

# LALApps — LSC Algorithm Library Applications

Contact: Jolien Creighton [jolien@gravity.phys.uwm.edu](mailto:jolien@gravity.phys.uwm.edu)

November 18, 2003

# Authors

Sukanta Bose  
Patrick Brady  
Duncan Brown  
Jolien Creighton  
Steve Fairhurst  
Isabel Leonor  
Jeffrey Noel  
Peter Shawhan  
John T. Whelan  
Mark Stephen Williamsen

# Contents

<b>1</b>	<b>How to get, build and use software for the LSC Data Grid</b>	<b>1</b>
1.1	Getting and installing the software . . . . .	1
1.1.1	Installation of required development tools . . . . .	1
1.1.2	Configuring your environment . . . . .	2
1.1.3	LAL . . . . .	3
1.1.4	LALApps . . . . .	4
1.1.5	Updating LAL/LALApps . . . . .	5
<b>2</b>	<b>LALApps utilities</b>	<b>6</b>
2.1	Header <code>lalapps.h</code> . . . . .	7
2.1.1	Function <code>set_debug_level</code> . . . . .	8
2.1.2	Function <code>clear_status</code> . . . . .	10
2.1.3	Macro <code>RCSID</code> . . . . .	11
2.1.4	Macro <code>PRINT_VERSION</code> . . . . .	12
2.1.5	Macro <code>LAL_CALL</code> . . . . .	13
2.2	Source <code>hello.c</code> . . . . .	15
2.3	Python Module <code>pipeline</code> . . . . .	17
2.3.1	Functions . . . . .	17
2.3.2	Variables . . . . .	17
2.3.3	Class <code>AnalysisChunk</code> . . . . .	17
2.3.4	Class <code>AnalysisJob</code> . . . . .	17
2.3.5	Class <code>AnalysisNode</code> . . . . .	18
2.3.6	Class <code>CondorDAG</code> . . . . .	19
2.3.7	Class <code>CondorDAGError</code> . . . . .	20
2.3.8	Class <code>CondorDAGJob</code> . . . . .	20
2.3.9	Class <code>CondorDAGNode</code> . . . . .	20
2.3.10	Class <code>CondorDAGNodeError</code> . . . . .	21
2.3.11	Class <code>CondorError</code> . . . . .	22
2.3.12	Class <code>CondorJob</code> . . . . .	22
2.3.13	Class <code>CondorJobError</code> . . . . .	23
2.3.14	Class <code>CondorSubmitError</code> . . . . .	23
2.3.15	Class <code>ScienceData</code> . . . . .	24
2.3.16	Class <code>ScienceSegment</code> . . . . .	24
2.3.17	Class <code>SegmentError</code> . . . . .	25

<b>3</b>	<b>LALApps programs</b>	<b>26</b>
3.1	Program <code>lalapps_hello</code>	27
3.2	Program <code>lalapps_animate</code>	29
3.3	Python Module <code>inspiral</code>	31
3.3.1	Variables	31
3.3.2	Class <code>DataFindJob</code>	31
3.3.3	Class <code>DataFindNode</code>	31
3.3.4	Class <code>IncaJob</code>	32
3.3.5	Class <code>IncaNode</code>	33
3.3.6	Class <code>InspirError</code>	33
3.3.7	Class <code>InspirJob</code>	34
3.3.8	Class <code>InspirNode</code>	34
3.3.9	Class <code>TmpltBankJob</code>	35
3.3.10	Class <code>TmpltBankNode</code>	35
3.3.11	Class <code>TrigToTmpltJob</code>	36
3.3.12	Class <code>TrigToTmpltNode</code>	36
3.4	Inspir Search Programs	38
3.4.1	Program <code>lalapps_inspir_pipe</code>	39
3.4.2	Program <code>lalapps_tmpltbank</code>	41
3.4.3	Program <code>lalapps_inspir</code>	43
3.4.4	Program <code>lalapps_inca</code>	45
3.4.5	Program <code>lalapps_inspinj</code>	48
3.4.6	Program <code>lalapps_snglInspirReader</code>	50
3.4.7	Program <code>lalapps_inspinj_find</code>	52
3.4.8	Program <code>lalapps_inspmultiawg</code>	54
3.5	Power Tools	57
3.5.1	Program <code>lalapps_power</code>	58
3.5.2	Program <code>lalapps_snglBurstHistogram</code>	62
3.5.3	Program <code>lalapps_burca</code>	64
3.5.4	Program <code>lalapps_binj</code>	66
3.6	Programs Related to Stochastic Background Searches	68
3.6.1	Program <code>lalapps_olapredfcn</code>	68
3.7	Program <code>lalapps_ring</code>	70
3.8	Program <code>lalapps_trump</code>	73
3.9	Program <code>lalapps_findchirp_post</code>	75

# Chapter 1

## How to get, build and use software for the LSC Data Grid

### 1.1 Getting and installing the software

We describe the installation of LAL and LALAPPS for use in gravitational-wave data analysis. These packages are under active development at the present time and the latest functionality is available only from CVS. These instructions assume a standard RedHat installation including development tools.

#### 1.1.1 Installation of required development tools

A useful installation of LAL and LALAPPS requires FFTW, the FRAME and METAIO libraries. Pre-compiled versions of these libraries are available as RPM's from

<http://www.lsc-group.phys.uwm.edu/lal/rpms/>

The binary versions

<code>fftw-2.1.3-2lal.i386.rpm</code>	<code>28-Aug-2003</code>	<code>17:35</code>	<code>810k</code>
<code>frame-6.08-1lal.i386.rpm</code>	<code>28-Aug-2003</code>	<code>17:35</code>	<code>464k</code>
<code>metaio-4.13-1lal.i386.rpm</code>	<code>28-Aug-2003</code>	<code>17:35</code>	<code>24k</code>
<code>stow-1.3.3-1lal.i386.rpm</code>	<code>29-Aug-2003</code>	<code>12:27</code>	<code>29k</code>

are compiled on RedHat7.3 and RedHat 9.0 architecture, but should work on other versions of RedHat. Download the RPM's, log in as root on your computer, then type

```
rpm -Uvh fftw-2.1.3-2lal.i386.rpm
rpm -Uvh frame-6.08-1lal.i386.rpm
rpm -Uvh metaio-4.13-1lal.i386.rpm
rpm -Uvh stow-1.3.3-1lal.i386.rpm
```

Watch for error messages. If everything installs correctly, skip to Sec. 1.1.2, otherwise continue through the following steps.

If the binary RPM's do not work for some reason or your particular O/S is not supported, the source RPM's are also available at the LAL web site. Download

```
fftw-2.1.3-2lal.src.rpm    28-Aug-2003 17:35    810k
frame-6.08-1lal.src.rpm   28-Aug-2003 17:35    464k
metaio-4.13-1lal.src.rpm  28-Aug-2003 17:35     24k
stow-1.3.3-1lal.src.rpm   29-Aug-2003 12:27    139k
```

to your computer. As root, type

```
rpm -Uvh fftw-2.1.3-2lal.src.rpm
rpm -Uvh frame-6.08-1lal.src.rpm
rpm -Uvh metaio-4.13-1lal.src.rpm
rpm -Uvh stow-1.3.3-1lal.src.rpm
```

Watch for errors. If this fails, there is a major problem: e-mail [lal-discuss@gravity.phys.uwm.edu](mailto:lal-discuss@gravity.phys.uwm.edu). If it succeeds, as root

```
cd /usr/src/redhat/SPECS
rpm -bb fftw.spec
rpm -bb frame.spec
rpm -bb metaio.spec
rpm -bb stow.spec
cd /usr/src/redhat/RPMS/i386
rpm -Uvh fftw-2.1.3-2lal.i386.rpm
rpm -Uvh frame-6.08-1lal.i386.rpm
rpm -Uvh metaio-4.13-1lal.i386.rpm
rpm -Uvh stow-1.3.3-1lal.i386.rpm
```

If there are problems, try the following:

1. Later versions of the RPM software require `rpm -bb` to be replaced by `rpmbuild -bb`.
2. If `rpm -bb package` (or `rpmbuild -bb package`) fail, you might be able to build by hand:

```
cd ../BUILD/package-dir
./configure
make
make install
```

where *package* is the package for which `rpmbuild -bb` did not work. The *package-dir* that belongs to *package* should be obvious with the exception of `frame`, for which it is `v6r08`. You do not need to do `rpm -Uvh package.i386.rpm` for packages installed in this way (this binary rpm should not even exist).

3. If none of this works, e-mail [lal-discuss@gravity.phys.uwm.edu](mailto:lal-discuss@gravity.phys.uwm.edu).

If it succeeds, then you can go ahead to the next stage.

## 1.1.2 Configuring your environment

While developing, we recommend that you install the LAL and LALAPPS somewhere under your home directory. If you follow the instructions below, the LAL and LALAPPS libraries will be in `$LALPREFIX/lib`; the documentation in `$LALPREFIX/doc`; the header files in `$LALPREFIX/include`; the binaries in `$LALPREFIX/bin`. The environment `LALPREFIX` should be set to the absolute path where you want to install these things. For example, user `patrick` might put the source code in `/home/patrick/src` and set `LALPREFIX` to `/home/patrick`.

To make the appropriate binaries accessible to you, check that your path is set correctly. It is also useful to add environment variables for the CVS servers. If you have a lal CVS login, change `anonymous@gravity.phys.uwm.edu` to `your_user_name@gravity.phys.uwm.edu` in the environment variables below. If you are using `bash`, add the following lines to your `.bash.profile` file

```
export LALPREFIX=${HOME}          # <---- Change this as appropriate
export PATH=${LALPREFIX}/bin:${PATH}
export LALCVS=":pserver:anonymous@gravity.phys.uwm.edu:/usr/local/cvs/lal"
```

If you are using `bash` or a derivative, add the following lines to your `.bashrc` file

```
setenv LALPREFIX $HOME
setenv PATH $LALPREFIX/bin:$PATH
setenv LALCVS ":pserver:anonymous@gravity.phys.uwm.edu:/usr/local/cvs/lal"
```

These environment variables must be set before running anything, so it is a good idea to log out and log back in again before continuing. **Note:** With `libtool` version 1.4.2, or greater, there is no need to have the `LD_LIBRARY_PATH` environment variable set. Library paths are hard coded into the libraries using the `-rpath` compiler option. This is the correct way to do this, using `LD_LIBRARY_PATH` is incorrect.

### 1.1.3 LAL

In the following commands, remember `LALPREFIX` is the absolute directory path where you want to install the LAL library. If `LALPREFIX` does not exist, you must create it:

```
mkdir -p $LALPREFIX
```

Then create the directory into which you wish to put the source files for LAL and LALApps:

```
mkdir $LALPREFIX/src
```

The LAL software is maintained in a CVS repository – CVS stands for Concurrent Version-control System which is a tool to allow multiple developers to manipulate the same software, merging differences and identifying conflicts between changes if they arise. Obtain the LAL package from the CVS repository as follows:

```
cd $LALPREFIX/src
cvs -d $LALCVS login
```

At this point, you will be asked for a password. The password for the `anonymous` user is `lal`. If you have your own username, use the password that you have been given. The version of LAL that is checked-out is identified by the argument to the `-r` option in the commands below. The `HEAD` tag checks out the current development version. If you want to check out a released version, replace the `HEAD` tag by `release-X-Y` where `X` and `Y` are integers which identify the release version number as `X.Y`.

```
cvs -d $LALCVS checkout -rHEAD lal
```

You have now obtained the latest development version of LAL.

#### Build and Install

To build and install the LAL software suite,

```
cd $LALPREFIX/src/lal
./00boot
./configure --prefix=$LALPREFIX \
            --enable-frame --enable-metaio \
            --enable-static --disable-shared \
            --with-extra-cflags=-g
make
```

```
make check
make dvi
make install prefix=$LALPREFIX/stow_pkgs/lal-howto
cd $LALPREFIX/stow_pkgs
stow lal-howto
```

This completes the installation and testing of LAL.

### 1.1.4 LALApps

In the following commands, remember `LALPREFIX` is the absolute directory path where you want to install the LALApps binaries. We assume that you have followed the build instructions for LAL and that the directories `LALPREFIX` and `$LALPREFIX/src` exist.

Obtain the LALApps package from the CVS repository as follows:

```
cd $LALPREFIX/src
cvs -d $LALCVS login
```

At this point, you will be asked for a password. The password for the `anonymous` user is `lal`. If you have your own username, use the password that you have been given. The version of LALApps that is checked-out is identified by the argument to the `-r` option in the commands below. The `HEAD` tag checks out the current development version. If you want to check out a released version, replace the `HEAD` tag by `release-X-Y` where `X` and `Y` are integers which identify the release version number as `X.Y`.

```
cvs -d $LALCVS checkout -rHEAD lalapps
```

You have now obtained the latest development version of LALApps.

#### Build and Install

To build and install the LALApps suite,

```
cd $LALPREFIX/src/lalapps
./00boot
./configure --prefix=$LALPREFIX \
  --with-extra-cppflags="-I$LALPREFIX/include" \
  --with-extra-ldflags="-L$LALPREFIX/lib" \
  --enable-frame --enable-metaio \
  --enable-static --disable-shared \
  --with-extra-cflags="-g -static"
make
make check
make dvi
make install prefix=$LALPREFIX/stow_pkgs/lalapps-howto
cd $LALPREFIX/stow_pkgs
stow lalapps-howto
```

This completes the installation and testing of LALApps.

Note if you are building lalapps on a machine with Condor installed and wish to run in the standard universe, append the option `--enable-condor` to the configure for lalapps before you type make. The lalapps executables will then be linked with `condor_compile` and checkpointing will be available.

### 1.1.5 Updating LAL/LALApps

Since LAL and LALAPPS are constantly being improved, you will eventually want to update the versions you have installed. These instructions give you some guidance on the process. As you become a more experienced developer, you will develop your own tricks for making this more efficient. To do a complete update:

1. Update the source code on your computer

```
cd $LALPREFIX/src/lal
make cvs-clean
cvs update -Ad
cd $LALPREFIX/src/lalapps
make cvs-clean
cvs update -Ad
```

2. Remove the old versions

```
cd $LALPREFIX/stow_pkgs
stow --delete lal-howto
stow --delete lalapps-howto
```

3. Repeat the **Build and Install** instructions for LAL and then LALAPPS, but **change the prefix argument** to `make install`. For example, if you compiled on 28 August 2003, you might want to use

```
make install prefix=$LALPREFIX/stow_pkgs/lalapps-030828
cd $LALPREFIX/stow_pkgs
stow lalapps-030828
```

## **Chapter 2**

# **LALApps utilities**

Several utilities (macros, global variables, and functions) are provided to assist in writing programs in LALApps, and for maintaining a standard look-and-feel. This chapter describes these utilities and concludes with the listing of an example program.

## 2.1 Header `lalapps.h`

Provides utilities for writing programs for LALApps.

Several macros, global variables, and function prototypes are given that will assist in writing LALApps programs, and will aid in maintaining a standard look-and-feel.

To use these utilities, include the header `lalapps.h` and make sure the program links to the object `lalapps.o`.

### 2.1.1 Function `set_debug_level`

#### Name

`set_debug_level` — sets the LAL debug level

#### Synopsis

```
#include <lalapps.h>
extern int lalDebugLevel;
int set_debug_level( const char *s );
```

#### Description

The function `set_debug_level` sets the LAL debug level to a value determined by the string `s`, which can be an absolute debug level (a string representing an integer) or a string of LAL debug level flags. Allowed flags are:

##### **NDEBUG**

No debugging information is printed and memory debugging code is disabled.

##### **ERROR**

Error messages are printed.

##### **WARNING**

Warning messages are printed.

##### **INFO**

Information messages are printed.

##### **TRACE**

Function call tracing messages are printed.

##### **MEMINFO**

Memory allocation information messages are printed.

##### **MEMDBG**

Debugging of memory allocation routines is enabled but no messages are printed.

The following pre-defined composite levels are available:

##### **MSGLVL1**

Equivalent to `ERROR`.

##### **MSGLVL2**

Equivalent to `ERROR` | `WARNING`.

##### **MSGLVL3**

Equivalent to `ERROR` | `WARNING` | `INFO`.

##### **ALLDBG**

All debugging messages are printed.

If the argument to `set_debug_level` is `NULL`, then the string stored in the environment variable `LAL_DEBUG_LEVEL` is used. If this environment is not defined, or if no flags or values are specified in the string, the debug level is set to 0, which is equivalent to `NDEBUG`. (This is also the default value for `lalDebugLevel` unless it is set to some other value.)

For example, the statement

```
set_debug_level( "ERROR | INFO" );
```

will set the debug level so that error and information messages are printed (but not warning messages). Another example is the statement

```
set_debug_level( "2" );
```

which would set the debug level to 2 (warning messages are printed).

**Return Value**

The return value is the (integer) debug level that is assigned to `lalDebugLevel`.

**Environment****LAL\_DEBUG\_LEVEL**

Default LAL debug level string to use.

