }
use IO::Socket;
use Carp;

############################
### Local, Modifiable Variables
############################
# This is the host port to use
my $port  = shift || 34512;
# This is the host interface to use
my $host_ip    = shift || "localhost";
# This is the log file filehandle to use
my $logfile = ">>gamma_ray_burst.log";
# This is the command to interface with the control room
my $ezcawrite = "echo";
```

```
# Determines whether debug info is printed or not
my $DEBUG = 1;

# define event types
my $KEEPALIVE = 1;
# Ranges of event ids
my $GRB_LOW = 2;
my $GRB_HIGH = 19;
my $SN_LOW  = 20;
my $SN_HIGH  = 39;
# Event sources:
my $GRB_HETE = 2;
my $GRB_INTEGRAL = 3;
my $GRB_SWIFT = 4;
my $SN_SNEWS = 20;
```

Next, the various subroutines in this file are declared or defined. Note that the subroutines `initialize_epics`, `keep_alive`, `epics_alarm` and `epics_error` are pseudocode to communicate with the control room.

```
### List of subroutines
# Performs any initialization routines needed for EPICS or LabView
sub initialize_epics;
# Sends a keep-alive signal to the control room via EPICS or  ...
   LabView
sub keep_alive;
# Sends an alarm to the control room via EPICS or LabView
sub epics_alarm;
# Sends an error message to the control room via EPICS or LabView
sub epics_error;
# Writes a message to the logfile
sub logmsg {
open(LOG,$logfile) or print "Cannot open logfile $logfile\n";
print LOG " ".scalar gmtime," UTC | @_ \n";
close LOG;
}
```

Next a TCP/IP socket is opened on the local machine, and any initialization script is executed.

```
############################
### Open a server socket
############################
#:NOTE: This variable is present to allow file comparisons with  ...
   previous CVS
#:NOTE: versions.  The variable was changed to $new_sock to conform ...
    with
#:NOTE: the server on red.
my ($new_client);
my $sock  = new IO::Socket::INET (
                                LocalHost => $host_ip,
```

```
                                    LocalPort => $host_port,
                                    Proto => 'tcp',
                                    Listen => 1,
                                    Reuse => 1,
                                    Timeout => 1200,
                                     ) or die "cannot create socket";
$port = $sock->sockport();
logmsg "server started on port $port";
### Initialize controls interface
my $eventcounter = &initialize_epics();
```

The receiver script starts waiting for connections initiated by the parsing script. If no connections occur before the twenty-minute timeout period, the receiver script will assume there is an error with the TCP/IP connection and alert the control room (cf. below). When a connection occurs, this test script logs the event and sends some preliminary information to the parsing script. The receiver script next reads a line of text from the parsing script. A ten-second time limit is set to finish reading the text.

```
############################
### Wait for clients
############################
my $line;
my $type;
my $eventtime;
my $auxtime;
my $new_sock;
while (1) { # loop over waiting for TCP/IP packets
  while ($new_sock = $sock->accept()){
    ### Log initial details.
    $new_sock->autoflush(1);
    $remote_host = gethostbyaddr($new_sock->peeraddr,AF_INET) ||  ...
       $new_sock->peerhost;
    $remote_port = $new_sock->peerport;
    logmsg "connect from $remote_host, port $remote_port";
    #logmsg "connection from ".$new_sock->peerhost()." sock ". ...
       $new_sock->peerport();

    ### Send opening info to client
    print $new_sock "Connected to $0 at $host_ip, port $port\tat ",
        scalar gmtime, "\r\n";

    ### Handle input from client; set an alarm to wakeup if hung
    eval {
        local $SIG{ALRM} = sub {
            logmsg "alarm clock restart";
            die "alarm clock restart";
        };
        alarm 10;
```

```
        ### Read data from socket; untaint automatically
        $line = <$new_sock>;
        chomp $line;

        alarm 0;
    };

    if($DEBUG){print "RECEIVED: $line\n";}
```

Next, the receiver script parses the text into three fields: the event type, the time at which the event was detected, and the time at which staff may resume maintenance or commissioning (the *stand-down time*). Based on the type of event, two possible signals can be communicated with the control room; either a keep-alive signal can be communicated as, e.g., a toggled icon, or an alarm signal can be communicated audibly or inaudibly. Currently only gamma-ray bursts will trigger an audible alarm.

```
    # split out line
    ($type,$eventtime,$auxtime) = split / /,$line,3;
    if($DEBUG){print "TYPE = $type\n";}
    if($DEBUG){print "EVENTTIME = $eventtime\n";}
    if($DEBUG){print "AUXTIME   = $auxtime\n\n";}

    # decode type
    if ($type == $KEEPALIVE) {
      # KEEP-ALIVE EVENT
      if($DEBUG){print "keep alive Event\n";}
      &keep_alive($type,$eventtime,$auxtime);
    } elsif (($type >= $GRB_LOW)&&($type <= $GRB_HIGH)) {
      # GRB EVENT

      # raise alarm (will clear at end of processing cycle)
    ### Go to EPICS alarm subroutine
      $eventcounter = &epics_alarm($type, $eventtime, $auxtime);

      logmsg "$line";

    } elsif (($type >= $SN_LOW)&&($type <= $SN_HIGH)) {
      # SN EVENT
      if($DEBUG){print "Supernovae Event, not yet implemented\n";}
      if($DEBUG){print "Full message \n";}
      if($DEBUG){print "$line\n";}
    } else {
      # INVALID EVENT
      print "Invalid Event type $type\n";
      print "Full message \n";
      print "$line\n";
    }
```

The communication with the control room finished, the receiver script is free to acknowledge the signal to the client and close the socket connection. If the script process falls between the two `while()` loops, it signifies that the socket connection has timed out and an error is flagged to the control room (via `epics_error`).

```
149     # send acknowledgment to client
150     print $new_sock "ACK RED ".$line."\n";


        ### Close the client connection (child process)
        close $new_sock;
155   } # end while ($new_sock = $sock->accept()){
      # accept call has timed out, we have not received a keepalive
      # packet. Raise alarm and go back into accept loop.
      &epics_error("TCP/IP timeout error");

160 } # end while(1)
```

B GCN alerts

This section contains examples of GCN alerts sent via email to the parsing script during the S4 run. The header information has been somewhat abridged for readability, but the content of the notices are reproduced in full. The messages represent the four different sources which the parsing script currently recognizesin full: i) the HETE-2 project, ii) the Swift project, iii) the INTEGRAL project and iv) GCN internally-generated test messages. The email format for these sources are described within the GCN website [3].

This first set of messages are from the HETE-2 project. Three of the four messages concerning a trigger (`3689`) are shown. This is the initial message which was sent. As with all emails which the parsing script processes, the senders address is checked first. Only addresses from the official GCN and SNEWS mail agents are currently accepted; other emails are deleted from the INBOX without further processing. The parsing script then identifies the subject line (`GCN/HETE_POSITION`) and finds the appropriate handler defined in `parse_GCN.pl`. The handler, in turn, uses the content of the message to fill a Perl hash variable describing the trigger. Three different timestamps are translated into GPS: i) the email timestamp (from the `Date:` header), ii) the `NOTICE_DATE:` timestamp and iii) the `GRB_DATE:` and `GRB_TIME:` timestamps, which must be combined into the *event_timestamp*. Note that in this case, the email was generated less than thirty seconds after the event was detected by the spacecraft. This is typical for the first alerts for a given trigger.

```
From vxw@capella.gsfc.nasa.gov  Sat Mar  5 22:28:39 2005
Return-Path: <vxw@capella.gsfc.nasa.gov>
Date: Sun, 6 Mar 2005 01:28:31 -0500
From: Bacodine <vxw@capella.gsfc.nasa.gov>
To: wk_sz@ligo.caltech.edu
Subject: GCN/HETE_POSITION
```