### 2.1.2 Function `clear_status`

#### Name

`clear_status` — clears the LAL status structure after a failed LAL function call

#### Synopsis

```
#include <lalapps.h>
extern const LALStatus blank_status;
int clear_status( LALStatus *status );
```

#### Description

Clears the LAL status structure and iteratively frees attached (sic) any linked status structures. This is to be used after a failed LAL function call to restore the status structure to a useable form. The structure `blank_status` contains a blank status structure that can be used to initialize a status structure in the program.

#### Example

The following program calls a routine `LALFailUnlessNegative` twice, once with a positive argument (which causes the routine to fail) and once with a negative argument (which causes the routine to pass). The function `clear_status` is used to clean up the status structure after the failure and the constant structure `blank_status` is used to initialize the status structure.

```
#include <lalapps.h>
#include <lal/LALStdlib.h>

extern const LALStatus blank_status;

void LALFailUnlessNegative( LALStatus *status, INT4 n )
{
    INITSTATUS( status, "LALFail", "$Id$" );
    ATTATCHSTATUSPTR( status );
    ASSERT( n, status, 1, "Non-negative n" );
    if ( n > 0 )
    {
        TRY( LALFailUnlessNegative( status->statusPtr, n - 1 ), status );
    }
    DETATCHSTATUSPTR( status );
    RETURN( status );
}

int main( void )
{
    LALStatus status = blank_status;
    LALFailUnlessNegative( &status, 5 );
    clear_status( &status );
    LALFailUnlessNegative( &status, -2 );
    return status.statusCode;
}
```

### 2.1.3 Macro `RCSID`

**Name**

`RCSID` — set the RCS Id variable

**Synopsis**

```
#include <lalapps.h>
#ifndef RCSID
#define RCSID( id ) static volatile const char *rcsid = (id)
#endif
```

**Description**

`RCSID` sets the static (i.e., internal-linkage) variable `rcsid` to the RCS Id string, `$Id$`, which is given as the argument `id`. The string `$Id$` is expanded by RCS to contain the identification of the source file along with its revision number. For example:

```
RCSID( "$Id$" );
```

### 2.1.4 Macro `PRINT_VERSION`

#### Name

`PRINT_VERSION` — prints the LALApps version of the program

#### Synopsis

```
#include <lalapps.h>
static volatile const char *rcsid="$Id$";
#ifndef PRINT_VERSION
#define PRINT_VERSION( program ) \
    fprintf( stderr, PACKAGE " %s version " VERSION "\n%s\n", program, rcsid )
#endif
```

#### Description

`PRINT_VERSION` prints the version information for `program` in a standard format, along with the RCS Id information. For example, for the program `lalapps_hello`, the version information

```
lalapps hello version 0.1
$Id$
```

is printed with the command `lalapps_hello -V`. The source code to print this is

```
PRINT_VERSION( "hello" );
```

Note that `PRINT_VERSION` requires the string variable `rcsid` to be set.

### 2.1.5 Macro `LAL_CALL`

#### Name

`LAL_CALL` — call a LAL routine and handle any errors

#### Synopsis

```
#include <lalapps.h>

extern int vrblvl;
extern int ( *lal_errhandler )( LALStatus *stat, const char *func,
    const char *file, const int line, volatile const char *id );
extern lal_errhandler_t lal_errhandler;

static volatile const char *rcsid="$Id$";

#ifdef LAL_CALL
#define LAL_CALL( function, statusptr ) \
    ((function), lal_errhandler(statusptr, #function, __FILE__, __LINE__, rcsid))
#endif
```

#### Description

`LAL_CALL` executes the LAL function `function` and executes the error handler `lal_errhandler`, which examines the status structure `statusptr` to see if an error occurred. Typically the error handler will return with value 0 if there was no error; otherwise it will print a trace of the execution stack and then perform a specific action. The action performed depends on the error handler, which can be set to one of the following:

##### `LAL_ERR_DFLT`

The default error handler (same as `LAL_ERR_ABORT`).

##### `LAL_ERR_ABORT`

Raises `SIGABRT` if there is an error.

##### `LAL_ERR_EXIT`

Exits with the returned status code if there is an error.

##### `LAL_ERR_RTRN`

Returns the status code.

Note that `LAL_CALL` requires the string variable `rcsid` to be set.

#### Return Value

If `LAL_CALL` returns (rather than terminating execution), the return value is equal to the status code returned by the LAL function.

#### Example

The following example program illustrates the use of `LAL_CALL`. The routine `LALInvert` is called incorrectly twice. The first time the division by zero error is caught. The second time, the unexpected null pointer error is not caught and the default error handler aborts the program.

```
#include <stdlib.h>
#include <lalapps.h>
#include <lal/LALStdlib.h>

RCSID( "$Id$" );
```

```
extern int vrblvl;
extern const LALStatus blank_status;

void LALInvert( LALStatus *status, REAL4 *y, REAL4 x )
{
    INITSTATUS( status, "LALInvert", rcsid );
    ASSERT( y, status, 1, "Null pointer" );
    if ( input == 0 )
    {
        ABORT( status, 1, "Division by zero" );
    }
    *y = 1 / x;
    RETURN( status );
}

int main( void )
{
    LALStatus status = blank_status;
    REAL4 x;
    int code;

    vrblvl = 1;

    lal_errhandler = LAL_ERR_RTRN;
    code = LAL_CALL( LALInvert( &status, &x, 0 ), &status );
    if ( code == 2 )
    {
        puts( "division by zero" );
        clear_status( &status );
    }
    else if ( code )
    {
        exit( code );
    }

    lal_errhandler = LAL_ERR_DFLT;
    LAL_CALL( LALInvert( &status, NULL, 1 ), &status );

    return 0;
}
```

## 2.2 Source `hello.c`

This is the source code for the program `lalapps_hello`:

```
1  #include <stdio.h>
   #include <unistd.h>
   #include <lalapps.h>
   #include <lal/LALStdlib.h>
   #include <lal/LALHello.h>

   RCSID( "$Id: hello.c,v 1.1 2002/01/16 19:11:33 jolien Exp $" );

   #define usgfmt \
10  "Usage: %s [options]\n" \
   "Options [default in brackets]:\n" \
   " -h          print this message\n" \
   " -V          print version info\n" \
   " -v          verbose\n" \
   " -d dbglvl   set debug level to dbglvl [0]\n" \
   " -o outfile  use output file outfile [stdout]\n"

   #define usage( program ) fprintf( stderr, usgfmt, program )

20  extern char *optarg;
   extern int optind, opterr, optopt;
   extern int vrbflg;

   int main( int argc, char *argv[] )
   {
     const char *program = argv[0];
     const char *outfile = NULL;
     const char *dbglvl = NULL;
     lal_errhandler_t default_handler;
30  LALStatus status = blank_status;
     int code;
     int opt;

     /* parse options */
     while ( 0 < ( opt = getopt( argc, argv, "hVvd:o:" ) ) )
     {
       switch ( opt )
       {
40         case 'h':
           usage( program );
           return 0;
         case 'V':
           PRINT_VERSION( "hello" );
           return 0;
         case 'v':
           vrbflg = 1;

```

```
        break;
    case 'd':
        dbglvl = optarg;
50     break;
    case 'o':
        outfile = optarg;
        break;
    default:
        usage( program );
        return 1;
    }
}
if ( optind < argc )
60 {
    usage( program );
    return 1;
}

/* set debug level */
set_debug_level( dbglvl );

/* try to call LALHello; catch error LALHELLOH_EOPEN */
default_handler = lal_errhandler;
70 lal_errhandler = LAL_ERR_RTRN;
code = LAL_CALL( LALHello( &status, outfile ), &status );
if ( code == -1 && (status.statusPtr)->statusCode == LALHELLOH_EOPEN )
{
    fprintf( stderr, "warning: couldn't open file %s for output"
            "(using stdout)\n", outfile );
    clear_status( &status );
    lal_errhandler = LAL_ERR_EXIT;
    LAL_CALL( LALHello( &status, NULL ), &status );
}
80 else if ( code )
{
    exit( code );
}
lal_errhandler = default_handler; /* restore default handler */

LALCheckMemoryLeaks();
return 0;
}
```

## 2.3 Python Module [pipeline](#)

This modules contains objects that make it simple for the user to create python scripts that build Condor DAGs to run code on the LSC Data Grid.

### 2.3.1 Functions

<b>s2play</b> ( <i>t</i> )
Return 1 if <i>t</i> is in the S2 playground, 0 otherwise <i>t</i> = GPS time to test if playground

### 2.3.2 Variables

Name	Description
<code>__author__</code>	<b>Value:</b> 'Duncan Brown <duncan@gravity.phys.uwm.edu>'
<code>__date__</code>	<b>Value:</b> '\$Date: 2003/10/01 08:46:06 \$'
<code>__version__</code>	<b>Value:</b> '1.18'

### 2.3.3 Class AnalysisChunk

An AnalysisCunk is the unit of data that a node works with, usually some subset of a ScienceSegment.

#### Methods

<b>__init__</b> ( <i>self, start, end</i> )
start = GPS start time of the chunk. end = GPS end time of the chunk.
<b>__len__</b> ( <i>self</i> )
Returns the length of this AnalysisChunk in seconds.
<b>__repr__</b> ( <i>self</i> )
<b>dur</b> ( <i>self</i> )
Returns the length (duration) of the chunk in seconds.
<b>end</b> ( <i>self</i> )
Returns the GPS end time of the chunk.
<b>start</b> ( <i>self</i> )
Returns the GPS start time of the chunk.

### 2.3.4 Class AnalysisJob

Describes a generic analysis job that filters LIGO data as configured by an ini file.

**Methods**

<b>__init__(self, cp)</b>
cp = ConfigParser object that contains the configuration for this job.
<b>calibration(self, ifo)</b>
Returns the name of the calibration file to use for the given IFO. ifo = name of interferometer (e.g. L1, H1 or H2).
<b>channel(self)</b>
Returns the name of the channel that this job is filtering. Note that channel is defined to be IFO independent, so this may be LSC-AS_Q or IOO-MC_F. The IFO is set on a per node basis, not a per job basis.
<b>get_config(self, sec, opt)</b>
Get the configuration variable in a particular section of this jobs ini file. sec = ini file section. opt = option from section sec.

**2.3.5 Class AnalysisNode**

pipeline.CondorDAGNode

**AnalysisNode**

Contains the methods that allow an object to be built to analyse LIGO data in a Condor DAG.

**Methods**

<b>__init__(self)</b>
Overrides: pipeline.CondorDAGNode.__init__
<b>get_end(self)</b>
Get the GPS end time of the node.
<b>get_ifo(self)</b>
Returns the two letter IFO code for this node.
<b>get_input(self)</b>
Get the file that will be passed as input.
<b>get_output(self)</b>
Get the file that will be passed as output.
<b>get_start(self)</b>
Get the GPS start time of the node.
<b>set_cache(self, file)</b>
Set the LAL frame cache to use. The frame cache is passed to the job with the <code>-frame-cache</code> argument. file = calibration file to use.

**set\_end**(*self*, *time*)

Set the GPS end time of the analysis node by setting a `-gps-end-time` option to the node when it is executed. `time` = GPS end time of job.

**set\_ifo**(*self*, *ifo*)

Set the channel name to analyze and add a calibration file for that channel. The name of the ifo is prepended to the channel name obtained from the job configuration file and passed with a `-channel-name` option. A calibration file is obtained from the ini file and passed with a `-calibration-cache` option. `ifo` = two letter ifo code (e.g. L1, H1 or H2).

**set\_input**(*self*, *file*)

Add an input to the node by adding a `-input` option. `file` = option argument to pass as input.

**set\_output**(*self*, *file*)

Add an output to the node by adding a `-output` option. `file` = option argument to pass as output.

**set\_start**(*self*, *time*)

Set the GPS start time of the analysis node by setting a `-gps-start-time` option to the node when it is executed. `time` = GPS start time of job.

**Inherited from CondorDAGNode:** `__repr__`, `add_parent`, `add_var_arg`, `add_var_opt`, `job`, `set_log_file`, `set_name`, `set_retry`, `write_job`, `write_parents`, `write_vars`

### 2.3.6 Class CondorDAG

A CondorDAG is a Condor Directed Acyclic Graph that describes a collection of Condor jobs and the order in which to run them. All Condor jobs in the DAG must write their Condor logs to the same file. NOTE: The log file must not be on an NFS mounted system as the Condor jobs must be able to get an exclusive file lock on the log file.

#### Methods

**\_\_init\_\_**(*self*, *log*)

`log` = path to log file which must not be on an NFS mounted file system.

**add\_node**(*self*, *node*)

Add a CondorDAGNode to this DAG. The CondorJob that the node uses is also added to the list of Condor jobs in the DAG so that a list of the submit files needed by the DAG can be maintained. Each unique CondorJob will be added once to prevent duplicate submit files being written. `node` = CondorDAGNode to add to the CondorDAG.

**set\_dag\_file**(*self*, *path*)

Set the name of the file into which the DAG is written. `path` = path to DAG file.

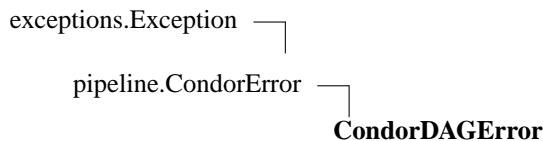
**write\_dag**(*self*)

Write all the nodes in the DAG to the DAG file.

**write\_sub\_files**(*self*)

Write all the submit files used by the dag to disk. Each submit file is written to the file name set in the CondorJob.

### 2.3.7 Class CondorDAGError

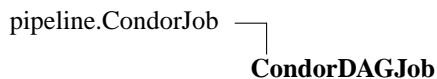


#### Methods

**Inherited from Exception:** `__getitem__`, `__str__`

**Inherited from CondorError:** `__init__`

### 2.3.8 Class CondorDAGJob



A Condor DAG job never notifies the user on completion and can have variable options that are set for a particular node in the DAG. Inherits methods from a CondorJob.

#### Methods

<b><code>__init__(self, universe, executable)</code></b>
--

universe = the condor universe to run the job in. executable = the executable to run in the DAG.
--

Overrides: pipeline.CondorJob. <code>__init__</code>
--

<b><code>add_var_arg(self)</code></b>
---------------------------------------

Add a command to the submit file to allow variable (macro) arguments to be passed to the executable.
--

<b><code>add_var_opt(self, opt)</code></b>
--

Add a variable (or macro) option to the condor job. The option is added to the submit file and a different argument to the option can be set for each node in the DAG. opt = name of option to add.
---

**Inherited from CondorJob:** `add_arg`, `add_condor_cmd`, `add_ini_opts`, `add_opt`, `get_stderr_file`, `get_stdout_file`, `get_sub_file`, `set_log_file`, `set_notification`, `set_stderr_file`, `set_stdout_file`, `set_sub_file`, `write_sub_file`

### 2.3.9 Class CondorDAGNode

**Known Subclasses:** AnalysisNode

A CondorDAGNode represents a node in the DAG. It corresponds to a particular condor job (and so a particular submit file). If the job has variable (macro) options, they can be set here so each nodes executes with the correct options.

#### Methods

<b><code>__init__(self, job)</code></b>
---

job = the CondorJob that this node corresponds to.
--

<b><code>__repr__(self)</code></b>
------------------------------------

**add\_parent**(*self*, *node*)

Add a parent to this node. This node will not be executed until the parent node has run successfully. *node* = CondorDAGNode to add as a parent.

**add\_var\_arg**(*self*, *arg*)

Add a variable (or macro) argument to the condor job. The argument is added to the submit file and a different value of the argument can be set for each node in the DAG. *arg* = name of option to add.

**add\_var\_opt**(*self*, *opt*, *value*)

Add the a variable (macro) options for this node. If the option specified does not exist in the CondorJob, it is added so the submit file will be correct when written. *opt* = option name. *value* = value of the option for this node in the DAG.

**job**(*self*)

Return the CondorJob that this node is associated with.

**set\_log\_file**(*self*, *log*)

Set the Condor log file to be used by this CondorJob. *log* = path of Condor log file.

**set\_name**(*self*)

Generate a unique name for this node in the DAG.

**set\_retry**(*self*, *retry*)

Set the number of times that this node in the DAG should retry. *retry* = number of times to retry node.

**write\_job**(*self*, *fh*)

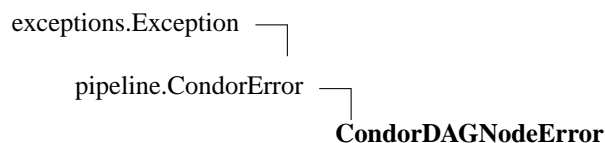
Write the DAG entry for this node's job to the DAG file descriptor. *fh* = descriptor of open DAG file.

**write\_parents**(*self*, *fh*)

Write the parent/child relations for this job to the DAG file descriptor. *fh* = descriptor of open DAG file.

**write\_vars**(*self*, *fh*)

Write the variable (macro) options and arguments to the DAG file descriptor. *fh* = descriptor of open DAG file.

**2.3.10 Class CondorDAGNodeError****Methods**

**Inherited from Exception:** `__getitem__`, `__str__`

**Inherited from CondorError:** `__init__`

### 2.3.11 Class CondorError

exceptions.Exception

└─ CondorError

**Known Subclasses:** CondorDAGError, CondorDAGNodeError, CondorJobError, CondorSubmitError  
Error thrown by Condor Jobs

#### Methods

`__init__(self, args=None)`

Overrides: exceptions.Exception.\_\_init\_\_

**Inherited from Exception:** \_\_getitem\_\_, \_\_str\_\_

### 2.3.12 Class CondorJob

**Known Subclasses:** CondorDAGJob

Generic condor job class. Provides methods to set the options in the condor submit file for a particular executable

#### Methods

`__init__(self, universe, executable, queue)`

universe = the condor universe to run the job in. executable = the executable to run. queue = number of jobs to queue.

`add_arg(self, arg)`

Add an argument to the executable. Arguments are appended after any options and their order is guaranteed. arg = argument to add.

`add_condor_cmd(self, cmd, value)`

Add a Condor command to the submit file (e.g. a class add or environment). cmd = Condor command directive. value = value for command.

`add_ini_opts(self, cp, section)`

Parse command line options from a given section in an ini file and pass to the executable. cp = ConfigParser object pointing to the ini file. section = section of the ini file to add to the options.

`add_opt(self, opt, value)`

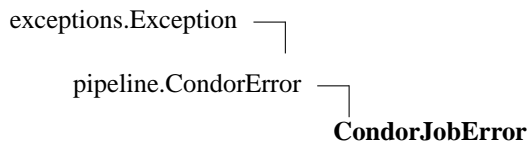
Add a command line option to the executable. The order that the arguments will be appended to the command line is not guaranteed, but they will always be added before any command line arguments. The name of the option is prefixed with double hyphen and the program is expected to parse it with `getopt.long()`. arg = command line option to add. value = value to pass to the option (None for no argument).

`get_stderr_file(self)`

Get the file to which Condor directs the stderr of the job.

<b>get_stdout_file</b> ( <i>self</i> )
Get the file to which Condor directs the stdout of the job.
<b>get_sub_file</b> ( <i>self</i> )
Get the name of the file which the Condor submit file will be written to when <code>write_sub_file()</code> is called. <code>path = path</code> to submit file.
<b>set_log_file</b> ( <i>self</i> , <i>path</i> )
Set the Condor log file. <code>path = path</code> to log file.
<b>set_notification</b> ( <i>self</i> , <i>value</i> )
Set the email address to send notification to. <code>value = email address or never</code> for no notification.
<b>set_stderr_file</b> ( <i>self</i> , <i>path</i> )
Set the file to which Condor directs the stderr of the job. <code>path = path</code> to stderr file.
<b>set_stdout_file</b> ( <i>self</i> , <i>path</i> )
Set the file to which Condor directs the stdout of the job. <code>path = path</code> to stdout file.
<b>set_sub_file</b> ( <i>self</i> , <i>path</i> )
Set the name of the file to write the Condor submit file to when <code>write_sub_file()</code> is called. <code>path = path</code> to submit file.
<b>write_sub_file</b> ( <i>self</i> )
Write a submit file for this Condor job.

### 2.3.13 Class `CondorJobError`

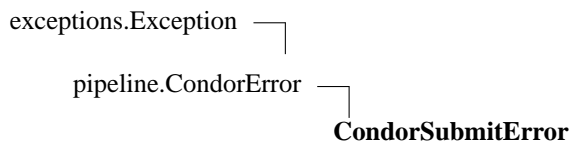


#### Methods

**Inherited from Exception:** `__getitem__`, `__str__`

**Inherited from CondorError:** `__init__`

### 2.3.14 Class `CondorSubmitError`



**Methods****Inherited from Exception:** `__getitem__`, `__str__`**Inherited from CondorError:** `__init__`**2.3.15 Class ScienceData**

An object that can contain all the science data used in an analysis. Can contain multiple ScienceSegments and has a method to generate these from a text file produces by the LIGOTOOLS segwizard program.

**Methods**

<code>__init__(self)</code>
<code>__getitem__(self, i)</code> Allows direct access to or iteration over the ScienceSegments associated with the ScienceData.
<code>__len__(self)</code> Returns the number of ScienceSegments associated with the ScienceData.
<code>__repr__(self)</code>
<code>make_chunks(self, length, overlap, play)</code> Divide each ScienceSegment contained in this object into AnalysisChunks. length = length of chunk in seconds. overlap = overlap between segments. play = if true, only generate chunks that overlap with S2 playground data.
<code>read(self, file)</code> Parse the science segments from the segwizard output contained in file. file = input text file containing a list of science segments generated by segwizard.

**2.3.16 Class ScienceSegment**

A ScienceSegment is a period of time where the experimenters determine that the interferometer is in a state where the data is suitable for scientific analysis. A science segment can have a list of AnalysisChunks associated with it that break the segment up into (possibly overlapping) smaller time intervals for analysis.

**Methods**

<code>__init__(self, segment)</code> segment = a tuple containing the (segment id, gps start time, gps end time, duration) of the segment.
<code>__getitem__(self, i)</code> Allows iteration over and direct access to the AnalysisChunks contained in this ScienceSegment.
<code>__len__(self)</code> Returns the number of AnalysisChunks contained in this ScienceSegment.
<code>__repr__(self)</code>

**add\_chunk**(*self*, *start*, *end*)

Add an AnalysisChunk to the list associated with this ScienceSegment. *start* = GPS start time of chunk. *end* = GPS end time of chunk.

**dur**(*self*)

Returns the length (duration) in seconds of this ScienceSegment.

**end**(*self*)

Returns the GPS end time of this ScienceSegment.

**id**(*self*)

Returns the ID of this ScienceSegment.

**make\_chunks**(*self*, *length*=0, *overlap*=0, *play*=0)

Divides the science segment into chunks of *length* seconds overlapped by *overlap* seconds. If the *play* option is set, only chunks that contain S2 playground data are generated. If the user has a more complicated way of generating chunks, this method should be overridden in a sub-class. Any data at the end of the ScienceSegment that is too short to contain a chunk is ignored. The length of this unused data is stored and can be retrieved with the `unused()` method. *length* = length of chunk in seconds. *overlap* = overlap between chunks in seconds. *play* = only generate chunks that overlap with S2 playground data.

**start**(*self*)

Returns the GPS start time of this ScienceSegment.

**unused**(*self*)

Returns the length of data in the science segment not used to make chunks.

### 2.3.17 Class SegmentError

```

exceptions.Exception ─┐
                       │
                       └─ SegmentError
  
```

#### Methods

**\_\_init\_\_**(*self*, *args*=None)

Overrides: `exceptions.Exception.__init__`

**Inherited from Exception:** `__getitem__`, `__str__`

## **Chapter 3**

# **LALApps programs**

This chapter describes the programs that are part of the LALApps package.

## 3.1 Program `lalapps_hello`

### Name

`lalapps_hello` — prints “hello LSC!”

### Synopsis

`lalapps_hello` [-h] [-V] [-v] [-d *dbglvl*] [-o *outfile*]

### Description

`lalapps_hello` prints “hello LSC!” to the screen or to an output file.

### Options

- h**  
Print a help message.
- V**  
Print the version information.
- v**  
Verbose output.
- d *dbglvl***  
Set LAL debug level to *dbglvl*.
- o *outfile***  
Write the output to file *outfile*.

### Debug levels

The LAL debug level can be specified as an integer or as a string of flags:

#### **NDEBUG**

No debugging information is printed and memory debugging code is disabled.

#### **ERROR**

Error messages are printed.

#### **WARNING**

Warning messages are printed.

#### **INFO**

Information messages are printed.

#### **TRACE**

Function call tracing messages are printed.

#### **MEMINFO**

Memory allocation information messages are printed.

#### **MEMDBG**

Debugging of memory allocation routines is enabled but no messages are printed.

The following composite levels are available:

**MSGLVL1**

Equivalent to `ERROR`

**MSGLVL2**

Equivalent to `ERROR | WARNING`

**MSGLVL3**

Equivalent to `ERROR | WARNING | INFO`

**ALLDBG**

All debugging messages are printed.

For example, the command

```
lalapps_hello -d "ERROR | INFO"
```

will set the debug level so that error and information messages are printed.

**Environment**

**LAL\_DEBUG\_LEVEL**

Default LAL debug level to use.

**Author**

Jolien Creighton

## 3.2 Program `lalapps_animate`

### Name

`lal_animate` — produces an animated display showing the time series output of a selected channel in a lower window, and a simultaneously calculated FFT power spectrum in the upper window

### Synopsis

```
lal_animate [--help] [--channel name] [--duration secs] [--epoch sec nsec] [--framedir dirname] [--highpass freq attenuation] [--numpts npoints] | xmgr -pipe
```

### Description

`lal_animate` produces an animated display showing the time series output of a selected channel in a lower window, and a simultaneously calculated FFT power spectrum in the upper window. The output from this program must be piped into the graphing program `xmgr`.

### Options

- `--help`  
Print a help message.
- `--channel` *name*  
Name of frame channel
- `--duration` *secs*  
How many seconds to look at
- `--epoch` *sec nsec*  
Starting epoch
- `--framedir` *dirname*  
Directory containing frame files
- `--highpass` *freq attenuation*  
High-pass filter parameters
- `--numpts` *npoints*  
Points per graph to display

### Example

To run the program, type:

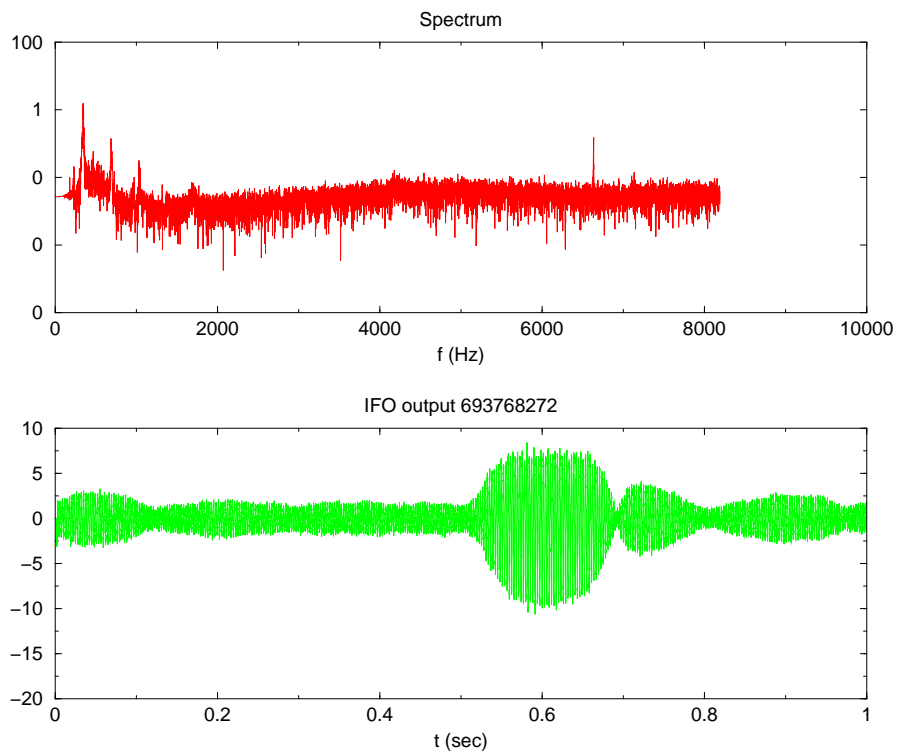
```
lalapps_animate --channel H2:LSC-AS_Q --framedir ./h1 --numpts 16384 \  
--epoch 693768272 0 --duration 1 --highpass 300 0.01 | xmgr -pipe
```

This will search in directory `./h1` for frame files containing the channel `H2:LSC-AS_Q` and pipe the data starting at 693768272 GPS seconds and 0 GPS nanoseconds to `xmgr` in segments containing 16384 points until 1 seconds of data has been reviewed. The data is highpass filtered to above 300 Hz with an attenuation of 0.1; the output is shown in Fig. 3.2

### Author

Bruce Allen and Patrick Brady

Figure 3.1: Example of output from `lalapps_animate` program



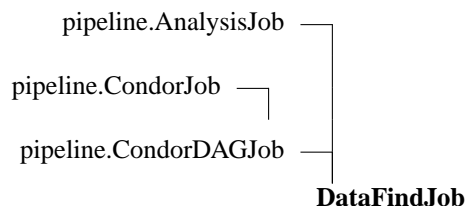
## 3.3 Python Module `inspiral`

Classes needed for the inspiral analysis pipeline. This script produced the necessary condor submit and dag files to run the standalone inspiral code on LIGO data

### 3.3.1 Variables

Name	Description
<code>__author__</code>	<b>Value:</b> 'Duncan Brown <duncan@gravity.phys.uwm.edu>'
<code>__date__</code>	<b>Value:</b> '\$Date: 2003/10/01 09:05:08 \$'
<code>__version__</code>	<b>Value:</b> '1.14'

### 3.3.2 Class `DataFindJob`



A LALdataFind job used by the inspiral pipeline. The static options are read from the section [datafind] in the ini file. The stdout from LALdataFind contains the paths to the frame files and is directed to a file in the cache directory named by site and GPS start and end times. The stderr is directed to the logs directory. The job always runs in the scheduler universe. The path to the executable is determined from the ini file.

#### Methods

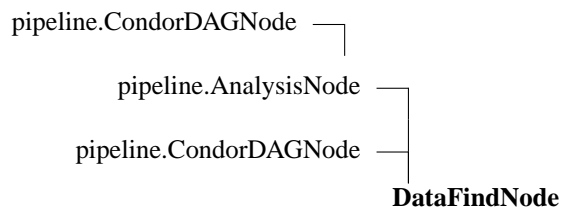
<code>__init__(self, cp)</code>
cp = ConfigParser object from which options are read.
Overrides: pipeline.CondorDAGJob.__init__

**Inherited from AnalysisJob:** calibration, channel, get\_config

**Inherited from CondorDAGJob:** add\_var\_arg, add\_var\_opt

**Inherited from CondorJob:** add\_arg, add\_condor\_cmd, add\_ini\_opts, add\_opt, get\_stderr\_file, get\_stdout\_file, get\_sub\_file, set\_log\_file, set\_notification, set\_stderr\_file, set\_stdout\_file, set\_sub\_file, write\_sub\_file

### 3.3.3 Class `DataFindNode`



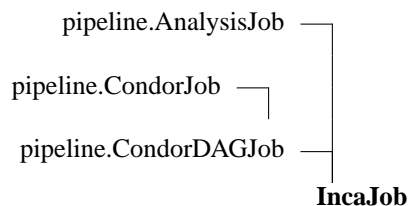
A DataFindNode runs an instance of datafind in a Condor DAG.

**Methods**

<b><code>__init__(self, job)</code></b>
job = A CondorDAGJob that can run an instance of LALdataFind. Overrides: pipeline.AnalysisNode.__init__
<b><code>get_output(self)</code></b>
Return the output file, i.e. the file containing the frame cache data. Overrides: pipeline.AnalysisNode.get_output
<b><code>set_end(self, time)</code></b>
Set the end time of the datafind query. time = GPS end time of query. Overrides: pipeline.AnalysisNode.set_end
<b><code>set_ifo(self, ifo)</code></b>
Set the IFO to retrieve data for. Since the data from both Hanford interferometers is stored in the same frame file, this takes the first letter of the IFO (e.g. L or H) and passes it to the <code>-instrument</code> option of LALdataFind. ifo = IFO to obtain data for. Overrides: pipeline.AnalysisNode.set_ifo
<b><code>set_start(self, time)</code></b>
Set the start time of the datafind query. time = GPS start time of query. Overrides: pipeline.AnalysisNode.set_start

**Inherited from AnalysisNode:** `get_end`, `get_ifo`, `get_input`, `get_start`, `set_cache`, `set_input`, `set_output`

**Inherited from CondorDAGNode:** `__repr__`, `add_parent`, `add_var_arg`, `add_var_opt`, `job`, `set_log_file`, `set_name`, `set_retry`, `write_job`, `write_parents`, `write_vars`

**3.3.4 Class IncaJob**

A `lalapps.inca` job used by the `inspiral` pipeline. The static options are read from the section `[inca]` in the ini file. The stdout and stderr from the job are directed to the logs directory. The job always runs in the scheduler universe. The path to the executable is determined from the ini file.