```
TITLE:          GCN/HETE BURST POSITION NOTICE
NOTICE_DATE:    Sun 06 Mar 05 06:28:18 UT
NOTICE_TYPE:    HETE S/C_Alert
TRIGGER_NUM:    3689,   Seq_Num: 1
GRB_DATE:       13435 TJD;    65 DOY;   05/03/06
GRB_TIME:       23293.72 SOD {06:28:13.72} UT
TRIGGER_SOURCE: Trigger on the 6-80 keV band.
GAMMA_RATE:     426  [cnts/s]  on a 1.300 [sec] timescale
SC_LONG:        272 [deg East]
SUN_POSTN:      346.91d {+23h 07m 40s}   -5.61d {-05d 36' 18"}
MOON_POSTN:     291.97d {+19h 27m 53s}  -27.12d {-27d 07' 05"}
MOON_ILLUM:     22 [%]
COMMENTS:       No s/c ACS pointing info available yet.
COMMENTS:       Probable GRB.
```

The next email shown is actually third in the series. This now contains information from HETE's wide-field X-ray monitor (WXM) detectors, namely direction information, within a half-degree error circle.

```
From vxw@capella.gsfc.nasa.gov  Sat Mar  5 22:31:27 2005
Return-Path: <vxw@capella.gsfc.nasa.gov>
Date: Sun, 6 Mar 2005 01:31:19 -0500
From: Bacodine <vxw@capella.gsfc.nasa.gov>
To: wk_sz@ligo.caltech.edu
Subject: GCN/HETE_POSITION

TITLE:          GCN/HETE BURST POSITION NOTICE
NOTICE_DATE:    Sun 06 Mar 05 06:31:06 UT
NOTICE_TYPE:    HETE S/C_Last
TRIGGER_NUM:    3689,   Seq_Num: 3
GRB_DATE:       13435 TJD;    65 DOY;   05/03/06
GRB_TIME:       23293.72 SOD {06:28:13.72} UT
TRIGGER_SOURCE: Trigger on the 6-80 keV band.
GAMMA_RATE:     426  [cnts/s]  on a 1.300 [sec] timescale
SC_-Z_RA:       167 [deg]
SC_-Z_DEC:        7 [deg]
SC_LONG:        272 [deg East]
WXM_CNTR_RA:    163.619d {+10h 54m 29s} (J2000),
                163.690d {+10h 54m 46s} (current),
                162.929d {+10h 51m 43s} (1950)
WXM_CNTR_DEC:   +31.608d {+31d 36' 28"} (J2000),
                +31.580d {+31d 34' 49"} (current),
                +31.875d {+31d 52' 28"} (1950)
WXM_MAX_SIZE:   28.00 [arcmin] diameter
WXM_LOC_SN:     4 sig/noise (pt src in image)
WXM_IMAGE_SN:   X= 3.0 Y= 3.0 [sig/noise]
WXM_LC_SN:      X= 7.2 Y= 6.5 [sig/noise]
SUN_POSTN:      346.91d {+23h 07m 40s}   -5.61d {-05d 36' 18"}
SUN_DIST:       153.85 [deg]
```

```
MOON_POSTN:     291.97d {+19h 27m 53s}  -27.12d {-27d 07' 05"}
MOON_DIST:      135.11 [deg]
MOON_ILLUM:     22 [%]
GAL_COORDS:     195.88,64.21 [deg] galactic lon,lat of the burst
ECL_COORDS:     152.31,22.66 [deg] ecliptic lon,lat of the burst
COMMENTS:       Probable GRB.
COMMENTS:       WXM error box is circular; not rectangular.
```

The parsing script converted the information from the previous email into the following two formats. The first format is a *parsed notice*, which resides in its own file in the OUTBOX directory tree. The second format is a LIGO lightweight table entry, which consists of a single line. The entry shown here has been divided into 95-character lines. Fields which are blank in the parsed notice are translated into either empty strings or 0's in the table entry.

```
  det_alts      : WXM
  det_band      : 6-80 keV
  det_fluence   : 426  [cnts/s]
  det_fluence_int       : 1.300 [sec]
  det_name      : FREGATE
  det_peak      :
  det_peak_int  :
  det_snr       :
  email_timefmt : GPS
  email_timestamp       : 794125892
  event_dec     : 31.608
  event_dec_err :
  event_epoch   : J2000
  event_err_type        : error disk
  event_ra      : 163.619
  event_ra_err  :
  event_timefmt : GPS
  event_timestamp       : 794125706.72
  event_type    : GRB
  event_z       :
  event_z_err   :
  notice_comments       : WXM error box is circular; not  ...
     rectangular.
  notice_id     : 3689
  notice_sequence       : 3
  notice_timefmt        : GPS
  notice_timestamp      : 794125879
  notice_type   : HETE S/C_Last
  notice_url    : http://gcn.gsfc.nasa.gov/other/3689.hete
  obs_fov_dec   : 7
  obs_fov_dec_width     : 90
  obs_fov_ra    : 167
  obs_fov_ra_width      : 180
  obs_loc_ele   :
  obs_loc_lat   : 0
```

```
  obs_loc_long  : 272

"WXM","6-80 keV","426  [cnts/s]","1.300 [sec]","FREGATE ...
   ","","","",794125892,3.16080e+01,0.00000
e+00,"J2000","error disk","","",1.63619e+02,0.00000e ...
   +00,"",794125706,720000028,"GRB",0.00000e+0
0,0.00000e+00,"","","","WXM error box is circular; not rectangul ...
   ","3689","3",794125879,"HETE S/
C_Last","http://gcn.gsfc.nasa.gov/other/3689.hete",7.00000e ...
   +00,9.00000e+01,1.67000e+02,1.80000e
+02,0.00000e+00,0.00000e+00,2.72000e+02,
```

Finally, the fourth email issued a retraction statement. This is general practice among the gamma-ray burst detector projects, but is not required. This retraction occured over forty minutes after the burst was initially detected.

```
From vxw@capella.gsfc.nasa.gov  Sat Mar  5 23:11:44 2005
Return-Path: <vxw@capella.gsfc.nasa.gov>
Date: Sun, 6 Mar 2005 02:11:36 -0500
From: Bacodine <vxw@capella.gsfc.nasa.gov>
To: wk_sz@ligo.caltech.edu
Subject: GCN/HETE_POSITION

TITLE:          GCN/HETE BURST POSITION NOTICE
NOTICE_DATE:    Sun 06 Mar 05 07:11:20 UT
NOTICE_TYPE:    HETE Ground Analysis
TRIGGER_NUM:    3689,   Seq_Num: 4
GRB_DATE:       13435 TJD;    65 DOY;   05/03/06
GRB_TIME:       23293.72 SOD {06:28:13.72} UT
TRIGGER_SOURCE: Trigger on the 6-80 keV band.
GAMMA_RATE:     426  [cnts/s]  on a 1.300 [sec] timescale
SC_-Z_RA:       167 [deg]
SC_-Z_DEC:        7 [deg]
SC_LONG:        272 [deg East]
SUN_POSTN:      346.91d {+23h 07m 40s}   -5.61d {-05d 36' 18"}
MOON_POSTN:     291.97d {+19h 27m 53s}  -27.12d {-27d 07' 05"}
MOON_ILLUM:     22 [%]
COMMENTS:       Definitely not a GRB.
COMMENTS:       There is no position known for this trigger at this ...
    time.
COMMENTS:       Burst_Invalidity flag is true.
COMMENTS:       Particle event.
```

The next sequence of emails is from the Swift project. The text has many fields in common with the HETE-2 alerts, but an important difference is that direction information is provided nearly immediately, in several different coordinate systems.

```
From vxw@capella.gsfc.nasa.gov  Wed Mar  9 02:45:56 2005
Return-Path: <vxw@capella.gsfc.nasa.gov>
Date: Wed, 9 Mar 2005 05:45:48 -0500
```

```
From: Bacodine <vxw@capella.gsfc.nasa.gov>
To: wk_sz@ligo.caltech.edu
Subject: GCN/SWIFT_BAT_POSITION