**Methods**

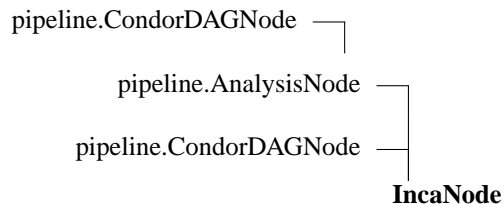
<b><code>__init__(self, cp)</code></b>
cp = ConfigParser object from which options are read. Overrides: pipeline.CondorDAGJob.__init__

**Inherited from AnalysisJob:** `calibration`, `channel`, `get_config`

**Inherited from CondorDAGJob:** `add_var_arg`, `add_var_opt`

**Inherited from CondorJob:** `add_arg`, `add_condor_cmd`, `add_ini_opts`, `add_opt`, `get_stderr_file`, `get_stdout_file`, `get_sub_file`, `set_log_file`, `set_notification`, `set_stderr_file`, `set_stdout_file`, `set_sub_file`, `write_sub_file`

### 3.3.5 Class `IncaNode`



An `IncaNode` runs an instance of the `inspiral` coincidence code in a Condor DAG.

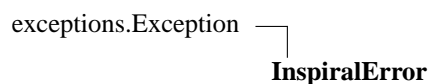
#### Methods

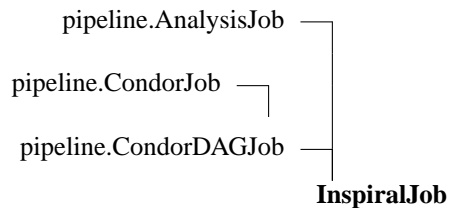
<b><code>__init__(self, job)</code></b>
<code>job</code> = A <code>CondorDAGJob</code> that can run an instance of <code>lalapps_inca</code> . Overrides: <code>pipeline.AnalysisNode.__init__</code>
<b><code>get_ifo_a(self)</code></b>
Returns the IFO code of the primary interferometer.
<b><code>get_ifo_b(self)</code></b>
Returns the IFO code of the primary interferometer.
<b><code>get_output(self)</code></b>
Returns the file name of output from the <code>inca</code> code. This must be kept synchronized with the name of the output file in <code>inca.c</code> . Overrides: <code>pipeline.AnalysisNode.get_output</code>
<b><code>set_ifo_a(self, ifo)</code></b>
Set the interferometer code to use as IFO A. <code>ifo</code> = IFO code (e.g. L1, H1 or H2).
<b><code>set_ifo_b(self, ifo)</code></b>
Set the interferometer code to use as IFO B. <code>ifo</code> = IFO code (e.g. L1, H1 or H2).

**Inherited from AnalysisNode:** `get_end`, `get_ifo`, `get_input`, `get_start`, `set_cache`, `set_end`, `set_ifo`, `set_input`, `get_output`, `set_start`

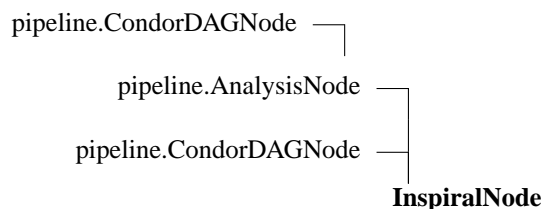
**Inherited from CondorDAGNode:** `__repr__`, `add_parent`, `add_var_arg`, `add_var_opt`, `job`, `set_log_file`, `set_name`, `set_retry`, `write_job`, `write_parents`, `write_vars`

### 3.3.6 Class `InspiralError`



**Methods**`__init__(self, args=None)`Overrides: `exceptions.Exception.__init__`**Inherited from Exception:** `__getitem__`, `__str__`**3.3.7 Class InspiralJob**

A `lalapps_inspirals` job used by the `inspiral` pipeline. The static options are read from the sections `[data]` and `[inspiral]` in the ini file. The `stdout` and `stderr` from the job are directed to the logs directory. The job runs in the universe specified in the ini file. The path to the executable is determined from the ini file.

**Methods**`__init__(self, cp)``cp` = `ConfigParser` object from which options are read.Overrides: `pipeline.CondorDAGJob.__init__`**Inherited from AnalysisJob:** `calibration`, `channel`, `get_config`**Inherited from CondorDAGJob:** `add_var_arg`, `add_var_opt`**Inherited from CondorJob:** `add_arg`, `add_condor_cmd`, `add_ini_opts`, `add_opt`, `get_stderr_file`, `get_stdout_file`, `get_sub_file`, `set_log_file`, `set_notification`, `set_stderr_file`, `set_stdout_file`, `set_sub_file`, `write_sub_file`**3.3.8 Class InspiralNode**

An `InspiralsNode` runs an instance of the `inspiral` code in a Condor DAG.

**Methods**`__init__(self, job)``job` = A `CondorDAGJob` that can run an instance of `lalapps_inspirals`.Overrides: `pipeline.AnalysisNode.__init__`

**get\_output(*self*)**

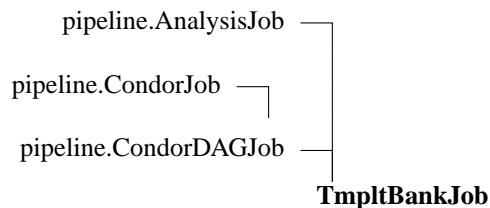
Returns the file name of output from the inspiral code. This must be kept synchronized with the name of the output file in `inspiral.c`.

Overrides: `pipeline.AnalysisNode.get_output`

**set\_bank(*self*, *bank*)**

**Inherited from `AnalysisNode`:** `get_end`, `get_ifo`, `get_input`, `get_start`, `set_cache`, `set_end`, `set_ifo`, `set_input`, `set_output`, `set_start`

**Inherited from `CondorDAGNode`:** `__repr__`, `add_parent`, `add_var_arg`, `add_var_opt`, `job`, `set_log_file`, `set_name`, `set_retry`, `write_job`, `write_parents`, `write_vars`

**3.3.9 Class `TmpltBankJob`**

A `lalapps_tmpltbank` job used by the inspiral pipeline. The static options are read from the sections `[data]` and `[tmpltbank]` in the ini file. The stdout and stderr from the job are directed to the logs directory. The job runs in the universe specified in the ini file. The path to the executable is determined from the ini file.

**Methods****\_\_init\_\_(*self*, *cp*)**

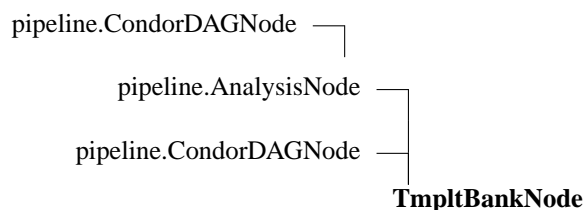
`cp` = `ConfigParser` object from which options are read.

Overrides: `pipeline.CondorDAGJob.__init__`

**Inherited from `AnalysisJob`:** `calibration`, `channel`, `get_config`

**Inherited from `CondorDAGJob`:** `add_var_arg`, `add_var_opt`

**Inherited from `CondorJob`:** `add_arg`, `add_condor_cmd`, `add_ini_opts`, `add_opt`, `get_stderr_file`, `get_stdout_file`, `get_sub_file`, `set_log_file`, `set_notification`, `set_stderr_file`, `set_stdout_file`, `set_sub_file`, `write_sub_file`

**3.3.10 Class `TmpltBankNode`**

A `TmpltBankNode` runs an instance of the template bank generation job in a Condor DAG.

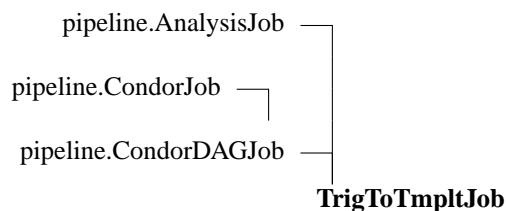
**Methods**

<code>__init__(self, job)</code>
job = A CondorDAGJob that can run an instance of <code>lalapps_tmplbank</code> .
Overrides: <code>pipeline.AnalysisNode.__init__</code>

<code>get_output(self)</code>
Returns the file name of output from the template bank code. This must be kept synchronized with the name of the output file in <code>tmplbank.c</code> .
Overrides: <code>pipeline.AnalysisNode.get_output</code>

**Inherited from AnalysisNode:** `get_end`, `get_ifo`, `get_input`, `get_start`, `set_cache`, `set_end`, `set_ifo`, `set_input`, `set_output`, `set_start`

**Inherited from CondorDAGNode:** `__repr__`, `add_parent`, `add_var_arg`, `add_var_opt`, `job`, `set_log_file`, `set_name`, `set_retry`, `write_job`, `write_parents`, `write_vars`

**3.3.11 Class TrigToTmplJob**

A `lalapps.trigtotmpl` job used by the `inspiral` pipeline. The static options are read from the section `[trigtotmpl]` in the ini file. The stdout and stderr from the job are directed to the logs directory. The job always runs in the scheduler universe. The path to the executable is determined from the ini file.

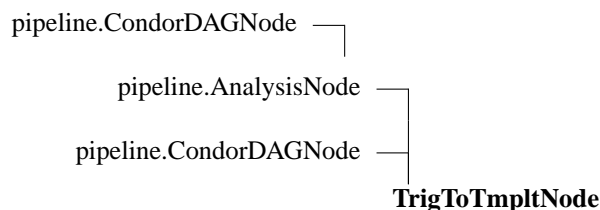
**Methods**

<code>__init__(self, cp)</code>
cp = ConfigParser object from which options are read.
Overrides: <code>pipeline.CondorDAGJob.__init__</code>

**Inherited from AnalysisJob:** `calibration`, `channel`, `get_config`

**Inherited from CondorDAGJob:** `add_var_arg`, `add_var_opt`

**Inherited from CondorJob:** `add_arg`, `add_condor_cmd`, `add_ini_opts`, `add_opt`, `get_stderr_file`, `get_stdout_file`, `get_sub_file`, `set_log_file`, `set_notification`, `set_stderr_file`, `set_stdout_file`, `set_sub_file`, `write_sub_file`

**3.3.12 Class TrigToTmplNode**

A `TrigToTmplNode` runs an instance of the triggered bank generator in a Condor DAG.

**Methods**

<code>__init__(self, job)</code>
----------------------------------

job = A CondorDAGJob that can run an instance of `lalapps.trigtotmplt`.

Overrides: `pipeline.AnalysisNode.__init__`

**Inherited from `AnalysisNode`:** `get_end`, `get_ifo`, `get_input`, `get_output`, `get_start`, `set_cache`, `set_end`, `set_ifo`, `set_input`, `set_output`, `set_start`

**Inherited from `CondorDAGNode`:** `__repr__`, `add_parent`, `add_var_arg`, `add_var_opt`, `job`, `set_log_file`, `set_name`, `set_retry`, `write_job`, `write_parents`, `write_vars`

## 3.4 Inspiral Search Programs

This section of LALAPPS contains programs that can be used to search interferometer data for inspiral signals using templated matched filtering and associated veto strategies.

### 3.4.1 Program `lalapps_inspiral_pipe`

#### Name

`lalapps_inspiral_pipe` — python script to generate Condor DAGs to run the inspiral pipeline.

#### Synopsis

```

-h, --help                display this message
-v, --version             print version information and exit
-u, --user-tag TAG       tag the job with TAG (overrides value in ini file)

-d, --datafind           run LALdataFind to create frame cache files
-t, --template-bank     run lalapps_tmpltbank to generate a template bank
-i, --inspiral          run lalapps_inspiral on the first IFO
-T, --triggered-bank    run lalapps_trigtotmplt to generate a triggered bank
-I, --triggered-inspiral run lalapps_inspiral on the second IFO
-C, --coincidence       run lalapps_inca on the triggers from both IFOs

-j, --injections        add simulated inspirals from injection file

-p, --playground-only   only create chunks that overlap with playground
-P, --priority PRIO     run jobs with condor priority PRIO

-f, --config-file FILE  use configuration file FILE
-l, --log-path PATH     directory to write condor log file

```

#### Description

`lalapps_inspiral_pipe` generates a Condor DAG to run the inspiral analysis pipeline. The configuration file should specify the parameters needed to run the jobs and must be specified with the `--config-file` option. A file containing science segments to be analyzed should be specified in the `[input]` section of the configuration file with a line such as

```
segments = S2H1L1v03_selectedsegs.txt
```

This should contain four whitespace separated columns:

```
segment_id    gps_start_time  gps_end_time    duration
```

that define the science segments to be used. Lines starting with an octothorpe are ignored.

The analysis chunk size is determined from the number of data segments and their length and overlap specified in config file. A chunk length is typically this is 1024 seconds for S2. The chunks start and stop times are computed from the science segment times and used to build the DAG.

Once the DAG file has been created it should be submitted to the Condor pool with the `condor_submit_dag` command.

#### Options

##### `--help`

Display a brief usage summary.

**Example**

Generate a DAG to run an inspiral search on the first IFO. The generated DAG is then submitted with [condor\\_submit\\_dag](#)

```
lalapps_inspiral_pipe --log-path /people/duncan/dag_logs \  
--datafind --template-bank --inspiral --playground-only \  
--config-file l1_s2.ini
```

```
condor_submit_dag l1_s2.dag
```

**Author**

Duncan Brown

### 3.4.2 Program `lalapps_tmpltbank`

#### Name

`lalapps_tmpltbank` — program to generate inspiral template banks.

#### Synopsis

```

--help                display this message
--verbose             print progress information
--debug-level LEVEL  set the LAL debug level to LEVEL
--user-tag STRING    set the process_params usertag to STRING
--comment STRING     set the process table comment to STRING

--gps-start-time SEC  GPS second of data start time
--gps-end-time SEC    GPS second of data end time
--pad-data T          pad the data start and end time by T seconds

--frame-cache         obtain frame data from LAL frame cache FILE
--calibration-cache FILE obtain calibration from LAL frame cache FILE
--channel-name CHAN  read data from interferometer channel CHAN

--sample-rate F       filter data at F Hz, downsampling if necessary
--resample-filter TYPE set resample filter to TYPE [ldas|butterworth]

--disable-high-pass   turn off the IIR highpass filter
--enable-high-pass F  high pass data above F Hz using an IIR filter
--spectrum-type TYPE  use PSD estimator TYPE [mean|median]

--segment-length N    set data segment length to N points
--number-of-segments N set number of data segments to N

--low-frequency-cutoff F do not filter below F Hz
--high-frequency-cutoff F upper frequency cutoff in Hz

--minimum-mass MASS  set minimum component mass of bank to MASS
--maximum-mass MASS  set maximum component mass of bank to MASS
--minimal-match M     generate bank with minimal match M

--order ORDER         set post-Newtonian order of the waveform to ORDER
                      (newtonian|oneHalfPN|onePN|onePointFivePN|
                      twoPN|twoPointFive|threePN|threePointFivePN)
--approximant APPROX set approximant of the waveform to APPROX
                      (TaylorT1|TaylorT2|TaylorT3|TaylorF1|TaylorF2|
                      PadeT1|PadeT2|EOB|BCV|SpinTaylorT3)
--space SPACE         grid up template bank with mass parameters SPACE
                      (Tau0Tau2|Tau0Tau3)

--write-raw-data      write raw data to a frame file
--write-response       write the computed response function to a frame
--write-spectrum       write the uncalibrated psd to a frame
--write-strain-spectrum write the calibrated strain psd to a text file

```

**Description**

`lalapps_tmpltbank` is a stand alone code for generating inspiral template banks for LIGO data with the LAL bank package. The code generates a calibrated power spectrum at the specified time for the requested channel and uses this to compute the template bank. See the LAL bank package documentation for detailed information on the algorithms used to generate the banks.

**Options****`--help`**

Display a brief usage summary.