TITLE:           GCN/SWIFT NOTICE
NOTICE_DATE:     Wed 09 Mar 05 10:45:40 UT
NOTICE_TYPE:     Swift-BAT GRB Position
TRIGGER_NUM:     107873,   Seg_Num: 0
GRB_RA:          182.622d {+12h 10m 29s} (J2000),
                 182.682d {+12h 10m 44s} (current),
                 182.033d {+12h 08m 08s} (1950)
GRB_DEC:         +77.591d {+77d 35' 27"} (J2000),
                 +77.562d {+77d 33' 43"} (current),
                 +77.869d {+77d 52' 08"} (1950)
GRB_ERROR:       4.00 [arcmin radius, statistical only]
GRB_INTEN:       200545 [cnts]    Peak=1241 [cnts/sec]
TRIGGER_DUR:     26.880 [sec]
TRIGGER_INDEX:   493     E_range: 50-350 keV
BKG_INTEN:       303166 [cnts]
BKG_TIME:        38488.00 SOD {10:41:28.00} UT
BKG_DUR:         64 [sec]
GRB_DATE:        13438 TJD;    68 DOY;   05/03/09
GRB_TIME:        38601.53 SOD {10:43:21.53} UT
GRB_PHI:         170.09 [deg]
GRB_THETA:        21.16 [deg]
SOLN_STATUS:     0x83
RATE_SIGNIF:     20.85 [sigma]
IMAGE_SIGNIF:    6.00 [sigma]
MERIT_PARAMS:     +1  +0  +0  +5  +3  +7  +0  +0 +28  +1
SUN_POSTN:       349.85d {+23h 19m 25s}   -4.37d {-04d 22' 00"}
SUN_DIST:        106.48 [deg]
MOON_POSTN:      339.53d {+22h 38m 08s}  -12.60d {-12d 36' 13"}
MOON_DIST:       113.98 [deg]
GAL_COORDS:      125.76, 39.31 [deg] galactic lon,lat of the burst  ...
   (or transient)
ECL_COORDS:      119.50, 64.15 [deg] ecliptic lon,lat of the burst  ...
   (or transient)
COMMENTS:        SWIFT-BAT GRB Coordinates.
COMMENTS:        This is a rate trigger.
COMMENTS:        A point_source was found.
COMMENTS:        This does not match any source in the on-board  ...
   catalog.
COMMENTS:        This does not match any source in the ground  ...
   catalog.
COMMENTS:        This is a GRB.
COMMENTS:        Since the IMAGE_SIGNIF is less than 7 sigma, this  ...
   is a questionable detection.
```

The following are headers from subsequent emails sent from Swift. The parsing script currently ignores these, as they describe the intent of the spacecraft to slew its x-ray and ultraviolet cameras into position to look for an afterglow transient. Another email provides a URL for the lightcurve detected by the Burst Alert Telescope (BAT).

```
Date: Wed, 9 Mar 2005 05:45:44 -0500
From: Bacodine <vxw@capella.gsfc.nasa.gov>
To: wk_sz@ligo.caltech.edu
Subject: GCN/SWIFT_FOM_OBSERVE

Date: Wed, 9 Mar 2005 05:49:09 -0500
From: Bacodine <vxw@capella.gsfc.nasa.gov>
To: wk_sz@ligo.caltech.edu
Subject: GCN/SWIFT_BAT_LIGHTCURVE

Date: Wed, 9 Mar 2005 06:11:13 -0500
From: Bacodine <vxw@capella.gsfc.nasa.gov>
To: wk_sz@ligo.caltech.edu
Subject: GCN/SWIFT_SC_SLEW

Date: Wed, 9 Mar 2005 06:27:40 -0500
From: Bacodine <vxw@capella.gsfc.nasa.gov>
To: wk_sz@ligo.caltech.edu
Subject: GCN/SWIFT_SC_SLEW

Date: Wed, 9 Mar 2005 06:33:05 -0500
From: Bacodine <vxw@capella.gsfc.nasa.gov>
To: wk_sz@ligo.caltech.edu
Subject: GCN/SWIFT_SC_SLEW

Date: Wed, 9 Mar 2005 06:33:09 -0500
From: Bacodine <vxw@capella.gsfc.nasa.gov>
To: wk_sz@ligo.caltech.edu
Subject: GCN/SWIFT_SC_SLEW
```

This is a retraction notice for the same Swift trigger. Note that the retraction was issued nearly four hours after the event.

```
From vxw@capella.gsfc.nasa.gov  Wed Mar  9 06:29:36 2005
Return-Path: <vxw@capella.gsfc.nasa.gov>
Date: Wed, 9 Mar 2005 09:29:27 -0500
From: Bacodine <vxw@capella.gsfc.nasa.gov>
To: wk_sz@ligo.caltech.edu
Subject: GCN/SWIFT_BAT_POSITION

TITLE:           GCN/SWIFT NOTICE
NOTICE_DATE:     Wed 09 Mar 05 14:29:20 UT
NOTICE_TYPE:     Swift-BAT GRB Position RETRACTION
TRIGGER_NUM:     107873,   Seg_Num: 0
```

```
GRB_RA:          182.622d {+12h 10m 29s} (J2000),
                 182.682d {+12h 10m 44s} (current),
                 182.034d {+12h 08m 08s} (1950)
GRB_DEC:         +77.591d {+77d 35' 27"} (J2000),
                 +77.562d {+77d 33' 43"} (current),
                 +77.869d {+77d 52' 09"} (1950)
GRB_ERROR:       9.99 [arcmin radius, statistical only]
GRB_INTEN:       200545 [cnts]    Peak=1241 [cnts/sec]
TRIGGER_DUR:     26.876 [sec]
TRIGGER_INDEX:   493     E_range: 50-350 keV
BKG_INTEN:       303166 [cnts]
BKG_TIME:        38488.00 SOD {10:41:28.00} UT
BKG_DUR:         64 [sec]
GRB_DATE:        13438 TJD;    68 DOY;   05/03/09
GRB_TIME:        38601.52 SOD {10:43:21.52} UT
GRB_PHI:         170.08 [deg]
GRB_THETA:        21.15 [deg]
SOLN_STATUS:     0xA3
RATE_SIGNIF:     20.85 [sigma]
IMAGE_SIGNIF:    6.00 [sigma]
MERIT_PARAMS:     +1  +0  +0  +5  +3  +7  +0  +0 +28  +1
SUN_POSTN:       349.85d {+23h 19m 25s}   -4.37d {-04d 22' 00"}
SUN_DIST:        106.48 [deg]
MOON_POSTN:      339.53d {+22h 38m 08s}  -12.60d {-12d 36' 13"}
MOON_DIST:       113.98 [deg]
GAL_COORDS:      125.76, 39.31 [deg] galactic lon,lat of the burst  ...
   (or transient)
ECL_COORDS:      119.50, 64.15 [deg] ecliptic lon,lat of the burst  ...
   (or transient)
COMMENTS:        SWIFT-BAT GRB Coordinates.
COMMENTS:        This is a rate trigger.
COMMENTS:        A point_source was found.
COMMENTS:        This does not match any source in the on-board  ...
   catalog.
COMMENTS:        This does not match any source in the ground  ...
   catalog.
COMMENTS:        This is not a GRB.
COMMENTS:        Since the IMAGE_SIGNIF is less than 7 sigma, this  ...
   is a questionable detection.
COMMENTS:        This is a RETRACTION of a previous notice that  ...
   identified this as a GRB --
  it is NOT a GRB.
COMMENTS:        This Notice was ground-reprocessed from flight- ...
   data.
COMMENTS:
COMMENTS:        This is NOT a real GRB -- it is a noise  ...
   fluctuation on the edge of the SAA.
```

```
COMMENTS:        The questionable aspect of the low detection  ...
   significance has been confirmed
  by ground analysis.
COMMENTS:        The non-astrophysical origin of this trigger is  ...
   further confirmed by the XRT
  non-detection of a source.
COMMENTS:        REPEAT:  This trigger is NOT a GRB.
```

The next email is the initial alert provided by the INTEGRAL project for a trigger. INTEGRAL also provides direction information in its alerts. However, INTEGRAL is not a dedicated gamma-ray burst detector, as it has several targets it observes.

```
From vxw@capella.gsfc.nasa.gov  Thu Feb 24 08:25:28 2005
Return-Path: <vxw@capella.gsfc.nasa.gov>
Date: Thu, 24 Feb 2005 11:25:19 -0500
From: Bacodine <vxw@capella.gsfc.nasa.gov>
To: wk_sz@ligo.caltech.edu
Subject: GCN/INTEGRAL_POSITION