**Example**

```
lalapps_tmpltbank \  
--gps-start-time 734357353 --gps-end-time 734358377 \  
--frame-cache cache/L-734357345-734361107.cache \  
--segment-length 1048576 --number-of-segments 7 \  
--pad-data 7 --sample-rate 4096 --resample-filter ldas \  
--enable-high-pass 5.000000e+01 --spectrum-type median \  
--low-frequency-cutoff 7.000000e+01 --high-frequency-cutoff 2.048000e+03 \  
--minimum-mass 1.000000e+00 --maximum-mass 3.000000e+00 \  
--minimal-match 9.700000e-01 --calibration-cache \  
/ldas_outgoing/calibration/cache_files/L1-CAL-V03-729273600-734367600.cache \  
--space Tau0Tau3 --approximant TaylorT1 --order twoPN \  
--channel-name L1:LSC-AS_Q --debug-level 33
```

**Author**

Duncan Brown

### 3.4.3 Program `lalapps_inspiral`

#### Name

`lalapps_inspiral` — stand alone inspiral search code

#### Synopsis

```

--help                display this message
--verbose             print progress information
--debug-level LEVEL  set the LAL debug level to LEVEL
--user-tag STRING    set the process_params usertag to STRING
--comment STRING     set the process table comment to STRING

--gps-start-time SEC  GPS second of data start time
--gps-start-time-ns NS GPS nanosecond of data start time
--gps-end-time SEC    GPS second of data end time
--gps-end-time-ns NS  GPS nanosecond of data end time
--pad-data T          pad the data start and end time by T seconds

--frame-cache         obtain frame data from LAL frame cache FILE
--calibration-cache FILE obtain calibration from LAL frame cache FILE
--channel-name CHAN   read data from interferometer channel CHAN

--injection-file FILE inject simulated inspiral signals from FILE

--bank-file FILE      read template bank parameters from FILE
--minimal-match M     override bank minimal match with M (sets delta)
--start-template N    start filtering at template number N in bank
--stop-template N     stop filtering at template number N in bank

--sample-rate F       filter data at F Hz, downsampling if necessary
--resample-filter TYPE set resample filter to TYPE (ldas|butterworth)

--disable-high-pass   turn off the IIR highpass filter
--enable-high-pass F  high pass data above F Hz using an IIR filter
--spectrum-type TYPE  use PSD estimator TYPE (mean|median)

--segment-length N    set data segment length to N points
--number-of-segments N set number of data segments to N
--segment-overlap N   overlap data segments by N points

--low-frequency-cutoff F do not filter below F Hz
--inverse-spec-length T set length of inverse spectrum to T seconds
--dynamic-range-exponent X set dynamic range scaling to 2^X

--chisq-bins P        set number of chisq veto bins to P
--snr-threshold RHO   set signal-to-noise threshold to RHO
--chisq-threshold X   threshold on  $\chi^2 < X * (p + \rho^2 * \delta^2)$ 
--enable-event-cluster turn on maximization over chirp length
--disable-event-cluster turn off maximization over chirp length

```

<code>--enable-output</code>	write the results to a LIGO LW XML file
<code>--disable-output</code>	do not write LIGO LW XML output file
<code>--write-raw-data</code>	write raw data to a frame file
<code>--write-filter-data</code>	write data that is passed to filter to a frame
<code>--write-response</code>	write the computed response function to a frame
<code>--write-spectrum</code>	write the uncalibrated psd to a frame
<code>--write-snr<sup>2</sup></code>	write the snr time series for each data segment
<code>--write-chisq</code>	write the $r^2$ time series for each data segment

### Description

`lalapps_inspiral` is a stand alone code for performing matched filtering of LIGO data for gravitational wave signals and Monte Carlo analysis.

### Options

#### `--help`

Display a brief usage summary.

### Example

```
lalapps_inspiral \  
--enable-output --inverse-spec-length 16 --segment-length 1048576 \  
--low-frequency-cutoff 7.000000e+01 --pad-data 8 \  
--bank-file L1-TMPLTBANK-734357353-1024.xml \  
--sample-rate 4096 --chisq-threshold 20.0 --resample-filter ldas \  
--channel-name L1:LSC-AS_Q --calibration-cache \  
/ldas_outgoing/calibration/cache_files/L1-CAL-V03-729273600-734367600.cache \  
--segment-overlap 524288 --snr-threshold 8.0 \  
--frame-cache cache/L-734357345-734361107.cache \  
--number-of-segments 7 --dynamic-range-exponent 6.900000e+01 \  
--enable-high-pass 5.000000e+01 --debug-level 33 \  
--gps-start-time 734357353 --gps-end-time 734358377 \  
--chisq-bins 8 --spectrum-type median --enable-event-cluster \  
--minimal-match 9.700000e-01
```

### Author

Duncan Brown

### 3.4.4 Program `lalapps_inca`

#### Name

`lalapps_inca` — program does inspiral coincidence analysis.

#### Synopsis

```
lalapps_inca [--help] [--verbose] [--comment COMMENT] [--debug-level LEVEL]
[--no-playground] [--playground-only] --ifo-a IFOA --ifo-b IFOB
[--epsilon  $\epsilon$ ] [--kappa  $\kappa$ ] [--dm  $\delta m$ ] [--dt  $\delta t$ ]
--gps-start-time SECONDS --gps-end-time SECONDS [--write-uniq-triggers]
(LIGO LIGHTWEIGHT XML FILES)
```

#### Description

`lalapps_inca` performs coincidence on triggers from the inspiral search code. At present it works for only two interferometers. The names of the two interferometers must be given. Output is written to a LIGO lightweight XML files. Two XML output files are written. The output files contain `process`, `process_params` and `search_summary` tables that describe the search. The primary ifo output file contains the triggers from IFOA that are found to be in coincidence with triggers in IFOB. The secondary output file contains the triggers from IFOB that are found to be in coincidence with the triggers from IFOA. Each trigger in the IFOA file corresponds to the coincident trigger in the IFOB file, so there may be duplicate IFOA triggers. To prevent this, specify the `--write-uniq-triggers` option.

The output files are named in the standard way for inspiral pipeline output. The primary triggers are in a file named

`IFOA-INCA_USERTAG-GPSSTARTTIME-DURATION.xml`

and the secondary triggers are in a file named

`IFOB-INCA_USERTAG-GPSSTARTTIME-DURATION.xml`

If a `--user-tag` is not specified on the command line, the `_USERTAG` part of the filename will be omitted.

The default behavior outputs triggers during playground times only. To obtain those triggers that are not in the playground, use the `--no-playground` flag.

`lalapps_inca` calls the LAL function `LALCompareSnglInspiral()` to test if two triggers are coincident. This first tests that the time of the triggers is coincidence to within  $\delta t$ . It then tests that both the template masses are coincident to within  $\delta m$ . It then tests to that

$$\frac{|D_{\text{IFOA}} - \hat{D}_{\text{IFOA}}|}{D_{\text{IFOA}}} < \frac{\epsilon}{\rho_{\text{IFOB}}} + \kappa. \quad (3.1)$$

This is equivalent to testing that

$$|\rho_{\text{IFOB}} - \hat{\rho}_{\text{IFOB}}| < \epsilon + \kappa \rho_{\text{IFOB}}. \quad (3.2)$$

If all three tests pass, the events are considered to be coincident and written to the output file.

#### Options

- no-playground**  
Optional. Record all triggers that are not in playground data. The default behaviour returns only those triggers which lie in the playground data set.
- playground-only**  
Optional. Record only triggers that occur in the playground times. This is the default behaviour.
- ifo-a IFOA**  
Required. This is the name of the interferometer to use as the interferometer A in the coincidence algorithm. It must be a two letter IFO code e.g. **L1**, **H1**, etc.
- ifo-b IFOB**  
Required. This is the name of the interferometer to use as the interferometer B in the coincidence algorithm. It must be a two letter IFO code e.g. **L1**, **H1**, etc.
- epsilon  $\epsilon$**   
Optional. Set the value of  $\epsilon$  in the effective distance test. If not given the default of  $\epsilon = 2$  will be used.
- kappa  $\kappa$**   
Optional. Set the value of  $\kappa$  in the effective distance test. If not given the default of  $\kappa = 0.01$  will be used.
- dm  $\delta m$**   
Optional. Accept triggers as coincident if both mass parameters agree within  $\delta m$ . If not supplied, then  $\delta m = 0$ .
- dt  $\delta t$**   
Optional. Accept triggers as coincident if their end times agree within  $\delta t$  milliseconds. If not supplied, then  $\delta t = 0$ .
- gps-start-time GPS SECONDS**  
Required. Look for coincident triggers with end times after GPS SECONDS.
- gps-end-time GPS SECONDS**  
Required. Look for coincident triggers with end times before GPS SECONDS.
- write-uniq-triggers**  
Optional. The default behaviour is to only write all triggers from IFO A. However, a trigger from IFO A may match two or more triggers from IFO B, so it may be duplicated in the output. Specifying this option causes only unique IFO A triggers to be written.
- comment STRING**  
Optional. Add STRING to the comment field in the process table. If not specified, no comment is added.
- user-tag STRING**  
Optional. Set the user tag for this job to be STRING. May also be specified on the command line as `-userTag` for LIGO database compatibility.
- help**  
Optional. Print a help message.
- debug-level LEVEL**  
Optional. Set the LAL debug level to LEVEL. If not specified the default is 1.

## Arguments

### [LIGO Lightweight XML files]

The arguments to the program should be a list of LIGO Lightweight XML files containing the triggers from the two interferometers. The input files can be in any order and do not need to be time ordered as `inca` will sort all the triggers once they are read in. If the program encounters a LIGO Lightweight XML containing triggers from an unknown interferometer (i.e. not IFO A or IFO B) it will exit with an error.

## Example

```
lalapps_inca \  
--playground-only --gps-start-time 734357353 --drhominus 5.0 \  
--dm 0.03 --gps-end-time 734358377 --ifo-b H1 --dt 20.0 \  
--ifo-a L1 --drhoplus 5.0 --debug-level 33
```

## Algorithm

The code maintains two pointers to triggers from each ifo, `currentTrigger[0]` and `currentTrigger[1]`, corresponding to the current trigger from IFO A and B respectively.

1. An empty linked list of triggers from each interferometer is created. Each input file is read in and the code determines which IFO the triggers in the file correspond to. The triggers are appended to the linked list for the corresponding interferometer.
2. If there are no triggers read in from either of the interferometers, the code exits cleanly.
3. The triggers for each interferometer is sorted by the `end_time` of the trigger.
4. `currentTrigger[0]` is set to point to the first trigger from IFO A that is after the specified GPS start time for coincidence. If no trigger is found after the start time, the code exits cleanly.
5. Loop over each trigger from IFO A that occurs before the specified GPS end time for coincidence:
  - (a) `currentTrigger[1]` is set to point to the first trigger from IFO B that is within the time coincidence window,  $\delta t$ , of `currentTrigger[0]`. If no IFO B trigger exists within this window, `currentTrigger[0]` is incremented to the next trigger from IFO A and the loop over IFO A triggers restarts.
  - (b) If the trigger `currentTrigger[0]` is, is not in the playground data, start looping over triggers from IFO B.
    - i. For each trigger from IFO B that is within  $\delta t$  of `currentTrigger[0]`
    - ii. Call `LALCompareSnglInspiral()` to check if the triggers match as determined by the options on the command line. If the trigger match, record them for later output as coincident triggers.
  - (c) Increment `currentTrigger[0]` and continue loop over triggers from IFO A.

## Author

Patrick Brady and Duncan Brown

### 3.4.5 Program `lalapps_inspinj`

#### Name

`lalapps_inspinj` — produces inspiral injection data files.

#### Synopsis

```
lalapps_inspinj [--help] --source-file SFILE --mass-file MFILE --gps-start-time TSTART
--gps-end-time TEND [--time-step TSTEP] [--seed SEED] [--waveform WAVE] [--usertag
TAG] [--ilwd]
```

#### Description

`lalapps_inspinj` generates a number of inspiral parameters suitable for using in a Monte Carlo injection to test the efficiency of a inspiral search. The various parameters (detailed below) are randomly chosen and are appropriate for a particular population of binary neutron stars whose spatial distribution includes the Milky Way and a number of extragalactic objects that are input in a datafile. The possible mass pairs for the binary neutron star com- panions are also specified in a (different) datafile.

The output of this program is a list of the injected events, starting at the specified start time, ending at the specified end time, and containing one set of random inspiral parameters every specified time step. The output is written to a file name in the standard inspiral pipeline format:

```
HL-INJECTIONS_USERTAG_SEED-GPSSTART-DURATION.xml
```

where `USERTAG` is `TAG` as specified on the command line, `SEED` is the value of the random number seed chosen and `GPSSTART` and `DURATION` describes the GPS time interval that the file covers. The file is in the standard LIGO lightweight XML format containing a `sim_inspiral` table that describes the injections. In addition, an ascii log file called `injlog.txt` is also written. If a `--user-tag` is not specified on the command line, the `_USERTAG` part of the filename will be omitted.

#### Options

##### `--help`

Print a help message.

##### `--source-file SFILE`

Optional. Data file containing spatial distribution of extragalactic objects. Default is the file `inspsrcs.dat` provided by LALApps.

##### `--mass-file MFILE`

Optional. Data file containing mass pairs for the binary neutron star companions. Default is the file `BNSMasses.dat` provided by LALApps.

##### `--gps-start-time TSTART`

Optional. Start time of the injection data to be created. Defaults to the start of S2, Feb 14 2003 16:00:00 UTC (GPS time 729273613)

##### `--gps-end-time TEND`

Optional. End time of the injection data to be created. Defaults to the end of S2, Apr 14 2003 15:00:00 UTC (GPS time 734367613).

##### `--time-step TSTEP`

Optional. Sets the time step interval between injections. The injections will occur at  $TSTEP/\pi$  second intervals. Defaults to  $2630/\pi$ .

**--seed SEED**

Optional. Seed the random number generator with the integer SEED. Defaults to 1.

**--waveform WAVE**

Optional. The string WAVE will be written into the `waveform` column of the `sim.inspiral` table output. This is used by the inspiral code to determine which type of waveforms it should inject into the data. Defaults is `GeneratePPNtwoPN`.

**--user-tag STRING**

Optional. Set the user tag for this job to be STRING. May also be specified on the command line as `-userTag` for LIGO database compatibility.

**--ilwd**

Optional. If this option is given, `lalapps_inspinj` also produces two ILWD-format files, `injepochs.ilwd` and `injparams.ilwd`, that contain, respectively, the GPS times suitable for inspiral injections, and the intrinsic inspiral signal parameters to be used for those injections.

The file `injepochs.ilwd` contains a sequence of integer pairs representing the injection GPS time in seconds and residual nano-seconds. The file `injparams.ilwd` contains the intrinsic binary parameters for each injection, which is a sequence of eight real numbers representing (in order) (1) the total mass of the binary system (in solar masses), (2) the dimensionless reduced mass — reduced mass per unit total mass — in the range from 0 (extreme mass ratio) to 0.25 (equal masses), (3) the distance to the system in meters, (4) the inclination of the binary system orbit to the plane of the sky in radians, (5) the coalescence phase in radians, (6) the longitude to the direction of the source in radians, (7) the latitude to the direction of the source in radians, (8) and the polarization angle of the source in radians.

**Example**

```
lalapps_inspinj --seed 45\  
--source-file inspsrcs.dat --mass-file BNSMasses.dat
```

**Environment****LALAPPS\_DATA\_PATH**

Directory to look for the default mass file `BNSMasses.dat` and the default source file `inspsrcs.dat`.

**Author**

Jolien Creighton, Patrick Brady, Duncan Brown

### 3.4.6 Program `lalapps_snglInspiralReader`

#### Name

`lalapps_snglInspiralReader` — manipulates LIGO lightweight XML files of inspiral triggers allowing cuts and clustering.

#### Synopsis

```
lalapps_snglInspiralReader --input INFILE --table TABLENAME
--outfile OUTFILE [--snrstar SNRSTAR] [--noplayground] [--sort] [--cluster MSEC]
[--clusteralgorithm CLUSTERCHOICE] [--help]
```

#### Description

`lalapps_snglInspiralReader` processes triggers from the inspiral search code. The `INFILE` should contain a list of the XML files containing the triggers; the format is one filename per line. The default behavior outputs triggers during playground times to the file `OUTFILE`; to obtain all triggers, use the `--noplayground` flag. To apply a cut on SNR, use the flag `--snrstar SNRSTAR`: only triggers with `SNR > SNRSTAR` will be recorded. Events can also be clustered within `MSEC` msec, in which case the `--sort` flag is recommended unless you are certain that the triggers are time-ordered. There is a choice of several clustering algorithms, which can be selected using `--clusteralgorithm`.

#### Options

##### `--input` INFILE

Required. A file containing a list of LIGO lightweight XML files with triggers to be processed. The format of `INFILE` is one file name per line.

##### `--table` TABLENAME

Required. The name of the XML table containing the inspiral events, this will usually be `sngl_inspiral`.

##### `--outfile` OUTFILE

Required. Name of the file to be used for output. The output format is LIGO lightweight XML with `sngl_inspiral`, `process` and `process_params` tables.

##### `--snrstar` SNRSTAR

Optional. A threshold cut on signal-to-noise. Only triggers with `SNR > SNRSTAR` are recorded in the output file.

##### `--noplayground`

Optional. Record all triggers. The default behaviour returns only those triggers which lie in the playground data set.

##### `--sort`

Optional. Sort the triggers in time (before clustering).

##### `--cluster` MSEC

Optional. Cluster triggers within `MSEC` msec window. The clustering algorithm identifies the first trigger in a cluster, then displaces it if another trigger within the clustering window satisfies the appropriate condition (described below).

##### `--clusteralgorithm` CHOICENUMBER

Optional. This determines which condition will be used in clustering of the triggers. The current choices are `snr_and_chisq` — displace event if its SNR is exceeded by an event with a smaller CHISQ; `snrsq_over_chisq` — displace event if the quantity  $(\text{SNR})^2/\text{CHISQ}$  is exceeded by a subsequent event's. The default is `snr_and_chisq`.

**--help**

Optional. Print a help message.

**Example**

```
lalapps_snglInspiralReader --input xmlfilelist \  
--table sngl_inspiral --outfile my.xml --snrstar 8.0 \  
--sort --cluster 20 --clusteralgorithm snrsq_over_chisq
```

**Author**

Patrick Brady

### 3.4.7 Program `lalapps_inspinj_find`

#### Name

`lalapps_inspinj_find` — compares LIGO lightweight XML files containing inspiral triggers with an XML file containing injected signals and tests for time coincidence.

#### Synopsis

```
lalapps_inspinj_find --input INFILE [--inject INJECTFILE] --outfile OUTFILE
[--snrstar SNRSTAR] [--sort] [--noplayground] [--deltat DT]
[--cluster CLUST] [--clusteralgorithm CLUSTERCHOICE] [--missedinjections MISSEDFILE]
[--hardware STARTTIME] [--help]
```

#### Description

`lalapps_inspinj_find` compares triggers from the inspiral search code with injections. The `INFILE` should contain a list of the XML files containing the triggers; the format is one filename per line. The triggers can be sorted, using the `--sort` flag. To apply a cut on SNR, use the flag `--snrstar SNRSTAR`: only triggers with `SNR > SNRSTAR` will be recorded. If the file `INJECTFILE` containing injections is provided then the program will compare the triggers with the injections, in which case the `--sort` flag is recommended unless you are certain that the triggers are time-ordered. Any trigger occurring within `DT` msec of an injection is retained. Additionally, any injection which is coincident with one or more triggers is also retained. The tables of triggers and injections are output to the file `OUTFILE`. Events can also be clustered within `CLUST` msec, after coincidence has been checked. If `CLUST` is twice `DT` then only one trigger per inspiral will survive. There is a choice of several clustering algorithms, which can be selected using `--clusteralgorithm`. Finally, specifying `--missedinjections`, creates a file `MISSEDFILE` containing a table of those injections which occurred during the times of the input files, and in the playground, which were not coincident with any triggers.

#### Options

##### `--input INFILE`

Required. A file containing a list of LIGO lightweight XML files with triggers to be processed. The format of `INFILE` is one file name per line.

##### `--inject INJECTFILE`

Optional. A file containing a list of inspiral events which were injected into the data. The `INJECTFILE` format is LIGO lightweight XML with a `sim_inspiral` table. If this file is not specified, the program runs in the same way as `lalapps_snglInspiralReader`.

##### `--outfile OUTFILE`

Required. Name of the file to be used for output. The output format is LIGO lightweight XML with `sngl_inspiral`, `sim_inspiral`, `process` and `process_params` tables.

##### `--snrstar SNRSTAR`

Optional. A threshold cut on signal-to-noise. Only triggers with `SNR > SNRSTAR` are checked for coincidence with the injections.

##### `--sort`

Optional. Sort the triggers in time (before checking coincidence).

##### `--noplayground`

Optional. Record all triggers. The default behaviour returns only those triggers which lie in the playground data set.

**--deltat** DT

Optional. This gives the maximum time difference DT allowed between the injection and trigger. If not specified, the default is 20msec. The time difference is calculated between the trigger time and the end time for the injection at the detector.

**--cluster** CLUST

Optional. Cluster triggers within CLUST msec window. The clustering algorithm identifies the first trigger in a cluster, then displaces it if another trigger within the clustering window satisfies the appropriate condition (described below).

**--clusteralgorithm** CHOICENUMBER

Optional. This determines which condition will be used in clustering of the triggers. The current choices are [snr\\_and\\_chisq](#) — displace event if its SNR is exceeded by an event with a smaller CHISQ; [snrsq\\_over\\_chisq](#) — displace event if the quantity  $(\text{SNR})^2/\text{CHISQ}$  is exceeded by a subsequent event's. The default is [snr\\_and\\_chisq](#).

**--missedinjections** MISSEDFILE

Optional. Output a [sim\\_inspirals](#) table in MISSEDFILE containing all injections which were not coincident with a trigger. Only injections occurring during during the times of the input files and within the playground (unless [--noplayground](#) is specified) are given.

**--hardware** STARTTIME

Optional. The STARTTIME should be the gps start time of the hardware injections in seconds. In this case, the INJECTFILE is expected to contain a list of the hardware injections, where the time given is the time after the start of injections that the coalescence occurs.

**--help**

Optional. Print a help message.

**Example**

```
lalapps_inspinj_find --input xmlfilelist \  
--inject injections.xml --outfile my.xml --snrstar 8.0 \  
--coincidence 20 --sort --cluster 20 --clusteralgorithm snrsq_over_chisq \  
--missedinjections missedinj.xml
```

**Author**

Steve Fairhurst

### 3.4.8 Program `lalapps_inspmultiawg`

#### Name

`lalapps_inspmultiawg` — injects specified inspiral chirps into zero data. Intended for producing the hardware injection data.

#### Synopsis

```

--help                display this message
--source SFILE        source file containing details of injection
--response RESPFILE   file containing the response function
--summary SUMFILE     write found injection to file
--ifo IFO             name of interferometer (optional)
--flow FSTART         start frequency of injection (default 40 Hz)
--fhigh FSTOP         end frequency of injection (default: end at ISCO)
--length LENGTH       length of the data (default 64 seconds)
--samplerate FREQ     rate at which data is sampled (default (16384Hz))
--debug-level DEBUG   give the lal debug level

```

#### Description

`lalapps_inspmultiawg` injects inspiral chirps into zero data. The details of several chirps can be specified using the command `--source`, otherwise, a single inspiral of two  $1.4 M_{\odot}$  solar mass neutron stars will be injected. Each chirp is injected into a new file containing zero data of `LENGTH` seconds, sampled at `FREQ` Hz, and each injection begins at the beginning of the data. The response function can be provided in `RESPFILE`. The chirp is output to a file named

`IFO_inspiral_NUMBER.txt`

where `IFO` is the name of the interferometer, and `NUMBER` is the injection number. A summary of the injections performed can be saved in `SUMFILE`.

#### Options

##### `--sourcefile SFILE`

Optional. Reads source information from the file `SFILE`. If absent, it injects a single  $1.4 M_{\odot}$ – $1.4 M_{\odot}$  inspiral, optimally oriented, at a distance of  $1.0 Mpc$ .

##### `--response RESPFILE`

Optional. Reads a detector response function from the file `RESPFILE`. If absent, it generates raw dimensionless strain.

##### `--summary SUMFILE`

Optional. The `SUMFILE` format is LIGO lightweight XML with `process`, `process_params` and `sim_inspiral` tables. The `sim_inspiral` table contains details of all the injections performed. The details of the injection are obtained from the source file.

##### `--length SEC`

Optional. Specify the length of data into which the signal will be injected. The default is 64 seconds.

##### `--samplerate FREQ`

Optional. Specify the rate at which the data is sampled. The default is 16384 Hz.

**--ifo IFO**

Optional. Give the name of the interferometer for which the injections are intended. This is only used in naming the output files.

**--flow FSTART**

Optional. Give the start frequency FSTART for the inspiral. The default is 40 Hz

**--fhigh SFTOP**

Optional. Give the end frequency SFTOP for the inspiral. The default behaviour is that the inspiral will continue to ISCO. If set to a negative number, the generator will use its absolute value as the terminating frequency, but will ignore post-Newtonian breakdown.

**--debug-level DEBUG**

Optional. Set the LAL debug level. The default is 33.

**--help**

Optional. Print a help message.

**Format for `sourcefile`:** The source file consists of any number of lines of data, each specifying a chirp waveform. Each line must begin with a character code (`CHAR` equal to one of `'i'`, `'f'`, or `'c'`), followed by 6 whitespace-delimited numerical fields: the epoch of the chirp (`INT8` seconds), the two binary masses (`REAL4`  $M_{\odot}$ ), the distance to the source (`REAL4` Mpc), and the source's inclination and phase at coalescence (`REAL4` degrees). The character codes have the following meanings:

- `'i'` The epoch represents the GPS time of the start of the chirp waveform.
- `'f'` The epoch represents the GPS time of the end of the chirp waveform.
- `'c'` The epoch represents the GPS time when the binaries would coalesce in the point-mass approximation.

Since the injection is started at time  $t = 0$ , it is recommended that the `'i'` option is used.

Thus a typical input line for two  $1.4M_{\odot}$  objects at 1.1 Mpc inclined  $30^{\circ}$  with an initial phase of  $45^{\circ}$ , beginning at 70 seconds (after the start of the injections), will have the following line in the input file:

```
i 70 1.4 1.4 1.1 30.0 45.0
```

The time parameter (in this case 70 sec) does not affect the output data in any way. It is simply stored in the `sim_inspiral` table of the SUMFILE, in order to make analysis of the injections easier.

**Format for `respfile`:** The response function  $R(f)$  gives the real and imaginary components of the transformation from ADC output  $o$  to tidal strain  $h$  via  $\tilde{h}(f) = R(f)\tilde{o}(f)$ . It is inverted internally to give the detector transfer function  $T(f) = 1/R(f)$ . The format `respfile` is a header specifying the GPS epoch  $t_0$  at which the response was taken (`INT8` nanoseconds), the lowest frequency  $f_0$  at which the response is given (`REAL8` Hz), and the frequency sampling interval  $\Delta f$  (`REAL8` Hz):

```
# epoch = t0
# f0 = f0
# deltaF = Δf
```

followed by two columns of `REAL4` data giving the real and imaginary components of  $R(f_0 + k\Delta f)$ .

**Format for the data output:** The data output in the files `IFO_inspiral_NUMBER.txt` is a single column of `REAL4` ADC data.

**Example**

```
lalapps_inspmultiawg --source s3.sources \  
--response response_l1.txt --summary summ.xml --ifo l1
```

**Author**

Steve Fairhurst

## 3.5 Power Tools

This section of LALAPPS contains programs that can be used to perform burst searches using the excess power algorithm.

Put a summary of the excess power algorithm here .....

### 3.5.1 Program `lalapps_power`

#### Name

`lalapps_power` — runs excess power code on chunks of real or simulated data.

#### Synopsis

```
lalapps_power --npts NPTS --nseg NSEG --olap OLAP --olapfctr OLAPFCTR \
  --minfbin MINFBIN --mintbin MINTBIN --flow FLOW --delf DELF --length LNGTH \
  --nsigma NSIGMA --alphdef ALPHDEF --segdcle SEGDCLE --threshold THRESHOLD \
  --etomstr ETOMSTR --channel CHANNEL --simtype SIMTYPE --spectype SPECTYPE \
  --window WINDOW --start_time SEC --start_time_ns NSEC --numpts NUMPTS \
  --srate SRATE [--printSpectrum] [--cluster] [--noise VAR] [--seed SEED] \
  [--calcache CALCACHE] [--injfile INJFILE] [--comment COMMENT] [--dbglevel DBGLEVEL]
```

#### Description

`lal_power` runs the excess power code from LAL on a chunk of real or simulated data. Consider searching for signals with the following properties:

- Maximum signal time duration  $T = 2^a$  seconds where  $a$  is a positive or negative integer; the sampling rate of the data stream is taken assumed  $srate = 2^b$  Hz.
- The frequency band of the signal is between  $f_{low}$  Hz and  $f_{high}$  Hz. Current versions of the code expect  $f_{high} - f_{low} = 2^d$  Hz where  $d$  is an integer.
- Minimum time duration,
- Minimum frequency bandwidth.

The input data for a real search should be in frame format. This data is located using the LAL frame cache file mechanism. The code can be used for Monte-Carlo simulations to determine search efficiency by providing a list of injections to be made; this injections list should be in LIGO lightweight format and can be generated using the `lalapps_binj` program described in Sec. 3.5.4.

The output data is written as `sngl_burst` triggers in LIGO lightweight XML files. The files are named according to a standardized naming convention

```
{IFO}-{comment}-POWER-{GPS Start Time}-{duration}.xml
```

So, if a search was run on the Hanford 4km interferometer and generated triggers starting at 731488397 and the triggers cover 33 seconds after that time, then the file name would be

```
H1-test_this_again-POWER-731488397-33.xml
```

where the comment was `test_this_again`. Note that the comment should not include spaces and should use underscores instead.

#### Options

`--npts` NPTS

Number of data points in a segment is determined by

$$NPTS = (T \times SRATE) .$$

`--nseg` NSEG

Number of overlapping segments into which data should be divided for filtering; must be an integer.

**--olap OLAP**

Number of points overlap between segments. This is an argument for completeness, but in general it should be  $NPTS/2$ .

**--olapfctr OLAPFCTR**

Amount of overlap between neighboring TF tiles; must be an integer. A reasonable value for this parameter is 3. See LAL [burstsearch](#) package for details.

**--minfbin MINFBIN**

Smallest extent in frequency of TF tiles to search; must be an integer. A reasonable value for this parameter is 2. The product  $MINFBIN \times MINTBIN$  is the minimum time-frequency volume to be searched. See LAL [burstsearch](#) package for details.

**--mintbin MINTBIN**

Smallest extent in time of TF tiles to search; must be an integer. A reasonable value for this parameter is 2. The product  $MINFBIN \times MINTBIN$  is the minimum time-frequency volume to be searched. See LAL [burstsearch](#) package for details.

**--flow FLOW**

Lowest frequency in Hz to be searched; a real number. This is obviously  $f_{low}$  Hz from our description of the desired signal parameters above.

**--delf DELF**

This input should be set to  $1/T$  Hz but it is ignored by the current version of the code; a real number.

**--length LNGTH**

may be determined by the following formula

$$LNGTH = T \times (f_{high} - f_{low}).$$

This is an integer which determines the maximum frequency bandwidth over which a signal is expected.

**--nsigma NSIGMA**

threshold number of sigma; a real number. Currently see LAL [burstsearch](#) package for details.

**--alphdef -ALPHDEF**

default alpha value for tiles with  $\sigma \geq \text{numSigmaMin}$ ; a real number. Currently see LAL [burstsearch](#) package for details.

**--segdcle SEGDCLE**

Number of segments analyzed at a time; must be an integer. The current code uses SEGDCLE overlapping segments to compute the average (or median) power spectral estimate for use inside the code.

**--threshold THRESHOLD**

Identify events with alpha less than this; a real number. Currently see LAL [burstsearch](#) package for details.

**--etomstr ETOMSTR**

Number of events to be accepted from a search over NPTS of data; must be an integer.

**--channel CHANNEL**

The name used to identify the data to be analyzed; a character string matching the channel name in the frame files, e.g. [H2:LSC-AS-Q](#).

**--start\_time SEC**

The GPS time corresponding to the start of the time series read from frames. Note: OLAP points are discarded at the beginning and end of the data to avoid data corrupted by the low-pass filtering.

**--start\_time\_ns NSEC**

The number of nanoseconds after SEC for the time series read from frames.

**--framecache FRCACHE**

A LAL format frame cache file. The FRCACHE can be a filename in the local directory, or a filename including absolute path. These cache files are explained in the *framedata* package in LAL and can be constructed by making calls to *LALdataFind* on some systems.

**--calcache CALCACHE**

A LAL format frame cache file. The CALCACHE can be a filename in the local directory, or a filename including absolute path. These cache files are explained in the *framedata* package in LAL. A calibration cache file should give information for the frames needed to construct the relevant calibration information from the reference calibrations and the  $\alpha$  and  $\beta$  coefficients.

**--injfile INJFILE**

A LIGO lightweight XML file containing a list of injections to be made. The file should contain a *sim\_burst* table which is used to determine information about the types of injections to be made. This file may be constructed by hand, or once can use the *lalapps\_binj* program described in Sec. 3.5.4.

**--simtype SIMTYPE**

Type of simulation. Set it to 0, although it is ignored in the current version of the code.

**--spectype SPECTYPE**

Spectrum estimator for whitening data; a character string [*useMean*, *useMedian*].

**--window WINDOW**

Type of window to use on the data; must be an integer. [Possible values are: 0=Rectangular, 1=Hann, 2=Welch, 3=Bartlett, 4=Parzen, 5=Papoulis, 6=Hamming.]

**--numpts NUMPTS**

The number of points of data to be extracted from the frame files may be determined by the following formula

$$\text{NUMPTS} = \text{NSEG} \times (\text{NPTS} - \text{OLAP}) + 3 \times \text{OLAP} .$$

Notice that this appears to be  $2 \times \text{OLAP}$  more points than you might expect. Since the data stream is high-pass filtered internally, the code ignores the first OLAP points and requires that there be an extra OLAP points at the end to avoid data corruption.

**--cluster**

Apply a clustering algorithm to tiles identified by the excess power code. The result is reduction of overlapping triggers to a single trigger which covers a square time-frequency volume which encompasses all overlapping trigger regions. The signal-to-noise and the confidence associated with a clustered trigger belong to the most significant excess-power trigger in the cluster. [This is a prototype option. Its behavior is not robustly tested. Treat with care and look at the code to insure understanding.]

**--comment COMMENT**

A user defined comment string. It should be less than 256 characters and should not contain spaces (replace spaces by underscores). This string will appear in the name of the file to which output information is written.

**--printSpectrum**

Print out the power spectrum to a file. Note: this needs to be enhanced. AT present it give some help with debugging and understanding the data.