TITLE:          GCN/INTEGRAL NOTICE
NOTICE_DATE:    Thu 24 Feb 05 16:25:08 UT
NOTICE_TYPE:    INTEGRAL Wakeup
TRIGGER_NUM:    2321,   Sub_Num: 0
GRB_RA:         258.1845d {+17h 12m 44s} (J2000),
                258.2715d {+17h 13m 05s} (current),
                257.3401d {+17h 09m 22s} (1950)
GRB_DEC:        -36.8670d {-36d 52' 00"} (J2000),
                -36.8729d {-36d 52' 21"} (current),
                -36.8080d {-36d 48' 28"} (1950)
GRB_ERROR:      2.70 [arcmin, radius, statistical only]
GRB_INTEN:      16.79 [sigma]
GRB_TIME:       59094.67 SOD {16:24:54.67} UT
GRB_DATE:       13425 TJD;    55 DOY;   05/02/24
SC_RA:          253.32 [deg] (J2000)
SC_DEC:         -37.79 [deg] (J2000)
SUN_POSTN:      337.93d {+22h 31m 44s}   -9.25d {-09d 14' 58"}
SUN_DIST:        76.22 [deg]
MOON_POSTN:     164.38d {+10h 57m 31s}  +10.43d {+10d 25' 32"}
MOON_DIST:       99.32 [deg]
GAL_COORDS:     349.58,  1.36 [deg] galactic lon,lat of the burst
ECL_COORDS:     260.29,-13.83 [deg] ecliptic lon,lat of the burst
COMMENTS:       INTEGRAL GRB Coordinates.
COMMENTS:       Possibly real GRB event
```

The following is a test notice sent by the GCN system. "BACODINE" is a remnant of the BATSE project, which initiated this real-time gamma-ray burst alert system. Test notices are sent roughly every four hours. A handler in `parse_GCN.pl` essentially ignores the text of the email message, which allows the notice to be deleted and recorded in the LOGFILE.

```
From vxw@capella.gsfc.nasa.gov Fri Dec  9 16:34:24 2005
```

```
Return-Path: <vxw@capella.gsfc.nasa.gov>
Date: Fri, 9 Dec 2005 19:34:27 -0500
From: Bacodine <vxw@capella.gsfc.nasa.gov>
To: wk_sz@ligo.caltech.edu
Subject: BACODINE_POSITION

TITLE:          BACODINE BURST POSITION NOTICE
NOTICE_DATE:    Sat 10 Dec 05 00:34:22 UT
NOTICE_TYPE:    Original
TRIGGER_NUM:    -1
GRB_RA:          35.91d {+02h 23m 38s} (J2000),
                 36.00d {+02h 24m 00s} (current),
                 35.16d {+02h 20m 38s} (1950)
GRB_DEC:        +34.97d {+34d 58' 24"} (J2000),
                +35.00d {+35d 00' 00"} (current),
                +34.75d {+34d 44' 48"} (1950)
GRB_ERROR:      5.0 [deg radius, statistical only]
GRB_INTEN:      1000 [cnts]    Peak=1000 [cnts/sec]
GRB_TIME:       2062.00 SOD {00:34:22.00} UT
GRB_DATE:       13714 TJD;   344 DOY;   05/12/10
GRB_SC_AZ:        0.00 [deg]                          {XScan=0.00}
GRB_SC_EL:        0.00 [deg]  {Zenith_angle=90.00}  {Scan=90.00}
SC_X_RA:          0.00 [deg] (J2000)
SC_X_DEC:         0.00 [deg]
SC_Z_RA:          0.00 [deg]
SC_Z_DEC:         0.00 [deg]
SUN_POSTN:      257.01d {+17h 08m 02s}  -22.90d {-22d 54' 01"}
SUN_DIST:       142.43 [deg]
MOON_POSTN:       8.33d {+00h 33m 19s}   +3.35d {+03d 21' 15"}
MOON_DIST:       40.73 [deg]
PROG_VERSION:   9.73
PROG_LEVEL:     3
COMMENTS:       Test Coordinates.
```

References

[1] Peter Shawhan librarian. http://www.ldas-sw.ligo.caltech.edu/ligotools/, (retrieved November 2005). A collection of general LIGO/LSC software tools.

[2] Dave Barker and Rauha Rahkola. `grb_server_MI.pl`. Perl script in CDS software repository, (retrieved November 2005).

[3] Scott Barthelmy. GRB Coordinates Network. http://gcn.gsfc.nasa.gov/, (retrieved November 2005).