**--printData**

Print out the time series that is read in by the code.

**--verbose**

Print out informational messages as the code runs.

**--dbglevel**

Set the lalDebugLevel. The default is `LALMSGVL2`. A useful setting is 65 which turns off memory padding, but keeps memory tracking and error messages. If you want to turn off memory tracking completely, then use 33.

**Example**

To run the program, type:

```
lalapps_power --cluster --npts 4096 --nseg 64 --olap 2048 --olapfctr 3 \  
--minfbin 2 --mintbin 2 --flow 60 --delf 1 --length 1024 --srate 4096 \  
--nsigma 2 --alphdef 0.5 --segdcle 32 --threshold 1.0e-05 --etomstr 100 \  
--channel L1:LSC-AS_Q --simtype 0 --spectype useMedian --window 2 \  
--start_time 731488397 --start_time_ns 0 --numpts 137216 --noiseamp 1 \  
--seed 8 --comment test_this_again --dbglevel 33 --verbose
```

**Author**

Patrick Brady

### 3.5.2 Program `lalapps_snglBurstHistogram`

#### Name

`lalapps_snglBurstHistogram` — runs over LIGO lightweight files and histograms the triggers by frequency.

#### Synopsis

```
lalapps_snglBurstHistogram --input INFILE [--threshold THRESHOLD] \  
  --freq FSTART FSTOP DF [--tfhist OUTFILE] [--help]
```

#### Description

`lal_snglBurstHistogram` reads in LIGO lightweight files and constructs a histogram of the triggers over frequency. This histogram data is then printed in the file `freq-hist.txt`. It also produces a list of triggers for each playground segment separately and that is printed in the `OUTFILE`, if provided. The code should be extensible to do a number of similar things like this.

#### Options

**`--input` INFILE**

The name of a file containing a list of XML files to parse; one XML file per line of `INFILE`.

**`--table` TABLENAME**

The name of the table to be parsed from the XML file. This will be needed when the process table is written into the XML files by the power code.

**`--threshold` THRESHOLD**

Identify events with alpha less than this; a real number. The meaning of the threshold is the same as `lalapps_power` itself. (Optional)

**`--freq` FSTART FSTOP DF**

Parameters which define a frequency histogram. `FSTART` is the lowest frequency; `FSTOP` is the highest frequency; `DF` is the width of abin.

**`--tfhist` OUTFILE**

The name of a file which will contain the list of triggers per playground segment for each frequency bin.

**`--help`**

Print usage instructions.

#### Example

To run the program, type:

```
lalapps_snglBurstHistogram --input files.txt --freq 100.0 1000.0 25.0 --tfhist tf.txt
```

This will read in the xml files listed in `files` (see below) and construct a frequency histogram with lowest frequency 100 Hz, highest frequency 1000 Hz and bin width 25 Hz. This output will be in `freq-hist.txt`. It will also produce the triggers for each playground segment and those will be in `tf.txt`. The file `files.txt` must contain a list of XML files. Here are the first few lines from such a file:

```
simtest-693768279-11.xml  
simtest-693768311-21.xml  
simtest-693768343-31.xml  
simtest-693768375-41.xml
```

**Author**

Patrick Brady

### 3.5.3 Program `lalapps_burca`

#### Name

`lalapps_burca` — program does burst coincidence analysis.

#### Synopsis

```
lalapps_burca --ifo-a TRIGFILE.A --ifo-b TRIGFILE.B [--start-time STARTCOINCIDENCE] \
  [--stop-time ENDCOINCIDENCE] [--drhoplus DRHOPLUS] [--drhominus DRHOMINUS] \
  [--dt DELTAT] --outfile OUTFILE [--noplayground] [--help]
```

#### Description

`lalapps_burca` performs coincidence on triggers from the burst search code. (At present it works for only two interferometers.) It must be called with at least one input file from each instrument. The default behavior outputs triggers during playground times to the file `OUTFILE`; to obtain all triggers, use the `--noplayground` flag.

#### Options

##### `--ifo-a` TRIGFILE.A

Required. LIGO lightweight XML file with triggers from interferometer A. This argument can be called multiple times. Triggers are sorted *after* all files have been read in.

##### `--ifo-b` TRIGFILE.B

Required. LIGO lightweight XML file with triggers from interferometer B. This argument can be called multiple times. Triggers are sorted *after* all files have been read in.

##### `--start-time` STARTCOINCIDENCE

Optional. Look for coincident triggers with start times after `STARTCOINCIDENCE`. If not supplied, the `STARTCOINCIDENCE = 0`.

##### `--stop-time` ENDCOINCIDENCE

Optional. Look for coincident triggers with start times before `ENDCOINCIDENCE`. If not supplied, then `ENDCOINCIDENCE = 977788813`, i.e. 00:00 Dec 31, 2010 UTC.

##### `--drhoplus` DRHOPLUS

Optional. **Not yet implemented.**

##### `--drhominus` DRHOMINUS

Optional. **Not yet implemented.**

##### `--dt` DELTAT

Optional. Accept triggers as coincident if their start times agree within `DELTAT` in msec. If not supplied, then `DELTAT = 0`.

##### `--outfile` OUTFILE

Required. Name of the file to be used for output. The output format is LIGO lightweight XML with only a `sngl_burst` table.

##### `--noplayground`

Optional. Record all triggers. The default behaviour returns only those triggers which lie in the playground data set.

##### `--cluster` MSEC

Optional. **Not yet implemented.** Cluster triggers within `MSEC` msec window.

**--help**

Optional. Print a help message.

**Example**

```
lalapps_burca --ifo-a L-POWER-734357353-1024.xml \  
--ifo-b H-POWER-734357353-1024.xml --dt 10 --outfile my.xml \  
--start-time 734357353 --stop-time 734358353 --noplayground
```

**Author**

Patrick Brady and Saikat Ray-Majumder

### 3.5.4 Program `lalapps_binj`

#### Name

`lalapps_binj` — produces burst injection data files.

#### Synopsis

```
lalapps_binj [--help] --gps-start-time TSTART --gps-end-time TEND [--time-step TSTEP]
[--seed SEED] [--waveform WAVE] [--coordinates COORDINATES] [--freq FREQ] [--flow FLOW]
[--fhigh FHIGH] [--deltaf DELTAF] [--quality QUALITY] [--tau TAU] [--hpeak HPEAK] [--usertag
TAG]
```

#### Description

**Warning:** this is very beta code that we are just starting to test. This documentation could be (is likely) out of date.

`lalapps_binj` generates a number of burst parameters suitable for using in a Monte Carlo injection to test the efficiency of a burst search. The various parameters (detailed below) are specified on the command line or can be randomly chosen in a manner appropriate for an burst upper limit search.

The output of this program is a list of the injected events, starting at the specified start time, ending at the specified end time, and containing one set of burst parameters every specified time step. The output is written to a file name in the standard burst pipeline format:

```
HL-INJECTIONS_USERTAG_SEED-GPSSTART-DURATION.xml
```

where `USERTAG` is `TAG` as specified on the command line, `SEED` is the value of the random number seed chosen and `GPSSTART` and `DURATION` describes the GPS time interval that the file covers. The file is in the standard LIGO lightweight XML format containing a `sim_burst` table that describes the injections. If a `--user-tag` is not specified on the command line, the `USERTAG` part of the filename will be omitted.

#### Options

##### `--help`

Print a help message.

##### `--gps-start-time TSTART`

Optional. Start time of the injection data to be created. Defaults to the start of S2, Feb 14 2003 16:00:00 UTC (GPS time 729273613)

##### `--gps-end-time TEND`

Optional. End time of the injection data to be created. Defaults to the end of S2, Apr 14 2003 15:00:00 UTC (GPS time 734367613).

##### `--time-step TSTEP`

Optional. Sets the time step interval between injections. The injections will occur at  $TSTEP/\pi$  second intervals. Defaults to  $2630/\pi$ .

##### `--seed SEED`

Optional. Seed the random number generator with the integer `SEED`. Defaults to 1.

##### `--coordinates COORDINATES`

Optional. The coordinate system to specify for the injections. The default is `EQUATORIAL`, but another useful one is `HORIZON` to easily allow for injection from directly overhead.

**--flow FLOW**

Optional. The code can generate injections at multiple frequencies. This option sets the first frequency used in that case. Default value is 150 Hz.

**--fhigh FHIGH**

Optional. Only generate injections with frequencies below FHIGH. Default value is 1000 Hz.

**--deltaf DELTAF**

Optional. The linear spacing between frequencies used to make injections. Default value is 0 Hz.

**--waveform WAVE**

Optional. The string WAVE will be written into the `waveform` column of the `sim_burst` table output. This is used by the burst code to determine which type of waveforms it should inject into the data. Default is `SineGaussian`.

**--tau TAU**

Optional. The decay-time for sine-gaussian, gaussian, ringdown and ring-up waveforms.

**--quality QUALITY**

Optional. The quality factor for sine-gaussian, gaussian, ringdown and ring-up waveforms. This option overrides the decay-time TAU and recalculates the duration for each waveform using the formula

$$\tau = \frac{\text{QUALITY}}{\sqrt{2\pi}f_0}$$

where  $f_0$  is the frequency of the injection.

**--freq FREQ**

Optional. The central frequency for sine-gaussian, ringdown and ring-up waveforms.

**--hpeak HPEAK**

Optional. The peak dimensionless strain for sine-gaussian, gaussian, ringdown and ring-up waveforms.

**--user-tag STRING**

Optional. Set the user tag for this job to be STRING. May also be specified on the command line as `-userTag` for LIGO database compatibility.

**Example**

```
lalapps_binj --seed 45\  
--freq 150 --tau 0.1 --hpeak 1e-20 --user-tag SGBURST
```

**Author**

Jolien Creighton, Patrick Brady, Duncan Brown

## 3.6 Programs Related to Stochastic Background Searches

### 3.6.1 Program `lalapps_olapredfcn`

#### Name

`lal_olapredfcn` — computes overlap reduction function given a pair of known detectors.

#### Synopsis

```
lal_olapredfcn [-h] [-q] [-v] [-d debugLevel ] \
  -s siteID1 [-a azimuth1] -t siteID2 [-b azimuth2] \
  [-f fLow] -e deltaF -n numPoints -o outfile
```

#### Description

`lal_olapredfcn` computes the overlap reduction function  $\gamma(f)$  for a pair of known gravitational wave detectors. It uses the LAL function `LALOverlapReductionFunction()`, which is documented in the LAL Software Documentation under the `stochastic` package.

#### Options

**-h**

Print a help message.

**-q**

Run silently (redirect standard input and error to `/dev/null`).

**-v**

Run in verbose mode.

**-d *debugLevel***

Set the LAL debug level to *debugLevel*.

**-s *siteID1* -t *siteID2***

Use detector sites identified by *siteID1* and *siteID2*; ID numbers between `LALNumCachedDetectors` (defined in the `tools` package of LAL) refer to detectors cached in the constant array `lalCachedDetectors[]`. (At this point, these are all interferometers.) Additionally, the five resonant bar detectors of the IGEC collaboration can be specified. The bar geometry data (summarized in table 3.2) is used by the function `LALCreateDetector()` from the `tools` package of LAL to generate the Cartesian position vector and response tensor which are used to calculate the overlap reduction function. The ID numbers for the bars depend on the value of `LALNumCachedDetectors`; the correct ID numbers can be obtained by with the command

```
./lalapps_olapredfcn -h
```

**-a *azimuth1* -b *azimuth2***

If *siteID1* (*siteID2*) is a bar detector, assume it has an azimuth of *azimuth1* (*azimuth2*) degrees East of North rather than the default IGEC orientation given in table 3.2. Note that this convention, measuring azimuth in degrees clockwise from North is not the same as that used in LAL (which comes from the frame spec). Note also that any specified azimuth angle is ignored if the corresponding detector is an interferometer.

**-f *fLow***

Begin the frequency series at a frequency of *fLow* Hz; if this is omitted, the default value of 0 Hz is used.

Name	Longitude	Latitude	Azimuth
AURIGA	11°56'54"E	45°21'12"N	N44°E
NAUTILUS	12°40'21"E	41°49'26"N	N44°E
EXPLORER	6°12'E	46°27'N	N39°E
ALLEGRO	91°10'43.766W	30°24'45.110N	N40°W
NIOBE	115°49'E	31°56'S	N0°E

Table 3.2: Location and orientation data for the five IGEC resonant bar detectors, stored in the `lalCachedBars[]` array. The data are taken from <http://igec.lnl.infn.it/cgi-bin/browser.pl?Level=0,3,1> except for the latitude and longitude of ALLEGRO, which were taken from Finn & Lazzarini, gr-qc/0104040. Note that the elevation above the WGS-84 reference ellipsoid and altitude angle for each bar is not given, and therefore set to zero.

**-e *deltaF***

Construct the frequency series with a frequency spacing of *deltaF* Hz

**-n *numPoints***

Construct a frequency series with *numPoints* points.

**-o *outfile***

Write the output to file *outfile*. The format of this file is that output by the routine `LALPrintFrequencySeries()` in the `support` package of LAL, which consists of a header describing metadata followed by two-column rows, each containing the doublet  $\{f, \gamma(f)\}$ .

**Example usage**

To compute the overlap reduction function for LIGO Hanford and LIGO Livingston, with a resolution of 1 Hz from 0 Hz to 1024 Hz:

```
lalapps_olapredfcn -s 0 -t 1 -e 1 -n 1025 -o LHOLLO.dat
```

To compute the overlap reduction function for ALLEGRO in its optimal orientation of 72°08 West of South (see Finn & Lazzarini, gr-qc/0104040) and LIGO Livingston, with a resolution of 0.5 Hz from 782.5 Hz to 1032 Hz (assuming `lalNumCachedBars` is 6):

```
lalapps_olapredfcn -s 9 -a 252.08 -t 1 -f 782.5 -e 0.5 -n 500 -o ALLEGROLHO.dat
```

**Author**

John T. Whelan

## 3.7 Program `lalapps_ring`

### Name

`lalapps_ring` — filters data through a bank of ringdown filters.

### Synopsis

```
lalapps_ring [-h] [-V] [-v] [-d dbglvl] [-i infile] [-o outfile] [-k] [-s] [-f framefile] [-r respfile] [-c channel] [-n numpoints] [-t starttime] [-b b0[,b1]] [-- filterparams]
```

### Description

`lalapps_ring` uses matched filtering to search for ringdown waveforms in gravitational wave data.

### Options

- h**  
Print a help message.
- V**  
Print the version information.
- v**  
Verbose output.
- d *dbglvl***  
Set LAL debug level to *dbglvl*.
- i *infile***  
Read filter parameters from input file *infile* [stdin].
- o *outfile***  
Write the output to file *outfile* [stdout].
- k**  
Keep filtering results (for use with option **-s**).
- s**  
Save intermediate filtering results as `.dat` and `snr-` files.
- f *framefile***  
Read channel data from *framefile* [`*.gwf`].
- r *respfile***  
Read response function data from *respfile* [`response.asc`].
- c *channel***  
Use channel *channel* data [H1:LSC-AS\_Q].
- n *numpoints***  
Use *numpoints* points of data [65536].
- t *starttime***  
Use data starting at GPS time *starttime* [start of frame data].

**-b *b0,b1***

Filter only template numbers *b0* to *b1* in bank [0,end of bank].

**-- *filterparams***

Specify filter parameters as command line arguments *filterparams* (see below for filter parameters).

**Filter parameters**

The filter parameters can be specified either on the command line as arguments following the `--` option or in a resource file that is input using the `-i` option (or from stdin). As a resource file, each option-value pair should have their own line.

**-segsz *npts***

Set the size of segments analyzed to *npts* points.

**-speclen *len***

Set the size of inverse spectrum truncation to *len* points [0].

**-flow *flow***

Set the low frequency cutoff to *flow* Hz.

**-fmin *fmin***

Set the minimum frequency for the bank to *fmin* Hz.

**-fmax *fmax***

Set the maximum frequency for the bank to *fmax* Hz.

**-qmin *qmin***

Set the minimum quality for the bank to *qmin*.

**-qmax *qmax***

Set the maximum quality for the bank to *qmax*.

**-maxmm *maxmm***

Set the maximum allowed mismatch for the bank to *maxmm*.

**-thresh *thresh***

Set the ringdown event signal-to-noise ratio threshold to *thresh*.

**-scale *scale***

Scale the response function by a dynamic range factor of *scale* [1].

**Debug levels**

The LAL debug level can be specified as an integer or as a string of flags:

**NDEBUG**

No debugging information is printed and memory debugging code is disabled.

**ERROR**

Error messages are printed.

**WARNING**

Warning messages are printed.

**INFO**

Information messages are printed.

**TRACE**

Function call tracing messages are printed.

**MEMINFO**

Memory allocation information messages are printed.

**MEMDBG**

Debugging of memory allocation routines is enabled but no messages are printed.

The following composite levels are available:

**MSGLVL1**

Equivalent to `ERROR`

**MSGLVL2**

Equivalent to `ERROR | WARNING`

**MSGLVL3**

Equivalent to `ERROR | WARNING | INFO`

**ALLDBG**

All debugging messages are printed.

For example, the command

```
lalapps_ring -d "ERROR | INFO" ...
```

will set the debug level so that error and information messages are printed.

**Environment****LAL\_DEBUG\_LEVEL**

Default LAL debug level to use.

**Author**

Jolien Creighton

## 3.8 Program `lalapps_trump`

### Name

`lalapps_trump` — postprocess data generated by the inspiral search code to determine an upper limit or perform detection.

### Synopsis

```
lalapps_trump --veto filename --trigger filename --times filename overlap [--detection] [--help]
```

### Description

`lalapps_trump` manipulates the results of the inspiral search code and veto data stored in XML format to produce lists of candidate events for individual interferometers. It can also be used to perform a coincidence analysis using the output written for each interferometer.

### Options

#### `--help`

Print a help message.

#### `--veto filename`

This is required. Name of the file containing metadata about the vetos to be applied to inspiral triggers. This file can contain multiple lines. Each line is a semi-colon separated list of the form:

```
name;filename;column;threshold;minusdtime;plusdtime;
```

Here *name* is a character string; *filename* is a character string naming the xml file with the veto triggers; *column* is the database column name to use for veto construction; *threshold* is a number (float), vetos have the value in *column* bigger than this; *minusdtime* and *plusdtime* give the interval of time (in seconds) which should be ignored before and after the veto trigger.

#### `--trigger filename`

This is required. Name of the file containing metadata about the inspiral triggers and simulated injections to determine the pipeline efficiency. This file contains a single, semi-colon separated line:

```
name;trigfile;injfile;snr*;chisq*;dtime;injlog;
```

Here *name* is a character string; *trigfile* is a character string naming the xml file with the inspiral triggers; *injfile* is a character string naming the xml file with the inspiral triggers when software injections are made; *snr\** is a number (float), triggers have *SNR* greater than this; *chisq\** is a number (float), triggers have *CHISQ* less than this; *dtime* is a number (float) giving the interval of time (in seconds) between threshold crossings for clustering; *injlog* is a character string naming the ascii file with the information about software injections.

#### `--detection`

Use this option if you want to run in detection mode. The code will terminate after generating a list of triggers.

#### `--times filename overlap`

Name of the ascii file with the list of data chunks analyzed. The expected format is three columns: (i) flag to indicate if it was analyzed or not, (ii) GPS start time, and (iii) GPS stop time. The floating point number *overlap* indicates how much data in seconds is ignored in each chunk. The code assumes that *overlap/2* is ignored at the beginning and end of each chunk.

**Example**

To run the program, type:

```
lalapps_trump --veto llvetoes --trigger llcand \  
  --times segment_S1_play_03.out 32.0 --detection
```

This command will look for information about vetoes in the file `llvetoes`, information about triggers in the file `llcand`, and information about the times analyzed in `segment_S1_play_03.out` with an overlap of 32 seconds. Since detection is turned on, a list of candidates is generated in a file called `triggers.dat` in the direction from which you execute the command.

**Uses**

This code uses LAL and the dataflow library. The code in `event_utils.c` includes some code written by Peter Shawhan to access the xml files.

**Author**

Patrick Brady

## 3.9 Program `lalapps_findchirp_post`

### Name

`lalapps_findchirp_post` — post process event files from findchirp

### Synopsis

```
lalapps_inspinj [-h] [-V] [-v] [-d dbglvl] [-m] [-s minsnr-maxsnr] [-c minchisq-maxchisq] [-b bins]
[-e eventfile]
```

### Description

`lalapps_inspinj` post process event files from findchirp.

### Options

**-h**

Print a help message.

**-V**

Print the version information.

**-v**

Verbose output.

**-d *dbglvl***

Set LAL debug level to *dbglvl*.

**-m**

Maximize events over injections. This options caused only the loudest event (in SNR) in each window of length  $100/\pi = 31.8309886183791$  to be processed. The window starts at the time of the first event.

**-s *minsnr-mmaxsnr***

Set minimum and maximum of the range of signal to noise ratio for creation of the histogram bins. (Default is 7.014.0.)

**-c *minchisq-mmaxchisq***

Set minimum and maximum of the range of chi squared veto statistic for creation of the histogram bins. (Default is 0.050.0.)

**-b *bins***

Set the number of bins in the two dimensional histogram. (Default is 10.)

**-e *eventfile***

Set the path to the file containing the list of inspiral events. (Default is `eventfile.dat`.) The event file must be of the format `END_TIME END_TIME_NS EFF_DISTANCE MASS1 MASS2 MCHIRP ETA SNR CHISQ SI`. Each event should be separated by a new line. Lines beginning with an octothorpe are ignored.

### Debug levels

The LAL debug level can be specified as an integer or as a string of flags:

**NDEBUG**

No debugging information is printed and memory debugging code is disabled.

**ERROR**

Error messages are printed.

**WARNING**

Warning messages are printed.

**INFO**

Information messages are printed.

**TRACE**

Function call tracing messages are printed.

**MEMINFO**

Memory allocation information messages are printed.

**MEMDBG**

Debugging of memory allocation routines is enabled but no messages are printed.

The following composite levels are available:

**MSGLVL1**

Equivalent to `ERROR`

**MSGLVL2**

Equivalent to `ERROR | WARNING`

**MSGLVL3**

Equivalent to `ERROR | WARNING | INFO`

**ALLDBG**

All debugging messages are printed.

For example, the command

```
lalapps_inspinj -d "ERROR | INFO"
```

will set the debug level so that error and information messages are printed.

**Environment****LAL\_DEBUG\_LEVEL**

Default LAL debug level to use.

**Author**

Duncan Brown

# Bibliography

[1] Einstein A (1905) “On the electrodynamics of moving bodies.” *Ann Phys Leipzig* **17** 891–921

KEY: Einstein:1905

[2] LIGO Scientific Collaboration *LAL Software Documentation*

KEY: LAL

ANNOTATION: Manual for the LSC Algorithm Library (LAL)

URL <http://www.lsc-group.phys.uwm.edu/lal>

[3] Press W H, Teukolsky S A, Vetterling W T, and Flannery B P (1992) *Numerical Recipes in C: the art of scientific computing* (Cambridge University Press, Cambridge, England), 2nd ed.

KEY: Press:1992

# Index

- [ALLDBG](#), 8
- [blank\\_status](#), 10
- [clear\\_status](#), 10
- Debug Level, 8
  - [ALLDBG](#), 8
  - [ERROR](#), 8
  - [INFO](#), 8
  - [lalDebugLevel](#), 8
  - [MEMDBG](#), 8
  - [MEMINFO](#), 8
  - [MSGLVL1](#), 8
  - [MSGLVL2](#), 8
  - [MSGLVL3](#), 8
  - [NDEBUG](#), 8
  - [TRACE](#), 8
  - [WARNING](#), 8
- Environment
  - [LAL\\_DEBUG\\_LEVEL](#), 8
- [ERROR](#), 8
- Error Handler, 13
  - [LAL\\_ERR\\_ABRT](#), 13
  - [LAL\\_ERR\\_DFLT](#), 13
  - [LAL\\_ERR\\_EXIT](#), 13
  - [LAL\\_ERR\\_RTRN](#), 13
- Function
  - [clear\\_status](#), 10
  - [lal\\_errhandler](#), 13
  - [set\\_debug\\_level](#), 8
- [hello.c](#), 15
- [INFO](#), 8
- [inspiral](#) (*module*), 31–37
  - [DataFindJob](#) (*class*), 31
    - [\\_\\_init\\_\\_](#) (*method*), 31
  - [DataFindNode](#) (*class*), 31–32
    - [\\_\\_init\\_\\_](#) (*method*), 32
    - [get\\_output](#) (*method*), 32
    - [set\\_end](#) (*method*), 32
    - [set\\_ifo](#) (*method*), 32
    - [set\\_start](#) (*method*), 32
  - [IncaJob](#) (*class*), 32–33
    - [\\_\\_init\\_\\_](#) (*method*), 32
  - [IncaNode](#) (*class*), 33
    - [\\_\\_init\\_\\_](#) (*method*), 33
    - [get\\_ifo\\_a](#) (*method*), 33
    - [get\\_ifo\\_b](#) (*method*), 33
    - [get\\_output](#) (*method*), 33
    - [set\\_ifo\\_a](#) (*method*), 33
    - [set\\_ifo\\_b](#) (*method*), 33
  - [InspiralError](#) (*class*), 33–34
    - [\\_\\_init\\_\\_](#) (*method*), 34
  - [InspiralJob](#) (*class*), 34
    - [\\_\\_init\\_\\_](#) (*method*), 34
  - [InspiralNode](#) (*class*), 34–35
    - [\\_\\_init\\_\\_](#) (*method*), 34
    - [get\\_output](#) (*method*), 34
    - [set\\_bank](#) (*method*), 35
  - [TmplBankJob](#) (*class*), 35
    - [\\_\\_init\\_\\_](#) (*method*), 35
  - [TmplBankNode](#) (*class*), 35–36
    - [\\_\\_init\\_\\_](#) (*method*), 36
    - [get\\_output](#) (*method*), 36
  - [TrigToTmplJob](#) (*class*), 36
    - [\\_\\_init\\_\\_](#) (*method*), 36
  - [TrigToTmplNode](#) (*class*), 36–37
    - [\\_\\_init\\_\\_](#) (*method*), 37
- [inspiral\\_pipeline.py](#), 39
- [LAL\\_CALL](#), 13
- [LAL\\_DEBUG\\_LEVEL](#), 8
- [LAL\\_ERR\\_ABRT](#), 13
- [LAL\\_ERR\\_DFLT](#), 13
- [LAL\\_ERR\\_EXIT](#), 13
- [LAL\\_ERR\\_RTRN](#), 13
- [lal\\_errhandler](#), 13
- [lal\\_errhandler\\_t](#), 13
- [lalapps\\_animate](#), 29

- [lalapps\\_binj](#), 66
  - [lalapps\\_burca](#), 64
  - [lalapps\\_findchirp\\_post](#), 75
  - [lalapps\\_hello](#), 27
  - [lalapps\\_inca](#), 45
  - [lalapps\\_inspinj](#), 48
  - [lalapps\\_inspinj\\_find](#), 52
  - [lalapps\\_inspiral](#), 43
  - [lalapps\\_inspmultiawg](#), 54
  - [lalapps\\_olapredfcn](#), 68
  - [lalapps\\_power](#), 58
  - [lalapps\\_ring](#), 70
  - [lalapps\\_snglBurstHistogram](#), 62
  - [lalapps\\_snglInspiralReader](#), 50
  - [lalapps\\_tmpltbank](#), 41
  - [lalapps\\_trump](#), 73
  - [lalDebugLevel](#), 8
- Macro
- [LAL\\_CALL](#), 13
  - [PRINT\\_VERSION](#), 12
  - [RCSID](#), 11
- [MEMDBG](#), 8
  - [MEMINFO](#), 8
  - [MSGLVL1](#), 8
  - [MSGLVL2](#), 8
  - [MSGLVL3](#), 8
  - [NDEBUG](#), 8
- pipeline (*module*), 17–25
- [AnalysisChunk](#) (*class*), 17
    - [\\_\\_init\\_\\_](#) (*method*), 17
    - [\\_\\_len\\_\\_](#) (*method*), 17
    - [\\_\\_repr\\_\\_](#) (*method*), 17
    - [dur](#) (*method*), 17
    - [end](#) (*method*), 17
    - [start](#) (*method*), 17
  - [AnalysisJob](#) (*class*), 17–18
    - [\\_\\_init\\_\\_](#) (*method*), 18
    - [calibration](#) (*method*), 18
    - [channel](#) (*method*), 18
    - [get\\_config](#) (*method*), 18
  - [AnalysisNode](#) (*class*), 18–19
    - [\\_\\_init\\_\\_](#) (*method*), 18
    - [get\\_end](#) (*method*), 18
    - [get\\_ifo](#) (*method*), 18
    - [get\\_input](#) (*method*), 18
    - [get\\_output](#) (*method*), 18
    - [get\\_start](#) (*method*), 18
    - [set\\_cache](#) (*method*), 18
    - [set\\_end](#) (*method*), 18
    - [set\\_ifo](#) (*method*), 19
    - [set\\_input](#) (*method*), 19
    - [set\\_output](#) (*method*), 19
    - [set\\_start](#) (*method*), 19
- [CondorDAG](#) (*class*), 19
  - [\\_\\_init\\_\\_](#) (*method*), 19
  - [add\\_node](#) (*method*), 19
  - [set\\_dag\\_file](#) (*method*), 19
  - [write\\_dag](#) (*method*), 19
  - [write\\_sub\\_files](#) (*method*), 19
- [CondorDAGError](#) (*class*), 19–20
- [CondorDAGJob](#) (*class*), 20
  - [\\_\\_init\\_\\_](#) (*method*), 20
  - [add\\_var\\_arg](#) (*method*), 20
  - [add\\_var\\_opt](#) (*method*), 20
- [CondorDAGNode](#) (*class*), 20–21
  - [\\_\\_init\\_\\_](#) (*method*), 20
  - [\\_\\_repr\\_\\_](#) (*method*), 20
  - [add\\_parent](#) (*method*), 20
  - [add\\_var\\_arg](#) (*method*), 21
  - [add\\_var\\_opt](#) (*method*), 21
  - [job](#) (*method*), 21
  - [set\\_log\\_file](#) (*method*), 21
  - [set\\_name](#) (*method*), 21
  - [set\\_retry](#) (*method*), 21
  - [write\\_job](#) (*method*), 21
  - [write\\_parents](#) (*method*), 21
  - [write\\_vars](#) (*method*), 21
- [CondorDAGNodeError](#) (*class*), 21
- [CondorError](#) (*class*), 21–22
  - [\\_\\_init\\_\\_](#) (*method*), 22
- [CondorJob](#) (*class*), 22–23
  - [\\_\\_init\\_\\_](#) (*method*), 22
  - [add\\_arg](#) (*method*), 22
  - [add\\_condor\\_cmd](#) (*method*), 22
  - [add\\_ini\\_opts](#) (*method*), 22
  - [add\\_opt](#) (*method*), 22
  - [get\\_stderr\\_file](#) (*method*), 22
  - [get\\_stdout\\_file](#) (*method*), 22
  - [get\\_sub\\_file](#) (*method*), 23
  - [set\\_log\\_file](#) (*method*), 23
  - [set\\_notification](#) (*method*), 23
  - [set\\_stderr\\_file](#) (*method*), 23
  - [set\\_stdout\\_file](#) (*method*), 23
  - [set\\_sub\\_file](#) (*method*), 23
  - [write\\_sub\\_file](#) (*method*), 23
- [CondorJobError](#) (*class*), 23

- CondorSubmitError (*class*), 23–24
- s2play (*function*), 17
- ScienceData (*class*), 24
  - `__getitem__` (*method*), 24
  - `__init__` (*method*), 24
  - `__len__` (*method*), 24
  - `__repr__` (*method*), 24
  - `make_chunks` (*method*), 24
  - `read` (*method*), 24
- ScienceSegment (*class*), 24–25
  - `__getitem__` (*method*), 24
  - `__init__` (*method*), 24
  - `__len__` (*method*), 24
  - `__repr__` (*method*), 24
  - `add_chunk` (*method*), 24
  - `dur` (*method*), 25
  - `end` (*method*), 25
  - `id` (*method*), 25
  - `make_chunks` (*method*), 25
  - `start` (*method*), 25
  - `unused` (*method*), 25
- SegmentError (*class*), 25
  - `__init__` (*method*), 25
- PRINT.VERSION, 12
- Program
  - `inspiral_pipeline.py`, 39
  - `lalapps_animate`, 29
  - `lalapps_binj`, 66
  - `lalapps_burca`, 64
  - `lalapps_findchirp_post`, 75
  - `lalapps_hello`, 27
  - `lalapps_inca`, 45
  - `lalapps_inspinj`, 48
  - `lalapps_inspinj_find`, 52
  - `lalapps_inspiral`, 43
  - `lalapps_inspmultiawg`, 54
  - `lalapps_olapredfcn`, 68
  - `lalapps_power`, 58
  - `lalapps_ring`, 70
  - `lalapps_snglBurstHistogram`, 62
  - `lalapps_snglInspiralReader`, 50
  - `lalapps_tmpltbank`, 41
  - `lalapps_trump`, 73
- RCSID, 11
- `rctsid`, 11
- `set_debug_level`, 8
- Source
  - `hello.c`, 15
- TRACE, 8
- Type
  - `lal_errhandler_t`, 13
- Variable
  - `blank_status`, 10
  - `lalDebugLevel`, 8
  - `rctsid`, 11
  - `vrblvl`, 13
- `vrblvl`, 13
- WARNING, 8